UNIVERSIDAD
NACIONAL
DE COLOMBIA

Bitácora de ejecución

Propuesta Redes neuronales orientadas a el transporte de manizales

JULIÁN DAVID PULGARÍN PIRAZÁN

Director(a):

PhD. Ingeniería y Néstor Darío Duque Méndez

Universidad Nacional de Colombia

Facultad de Administración, Departamento de Informática y Computación

Manizales, Colombia

2023

Historia de revisiones

Versión	Autor(es)	Descripción	Fecha
1.0	Julian David Pulgarin	Creación del documento	10 de octubre de 2023
2.0	Julian David Pulgarin	Ejecución de pruebas con cambios en los hiperparametros	30 de octubre de 2023
3.0	Julian David Pulgarin	Ejecución de pruebas con cambios en el data set	9 de noviembre de 2023
4.0	Julian David Pulgarin	Ejecución de pruebas con cambios en los hiperparametros y el data	20 de noviembre de 2023

Índice

1. Introducción

Esta bitácora registra el proceso de pruebas, ajustes y resultados obtenidos mediante la implementación y evaluación de una red neuronal diseñada para abordar y generar predicciones en el sistema de transporte público en la ciudad de Manizales. El propósito principal de esta bitácora es documentar detalladamente todas las modificaciones realizadas a los hiperparámetros de la red neuronal, así como cualquier cambio significativo efectuado en el conjunto de datos utilizado para entrenar y validar el modelo. Cada entrada en esta bitácora estará asociada con una corrida específica de la red neuronal. Se registrarán los cambios en los hiperparámetros, arquitectura de la red, técnicas de regularización, estrategias de optimización, entre otros aspectos relevantes que puedan influir en el rendimiento del modelo, además, se documentaron los cambios en el conjunto de datos, como la adición de nuevas características o modificaciones en la estructura, con el objetivo de mantener un registro claro de la evolución del conjunto de datos a lo largo de las pruebas. Los resultados obtenidos de cada ejecución de la red neuronal, incluyendo métricas de rendimiento, como precisión, pérdida, tiempos de convergencia, entre otros, serán detalladamente registrados y analizados para comprender el impacto de los ajustes realizados.

Esta bitácora servirá como guía y referencia para entender el progreso, los desafíos encontrados y las lecciones aprendidas durante el desarrollo y la optimización de la red neuronal orientada al transporte público de la ciudad de Manizales.

2. bitácora Pruebas de ejecución de la red neuronal

Fecha: 10/10/2023
Ejecución No.:1
Detalles de la Prueba:
Descripción: En esta ejecución se realizó una prueba inicial con un conjunto de datos que no fue normalizado previamente. Se utilizaron columnas específicas del conjunto de datos para la predicción de la demanda en el transporte público de la ciudad de Manizales. Se empleó una red neuronal con una estructura específica y se configuraron los hiperparámetros sin ajustes posteriores.

Dataset Utilizado: Conjunto de datos original sin normalizar.

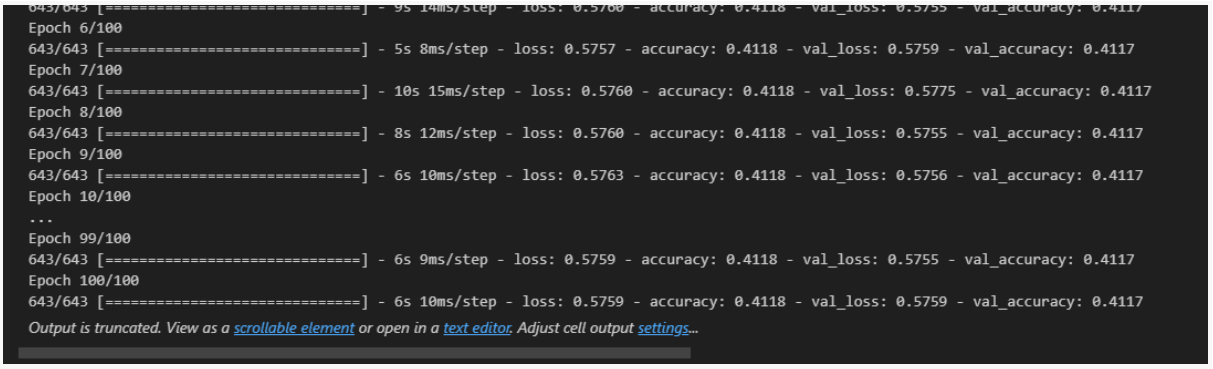
Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:



.....

Fecha: 10/10/2023
Ejecución No.:2
Detalles de la Prueba:
Descripción: En esta ejecución se realizó una prueba inicial con un conjunto de datos que no fue normalizado previamente. Se utilizaron columnas específicas del conjunto de datos para la predicción de la demanda en el transporte público de la ciudad de Manizales. Se empleó una red neuronal con una estructura específica y se configuraron los hiperparámetros sin ajustes posteriores.

Dataset Utilizado: Conjunto de datos original sin normalizar.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```python
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=4, activation='sigmoid'))
model.add(Dense(units=2, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
643/643 [==============================] - 4s 6ms/step - loss: 0.5808 - accuracy: 0.4118 - val_loss: 0.5757 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 4s 6ms/step - loss: 0.5812 - accuracy: 0.4118 - val_loss: 0.5760 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5815 - accuracy: 0.4118 - val_loss: 0.5762 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5814 - accuracy: 0.4118 - val_loss: 0.5831 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5805 - accuracy: 0.4118 - val_loss: 0.5770 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5817 - accuracy: 0.4118 - val_loss: 0.5814 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5819 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 13/100
...
643/643 [==============================] - 4s 7ms/step - loss: 0.5814 - accuracy: 0.4118 - val_loss: 0.5754 - val_accuracy: 0.4117
Epoch 85/100
643/643 [==============================] - 5s 8ms/step - loss: 0.5786 - accuracy: 0.4118 - val_loss: 0.5798 - val_accuracy: 0.4117
Epoch 86/100
```

Fecha: 10/10/2023
Ejecución No.:3
Detalles de la Prueba:
Descripción: En esta ejecución se realizó una prueba inicial con un conjunto de datos que no fue normalizado previamente. Se utilizaron columnas específicas del conjunto de datos para la predicción de la demanda en el transporte público de la ciudad de Manizales. Se empleó una red neuronal con una estructura específica y se configuraron los hiperparámetros sin ajustes posteriores.

Dataset Utilizado: Conjunto de datos original sin normalizar.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:

X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
643/643 [==============================] - 4s 6ms/step - loss: 0.5808 - accuracy: 0.4118 - val_loss: 0.5757 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 4s 6ms/step - loss: 0.5812 - accuracy: 0.4118 - val_loss: 0.5760 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5815 - accuracy: 0.4118 - val_loss: 0.5762 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5814 - accuracy: 0.4118 - val_loss: 0.5831 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5805 - accuracy: 0.4118 - val_loss: 0.5770 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5817 - accuracy: 0.4118 - val_loss: 0.5814 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5819 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 13/100
...
Epoch 99/100
643/643 [==============================] - 6s 9ms/step - loss: 0.5816 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 100/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5831 - accuracy: 0.4118 - val_loss: 0.5782 - val_accuracy: 0.4117
```

—----------

Fecha: 10/10/2023
Ejecución No.:4
Detalles de la Prueba:
Descripción: En esta ejecución se realizó una prueba inicial con un conjunto de datos que no fue normalizado previamente. Se utilizaron columnas específicas del conjunto de datos para la predicción de la demanda en el transporte público de la ciudad de Manizales. Se empleó una red neuronal con una estructura específica y se configuraron los hiperparámetros sin ajustes posteriores.

Dataset Utilizado: Conjunto de datos original sin normalizar.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=10, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
643/643 [==============================] - 4s 6ms/step - loss: 0.5808 - accuracy: 0.4118 - val_loss: 0.5757 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 4s 6ms/step - loss: 0.5812 - accuracy: 0.4118 - val_loss: 0.5760 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5815 - accuracy: 0.4118 - val_loss: 0.5762 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5814 - accuracy: 0.4118 - val_loss: 0.5831 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5805 - accuracy: 0.4118 - val_loss: 0.5770 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5817 - accuracy: 0.4118 - val_loss: 0.5814 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5819 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 13/100
...
Epoch 99/100
643/643 [==============================] - 6s 9ms/step - loss: 0.5816 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 100/100
643/643 [==============================] - 4s 7ms/step - loss: 0.5831 - accuracy: 0.4118 - val_loss: 0.5782 - val_accuracy: 0.4117
```

—----------

Fecha: 30/10/2023
Ejecución No.:5
Detalles de la Prueba:
Descripción: Esta ejecución se llevó a cabo como continuación de las pruebas anteriores. Se incorporó una nueva columna ('PromPasDia') al conjunto de datos sin

realizar previamente la normalización de los datos. Se utilizaron columnas específicas del conjunto de datos para la predicción de la demanda en el transporte público de la ciudad de Manizales. Se modificó la arquitectura de la red neuronal y se ajustaron los hiper parámetros con valores diferentes a la ejecución anterior.

Dataset Utilizado: Conjunto de datos actualizado con una nueva columna ('PromPasDia') sin normalizar..

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta','PromPasDia' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta","PromPasDia"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
643/643 [==============================] - 17s 27ms/step - loss: 0.5891 - accuracy: 0.4118 - val_loss: 0.6122 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 23s 36ms/step - loss: 0.5919 - accuracy: 0.4118 - val_loss: 0.5800 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 23s 36ms/step - loss: 0.5912 - accuracy: 0.4118 - val_loss: 0.6042 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 22s 34ms/step - loss: 0.5996 - accuracy: 0.4118 - val_loss: 0.5926 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 24s 37ms/step - loss: 0.5894 - accuracy: 0.4118 - val_loss: 0.5797 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 26s 40ms/step - loss: 0.5933 - accuracy: 0.4118 - val_loss: 0.5977 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 19s 29ms/step - loss: 0.5917 - accuracy: 0.4118 - val_loss: 0.5923 - val_accuracy: 0.4117
Epoch 13/100
...
Epoch 99/100
643/643 [==============================] - 10s 16ms/step - loss: 0.5892 - accuracy: 0.4118 - val_loss: 0.6413 - val_accuracy: 0.4117
Epoch 100/100
643/643 [==============================] - 9s 14ms/step - loss: 0.5871 - accuracy: 0.4118 - val_loss: 0.5774 - val_accuracy: 0.4117
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fecha: 30/10/2023
Ejecución No.:6
Detalles de la Prueba:
Descripción: Esta ejecución es una continuación de las pruebas anteriores. Se realizó el ajuste de los hiperparámetros, estableciendo un coeficiente de aprendizaje de 0.01 y modificando la arquitectura de la red neuronal. Además, se normalizó el conjunto de datos utilizando la técnica Min-Max.

Dataset Utilizado: Conjunto de datos actualizado con una nueva columna ('PromPasDia') normalizado con la técnica Min-Max..

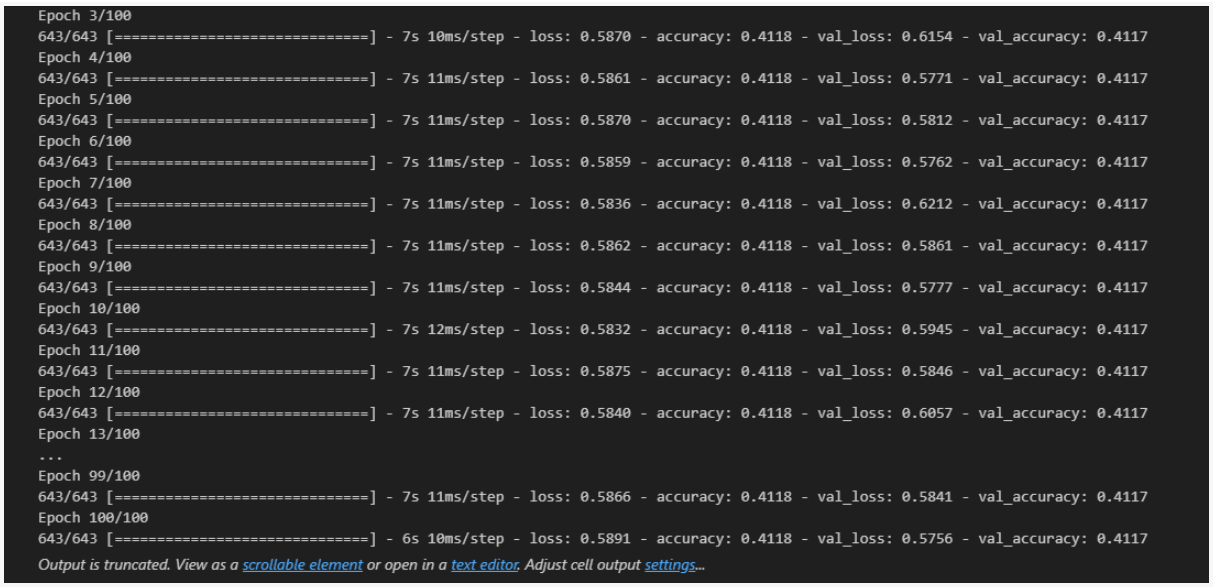
Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta','PromPasDia' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta","PromPasDia"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=6, activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:



—----------

Fecha: 9/11/2023
Ejecución No.:7
Detalles de la Prueba:
Descripción: En esta ejecución se realizaron cambios significativos en la arquitectura de la red neuronal. Se normalizó el conjunto de datos utilizando la técnica Min-Max y se modificó la función de activación de las capas ocultas por 'ReLU'..

Dataset Utilizado: Conjunto de datos actualizado con una nueva columna ('PromPasDia') normalizado con la técnica Min-Max.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta','PromPasDia' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta","PromPasDia"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```python
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
643/643 [==============================] - 5s 8ms/step - loss: 0.5810 - accuracy: 0.4118 - val_loss: 0.5774 - val_accuracy: 0.4117
Epoch 6/100
643/643 [==============================] - 5s 8ms/step - loss: 0.5808 - accuracy: 0.4118 - val_loss: 0.5755 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 6s 10ms/step - loss: 0.5809 - accuracy: 0.4118 - val_loss: 0.5757 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 5s 7ms/step - loss: 0.5814 - accuracy: 0.4118 - val_loss: 0.5758 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 5s 7ms/step - loss: 0.5815 - accuracy: 0.4118 - val_loss: 0.5769 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 5s 7ms/step - loss: 0.5811 - accuracy: 0.4118 - val_loss: 0.5756 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 5s 8ms/step - loss: 0.5804 - accuracy: 0.4118 - val_loss: 0.5756 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 6s 9ms/step - loss: 0.5824 - accuracy: 0.4118 - val_loss: 0.5774 - val_accuracy: 0.4117
Epoch 13/100
...
Epoch 99/100
643/643 [==============================] - 6s 9ms/step - loss: 0.5806 - accuracy: 0.4118 - val_loss: 0.5768 - val_accuracy: 0.4117
Epoch 100/100
643/643 [==============================] - 6s 9ms/step - loss: 0.5812 - accuracy: 0.4118 - val_loss: 0.5783 - val_accuracy: 0.4117
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fecha: 9/11/2023
Ejecución No.:8
Detalles de la Prueba:
Descripción: En esta ejecución se realizaron ajustes en los hiperparámetros de la red neuronal. Se aumentó el número de neuronas en las capas ocultas a 20 cada una y se modificó el coeficiente de aprendizaje a 0.001. Además, se mantuvo la función de activación 'ReLU' en las capas ocultas.

Dataset Utilizado: Conjunto de datos actualizado con una nueva columna ('PromPasDia') normalizado con la técnica Min-Max.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta','PromPasDia' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta",”PromPasDia”]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='sigmoid'))
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=1, activation='linear'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.1)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
Epoch 6/100
643/643 [==============================] - 4s 6ms/step - loss: 0.4819 - accuracy: 0.4118 - val_loss: 0.5248 - val_accuracy: 0.4117
Epoch 7/100
643/643 [==============================] - 4s 6ms/step - loss: 0.4802 - accuracy: 0.4118 - val_loss: 0.4867 - val_accuracy: 0.4117
Epoch 8/100
643/643 [==============================] - 4s 6ms/step - loss: 0.4832 - accuracy: 0.4118 - val_loss: 0.5028 - val_accuracy: 0.4117
Epoch 9/100
643/643 [==============================] - 4s 6ms/step - loss: 0.4787 - accuracy: 0.4118 - val_loss: 0.4908 - val_accuracy: 0.4117
Epoch 10/100
643/643 [==============================] - 3s 5ms/step - loss: 0.4790 - accuracy: 0.4118 - val_loss: 0.4929 - val_accuracy: 0.4117
Epoch 11/100
643/643 [==============================] - 3s 5ms/step - loss: 0.4793 - accuracy: 0.4118 - val_loss: 0.4970 - val_accuracy: 0.4117
Epoch 12/100
643/643 [==============================] - 3s 5ms/step - loss: 0.4820 - accuracy: 0.4118 - val_loss: 0.5027 - val_accuracy: 0.4117
Epoch 13/100
...
Epoch 99/100
643/643 [==============================] - 5s 8ms/step - loss: 0.4820 - accuracy: 0.4118 - val_loss: 0.4812 - val_accuracy: 0.4117
Epoch 100/100
643/643 [==============================] - 5s 8ms/step - loss: 0.4845 - accuracy: 0.4118 - val_loss: 0.4845 - val_accuracy: 0.4117
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

—----------

Fecha: 20/11/2023
Ejecución No.:9
Detalles de la Prueba:
Descripción: En esta ejecución se realizaron ajustes en los hiperparámetros de la red neuronal. Se aumentó el número de neuronas en las capas ocultas a 30 cada una y se modificó el coeficiente de aprendizaje a 0.01. Además, se mantuvo la función de activación 'ReLU' en las capas ocultas, en la de entrada y salida

Dataset Utilizado: Conjunto de datos actualizado con una nueva columna ('PromPasDia') normalizado con la técnica Min-Max.

Columnas Utilizadas: 'Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', 'DistRut_Km', 'TimeRuta','PromPasDia' (Variables predictoras), 'demanda' (Variable objetivo).

División en X e Y:
X = datos[['Ruta', 'Hora', 'Dia', 'Cond_Ruta', '#Veh_Disp_x_rut', "DistRut_Km", "TimeRuta","PromPasDia"]]
Y = datos['demanda']

Estructura de la Red Neuronal y entrenamiento:

```
model = Sequential()
#Se agrega capa inicial con 5 Neuronas funcion de activacion " sigmoid "
model.add(Dense(units=8, input_dim=8 , activation='relu'))
model.add(Dense(units=30, activation='relu'))
model.add(Dense(units=30, activation='relu'))
model.add(Dense(units=1, activation='relu'))

# Crea un optimizador Adam con  tasa de aprendizaje = 0.01
custom_optimizer = Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error', optimizer=custom_optimizer, metrics=['accuracy'])

##model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test))
model.fit(X_train, Y_train, epochs=1000, batch_size=32, verbose=0)

#Se Guarda el porcentaje de perdida para graficar

history = model.fit(X_train, Y_train, epochs=1000, batch_size=32, validation_data=(X_test, Y_test))
```

Resultados Obtenidos:

```
567/567 [==============================] - 3s 6ms/step - loss: 0.0977 - accuracy: 0.3000 - val_loss: 0.0788 - val_accuracy: 0.3000
Epoch 9/100
567/567 [==============================] - 3s 6ms/step - loss: 0.1081 - accuracy: 0.3000 - val_loss: 0.0877 - val_accuracy: 0.3000
Epoch 10/100
567/567 [==============================] - 3s 6ms/step - loss: 0.1089 - accuracy: 0.3000 - val_loss: 0.1136 - val_accuracy: 0.3000
Epoch 11/100
567/567 [==============================] - 4s 6ms/step - loss: 0.1001 - accuracy: 0.3000 - val_loss: 0.0830 - val_accuracy: 0.3000
Epoch 12/100
567/567 [==============================] - 4s 6ms/step - loss: 0.0979 - accuracy: 0.3000 - val_loss: 0.0838 - val_accuracy: 0.3000
Epoch 13/100
...
Epoch 99/100
567/567 [==============================] - 3s 6ms/step - loss: 0.0903 - accuracy: 0.3000 - val_loss: 0.0696 - val_accuracy: 0.3000
Epoch 100/100
567/567 [==============================] - 3s 5ms/step - loss: 0.0932 - accuracy: 0.3000 - val_loss: 0.0743 - val_accuracy: 0.3000
```