Fidel Alejandro Navarro Salazar

217202813

Seminario de Física Computacional

Proyecto Red Neuronal RNN

En este trabajo se entrenó una red neuronal tipo RNN con la finalidad de realizar predicciones de la temperatura máxima de la ciudad de Canberra, Australia. Para la construcción y entrenamiento de la red neuronal se utilizó la biblioteca de Tensorflow en Python.

Los datos utilizados se obtuvieron de Kaggle[1]. Estos corresponden a las temperaturas máximas de la ciudad de Canberra entre los años 2008 y 2017. Debido a que el conjunto de datos corresponde a distintas ciudades de Australia fue necesario procesar la información presentada. Se decidió utilizar los datos de la ciudad de Canberra debido a que era la ciudad con mayor información disponible. Si se desea conocer la metodología seguida para el procesamiento de los datos se aconseja revisar la primera sección del código adjunto.

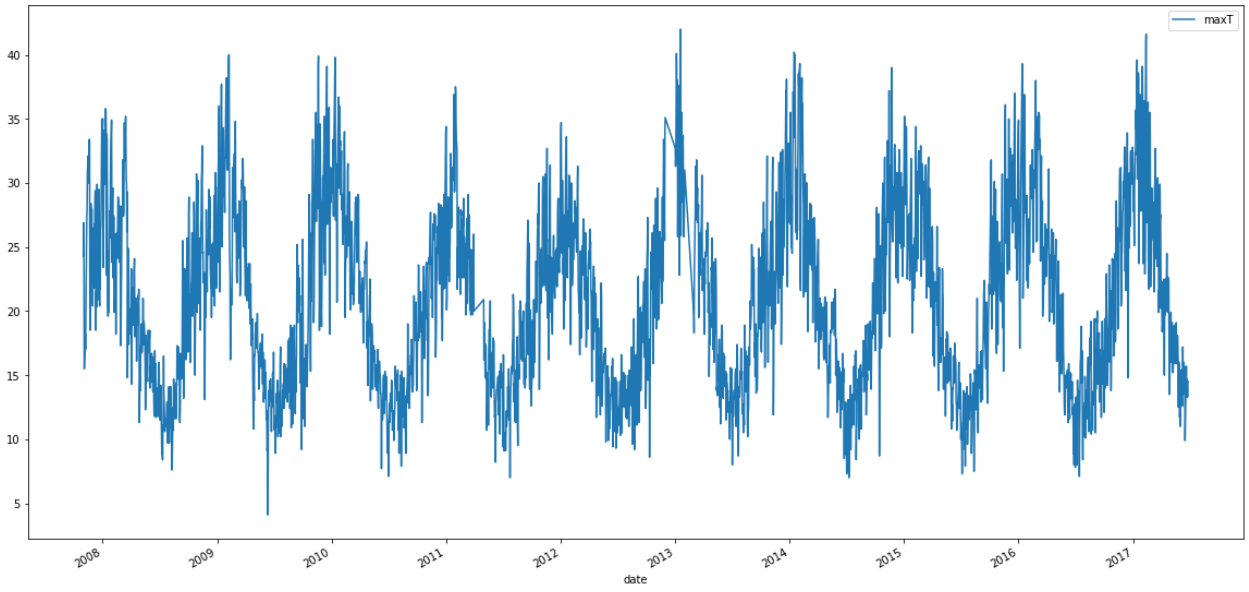

Fig 1: Gráfica de la temperatura máxima de la ciudad de Canberra entre 2008 y 2017.

Los datos utilizados están conformados por las temperaturas de 3427 días. Los datos fueron separados en dos conjuntos de datos: el primero corresponde al arreglo de entrenamiento con 80% de los datos, y el segundo al arreglo de validación con el 20% de los datos.

Para la definición del modelo base se utilizó un timestamp de 10, y un epoch de 200.

```
n_timestamp = 10
train_days = int(len(df2)*0.80)
test_days = int(len(df2)-train_days)
epoch = 200
```

Fig 2: Información del timestamp, epoch y número de dias de entrenamiento.

Posteriormente, se definió el modelo a utilizar. Para el modelo se utilizó una capa de LSTM normal (long short term memory) con función de activación relu (línea rectificadora) y dropout de 0.2.

```
#Definimos el modelo
model = Sequential()
model.add(LSTM(units = 50, activation = "relu", input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 50)                10400
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
```

Fig 3: Definición del modelo.

```
#Comenzaremos el entrenamiento
model.compile(optimizer = "adam", loss = "mean_squared_error")
history = model.fit(
    x_train, y_train, epochs = epoch, batch_size = 32,
    shuffle = False, validation_data = (x_test, y_test))
loss = history.history["loss"]
val_loss = history.history['val_loss']
epochs = range(len(loss))
```

Fig 4: Parámetros para entrenar el modelo.

Para el entrenamiento del modelo se utilizó el optimizador Adam y un tamaño (batch_size) de 32.

Para el entrenamiento de la temperatura máxima se obtuvo $R^2 = 0.81$ y un error cuadrático medio de 6.11. Del error cuadrático medio y la grafica de dispersión f) podemos observar que los valores predecidos presentan un ligero comportamiento lineal cuando se compara con los valores reales, sin embargo, estos presentan una dispersión notoria. Por otro lado, se observa de la gráfica b) que el modelo fue capaz de predecir el comportamiento de los datos.

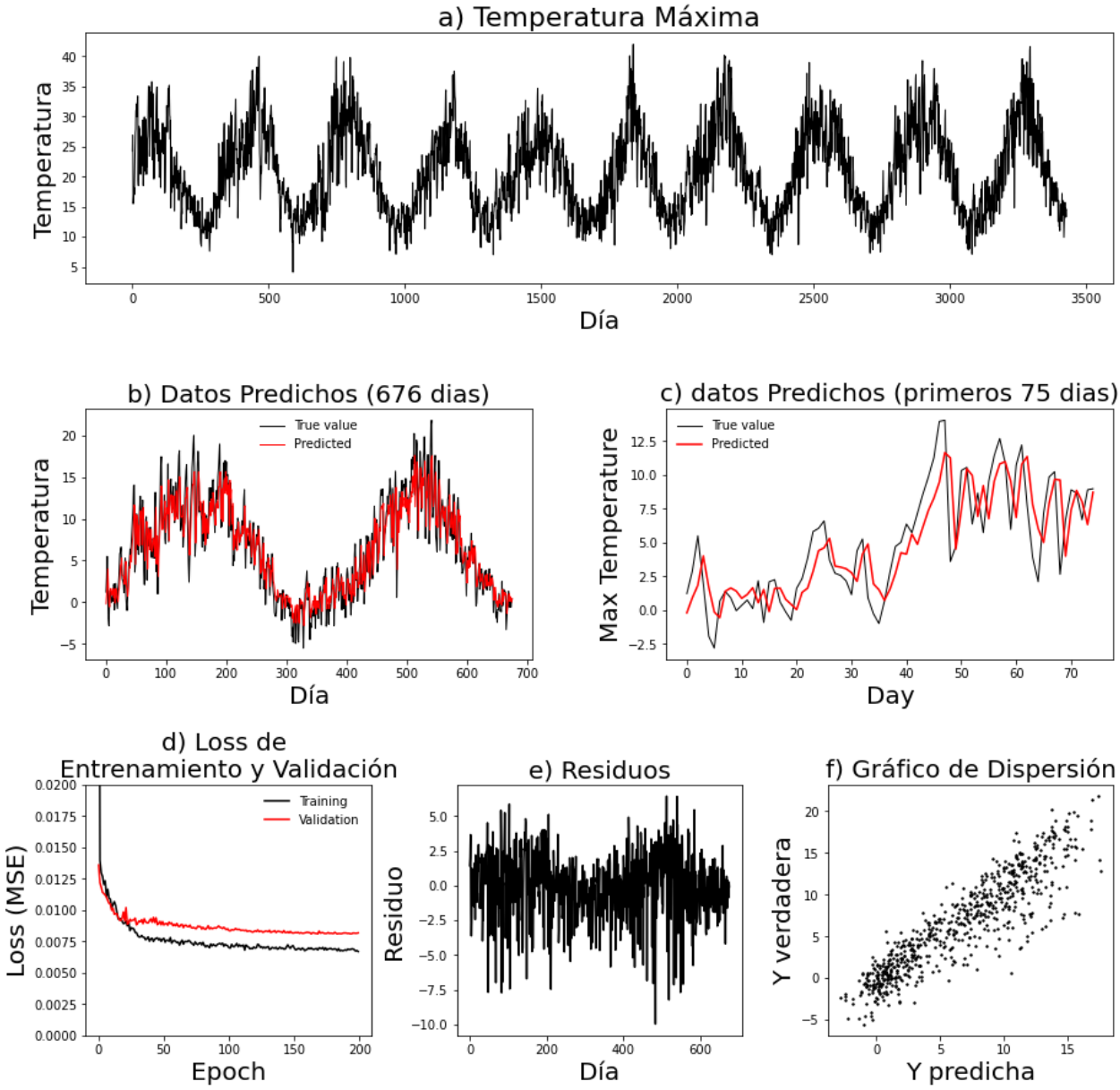

Fig 5: Resultados del entrenamiento del modelo. a) Valores reales de la temperatura máxima. b) Comportamiento de los datos predichos en comparación a los esperados para 676 días. c) Comparación de los primeros 75 datos predichos con los esperados. d) Comportamiento del loss para el entrenamiento y validación. e) Comportamiento de los residuos. f) Dispersión de los datos predichos junto a los reales.

Comparando los datos más de cerca, podemos visualizar en la gráfica c) que los datos siguen el comportamiento esperado; sin embargo, las magnitudes se encuentran por debajo de las esperadas. Esto se ve reforzado con la gráfica e) Residuos, donde podemos observar que la diferencia entre las magnitudes de los valores predichos con los reales es de +/- 5 grados. Por último, se observa de la gráfica d) que el loss se vuelve asintótico después de 200 epochs.

En resumen, el modelo base presentó un comportamiento aceptable y se buscará en las secciones posteriores mejorar los resultados realizando las siguientes variaciones:

1) Cambio de la configuración de la red neuronal a una LSTM Bidireccional.
2) Cambio de la configuración de la red neuronal a una LSTM Stacked.
3) Implementar el optimizador RMSprop.
4) Cambiar el tamaño (batch_size) de 32 a 64.

Variación 1: Red LSTM Bidirecional

Se modificó la capa del modelo, cambiando la configuración de la red neuronal a una bidireccional.

```
#Definimos el modelo
model = Sequential()
model.add(Bidirectional(LSTM(units = 50, activation = "relu"), input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 100)               20800
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 20,901
Trainable params: 20,901
Non-trainable params: 0
_________________________________________________________________
```

Fig 6: Modelo con configuración bidirecional.

Para el entrenamiento del modelo se utilizaron los mismos parámetros. Los resultados obtenidos se presentan en la figura 7.

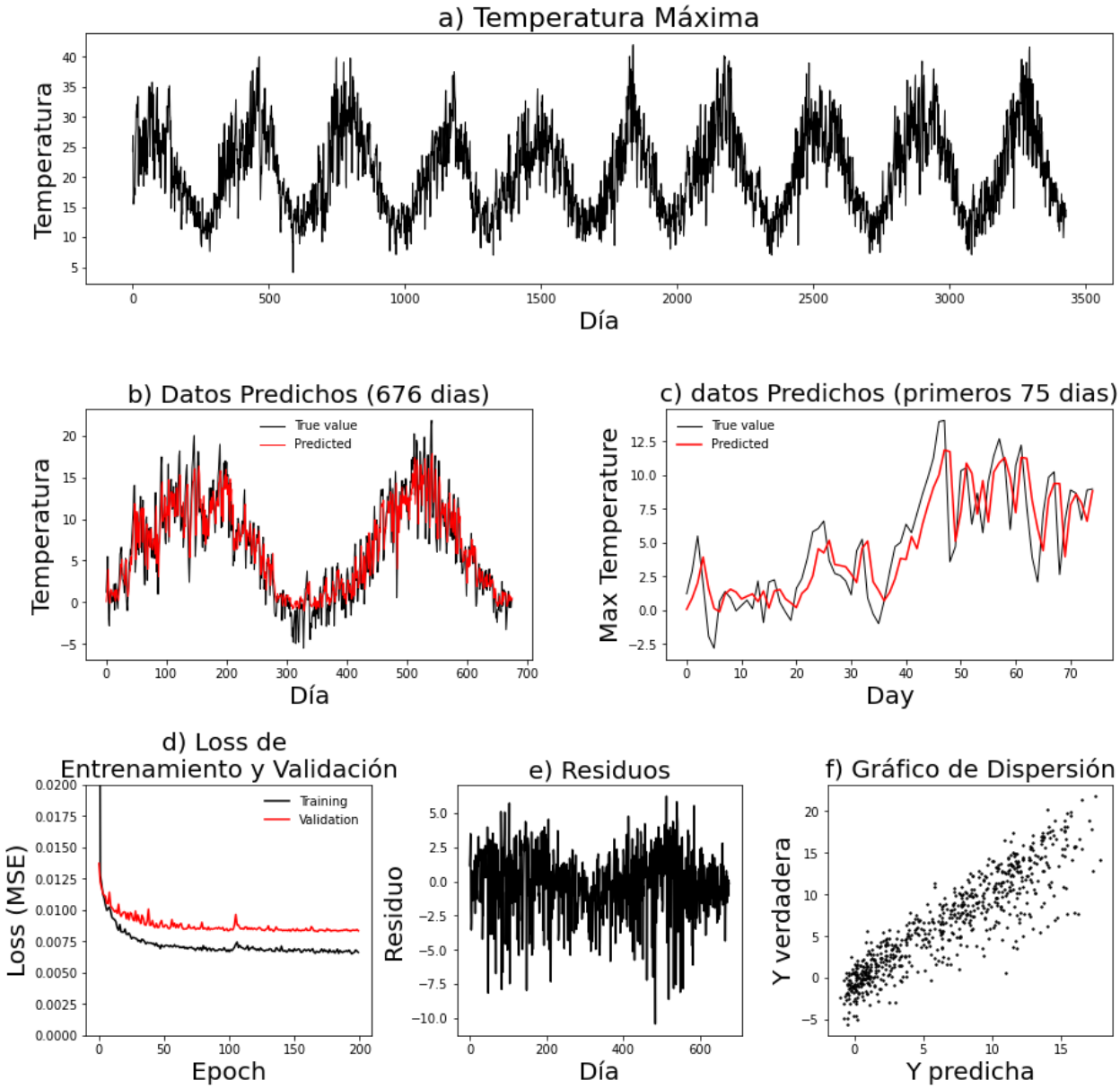

Fig 7: : Resultados del entrenamiento del modelo con red bidirecional. a) Valores reales de la temperatura máxima. b) Comportamiento de los datos predichos en comparación a los esperados para 676 días. c) Comparación de los primeros 75 datos predichos con los esperados. d) Comportamiento del loss para el entrenamiento y validación. e) Comportamiento de los residuos. f) Dispersión de los datos predichos junto a los reales.

Comparando los resultados de las figuras 5 y 7 no se muestra ninguna diferencia apreciable. Las diferencias más notorias se encuentran en la gráfica f), donde la red bidireccional mostró una mayor dispersión de los datos con error cuadrático medio de 6.22 y $R^2 = 0.81$.

Variación 2: Red LSTM Stacked

Se modificó la red neuronal agregando una capa adicional. Debido a que la diferencia entre el modelo base y la variación 1 no fue apreciable se entrenaron dos modelos, el primero con dos capas con configuración LSTM normal, y el segundo con dos capas con configuración LSTM bidireccional.

El modelo se entrenó utilizando los mismo parámetros del modelo base.

```
#Definimos el modelo
model = Sequential()
model.add(LSTM(units = 50, activation = "relu",
               return_sequences=True,
               input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, activation="relu"))
model.add(Dense(units = 1))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
lstm (LSTM)                  (None, 10, 50)            10400
_________________________________________________________________
dropout (Dropout)            (None, 10, 50)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
_________________________________________________________________
```

A)

```
#Definimos el modelo
model = Sequential()
model.add(Bidirectional(LSTM(units = 50, activation = "relu",
                             return_sequences=True),
                        input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(units = 50, activation="relu")))
model.add(Dense(units = 1))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
bidirectional (Bidirectional (None, 10, 100)           20800
_________________________________________________________________
dropout (Dropout)            (None, 10, 100)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 100)               60400
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 81,301
Trainable params: 81,301
Non-trainable params: 0
_________________________________________________________________
```

B)

Fig 8: Modelos con red LSTM Stacked. A) Modelo con dos capas LSTM normales.

B) Modelo con dos capas LSTM bidireccionales.

Los resultados para la configuración con dos capas LSTM normales pueden ser observados en la figura 9; se obtuvo una $R^2 = 0.81$ y un error cuadrático medio de 6.19. Por otro lado, los resultados de la configuración con dos capas LSTM bidireccionales se observan en la figura 10; se obtuvo una $R^2 = 0.81$ y un error cuadrático medio de 6.36.

Debido a que no se presentó ninguna diferencia significativa entre los modelos de la variación 1 (capa bidireccional), variación 2 (dos capas normales/bidireccionales) y el modelo base (capa normal) se decidió continuar con las variaciones haciendo uso del modelo base. Esta decisión fue realizada tomando en cuenta que el modelo base presentó el menor error cuadrático medio (6.11).

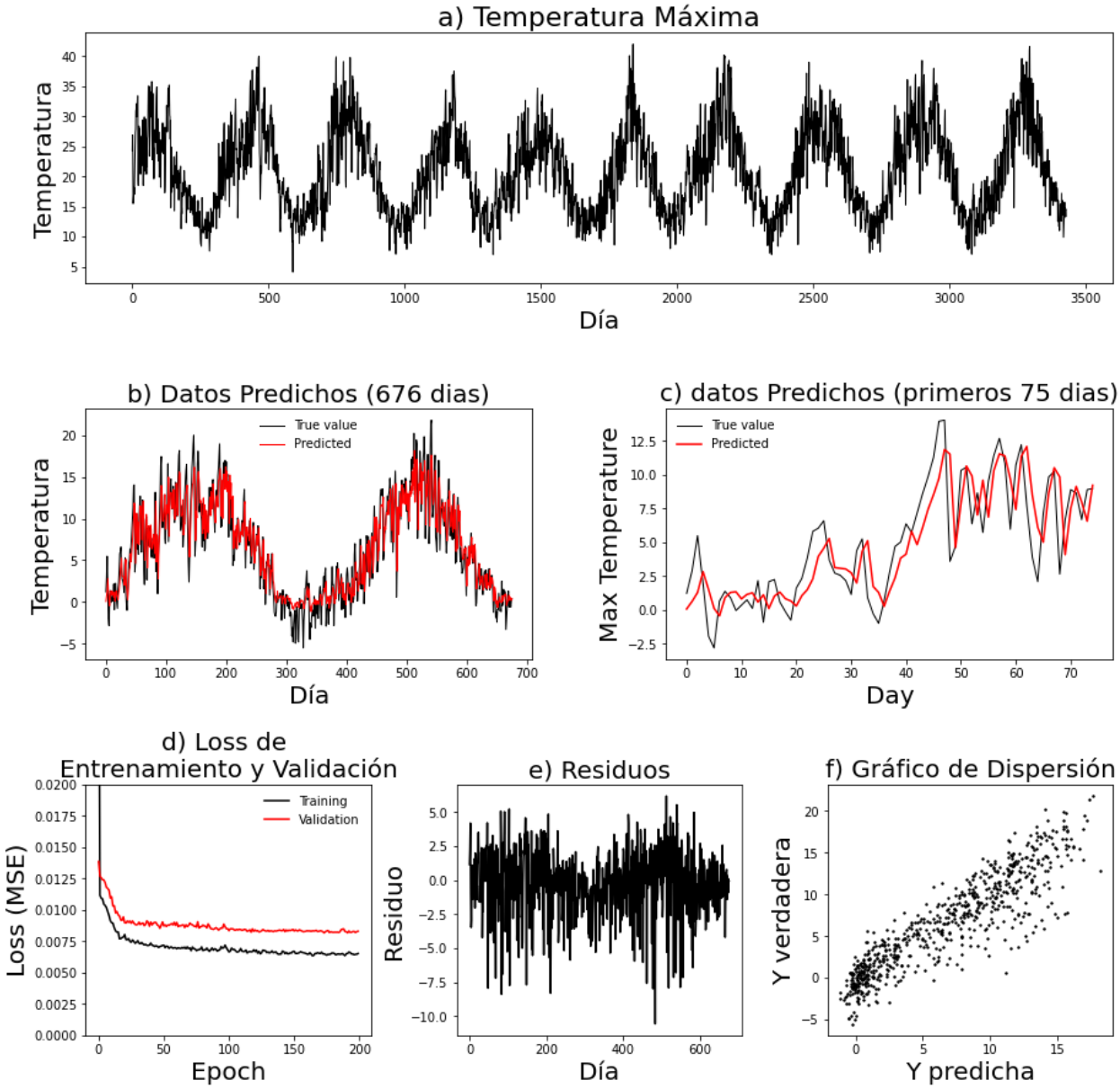

Fig 9: Modelo Stacked con capas normales.

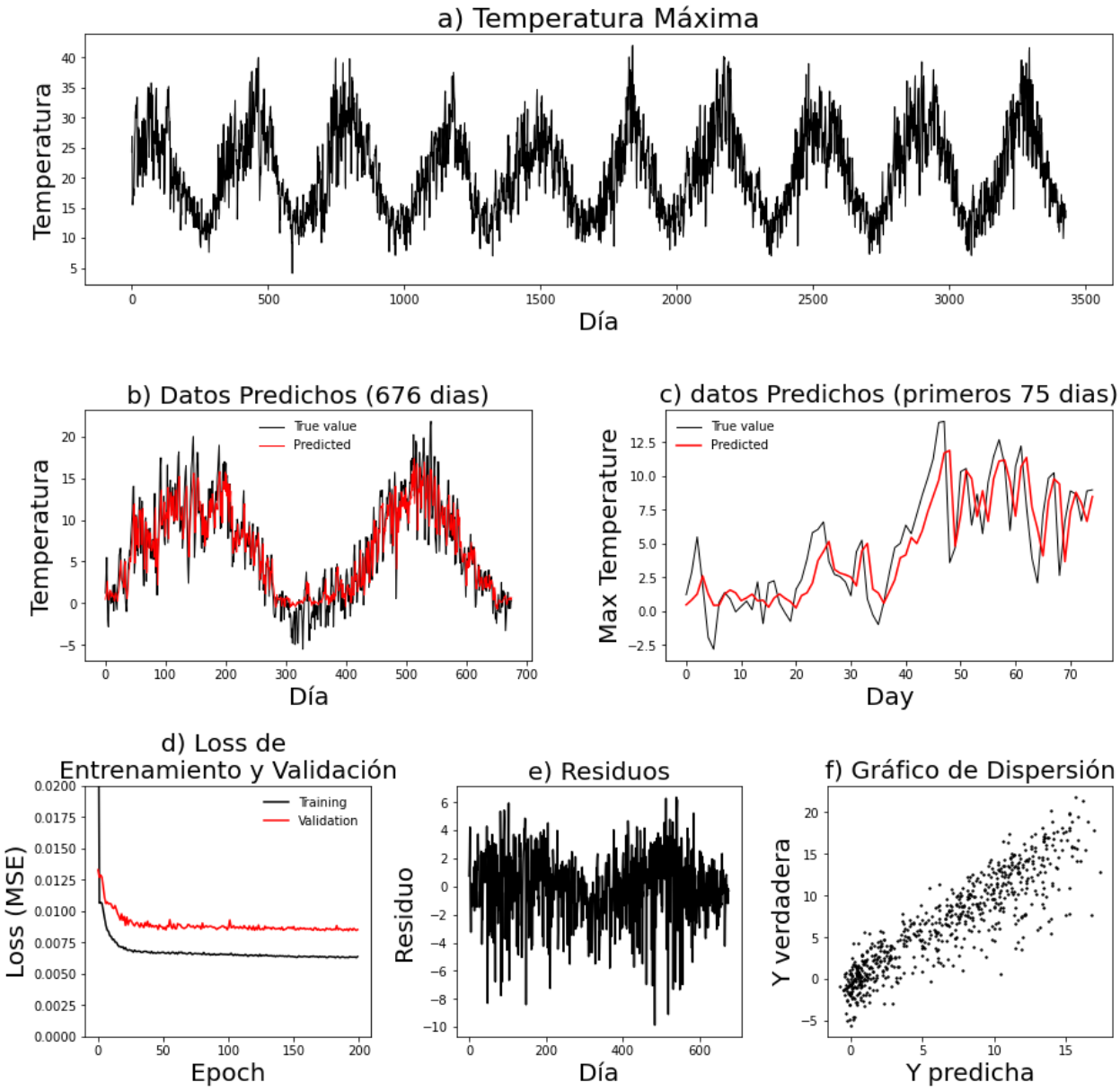

Fig 10: Modelo Stacked con capas bidireccionales.

Variación 3: Optimizador RMSprop

En esta variación del modelo se sustituyó el optimizador Adam por el RMSprop. Para el entrenamiento se utilizaron los mismos parámetros del modelo base. Los resultados del modelo se encuentran en la figura 12.

```
#Comenzaremos el entrenamiento
model.compile(optimizer = "RMSprop", loss = "mean_squared_error")
history = model.fit(
    x_train, y_train, epochs = epoch, batch_size = 32,
    shuffle = False, validation_data = (x_test, y_test))
loss = history.history["loss"]
val_loss = history.history['val_loss']
epochs = range(len(loss))
```

Fig 11: Implementación del optimizador RMSprop.

Para este modelo se obtuvo una $R^2 = 0.8$ y un error cuadrático medio de 6.48. Con el uso del optimizador RMSprop aumentó la dispersión de los datos en comparación con el modelo base que hace uso del optimizador Adam. Adicionalmente, se observó un aumento en el tiempo de entrenamiento al usar el optimizador RMSprop, con 52.46 segundos del optimizador Adam en comparación a 70.004 segundos del optimizador RMSprop.

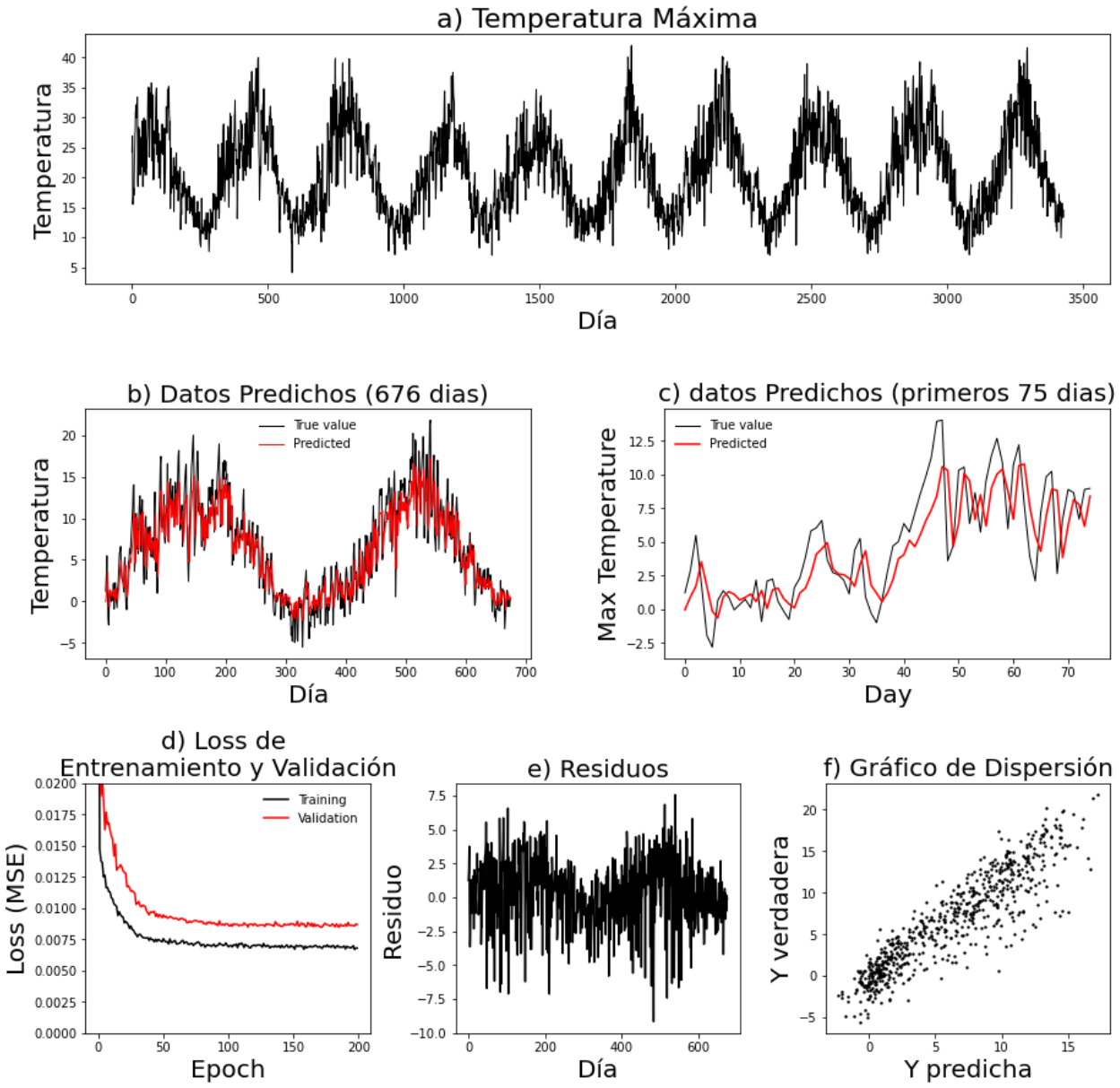

Fig. 11: Modelo con optimizador RMSprop.

Variación 4: Batch Size

Previamente en el modelo base se hizo uso de un batch size de 32, para esta variación se utilizará un batch size de 64. Esta variación tiene la finalidad de observar el comportamiento del modelo al aumentar el número de datos utilizados por ciclo. Para el entrenamiento se utilizó la configuración del modelo base. Los resultados del modelo se encuentran en la figura 13.

```
#Comenzaremos el entrenamiento
model.compile(optimizer = "adam", loss = "mean_squared_error")
history = model.fit(
    x_train, y_train, epochs = epoch, batch_size = 64,
    shuffle = False, validation_data = (x_test, y_test))
loss = history.history["loss"]
val_loss = history.history['val_loss']
epochs = range(len(loss))
```

Fig 12: Entrenamiento del modelo con batch size de 64.

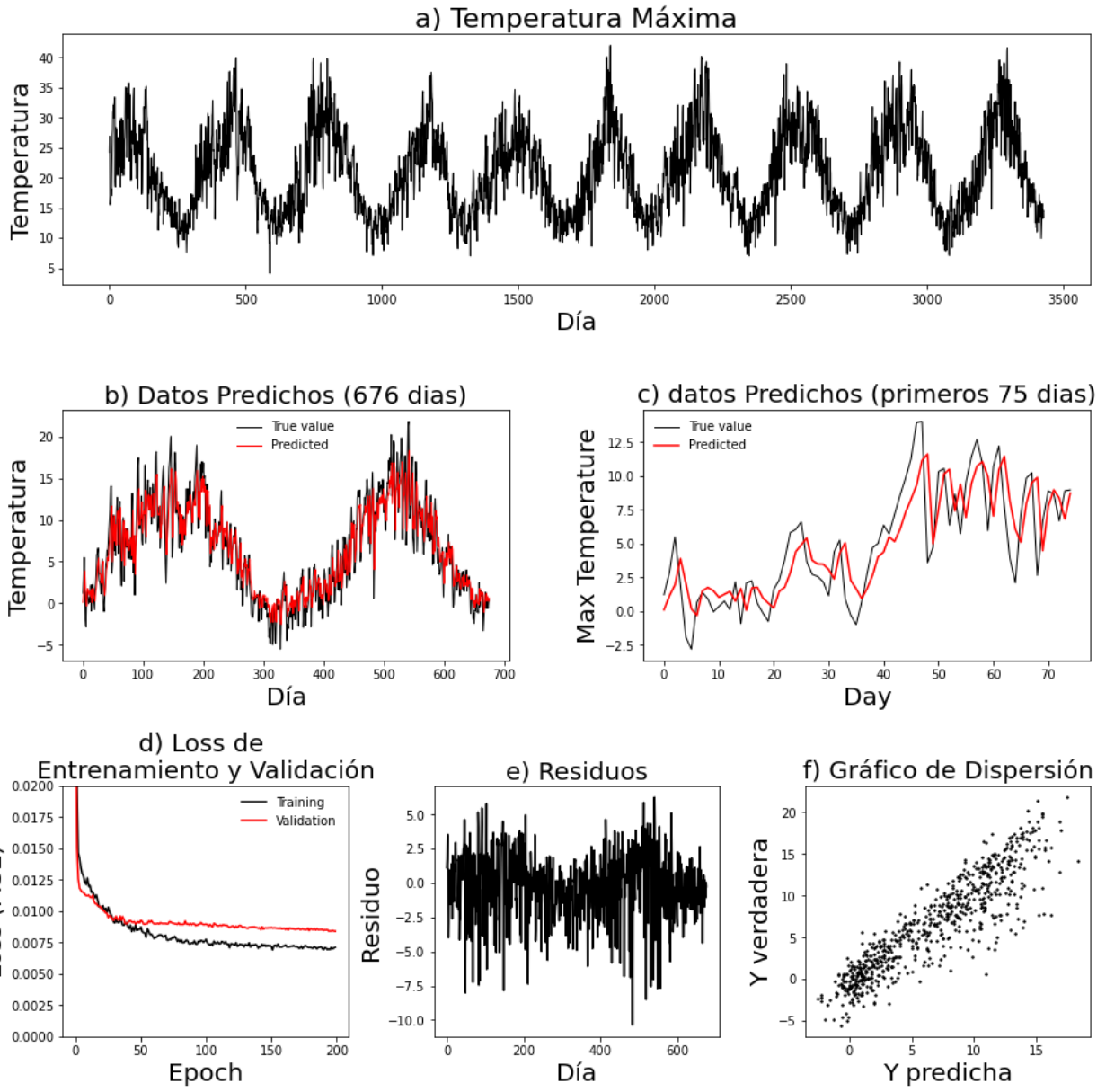

Fig 13: Resultados del modelo con batch size de 64.

Para este modelo se obtuvo una $R^2 = 0.81$ y un error cuadrático medio de 6.26. A pesar de aumentar la dispersión de los datos se observa una ligera disminución del loss para los datos de validación. Sin embargo, esta pequeña diferencia no es suficiente para justificar la preferencia del modelo con batch size 64 sobre el modelo base.

Conclusión

El modelo utilizado para el entrenamiento de la red neuronal consiste en los siguientes parámetros:

a) Modelo con una capa LSTM normal
b) Dropout = 0.2
c) Función de activación Relu
d) Epoch = 200
e) Batch size = 32
f) Optimizador Adam

Este modelo presentó una $R^2 = 0.81$ y un error cuadrático medio de 6.11; es decir, fue capaz de predecir el 81% de los datos de prueba. El modelo es útil para obtener una estimación del comportamiento de la temperatura máxima en los próximos días en la ciudad de Canberra. Sin embargo, debido a que presenta una dispersión alta y una R^2 por debajo del 90% se recomienda utilizar un mayor número de datos en futuros entrenamientos.

Referencias

1) Kaggle. “Rain in Australia”. (2020, 11 diciembre). Kaggle. https://www.kaggle.com/jsphyg/weather-dataset-rattle-package