# Extensión del hielo marino en el Hemisferio Sur.

- Integrantes: Valentina Aranda M.
  Valentina Astudillo J.
  Sebastian Lillo O.
- Profesor: Francisco Plaza

Jueves 29 Agosto, 2024

# Tabla de contenidos.

# Introducción.

¿Qué nos interesa estudiar?

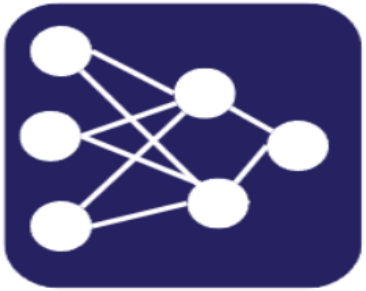
¿ Cómo lo llevaremos a cabo?

# Implementación.

MODELO UNIVARIADO+ UNISTEP Y MODELO UNIVARIADO + MULTISTEP

**X (entrada):**

6 meses de media mensual de la extensión del hielo



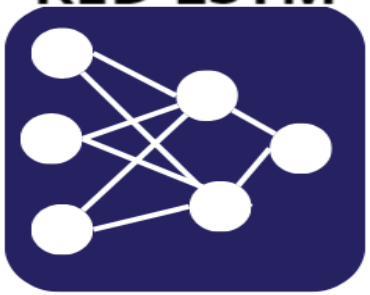
**RED LSTM**

**Y (salida):**

Extensión del hielo en el siguiente mes.

**X (entrada):**

12 meses de media mensual de la extensión del hielo



**RED LSTM**

**Y (salida):**

Extensión del hielo en los siguietnes 6 meses.

# Sets de Entrenamiento validación y prueba

## Sets de entrenamiento, validación y prueba

El objetivo de la partición del dataset en los subsets de entrenamiento validación y prueba es poder no sólo entrenar la Red LSTM sino validarla correctamente (verificando que no haya *underfitting* u *overfitting*).

```python
# Función para generar las particiones preservando las características
# de la serie de tiempo

def train_val_test_split(serie, tr_size=0.8, vl_size=0.1, ts_size=0.1 ):
    # Definir número de datos en cada subserie
    N = serie.shape[0]
    Ntrain = int(tr_size*N)  # Número de datos de entrenamiento
    Nval = int(vl_size*N)    # Número de datos de validación
    Ntst = N - Ntrain - Nval # Número de datos de prueba

    # Realizar partición
    train = serie[0:Ntrain]
    val = serie[Ntrain:Ntrain+Nval]
    test = serie[Ntrain+Nval:]

    return train, val, test
```

# Dataset supervisado

El objetivo de esta fase es ajustar nuestros sets de entrenamiento, validación y prueba al formato requerido por la Red LSTM para realizar el entrenamiento y posteriormente las predicciones.

```python
def crear_dataset_supervisado(array, input_length, output_length):

    # Inicialización
    X, Y = [], []    # Listados que contendrán los datos de entrada y salida del modelo
    shape = array.shape
    if len(shape)==1: # Si tenemos sólo una serie (univariado)
        fils, cols = array.shape[0], 1
        array = array.reshape(fils,cols)
    else: # Multivariado
        fils, cols = array.shape

    # Generar los arreglos
    for i in range(fils-input_length-output_length):
        X.append(array[i:i+INPUT_LENGTH,0:cols])
        Y.append(array[i+input_length:i+input_length+output_length,-1].reshape(output_length,1))

    # Convertir listas a arreglos de NumPy
    X = np.array(X)
    Y = np.array(Y)

    return X, Y
```

# Creación de dataset supervisado

- Redes LSTM, debemos garantizar que las variables que alimentan el modelo se encuentran en el mismo rango de valores. Esto facilitará la convergencia del algoritmo de optimización usado durante el entrenamiento, lo que a su vez mejorará las predicciones obtenidas con el modelo entrenado.

```
# Definición de los hiperparámetros INPUT_LENGTH y OUTPUT_LENGTH
INPUT_LENGTH = 6   # Registros de 6 meses consecutivos a la entrada
OUTPUT_LENGTH = 1    # El modelo va a predecir 1 mes a futuro


x_tr, y_tr = crear_dataset_supervisado(tr.values, INPUT_LENGTH, OUTPUT_LENGTH)
x_vl, y_vl = crear_dataset_supervisado(vl.values, INPUT_LENGTH, OUTPUT_LENGTH)
x_ts, y_ts = crear_dataset_supervisado(ts.values, INPUT_LENGTH, OUTPUT_LENGTH)

print('Tamaños entrada (BATCHES x INPUT_LENGTH x FEATURES) y de salida (BATCHES x OUTPUT_LENGTH x FEATURES)')
print(f'Set de entrenamiento - x_tr: {x_tr.shape}, y_tr: {y_tr.shape}')
print(f'Set de validación - x_vl: {x_vl.shape}, y_vl: {y_vl.shape}')
print(f'Set de prueba - x_ts: {x_ts.shape}, y_ts: {y_ts.shape}')
```

```
Tamaños entrada (BATCHES x INPUT_LENGTH x FEATURES) y de salida (BATCHES x OUTPUT_LENGTH x FEATURES)
Set de entrenamiento - x_tr: (317, 6, 1), y_tr: (317, 1, 1)
Set de validación - x_vl: (33, 6, 1), y_vl: (33, 1, 1)
Set de prueba - x_ts: (35, 6, 1), y_ts: (35, 1, 1)
```

```
Set de entrenamiento - x_tr: (306, 12, 1), y_tr: (306, 6, 1)
Set de validación - x_vl: (22, 12, 1), y_vl: (22, 6, 1)
Set de prueba - x_ts: (24, 12, 1), y_ts: (24, 6, 1)
```

# Escalamiento de los datos

- Antes de entrenar cualquier modelo de *Deep Learning*, incluyendo las Redes LSTM, debemos garantizar que las variables que alimentan el modelo se encuentran en el mismo rango de valores. Esto facilitará la convergencia del algoritmo de optimización usado durante el entrenamiento, lo que a su vez mejorará las predicciones obtenidas con el modelo entrenado.

```python
# Escalamiento del dataset con la función anterior

# Crear diccionario de entrada
data_in = {
    'x_tr': x_tr, 'y_tr': y_tr,
    'x_vl': x_vl, 'y_vl': y_vl,
    'x_ts': x_ts, 'y_ts': y_ts,
}

# Y escalar
data_s, scaler = escalar_dataset(data_in)

# Extraer subsets escalados
x_tr_s, y_tr_s = data_s['x_tr_s'], data_s['y_tr_s']
x_vl_s, y_vl_s = data_s['x_vl_s'], data_s['y_vl_s']
x_ts_s, y_ts_s = data_s['x_ts_s'], data_s['y_ts_s']
```

```
Min x_tr/x_vl/x_ts sin escalamiento: 3.09/3.09/3.09
Min x_tr/x_vl/x_ts con escalamiento: -0.9999999999999999/-0.9999999999999999/-0.9999999999999999

Min y_tr/y_vl/y_ts sin escalamiento: 3.09/3.09/3.09
Min y_tr/y_vl/y_ts con escalamiento: -0.9999999999999999/-0.9999999999999999/-0.9999999999999999
```

# Descripción del proceso de entrenamiento

## TABLA COMPARATIVA

| CARACTERÍSTICAS | UNISTEP | MULTISTEP |
|---|---|---|
| EPOCAS | 100 | 100 |
| FUNCIÓN DE PERDIDA | ECM | ECM |
| GPU / NO GPU | GPU | GPU |
| PLATAFORMA | GOOGLE-COLAB | GOOGLE COLAB |
| TIEMPO DE COMPUTO | 1 MIN APROX | 1 MIN 1/2 APROX |
| | | |

# Código

```python
# Creación del modelo
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import RMSprop
import tensorflow as tf

# Ajustar parámetros para reproducibilidad del entrenamiento
tf.random.set_seed(123)
tf.config.experimental.enable_op_determinism()

# El modelo
N_UNITS = 30  # Tamaño del estado oculto (h) y de la celdad de memoria (c)
INPUT_SHAPE = (x_tr_s.shape[1], x_tr_s.shape[2]) # 6 (medias mensuales) x 1 (feature)

modelo = Sequential()
modelo.add(LSTM(N_UNITS, input_shape=INPUT_SHAPE))
modelo.add(Dense(OUTPUT_LENGTH, activation='linear')) # activation = 'linear' pues queremos pronosticar (

# Pérdida: se usará el RMSE (root mean squared error) para el entrenamiento
# pues permite tener errores en las mismas unidades de los km
def root_mean_squared_error(y_true, y_pred):
    rmse = tf.math.sqrt(tf.math.reduce_mean(tf.square(y_pred-y_true)))
    return rmse

```

# Código

```
# Compilación
optimizador = RMSprop(learning_rate=5e-5)
modelo.compile(
    optimizer = optimizador,
    loss = root_mean_squared_error,
)

# Entrenamiento (aproximadamente 1 min usando GPU)
EPOCHS = 100 # Hiperparámetro
BATCH_SIZE = 16 y 64 para multistep # Hiperparámetro
historia = modelo.fit(
    x = x_tr_s,
    y = y_tr_s,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_data = (x_vl_s, y_vl_s),
    verbose=2
```

# Resultado Entrenamiento

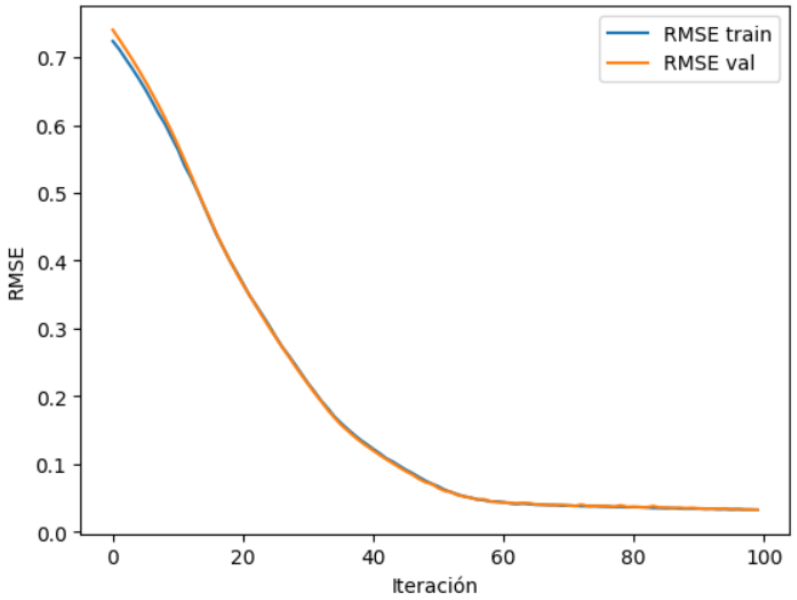
UNISTEP

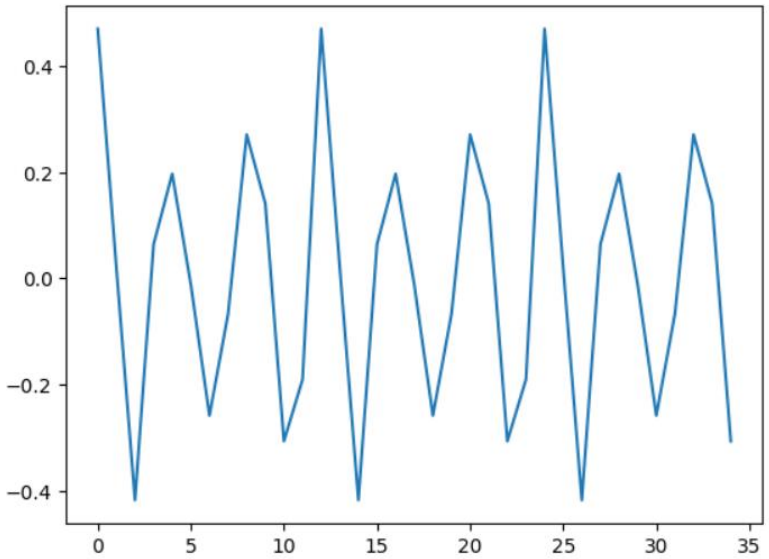
```
Epoch 1/100
20/20 - 10s - loss: 0.7236 - val_loss: 0.7402
Epoch 2/100
20/20 - 0s - loss: 0.7114 - val_loss: 0.7260
Epoch 3/100
20/20 - 0s - loss: 0.6970 - val_loss: 0.7113
Epoch 4/100
20/20 - 0s - loss: 0.6830 - val_loss: 0.6964
Epoch 98/100
20/20 - 0s - loss: 0.0329 - val_loss: 0.0332
Epoch 99/100
20/20 - 0s - loss: 0.0330 - val_loss: 0.0329
Epoch 100/100
20/20 - 0s - loss: 0.0327 - val_loss: 0.0327
```

MULTISTEP

```
Epoch 1/100
39/39 - 5s - 131ms/step - loss: 0.7158 - val_loss: 0.7135
Epoch 2/100
39/39 - 1s - 14ms/step - loss: 0.7043 - val_loss: 0.7020
Epoch 3/100
39/39 - 1s - 17ms/step - loss: 0.6924 - val_loss: 0.6895
Epoch 4/100
39/39 - 1s - 16ms/step - loss: 0.6792 - val_loss: 0.6753
Epoch 98/100
39/39 - 0s - 10ms/step - loss: 0.0127 - val_loss: 0.0151
Epoch 99/100
39/39 - 0s - 9ms/step - loss: 0.0125 - val_loss: 0.0149
Epoch 100/100
39/39 - 0s - 10ms/step - loss: 0.0123 - val_loss: 0.0148
```

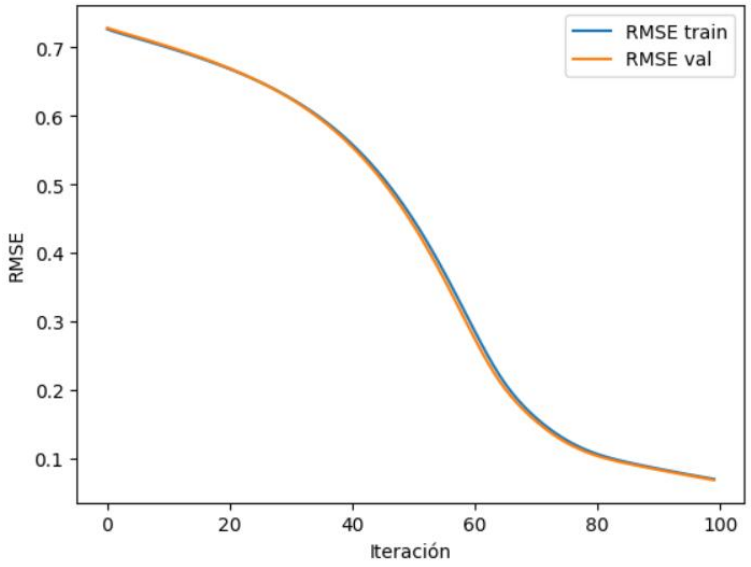
# Validación Unistep



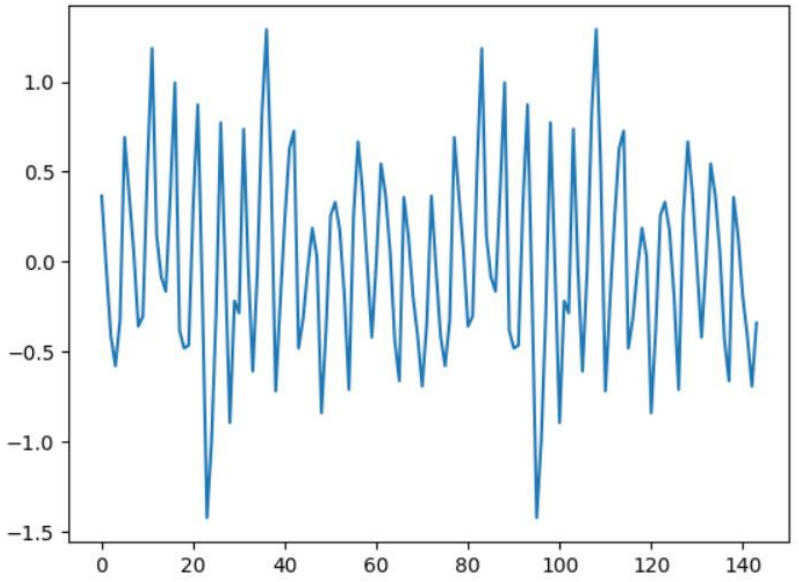

```
Comparativo desempeños:
  RMSE train:    0.032
  RMSE val:      0.033
  RMSE test:     0.032
```

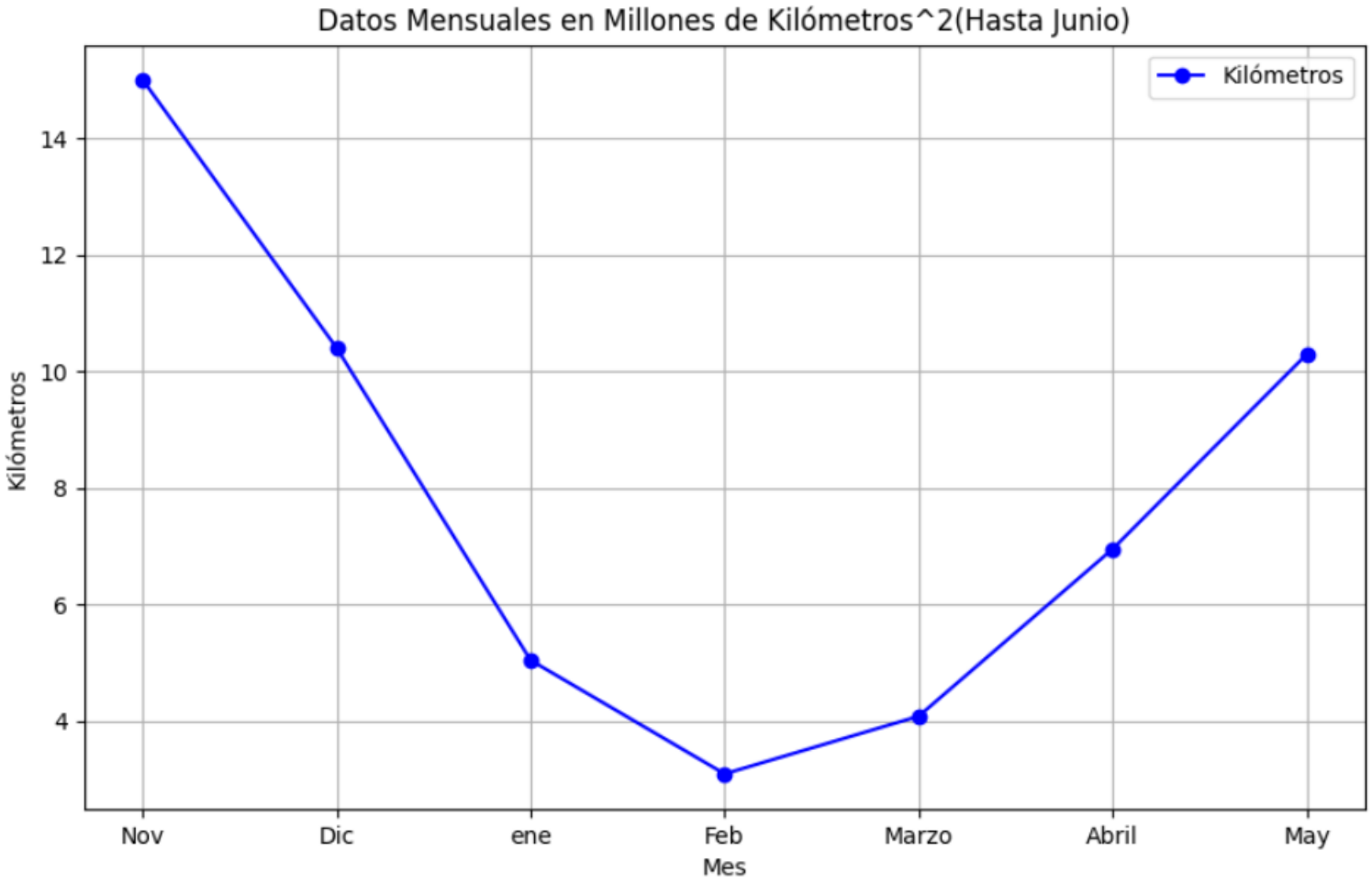
# Validación Multistep




Comparativo desempeños:

| | |
|---|---|
| RMSE train: | 0.069 |
| RMSE val: | 0.068 |
| RMSE test: | 0.069 |




Datos Mensuales en Millones de Kilómetros^2(Hasta Junio)
Kilómetros
14
12
10
8
6
4
Nov
Dic
ene
Feb
Marzo
Abril
May
Mes

# Conclusión

-Modelos

-Red Lstm en series de tiempo

-Extensión del hielo en el hemisferio sur.

2013

2022

# CONCLUSIONES