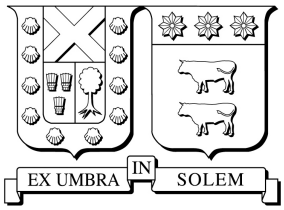


INF-477. Redes Neuronales Artificiales.
Tarea 3 - Redes Recurrentes

Prof. Ricardo Ñanculef (San Joaquín) & Carlos Valle (Casa Central)
jnancu@inf.utfsm.cl & *cvalle@inf.utfsm.cl*

Temas

- Diseño e implementación de redes recurrentes (RNN) usando *keras*.
- Apilamiento de RNN usando *keras*.
- Uso de capa embebida (word embedding) usando *keras*.

Formalidades

- Equipos de trabajo de: 2 personas.*
- Se debe preparar un (breve) Jupyter/IPython notebook que explique la actividad realizada y las conclusiones del trabajo.
- Se debe preparar una presentación de 20 minutos. Presentador será elegido aleatoriamente.
- Se debe mantener un respaldo de cualquier tipo de código utilizado, informe y presentación en Github.
- Fecha de entrega y discusión: Lunes 28 de Noviembre.
- Formato de entrega: envío de link Github al correo electrónico del ayudante (joaquin.velasquez@alumnos.inf.utfsm.cl), , con copia a y asunto: [Tarea3-INF477-I-2017].

*La modalidad de trabajo en equipo nos parece importante, porque en base a nuestra experiencia enriquece significativamente la experiencia de aprendizaje. Sin embargo, esperamos que esto no se transforme en una cruda división de tareas. Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

1. Análisis de Sentimientos usando RNN

Hoy en día, una aplicación relevante de las redes neuronales recurrentes es el modelamiento de texto y lenguaje natural. En esta sección abordaremos el problema de procesar el texto contenido en comentarios u opiniones (*review*) sobre una película para determinar su polaridad, es decir, determinar si refleja un sentiemiento positivo o negativo. Usaremos el *Large Movie Review Dataset*, también conocido como *IMDB dataset* que contiene 50000 comentarios de películas etiquetadas como buenas o malas (50 %-50 % train-testing). Este dataset fue recolectado por investigadores de Stanford [5] y utilizado en la competencia Kaggle [6].

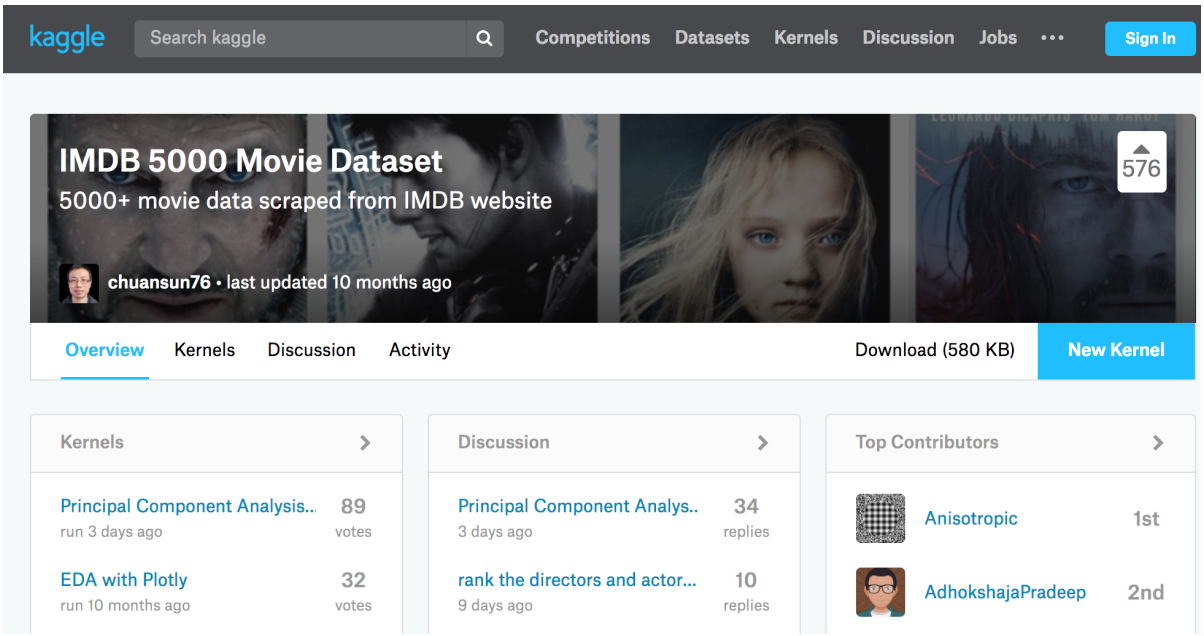


Figura 1: IMDB dataset

(a) Cargue los datos.

```
import numpy as np
from keras.datasets import imdb
np.random.seed(3)
(X_train, y_train), (X_test, y_test) = imdb.load_data(seed=15)
```

(b) Estudie la distribución del largo de los textos a procesar. Estudie también la frecuencia con la que aparecen las palabras en todo el dataset. ¿Se observa una ley Zipf? ¿Cambia el resultado cuando se separan los textos de acuerdo a su clase/categoría? Comente.

```
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
from matplotlib import pyplot
print("Review length: ")
result = map(len, X)
pyplot.boxplot(result)
pyplot.show()
```

(c) Cargue nuevamente el dataset, pero esta vez extrayendo solo las 3000 palabras más relevantes y acotando el largo máximo para un comentario en 500 palabras. Los comentarios con un largo menor deben ser rellenados con 0 (padding). ¿Porqué es necesario hacer esto último?

```
imdb.load_data(nb_words=3000, seed=15)
from keras.preprocessing import sequence
```

```
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
```

(d) Entrene una red LSTM para aprender a clasificar el texto y evalúe su desempeño. Esta red debe procesar la secuencia de 500 palabras a las que hemos reducido (o aumentado) cada review. Como las palabras corresponden a datos esencialmente categóricos, o al menos discretos, es necesario generar una representación vectorial de ellas. La primera capa de la red a construir debe por lo tanto incluir una transformación entrenable desde el espacio de representación original (discreto) a $\mathbb{R}^d$, con d la dimensionalidad del *embedding*.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vector_length, input_length=500))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=3, batch_size=64)
scores = model.evaluate(X_test, y_test, verbose=0)
```

(e) Varíe la dimensionalidad del embedding inicial y determine si aumenta o disminuye el error de clasificación. Comente.

(f) Modifique el número de palabras más frecuentes (top words) seleccionadas. Determine cómo afecta aquello el error de clasificación.

(g) Use Dropout para entrenar la LSTM. ¿El Dropout mejora el desempeño de la red? Señale cuales podrían ser las causas del comportamiento observado.

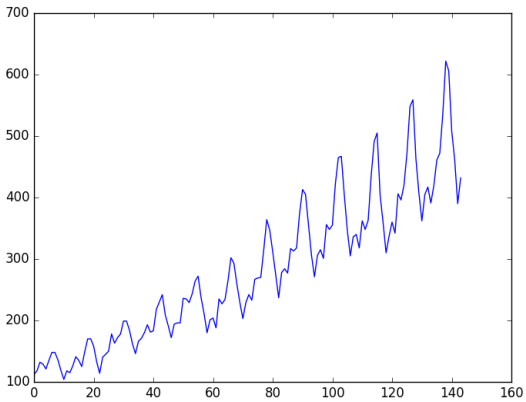
```
from keras.layers import Dropout

embedding_vector_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vector_length, input_length=500))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, nb_epoch=3, batch_size=64)
scores = model.evaluate(X_test, y_test, verbose=0)
```

(h) Proponga una modificación del modelo estudiado hasta el momento que mejore la capacidad predictiva del modelo.

2. Entrenamiento de RNNs en una Serie de Tiempo

En esta sección emplearemos redes neuronales recurrentes para modelar series de tiempo, es decir una serie de registros (típicamente valores reales) regularmente indexados en el tiempo. Parea ello utilizaremos el dataset denominado "international airline passengers" [4]. La tarea consiste en predecir el número de pasajeros (miles) en vuelos internacionales.



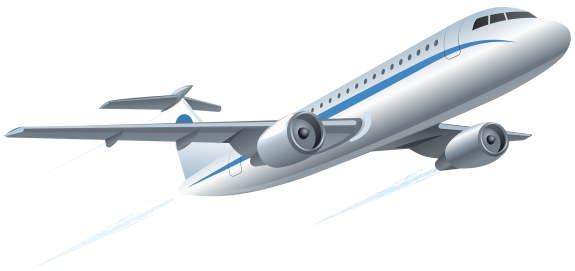


Figura 2: International airline passengers dataset

(a) Escriba una función que cargue los datos, los divida en conjuntos de entrenamiento y test, y los escale.

```
name_f=international-airline-passengers.csv
dataframe = pd.read_csv(name_f, sep=',', usecols=[1], engine='python', skipfooter=3)
dataframe[:] = dataframe[:].astype('float32')
df_train, df_test = dataframe[0:96].values, dataframe[96:].values

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1)).fit(df_train)
stream_train_scaled = scaler.transform(df_train)
stream_test_scaled = scaler.transform(df_test)
```

(b) Ahora nos gustaría manipular los datos, para que hagamos la predicción de la cantidad de pasajeros para el tiempo siguiente usando la cantidad de pasajeros de los últimos períodos de tiempo. El número de períodos de tiempos que usaremos se denomina lag. Por ejemplo, tendremos un lag igual a 3, si para predecir el valor x_{t+1} en el tiempo siguiente usamos la información del tiempo actual x_t y la de los dos períodos anteriores x_{t-1}, x_{t-2} como variables de entrada.

Realice una función que reciba una secuencia de valores y la transforme en dos arreglos *dataX* (inputs) y *dataY* (targets) donde el número de características de la matriz de entrada (columnas) sea el número de períodos de tiempos que se considerarán como información (lag).

```
def create_dataset(dataset, lag=1):
        return np.array(dataX), np.array(dataY)
```

Por ejemplo, si en el arreglo *dataset* tenemos la secuencia $112, 118, 132, 129, 121, 135, 148$

```
create_dataset(dataset, 3)
```

La función debiese generar $X = (X_1, X_2, X_3)$ e Y:

X_1	X_2	X_3	Y
112	118	132	129
118	132	129	121
132	129	121	135
129	121	135	148

(c) Usando la función anterior genere los conjuntos de entrenamiento y test para el problema.

```
lag = 3
trainX, trainY = create_dataset(stream_train_scaled, lag)
testX, testY = create_dataset(stream_test_scaled, lag)
```

(d) En estos momentos tenemos nuestros datos en la forma [ejemplos, atributos]. Sin embargo, la red LSTM necesita que los datos se encuentren en un arreglo de tres dimensiones [samples, time steps, features]. Transforme el train y test sets a la estructura deseada.

```
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
```

(e) Entrene una LSTM usando un lag de 3

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

model = Sequential()
model.add(LSTM(4, input_dim=lag, activation='tanh', inner_activation='sigmoid'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)
```

(f) Realice las predicciones del modelo para los conjuntos de entrenamiento y prueba. Denormalice los datos para que el error pueda ser computado en la escala original.

```
trainPredict = model.predict(trainX)

trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
```

(g) Compute el *root mean square error* (RMSE) sobre los conjuntos de entrenamiento y test.

```
from sklearn.metrics import mean_squared_error

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

(h) Grafique las predicciones del train y test set, y contrástelas con la serie de tiempo original.

```
# shift train predictions for plotting
trainPredictPlot = np.empty_like(dataframe.values)
trainPredictPlot[:, :] = np.nan

trainPredictPlot[lag:len(trainPredict)+lag, :] = trainPredict

```

```
# shift test predictions for plotting
testPredictPlot = np.empty_like(dataframe.values)
testPredictPlot[:, :] = np.nan

testPredictPlot[(len(trainPredict)+2*lag):, :] = testPredict
```

(h) Determine el número de bloques LSTM usando 5-fold Cross Validation.

```
nb=range(4,13,2)
model = Sequential()
model.add(LSTM(nb=range(4,13,2), input_dim=lag, activation='tanh', inner_activation='sigmoid'))
model.add(Dense(1))
```

(i) Compare el desempeño de la red LSTM variando el lag de 1 a 4. Comente brevemente.

(j) Usando un lag de 3, compare el desempe no de la LSTM con una red recurrente simple y una GRU. Comente.

```
from keras.layers import GRU
from keras.layers import SimpleRNN

GRU(output_dim, inner_init='orthogonal', activation='tanh')
SimpleRNN(output_dim, inner_init='orthogonal',activation='tanh')
```

(k) En lugar de aumentar el número de dimensiones como el el paso e), entrene la red con un timestep de 3 (con dimensión de entrada 1). ¿Se produce una mejora del error? ¿Los tiempos de computación son comparables? Comente brevemente.

```
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
model = Sequential()
model.add(LSTM(4, input_dim=1, activation='tanh', inner_activation='sigmoid'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)
```

(k) Entrene la red LSTM con memoria entre batches.

```
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, lag, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, lag, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
        model.fit(trainX, trainY, nb_epoch=1, batch_size=batch_size, verbose=2, shuffle=False)
        model.reset_states()
```

(l) Compare el resultado anterior usando un tamaño de batch de 3.

(m) Construya una LSTM apilada, y compárela con la obtenida en k). Comente brevemente lo sucedido.

```
model.add(LSTM(4, batch_input_shape=(batch_size, lag, 1), stateful=True, return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, lag, 1), stateful=True))
trainPredict = model.predict(trainX, batch_size=batch_size)
```

Referencias

[1] Hastie, T.; Tibshirani, R., Friedman, J. (2009), The Elements of Statistical Learning, Second Edition. Springer New York Inc.

[2] Bishop, Christopher M. (1995), Neural Networks for Pattern Recognition, Clarendon Press.

[3] Graves, A. (2008), PhD Thesis. Supervised sequence labeling with recurrent neural networks. Technical University Munich.

[4] Box, G. E. P., Jenkins, G. M. and Reinsel, G. C. (1976), Time Series Analysis, Forecasting and Control. Third Edition. Holden-Day. Series G.

[5] Maas, A. L., Daly, R. E., Pham, P. T. and Huang, D., Ng, A. Y. and Potts, C. (2011), Learning Word Vectors for Sentiment Analysis, Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1.

[6] `https://www.kaggle.com/c/word2vec-nlp-tutorial/data`

LaTeX