PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DIPLOMADO BIG DATA

ACTIVIDAD PRÁCTICA 2:
TÉCNICAS DE APRENDIZAJE PROFUNDO

Actividades previas y material de preparación para evaluación escrita.

1 Objetivo

Estimad@s, en esta actividad tendrán la oportunidad de poner en práctica sus conocimientos sobre aprendizaje profundo (deep learning). En particular, podrán experimentar con las técnicas que discutimos en clases para implementar Redes Neuronales Convolucionales (Convolutional Neural Nets o CNNs) y Redes Neuronales Recurrentes (Recurrent Neural Nets o RNNs).

2 Redes Neuronales Convolucionales (CNNs): AlexNet

Para ilustrar la implementación de CNNs en Keras, estudiaremos en profundidad una de las estructuras más populares: AlexNet [1]. La figura 1 muestra la estructura de AlexNet:

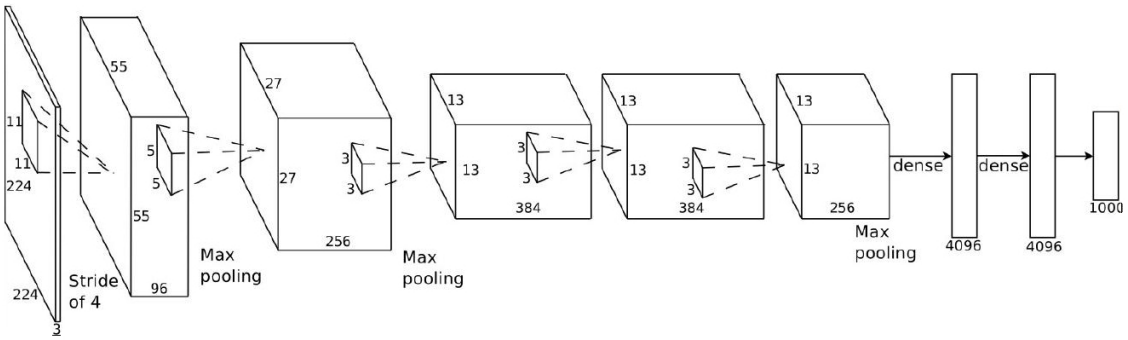

Figura 1: Arquitectura de la red convolucional AlexNet.

Para implementar esta red en Keras, comenzamos creando el contenedor (container) que será usado para encapsular el modelo:

```
modelAlexNet = Sequential()
```

Como comentamos en clases, para aprovechar de mejor forma la información en los bordes de la imagen de entrada, agregamos filas y columnas con ceros alrededor de ella. En el caso de AlexNet se agregan 2 filas y 2 columnas a cada lado de la imagen de entrada. Para esto usamos la función de Keras *ZeroPadding2D*, según el siguiente formato:

```
modelAlexNet.add(ZeroPadding2D((2,2),input_shape=(224, 224, 3)))
```

El primer parámetro indica el padding de 2 pixeles en cada borde (horizontal y vertical), mientras que el segundo parámetro (*input_shape*) indica el tamaño de la imagen de entrada.

Una vez que tenemos lista la entrada podemos definir el operador de convolución de la primera capa. En este caso corresponde a 96 filtros de 11x11, con un stride de 4 pixeles en la dirección horizontal y 4 en la dirección vertical. Para esto usamos la función de Keras *Convolution2D*, según la siguiente notación:

```
modelAlexNet.add(Convolution2D(96,11,11,subsample=(4,4),border_mode='valid'))
```

El parámetro *subsample=(4,4)* indica el tamaño del stride horizonal y vertical. Por su parte, el parámetro *border_mode='valid'* indica que el barrido de la convolución se aplica hasta la última posición válida antes de superar el borde de la imagen.

Como discutimos en clases, en sus capas convolucionales AlexNet utiliza rectificadores lineales (Relu) como función de activación. Adicionalmente, aplica una normalización sobre las activaciones de las neuronas sobre cada batch de entrenamiento. Esta normalización se encarga de generar batch con salidas de media=0 y desviación standard=1. Este tipo de normalización es usual en CNNs y Keras la implementa mediante la función *BatchNormalization()*. En Keras, las 2 tareas anteriores quedan dadas por:

```
modelAlexNet.add(Activation(activation='relu ' ))
modelAlexNet.add(BatchNormalization())
```

Como muestra la figura 1, el último paso de la primera capa de AlexNet consiste en la aplicación de un operador *Max-Pooling*. En este caso, el operador actúa sobre una vecindad de 3x3 utilizando un stride de 2 posiciones en las direcciones vertical y horizontal. Para esto usamos la función de Keras *MaxPooling2D*, según la siguiente notación:

```
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
```

Uniendo todo lo anterior y agregando la definición de las librería necesarias, obtenemos el siguiente código:

```
#Definicion de librerias con la funciones que seran utilizadas por Keras.
import keras
from keras.layers import Activation, Dense, Flatten, Dropout
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D

#Definicion de contenedor y primera capa de AlexNet.
modelAlexNet = Sequential()
modelAlexNet.add(ZeroPadding2D((2,2), input_shape=(224, 224, 3)))
modelAlexNet.add(Convolution2D(96,11,11,subsample=(4,4),border_mode='valid'))
modelAlexNet.add(Activation(activation='rel'))
modelAlexNet.add(BatchNormalization())
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
```

Actividad 1

El archivo AlexNetCapa1.py, disponible en el sitio web del curso, contiene el código anterior. Ejecute este código y verifique que las dimensiones de salida de la capa definida corresponden a las utilizadas por AlexNet. Para esto puede utilizar en Keras el comando: *print(modelAlexNet.output_shape)*. Este comando permite imprimir en pantalla la dimensiones de salida de la red definida hasta la ejecución del comando. A modo de ejemplo, para acceder a las dimensiones de salida de la red antes y después de aplicar el operador *Max-Pooling*, podemos ejecutar:

```
#Definicion de contenedor y primera capa de AlexNet.
modelAlexNet = Sequential()
modelAlexNet.add(ZeroPadding2D((2,2), input_shape=(224, 224, 3)))
modelAlexNet.add(Convolution2D(96,11,11,subsample=(4,4),border_mode='valid'))
modelAlexNet.add(Activation(activation='rel'))
modelAlexNet.add(BatchNormalization())
```

```
print(modelAlexNet.output\_shape) #dims de la red antes de max-pooling
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
print(modelAlexNet.output\_shape) #dims de la red despues de max-pooling
```

Siguiendo con la arquitectura de AlexNet en la figura 1 y las funciones definidas anteriormente, la segunda capa queda definida por:

```
modelAlexNet.add(ZeroPadding2D((2,2)))
modelAlexNet.add(Convolution2D(256, 5, 5, border_mode='valid'))
modelAlexNet.add(Activation(activation='relu '))
modelAlexNet.add(BatchNormalization())
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
```

Actividad 2
Verifique que las dimensiones de salida de esta segunda capa corresponden a las utilizadas por AlexNet. En su informe de laboratorio incorporé los output generados. Adicionalmente, utilice el comando *modelAlexNet.summary()* para generar un resumen de la red construida hasta este momento. ¿Cuántos parámetros (pesos) contiene esta red?

Actividad 3
Usando como guía las capas anteriores, construya en Keras la tercera capa de AlexNet. Tenga presente que esta tercera capa:

- Incluye un padding de 1 cero a cada lado del mapa de activaciones de entrada.
- No utiliza normalización batch.
- No incorpora una etapa de max-pooling.

En su informe de laboratorio reporte el código generado. Adicionalmente, utilice la función *output_shape* para verificar que la salida de la tercera capa corresponde a la arquitectura de la figura 1, y la función *summary* para obtener un resumen de los parámetros de la red.

Actividad 4
Para completar la fase convolucional de AlexNet nos queda definir las capas 4 y 5. Análogamente a la capa 3, la capa 4 también realiza un padding de 1 cero a cada lado de la entrada, no incluye normalización batch, y no incluye max-pooling. Por su parte, la capa 5 realiza un padding de 1 cero a cada lado de la entrada, no incorpora normalización batch, pero si incorpora una etapa de max-pooling con una ventana de 3x3 y un stride de 2 en las direcciones vertical y horizontal.

Genere el código de las capas 4 y 5, verifique que las salidas sean las adecuadas, y determine el número de parámetros de la red. Reporte sus observaciones y código generado.

Actividad 5
Ha sido una tarea ardua pero estamos cerca de completar la implementación de AlexNet, sólo nos queda la definición de las capas de conexión densa (multilayer perceptron o MLP).

Para la definición de la primera de estas capas, el primer paso es llevar a una representación 1D la salida 3D de la capa 5 (representada en figura 1 con un cubo). Para esto utilizamos la función de Keras *Flatten* (aplanar), según la siguiente sintaxis:

```
modelAlexNet.add(Flatten())
```

Utilice la función *output_shape* para verificar el efecto de la función *Flatten*. ¿Las dimensiones obtenidas corresponden a lo esperado?. Fundamente sus observaciones.

Actividad 6
Como vimos en clases, las capas densas son definidas en Keras mediante la función *Dense()*. Revise sus apuntes y genere el código apropiado para definir la primera capa densa que tiene como salida 4096 neuronas. Tal como en las capas convolucionales, AlexNet utiliza para la capa 6 una función de activación *Relu*. Adicionalmente, incorpora un proceso de *Dropout* con una probabilidad de 0.5.

Revise sus apuntes de clases, donde se ilustra el uso de Dropout en Keras, y genere el código restante para completar la definición de la capa 6 de AlexNet. Verifique que las salidas sean las adecuadas, y determine el número de parámetros de la red. Reporte sus observaciones y código generado.

Actividad 7
Defina las capas 7 y 8 de AlexNet. En el caso de la capa 7, como se aprecia en la figura 1, tiene una salida de 4096 neuronas. Esta capa utiliza función de activación Relu y Dropout con probabilidad 0.5. Finalmente, la capa 8 tiene una salida de 1000 neuronas. Esta capa utiliza función de activación *softmax* y no utiliza Dropout.

En su reporte de laboratorio incorpore el códigos generado, así como un análisis del número de filtros y parámetros de la red final. ¿ Qué capas utilizan más filtros y parámetros ?, ¿Qué justifica este tipo de arquitectura?. Comente y fundamente sus observaciones.

3 Redes Neuronales Recurrentes (RNNs)

Como comentamos en clases, las redes recurrentes nos permiten modelar secuencias, es decir, permiten capturar explícitamente relaciones temporales o espaciales en los datos. En esta actividad evaluaremos esta capacidad mediante el uso de un ejemplo incluido en las librerías de Keras. Específicamente, usaremos una RNN para predecir el flujo mensual de pasajeros en vuelos internacionales en EE.UU. Para esto usaremos el set de datos "international-airline-passengers.cvs" disponible en el sitio web del curso. Este set de datos contiene la demanda mensual de pasajes internacionales para un período de 12 años, correspondiente a Enero 1949 a Diciembre 1960. La figura 2 muestra una gráfica de este set de datos.

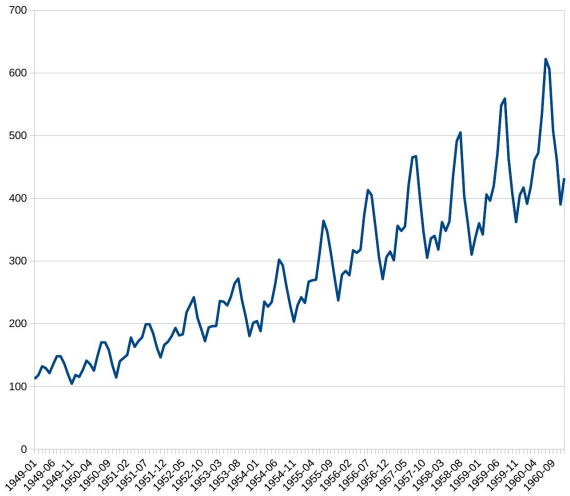

Figura 2: Total mensual de pasajeros en vuelos internacionales en EE.UU para el período Enero 1949 a Diciembre 1960. Cifras en miles de pasajeros.

La tendencia creciente de la curva muestra el aumento de pasajeros a lo largo de los años. Adicionalmente, cada año presenta peaks característicos causados por aumentos puntuales de la demanda debido al período de vacaciones y festividades especiales.

Para este set de datos, usaremos el período Enero 1949 a Diciembre 1956 para aprender un modelo del tipo RNN, y el resto de los datos para probar la capacidad predictiva del modelo obtenido. Específicamente, definiremos un modelo base, en el cual consideraremos el problema de predecir la demanda en el mes t según la información de demanda de los 4 meses anteriores. Luego estudiaremos variantes de este modelo base.

El siguiente código muestra la definición en Keras de una red neuronal recurrente para el modelo base.

```
modelRNN = Sequential()
modelRNN.add(SimpleRNN(5,input_dim=1,input_length=4,return_sequences=False))
```

Primeramente se define el container, que en este caso es denominado *modelRNN*. Luego se define la red recurrente según los siguientes parámetros:

- El primer parámetro indica la dimensionalidad del espacio de características del estado intermedio o oculto, en este caso dimensionalidad 5.
- *input_dim* indica la dimensionalidad de la entrada, en este caso 1 (demanda mensual).
- *input_length* indica el largo de la secuencia de entrada, en esta caso 4 meses anteriores.
- *return_sequences=False* indica que sólo el último nodo de la secuencia intermedia genera una salida (ver figura 3).

La siguiente figura muestra un diagrama de la RNN definida por el código anterior:

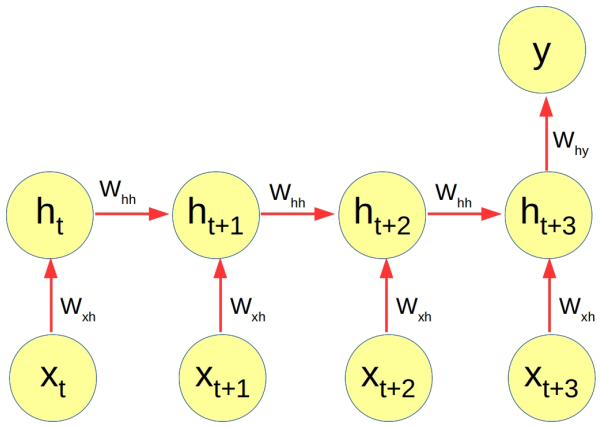

Figura 3: Arquitectura de la red recurrente definida para la red recurrente base.

Al analizar los embeddings podemos calcular el número de parámetros que es necesario ajustar durante el aprendizaje. La entrada corresponde a la medición de demanda de cada mes, por tanto, tiene dimensión 1 (*input_dim=1*). Por su parte, el estado intermedio h_t tiene dimensionalidad 5. Así el embedding W_{xh} transforma de dimensionalidad 1 a 5, por tanto, W_{xh} consiste de una matrix de 5x1 valores, es decir, $W_{xh} \in \mathbb{R}^{5x1}$, 5 parámetros. Por su parte, para el embedding entre estados intermedios tenemos que $W_{hh} \in \mathbb{R}^{5x5}$, por tanto, 25 parámetros. Finalmente, el embedding de salida $W_{hy} \in \mathbb{R}^{5x1}$, por tanto, 5 parámetros. Consecuentemente, el total de parámetros es 35.

Actividad 8
Verifique que el modelo definido anteriormente contiene 35 parámetros. Para ello utilice como guía el código anterior y la función *summary()* para acceder al número de parámetros.

Si se decide modificar la dimensionalidad del estado intermedio a un valor 4, ¿Cómo se afecta el número de parámetros?. Pruebe y fundamente su respuesta.

Para realizar la predicción de la demanda podemos agregar a la red anterior una capa densa. Dado que queremos predecir el valor de una variable que toma valores en el eje de los número reales, utilizamos sólo 1 neurona como salida y no incorporamos función de activación. De esta manera el modelo base de predicción queda dado por:

```
# create and fit the RNN
modelRNN = Sequential()
modelRNN.add(SimpleRNN(5,input_dim=1,input_length=4,return_sequences=False))
modelRNN.add(Dense(1))
```

Actividad 10
El archivo AirlinePrediction.py, disponible en el sitio web del curso, contiene el código anterior y las funciones necesarias para cargar el archivo de datos y mostrar los resultados obtenidos. Use este código para probar el rendimiento del modelo base. En términos de la presentación de resultados, la salida del archivo AirlinePrediction.py indica el error medio cuadrático en el set de entrenamiento y test. Además muestra una gráfica que incluye los datos originales (curva azul), la predicción en datos de entrenamiento (curva verde) y la predicción en datos de test (curva roja).

Ejecute el modelo base varias veces, ¿Por qué no se obtiene siempre el mismo resultado?. Fundamente su respuesta.

Luego pruebe las siguientes variantes al modelo base y analice su resultados:

- Modifique el largo de la secuencia de entrada. Este valor es determinado por el valor de la variable *history*. Por ejemplo, para cambiar el largo de la secuencia de entrada desde 4 (modelo base) a 10, debe reemplazar en AirlinePrediction.py *history=4* por *history=10*.
- Agregue una nueva capa densa de 100 unidades y vea su impacto en los resultados.
- Agregue Dropout.

Documente sus resultados y principales observaciones, no olvide fundamentar sus respuestas.

Actividad 11
La red anterior implementa una red convolucional tradicional. Como comentamos en clases, este tipo de modelo presenta limitaciones para ajustar los parámetros de la red utilizando métodos de descenso de gradiente (problema de desvanecimiento de gradiente o vanishing gradient problem). En la actualidad redes recurrentes tipo LSTM y GRU ofrecen soluciones más estables. Keras ofrece implementaciones de ambas alternativas, siendo muy simple su uso, basta cambiar el nombre en el llamado a la función. Por ejemplo, para reemplazar en el modelo base la red recurrente tradicional por un modelo LSTM, el código sería:

```
# create and fit a LSTM network
modelRNN = Sequential()
modelRNN.add(LSTM(5,input_dim=1,input_length=4,return_sequences=False))
modelRNN.add(Dense(1))
```

Compare el rendimiento de un modelos LSTM con respecto al modelo tradicional (SimpleRNN). Considere el rendimiento en la predicción y el número de parámetros utilizado.

Actividad 12
Considere la función: *modelRNN.fit(trainX, trainY, nb_epoch=150, batch_size=5, verbose=2)*

Explique brevemente el rol de los parámetros: *nb_epoch* y *batch_size*. Seleccione alguno de los modelos testeados y ejecute cambiando el valor de estos 2 parámetros, comente sus observaciones.

Bibliografía

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.