Práctica 2
Resolución de un problema de regresión

TÉCNICAS ESTADÍSTICAS PARA EL APRENDIZAJE II

Máster Universitario en Estadística Computacional
y Ciencia de Datos para la Toma de Decisiones

Resolución de un problema de regresión

En esta práctica crearemos un modelo de red neuronal para un **problema de regresión**.

A diferencia de un problema de clasificación, donde el objetivo es seleccionar una clase de una lista de clases, en un problema de regresión, el objetivo es **predecir la salida de un valor continuo.**

En esta práctica intentaremos **predecir el precio medio de las viviendas** (expresado en los datos en miles de $) en diferentes ubicaciones alrededor de los suburbios de Boston a finales de la década de 1970.

Disponemos de información sobre el barrio recogida en **variables numéricas**, como la tasa de criminalidad, el número medio de habitaciones por vivienda, el número de colegios en el barrio, etc. En este ejemplo tenemos pocos datos: sólo 506, divididos en 404 muestras de entrenamiento y 102 de prueba. Este *dataset*, que proviene de la biblioteca StatLib, está incluido en Keras.

A diferencia de la práctica 1, aquí cada **característica de entrada** (por ejemplo, la tasa de criminalidad) tiene una **escala diferente.**

Normalización de datos

No debemos introducir en una red neuronal valores con rangos muy diferentes. El modelo podría adaptarse a datos heterogéneos, pero dificultaría el aprendizaje.

Una práctica muy extendida es la **normalización por características**: para cada característica de los datos de entrada (cada columna de la matriz de datos de entrada), restamos su media y la dividimos por su desviación estándar, de forma que se centre en 0 y tenga una desviación estándar unitaria.

La normalización debe realizarse utilizando la media y la desviación estándar del conjunto de entrenamiento

IMPORTANTE: No debemos filtrar ninguna información de nuestro conjunto de prueba a otros conjuntos, es decir, nunca debemos usar ninguna cantidad calculada con los datos de prueba, incluso para algo tan simple como la normalización de datos.

RMSprop [*Root-Mean Square propagation*]

En el algoritmo de **gradiente descendente clásico** fijamos el valor de la tasa de aprendizaje, lo que puede dar como resultado una convergencia lenta.

Además, algunos gradientes pueden ser pequeños y otros pueden ser enormes, por lo que tratar de encontrar una tasa de aprendizaje global única para el algoritmo puede ser un problema muy difícil.

RMSprop se desarrolló como una técnica para el aprendizaje por minilotes. Es una extensión del método de gradiente descendente que se encuentra en el ámbito de los métodos de **tasa de aprendizaje adaptativo**, que han ido creciendo en popularidad en los últimos años.

Resolución de un problema de regresión

RMSprop divide la tasa de aprendizaje de cada peso por un promedio móvil de los cuadrados de los gradientes recientes para ese peso. Con esto se consigue equilibrar el tamaño del paso, disminuyéndolo para gradientes grandes para evitar explosiones y aumentándolo para gradientes pequeños para evitar la desaparición.

El uso de ese promedio móvil permite que el algoritmo olvide los gradientes iniciales y se concentre en los gradientes observados más recientemente durante el proceso de la búsqueda.

Adam [*Adaptive moment estimation*] es otro algoritmo muy popular en *deep learning*, que puede verse como una combinación de RMSprop con momentos adaptativos.

El algoritmo de gradiente descendente (***sgd***, *stochastic gradient descent*), ***rmsprop*** **y** ***adam*** se encuentran entre los algoritmos de optimización más utilizados.

Guardar modelos

Dado que los modelos de aprendizaje profundo pueden tardar horas, días e incluso semanas en entrenarse, **no podemos entrenar los modelos constantemente**. Por ello, es importante saber cómo guardarlos y cargarlos para, cuando vengan nuevos datos, poder hacer predicciones.

La función *save()* de Keras nos permite guardar un modelo después del entrenamiento.

Un modelo guardado contiene los **valores de los pesos**, la **configuración del modelo** (arquitectura) y la **configuración del optimizador**.

El modelo guardado se puede cargar más tarde con la función *load_model()* que devuelve el modelo con la misma arquitectura y pesos.

Resolución de un problema de regresión

Ejercicio 1.

Una forma de mejorar el rendimiento de una red neuronal es agregar más capas. Esto podría permitir que el modelo extraiga y recombine características de orden superior integradas en los datos.

Crea una nueva red neuronal añadiendo más neuronas a la primera capa y una segunda capa oculta, y evalúa las métricas en esta nueva topología de red más profunda y más amplia.

Evalúa dicho modelo en el conjunto de test y guarda el modelo con la función *save()* de Keras.

Ejercicio 2.

Cambia de optimizador, utilizando *"sgd"* (*stochastic gradient descent*) para encontrar los pesos y sesgos en la red neuronal inicial, es decir, con una sola capa oculta.

Evalúa las métricas en esta red neuronal, guarda el modelo con la función *save()* de Keras y compara los resultados con los que obtuviste con el optimizador *"rmsprop"*.