Práctica 4
Optimización de hiperparámetros

TÉCNICAS ESTADÍSTICAS PARA EL APRENDIZAJE II

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

Optimización de hiperparámetros

Los **hiperparámetros** son parámetros que se configuran antes de que comience el proceso de entrenamiento y no se pueden aprender de los datos (tasa de aprendizaje, tamaño del lote, cantidad de capas ocultas, funciones de activación, etc). Ajustar estos hiperparámetros es esencial para lograr un buen rendimiento de los modelos de aprendizaje profundo.

En esta práctica trabajaremos con **tasas de aprendizaje adaptativo** y con la búsqueda y evaluación eficiente de hiperparámetros del modelo utilizando ***grid search***.

Trabajaremos con el **conjunto de datos Reuters**, un conjunto de noticias breves y sus temas, publicado por Reuters en 1986. Se trata de un conjunto de datos muy utilizado para la clasificación de textos. Hay 46 temas diferentes; algunos temas están más representados que otros, pero cada tema tiene al menos 10 ejemplos en el conjunto de entrenamiento.

https://keras.io/api/datasets/reuters/

Preprocesamiento

Cada noticia es una lista de números enteros (índices de palabras) de longitudes diferentes, por lo que no podemos introducirlas directamente en una red neuronal, que espera procesar lotes contiguos de datos.

Una posibilidad aquí es usar **codificación multi-hot** en las listas para convertirlas en vectores de 0s y 1s.

Para ello utilizaremos la función
vectorize_sequences(sequences, dimension=10000)
extraida del libro *Deep Learning with Python* de François Chollet

Optimización de hiperparámetros

Preprocesamiento

vectorize_sequences(sequences, dimension=10000)

Valores de entrada:

- sequences: secuencia de entrada
- dimension: dimensión del resultado

Resultado:

- Secuencia de 0s y 1s de tamaño 10000, con valor 1 cuando la palabra está contenida en la secuencia considerada.

Esto significaría, por ejemplo, convertir la secuencia [8, 5] en un vector de 10.000 dimensiones que sería todo 0s excepto los índices 8 y 5, que serían 1s.

Además, como parte del preprocesamiento, aplicaremos **codificación one-hot a la etiqueta.**

Tasa de aprendizaje

La **estrategia predeterminada** es usar una **tasa de aprendizaje constante** para actualizar los pesos de la red en cada época de entrenamiento.

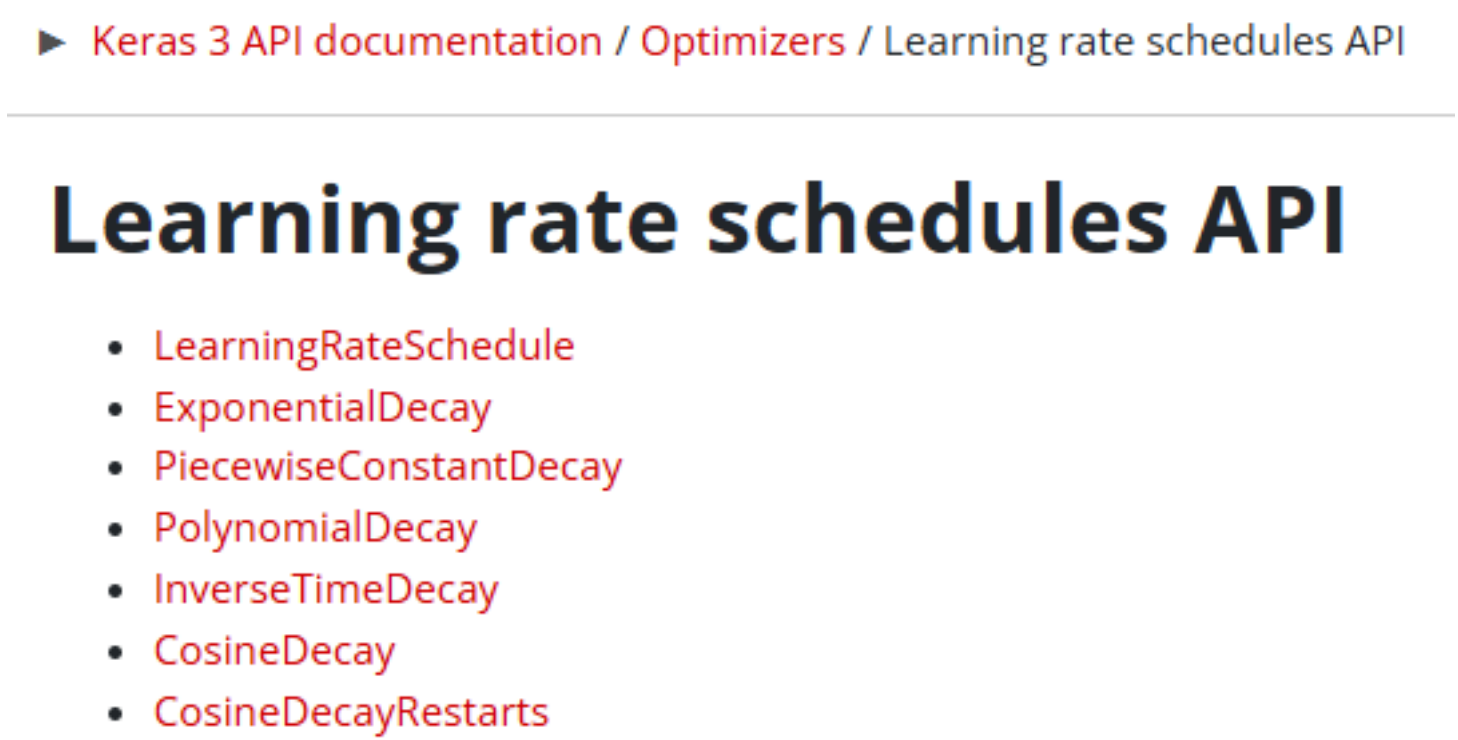
No obstante, adaptar la tasa de aprendizaje en el procedimiento de optimización puede aumentar el rendimiento y reducir el tiempo de entrenamiento. Esto se denomina **tasas de aprendizaje adaptables o adaptativas**.

La **adaptación más simple** y quizás más utilizada de las tasas de aprendizaje durante el entrenamiento son las **técnicas que reducen la tasa de aprendizaje a medida que avanza el entrenamiento.**

Optimización de hiperparámetros

Keras tiene incorporadas diferentes planificaciones para la tasa de aprendizaje basada en el tiempo. Cada planificación altera el ritmo de aprendizaje mediante unas reglas específicas.

https://keras.io/api/optimizers/learning_rate_schedules/



Optimización de hiperparámetros

En esta práctica veremos cómo aplicar un decrecimiento exponencial de la tasa de aprendizaje. Los parámetros principales de la función *ExponentialDecay()* son:

initial_learning_rate: la tasa de aprendizaje inicial al comienzo del entrenamiento.

decay_steps: el número de pasos después de los cuales la tasa de aprendizaje disminuirá.

decay_rate: la velocidad a la que disminuirá la tasa de aprendizaje. Por ejemplo, si *decay_rate* se establece en 0.96, la tasa de aprendizaje se multiplicará por 0.96 por cada *decay_steps* pasos.

Por ejemplo, si tenemos un total de 50000 muestras de entrenamiento y utilizamos un tamaño de lote de 64, eso implica que hay un total de 50000/64 = 782 pasos (*steps*) por época, es decir, aplicamos un total de 782 actualizaciones de pesos antes de que se complete una época.

Optimización de hiperparámetros

Aspectos a tener en cuenta:

Incrementar la tasa de aprendizaje inicial: debido a que la tasa de aprendizaje disminuirá, es aconsejable comenzar con un valor mayor desde el cual disminuir. Una tasa de aprendizaje mayor resultará en cambios mayores en los pesos, al menos al principio, lo que nos permitirá beneficiarnos de un mejor ajuste.

Utilizar un gran momentum: el uso de momentum ayudará al algoritmo de optimización a seguir realizando actualizaciones en la dirección correcta cuando su tasa de aprendizaje se reduzca a valores pequeños.

$$\Delta w(t) = -\eta \nabla E(w(t)) + \mu \Delta w(t-1)$$

Experimentar con diferentes planificaciones: es aconsejable probar con diferentes opciones de configuración y ver cuál funciona mejor en nuestro problema (la mejor opción dependerá tanto de los datos como de la arquitectura de nuestro modelo).

Optimización de hiperparámetros

En esta práctica también vamos a ver cómo crear un método de optimización de búsqueda de hiperparámetros evaluando todos ellos, es decir, lo que se conoce como **búsqueda sistemática [*grid search*]**.

En scikit-learn, la búsqueda sistemática se proporciona en la clase *GridSearchCV*, que realiza una búsqueda exhaustiva en una cuadrícula predefinida de valores de hiperparámetros. Evalúa el rendimiento del modelo mediante validación cruzada y selecciona la combinación de hiperparámetros que ofrece los mejores resultados.

Los modelos Keras se pueden utilizar en scikit-learn con la clase KerasClassifier o KerasRegressor del módulo **scikeras**.

Optimización de hiperparámetros

En primer lugar, tenemos que definir los diferentes valores para los parámetros que deseamos buscar. En concreto, nosotros vamos a explorar los siguientes hiperparámetros:

- Número de épocas
- Tamaño de lote
- Número de neuronas en las capas ocultas
- Optimizador para actualización de pesos

Después, pasaremos la configuración de *grid search* a través de un diccionario de hiperparámetros en el argumento *param_grid*.

Por último, *GridSearchCV* construirá y evaluará un modelo para cada combinación de parámetros.

Optimización de hiperparámetros

La búsqueda sistemática suele resultar en **muchos modelos y mucho cálculo**. Por ello, no es un esquema que se deba utilizar a la ligera debido al tiempo que supone.

Suele resultar útil realizar algunos **experimentos previos** que nos guíen en la búsqueda de valores adecuados para los parámetros.

En esta práctica también veremos cómo utilizar ***RandomizedSearchCV*** que, a diferencia de *GridSearchCV*, no prueba todos los valores de los parámetros, sino que toma muestras aleatorias de un número fijo de configuraciones.

RandomizedSearchCV es más eficiente que *GridSearchCV* para explorar grandes espacios de hiperparámetros y puede ser particularmente útil cuando los recursos computacionales son limitados.

Optimización de hiperparámetros

Algunos consejos para optimizar hiperparámetros

Utilizar una muestra del conjunto de datos: debido a que las redes son lentas de entrenar, es aconsejable entrenarlas con una muestra más pequeña de nuestro conjunto de datos de entrenamiento, solo para tener una idea de las direcciones generales de los parámetros en lugar de las configuraciones óptimas.

Comenzar con cuadrículas de grano grueso y, posteriormente, ampliar a cuadrículas de grano más fino.

Revisar toda la cuadrícula: no debemos centrarnos únicamente en el mejor resultado. Es aconsejable revisar toda la cuadrícula de resultados y buscar tendencias que sirvan de apoyo a las decisiones de configuración.

Optimización de hiperparámetros

Ejercicio 1.

Realiza una búsqueda sistemática con *GridSearchCV* o *RandomizedSearchCV* explorando otros hiperparámetros del modelo y guarda el modelo con la mejor combinación utilizando la función *save()* de Keras.

Ejercicio 2.

El conjunto de datos IMDB contiene 50.000 críticas muy polarizadas de la base de datos de películas de Internet. Se divide en 25.000 críticas para el entrenamiento y 25.000 críticas para el test. Cada conjunto consta de un 50% de críticas negativas y un 50% de críticas positivas. El conjunto de datos IMDB está disponible en Keras y ya ha sido preprocesado: las reseñas (secuencias de palabras) se han convertido en secuencias de enteros, donde cada entero representa una palabra específica en un diccionario y las etiquetas toman los valores 0 y 1, donde 0 significa negativo y 1 positivo.

```
# Datos
from keras.datasets import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)
```

Construye una red neuronal que clasifique las críticas de películas como positivas o negativas basándose en el contenido del texto de las críticas.