Selección de hiperparámetros

U-TAD

¿Qué es un hiperparámetro?

Debemos seleccionar variables, cuando hablamos de una arquitectura de red neuronal, que no se corresponden ni con las características de entrada X_i ni con los pesos w_i.

Por ejemplo, tenemos que seleccionar el **número de hidden layers** que tendrá nuestra red neuronal. También tendremos que seleccionar el **número de neuronas** de cada hidden layer, así como también la **función de activación** para cada una de las capas.

Además, tendremos que decidir variables como el **learning rate**, entre otros, de mi **algoritmo de optimización** también a elegir.

Todas estas es lo que denominamos hiperparámetros, que normalmente se corresponden con argumentos que le proporcionamos a las funciones que están implementadas en las diferentes librerías.

¿Cómo selecciono los hiperparámetros?

Keras Tuner es una biblioteca que te ayuda a elegir el conjunto óptimo de hiperparámetros para tu modelo. El proceso de seleccionar el conjunto correcto de hiperparámetros para su aplicación de aprendizaje automático se llama tuning de hiperparámetros o *hypertuning*.

Es un módulo de Keras que entrenará nuestra red neuronal con distintos hiperparámetros y procesará los resultados en un conjunto de validación y en un conjunto de pruebas. De modo, que nos devolverá finalmente el conjunto de hiperparámetros óptimo que maximice el rendimiento de nuestra red neuronal. Es decir, la que nos proporcione mejores resultados.

Pero si ponemos un rango muy grande a seleccionar por Keras Tuner, probablemente tardemos demasiado en realizar la selección. Por lo que tendremos que decidir entre los parámetros más relevantes y cuáles son las instrucciones para modificar y reducir ese rango de valores posibles para los hiperparámetros.

Caso Práctico: Selección de hiperparámetros con Keras Tuner

Optimización/Selección de hiperparámetros con Keras Tuner.ipynb

pip install keras-tuner

import kerastuner as kt (deprecated)

import keras_tuner as kt

def model_builder(hp):

model = keras.Sequential()

/* Transforma los datos, la información, en un vector (sin que tenga parámetros asociados), lo hemos hecho de otro modo, pero podéis ver otra forma de hacerlo: /*

model.add(keras.layers.Flatten(input_shape=(28, 28)))

… // Ver hp_units y hp_learning_rate

tuner = kt.Hyperband(...factor = 3 //hiperparámetro para su fórmula $1+\log_{(factor)}(max\_epochs)$…)

Hiperparámetros relevantes

Número de capas óptimo

- Una red neuronal artificial con una única hidden layer, teóricamente, puede llegar a construir funciones muy complejas (si el problema no es muy complejo, límite de decisión sencillo, podemos resolverlo con una sola capa)
- Las redes neuronales artificiales con un número elevado de hidden layers tienen una eficiencia de los parámetros más altas, permiten alcanzar un rendimiento más elevado con el mismo conjunto de datos de entrenamiento (siempre y cuando el conjunto de datos de sea grande y requiera de límites de decisión complejos, pero debemos tener en cuenta el rendimiento a la hora de entrenar: se necesitarán muchos más recursos computacionales)
- Un número elevado de hidden layers facilitan la generalización para nuevos conjuntos de datos: *transfer learning* (cada capa aprende de manera diferente: las primeras se dedicarán a identificar características de muy bajo nivel, y según avanzamos hacia las últimas van a predecir características más complejas, por lo que podríamos reutilizar el aprendizaje de las primeras capas, por ejemplo, ejes, colores, sombras, para distintos problemas de identificación, convergiendo en, por ejemplo, ruedas y, finalmente, una moto)
- ¿Cómo seleccionamos el número óptimo de capas? (en base a la complejidad y los datos)

Hiperparámetros relevantes

Número de neuronas óptimo

- El número de neuronas de la input layer y de la output layer está determinado por el tipo de entrada y salida que requiere nuestra tarea
- Lo más común es que el número de neuronas por cada hidden layer formen una pirámide, con más neuronas en las primeras capas que en las últimas (pensado por el *transfer learning:* las características simples son más)
- Sin embargo, con el paso del tiempo, se ha llegado a la conclusión de que utilizar el mismo número de neuronas por capa arroja resultados muy similares y reduce el espacio de hiperparámetros a evaluar
- Un número muy elevado de neuronas, normalmente, causarán *overfitting*

Otros hiperparámetros relevantes

- **Learning Rate:** Es, probablemente, el hiperparámetro más importante. La estrategia de selección del *learning rate* más utilizada consiste en comenzar con un valor muy pequeño (10^-5) e incrementarlo hasta un valor grande (1 ó 10)
- **Función de optimización:** Tal y como hemos visto en temas anteriores, existen varias funciones de optimización que pueden ayudar a mejorar el resultado del algoritmo
- **Batch size**: Un tamaño de *batch size* elevado permite que un componente hardware como la GPU sean más eficientes procesando los ejemplos de entrenamiento. Sin embargo, en la práctica, entrenar con un *batch size* elevado puede conducir a inestabilidades en el entrenamiento que generen un modelo que no sea capaz de generalizar bien. Como normal general, utilizar *batches* de tamaño cercanos a **32** suele ser la selección de preferencia
- **Función de activación:** Se han discutido en secciones anteriores las funciones de activación más populares y el efecto que produce cada una de ellas en el modelo. En general, la función de activación *relu* suele ser la selección de preferencia

Ejercicio: Selección de hiperparámetros con Keras Tuner y clasificación de texto

Tal y como hicimos para el conjunto de datos de **Reuters** (tema de optimización), carga el conjunto de datos, preprocésalo, y divide *test* en *test* y *val*.

Ahora construye una red neuronal artificial usando ***kerastuner***, como hemos visto en el caso práctico anterior y (selección de hiperparámetros con Keras Tuner), pero ahora añadiremos alguno más, como el número de capas, en *def model_builder(hp)*:

- Definición del modelo (no es necesario preprocesamiento, Flatten)
- Tuning del número de neuronas de las hidden layer
- **Tuning del número de capas:** *hp_layers = hp.Int('layers', min_value=1, max_valule=5, step=1)*
- Input layer: model.add(layers.Dense(units=128, activation='relu', input_shape=(10000,)))
- **Hidden layers:** *for i in range(hp_layers): model.add(layers.Dense(units=hp_units, activation='relu'))*
- Output layer: model.add(layers.Dense(46, activation='softmax'))
- Learning rate
- model.compile: …, *loss='categorical_crossentropy'*, …

Instancia *tuner = kt.Hyperband(...)* y ejecútalo con *tuner.search(...)*

Obtén los hiperparámetros óptimos (best_hps = *tuner.get_best_hyperparameters(...)*) y

entrena el modelo (*tuner.hypermodel.build(best_hps)*)

Obtén las gráficas con loss & val_loss y accuracy & val_accuracy, y evaúa cómo etiqueta nuestro modelo entre los 46 temas posibles de las noticias (deberías obtener sobre 0.8 de accuracy para el subconjunto de test). Ten en cuenta que lo normal es que produzca overfitting por ser un conjunto de datos muy pequeño.