Print to PDF

Regularización

Contents

Sobreajuste y subajuste

El sobreajuste y el subajuste son problemas comunes en el entrenamiento de modelos de *machine learning* que afectan la capacidad del modelo para generalizar correctamente a nuevos datos.

- *Sobreajuste (Overfitting)*:
  Ocurre cuando el modelo aprende en exceso los datos de entrenamiento, incluidos los ruidos del conjunto de entrenamiento. Esto resulta en un alto desempeño en el conjunto de entrenamiento, pero un bajo desempeño en datos nuevos o de prueba. En otras palabras, el modelo memoriza los datos en lugar de aprender patrones generales. Este problema puede presentarse cuando el modelo es demasiado complejo para la cantidad o la naturaleza de los datos de entrenamiento.
- *Subajuste (Underfitting)*:
  Sucede cuando el modelo es demasiado simple o no ha sido entrenado durante suficientes épocas, lo que le impide captar los patrones importantes en los datos de entrenamiento. Esto provoca un rendimiento bajo en los conjunto de entrenamiento y prueba. El subajuste puede surgir cuando el modelo es muy simple y no puede ajustarse a la complejidad de los datos o cuando hay demasiado ruido en el conjunto de entrenamiento.

Fig.1.Overfitting y underfitting Taye MM

Regularización

La regularización es un conjunto de técnicas utilizadas para abordar el problema del sobreajuste en los modelos de machine learning, reduciendo los problemas de generalización y tratando de afectar lo menos posible el error de entrenamiento. Sin embargo, un aumento en la capacidad de generalización puede provocar una ligera disminución en la precisión del modelo.

Tipos de Regularización

Modificación de la muestra o los datos:

- **Aumento de datos (Data Augmentation):** Esta técnica implica incrementar el tamaño del conjunto de datos disponibles mediante la creación de entradas artificiales. Esto ayuda a mejorar la robustez del modelo al exponerlo a una mayor variedad de ejemplos.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
import matplotlib.pyplot as plt
import numpy as np
import requests
from io import BytesIO
from PIL import Image
```

```
image_url = "https://github.com/Luisafrodriguezo1/programacionMAD/blob/main/logo%20igac12.png?raw=true"  # URL de
response = requests.get(image_url)
image = Image.open(BytesIO(response.content))
image = image.convert("RGB")

image = image.resize((150, 150))
```

```
image_array = img_to_array(image)
image_array = np.expand_dims(image_array, axis=0)  # Añadir una dimensión extra para representar el batch
```

```
# Configuración del generador de datos para aumentar las imágenes
datagen = ImageDataGenerator(
    rescale=1.0 / 255,          # Escalar los valores de los píxeles entre 0 y 1 para normalizar
    rotation_range=40,          # Rotaciones aleatorias de hasta 40 grados
    width_shift_range=0.2,      # Desplazamiento horizontal de hasta un 20%
    height_shift_range=0.2,     # Desplazamiento vertical de hasta un 20%
    shear_range=0.2,            # Cizalladura (deformación) de hasta un 20%
    zoom_range=0.2,             # Zoom aleatorio de hasta un 20%
    horizontal_flip=True        # Volteo horizontal aleatorio
)
```

```
# Generar imágenes aumentadas a partir de la imagen cargada
augmented_images = datagen.flow(image_array, batch_size=1)
#batch_size especifica cuántas imágenes deben ser generadas y devueltas por cada iteración del generador.
#En este caso, el valor es 1, lo que significa que en cada iteración el generador devuelve una sola imagen aument
```

```
fig, axes = plt.subplots(1, 6, figsize=(20, 5))  # Crear una fila de 6 imágenes

# Mostrar la imagen original
axes[0].imshow(image_array[0] / 255.0)  # Escalar la imagen para mostrar correctamente
axes[0].set_title("Original")  # Título para la imagen original
axes[0].axis("off")  # Quitar los ejes de la visualización

# Generar y mostrar 5 imágenes aumentadas
for i in range(5):
    augmented_image = next(augmented_images)[0]  # Obtener la siguiente imagen aumentada
    axes[i + 1].imshow(augmented_image)  # Mostrar la imagen aumentada
    axes[i + 1].set_title(f"Aumentada {i + 1}")
    axes[i + 1].axis("off")


plt.tight_layout()
plt.show()
```

- **Validación cruzada:** La validación cruzada es una técnica para evaluar modelos de machine learning mediante el entrenamiento de varios modelos en subconjuntos de los datos de entrada disponibles y evaluándolos con el subconjunto complementario de los datos. Por ejemplo:
- **K-Fold:** Este algoritmo comienza con la mezcla aleatoria de los datos y la inicialización del parámetro k, que es simplemente un número entero que define el número de particiones en las que se dividirán los datos. Por ejemplo, en un esquema de 5-fold, el conjunto de datos se divide en 5 partes, y se entrena el modelo 5 veces, cada vez utilizando una parte diferente como conjunto de prueba y las restantes como conjunto de entrenamiento.

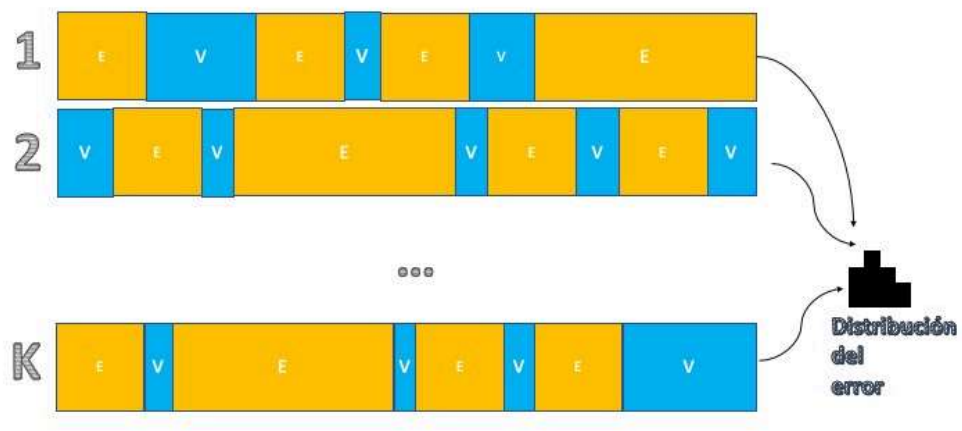

Fig.2. K-Fold Taye MM

Modificación del modelo

Dropout: Esta técnica consiste en desconectar aleatoriamente neuronas durante el entrenamiento para evitar que el modelo dependa demasiado de cualquier neurona en particular.Por ejemplo, si establecemos un valor de Dropout del 50%, estaremos indicando que en cada iteración del entrenamiento se deshabilitará el 50% de las neuronas de esa capa. Esto ayuda a prevenir el sobreajuste. `Dropout(0.5)`

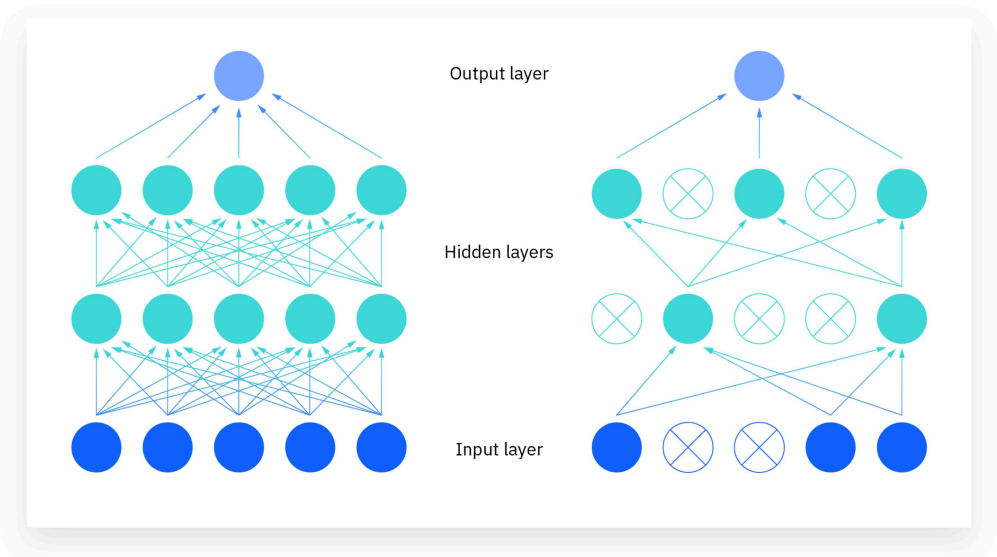

Fig.2. Dropout Taye MM

Regularización L2 (Ridge) y Regularización L1 (Lasso): Penaliza al error de los coeficientes

Regularización L2 (Ridge): Agrega una penalización al error del modelo que es proporcional al cuadrado de la magnitud de los coeficientes. Esto tiende a reducir la complejidad del modelo y a evitar que se ajuste demasiado a los datos de entrenamiento.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

Donde:

- $J(\theta)$ es la función de pérdida total.
- m es el número de ejemplos de entrenamiento.
- $y^{(i)}$ es el valor real de la i-ésima muestra.
- $h_\theta(x^{(i)})$ es la predicción del modelo para la i-ésima muestra.
- λ es el parámetro de regularización.
- θ_j son los parámetros del modelo.

Regularización L1 (Lasso): Agrega una penalización que es proporcional a la magnitud absoluta de los coeficientes. Esta técnica puede llevar a que algunos coeficientes se conviertan en cero, lo que implica que algunas características del modelo se eliminan completamente, ayudando a la selección de características y a la interpretación del modelo.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 + \lambda\sum_{j=1}^{n}|\theta_j|$$

Donde:

- $J(\theta)$ es la función de pérdida total.
- m es el número de ejemplos de entrenamiento.
- $y^{(i)}$ es el valor real de la i-ésima muestra.
- $h_\theta(x^{(i)})$ es la predicción del modelo para la i-ésima muestra.
- λ es el parámetro de regularización.
- θ_j son los parámetros del modelo.

```python
from keras.regularizers import l1, l2
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense, Dropout

model_r = Sequential([
    # Primera capa convolucional con activación ReLU y zancadas de 1
    Conv2D(64, (3, 3), activation='relu', input_shape=(8, 8, 10), padding='same', strides=(1, 1)),
    MaxPooling2D(pool_size=(2, 2)),

    # Segunda capa convolucional con activación Sigmoid y zancadas de 1
    Conv2D(64, (3, 3), activation='sigmoid', padding='same', strides=(1, 1)),
    MaxPooling2D(pool_size=(2, 2)),

    # Tercera capa convolucional con activación Tanh y zancadas de 1
    Conv2D(32, (3, 3), activation='tanh', padding='same'),

    # Normalización por lotes para estabilizar el entrenamiento
    BatchNormalization(),

    # Aplanar la salida para conectar con las capas densas
    Flatten(),

    # Capa densa completamente conectada con activación ReLU y regularización L1
    Dense(64, activation='relu', kernel_regularizer=l1(0.01)),

    # Capa Dropout para reducir el sobreajuste
    Dropout(0.2),

    # Capa densa con activación Sigmoid y regularización L2
    Dense(32, activation='sigmoid', kernel_regularizer=l2(0.01)),

    # Capa de salida con activación Softmax para clasificación en 4 clases
    Dense(4, activation='softmax')
])
```