INTRO A REDES NEURONALES

U1) Introducción a Redes Neuronales (scikit-learn)

1. ¿Que diferencia hay entre una red neuronal y un modelo de regresión lineal?
    a. Las redes neuronales son más complejas y pueden aprender relaciones no lineales.
2. ¿Qué diferencia hay entre un perceptrón y una regresión lineal?
    a. Que el perceptrón tiene una función de activación.
3. ¿Qué es la retropropagación del error?
    a. Un algoritmo para optimizar los pesos de una red neuronal
4. ¿Qué es una función de activación?
    a. Una función que transforma la salida de una neurona.
5. ¿Qué es el descenso del gradiente?
    a. Un algoritmo para optimizar los pesos de una red neuronal
6. ¿Qué es la tasa de aprendizaje?
    a. Un parámetro que controla la velocidad a la que la red neuronal aprende.
    b. Aumentarla puede hacer que no converja o que converja a un mínimo local. Inestabilidad. Pero es mucho más rápida.
    c. Generalmente se inicia con tasas bajas y después se va subiendo.
7. ¿Qué es la época en el entrenamiento de deep learning?
    a. Una iteración completa sobre el conjunto de datos de entrenamiento, mini-batches, subconjuntos aleatorios de datos.
    b. El algoritmo de optimización coge todo el dataset, pero en vez de utilizarlo todo, lo parte en cachitos batchs.
    c. Si hicieramos el entrenamiento en plan bruto, le meteríamos todos los datos al fit, sin embargo vamos a meterle batches. Por cada batch se hace un cálculo de los nuevos coeficientes y pasamos a la siguiente época.
    d. No podemos meterle todos los datos todo el tiempo.
8. ¿Por qué son importantes las funciones de activación no lineales en el deep learning?

    a. Introducen flexibilidad, permitiendo modelar relaciones complejas.
9. ¿Qué es un ‘batch’ en el contexto del entrenamiento de deep learning?
    a. Un subconjunto de los datos de entrenamiento utilizados en una iteración.
10. ¿Cuántos fits hace GridSearch?
    a. Uno por cada combinación de parámetros en el grid por cada fold. Finalmente hay un último entrenamiento con los mejores parámetros y todos los datos.
    b. Conviene hacer unos números antes de lanzar un GridSearch

CONCLUSIONES A LA PRÁCTICA

¿Ha habido una gran mejoría entre el RandomForest y el MLP?

- No, ambos modelos tenían un performance parecido.
- El tiempo de entrenamiento, en cambio, sí.
- El RF es más fácil de configurar.
- Las redes neuronales no funcionan bien con datos tabulares.
- Por lo tanto: menos reproducibilidad y menos explicabilidad.
- Para el dataset del Titanic, MLP no tiene sentido.
- Los modelos de deep son muy buenos cuando tienen que captar esas estructuras, por ejemplo ChatGPT.
- Los modelos de deep funcionan muy bien para cuando los datos no están estructurados (NLP, por ejemplo, no viene por columnas) pero sí tienen estructura (Ver Noam Chomsky).
- ¿Qué grandes grupos de aplicaciones (nucleares) tenemos en ML?
    - Regresión
    - Clasificación

    - Clustering (agrupaciones, como clasificar cuando no me sé las clases)
- Los modelos de deep learning:
    - Resuelven muy bien problemas con datos brutos
    - Combinaciones de varias etapas de los tres grandes problemas dan lugar a aplicaciones impresionantes.
    - Por ejemplo, detectar una persona es un problema de regresión con 4 coordenadas, las coordenadas de la persona en la imagen.
    - ¿Qué tipo de problema usa un algoritmo de deep learning para resolver un problema de resumir un texto?
        - Se reduce a dar la siguiente palabra. Empieza por la palabra más probable y después va escogiendo palabras hasta que algo le dice que pare. Es un problema de clasificación secuencial. Genera palabra 1, palabra 2, palabra n, palabra final. Elige palabras de un diccionario para cada palabra. Multi-categórico de 93.000 clases
    - ChatGPT, por ejemplo, genera probabilidades para la siguiente palabra. ChatGPT parece inteligente pero no lo es. Da la apariencia de inteligencia. Capta estructuras a una profundidad increíble y con un dataset gigante. “**TODO ESTÁ ESCRITO y BIEN ESCRITO”.**
    - ***s***
    - ***Retrieval-Augmented Generation****. Es un enfoque que combina técnicas de recuperación de información con técnicas de generación de texto para mejorar la calidad y la precisión de las respuestas generadas por modelos de lenguaje.*

        - *No se basa directamente en técnicas tradicionales de clasificación o regresión, pero sí puede involucrar elementos de ambas en diferentes etapas del proceso.*
    - Resumen:
        - DeepLearning es mejor para datos no estructurados pero que tienen una estructura subyacente. Penalty de tiempo de entrenamiento.
        - El DeepLearning tiene ya modelos pre-entrenados que te descargas y sólo entrenas la parte correspondiente a tu problema partiendo de la red ya pre-entrenada. Esto es muy potente y se llama transfer learning.
        - El Machine Learning es mejor con datos tabulares.
        - *Graph Neural Networks es una línea de investigación para DNN que se supone aumentará el performance con datos tabulares.*

GridSearchCV con MLP

Con redes neuronales vamos a trabajar en general en el siguiente orden.

- La topología
- Activation
- Solver
- Regularización
- Learning rate

Deep Neural Networks o Multi Layer Perceptron:

Cada Perceptrón es un modelo en sí mismo. Una regresión lineal con una función de activación con una sigmoide, es decir un LogisticRegressor.

Otra forma de ver una red neuronal es la de “transformador de features” que convierte las n features de entrada en las m features que tengo a la salida, aplicando una regresión logística.

El algoritmo en su conjunto se puede considerar un super extractor de features.

El DNN se puede considerar un modelo para el que no hace falta fijarse mucho en las features de entrada y se consigue un modelo potente sin hacer análisis visual. Es de vagos 🙂

No es de vagos, diseñar las capas requiere mucho trabajo.

Cuando 50 entradas y un primera hidden layer con una estructura de (25,50), estamos convirtiendo las 25 en una expansión a 50. Después puede venir una capa de compresión. Esto se llama prisma.

Si primero se expande y después va reduciéndose se llama pirámide.

Los números de capas y su número de neuronas no son una ciencia, es más empírico, con alguna guía.

Cuando empezamos a entrenar la red, ¿por qué todos los pesos ya están inicializados? Porque si todos los pesos valen 0, no puede funcionar el SGD. Por lo tanto también hay inicializadores, que puede cambiar mucho el panorama.
Las redes neuronales son tan resource-consuming que no se hace GridSearch. Entonces, se suele ser más selectivo, y después se tunean los parámetros poco a poco.

TEORÍA:

Épocas (epochs)

Una época se refiere a una pasada completa por todo el conjunto de datos de entrenamiento. Es decir, cuando el modelo ha visto todos los ejemplos de entrenamiento una vez, se considera que una época ha

terminado. Por lo general, los modelos de redes neuronales se entrenan durante múltiples épocas, lo que significa que el conjunto de datos se utiliza varias veces para actualizar los pesos del modelo. El número de épocas es un hiperparámetro que se puede ajustar durante el entrenamiento. Un mayor número de épocas puede permitir al modelo aprender mejor, pero también puede aumentar el riesgo de sobreajuste (overfitting) si se entrena demasiado.

Batches (minibatches)

Debido a que pasar todo el conjunto de datos de entrenamiento a través de la red neuronal de una vez puede ser computacionalmente costoso y lento, se suele dividir el conjunto de datos en subgrupos más pequeños llamados batches o minibatches. Cada batch contiene un número fijo de ejemplos y se utiliza para actualizar los pesos del modelo una vez. Este proceso se llama entrenamiento en minibatches (minibatch training) y permite aprovechar la computación paralela y reducir el consumo de memoria.

El tamaño del batch, también conocido como batch size, es otro hiperparámetro importante. Si el tamaño del batch es 1, cada actualización se realiza después de cada ejemplo (esto se conoce como entrenamiento online o estocástico). Si el tamaño del batch es igual al tamaño del conjunto de datos, se llama entrenamiento en batch completo o entrenamiento batch. Los tamaños de batch más comunes son intermedios, como 32, 64 o 128 ejemplos por batch.

Relación entre épocas y batches

Supongamos que tienes un conjunto de datos de 1000 ejemplos y eliges un tamaño de batch de 100:

- Una época completa implica pasar los 1000 ejemplos a través del modelo.
- Con un tamaño de batch de 100, esto se traduce en 10 batches por época.
- Después de procesar cada batch, los pesos del modelo se actualizan.

- ¿Cuántos batches se entrenan en cada época? Batch-size batches.

Parámetros de un entrenamiento para MLP:

- modelo__activation: Define la función de activación para las capas ocultas de la red neuronal. En este caso, se están considerando dos opciones: 'tanh' y 'relu'.
- modelo__alpha: Es el parámetro de regularización que controla la fuerza de la penalización L2 (también conocida como regularización de Ridge) en los pesos de la red neuronal. Aquí se están probando dos valores: 0.0001 y 0.05.
- modelo__hidden_layer_sizes: Es una tupla que indica el número de neuronas en cada capa oculta de la red neuronal. En este caso, parece haber una sola capa oculta con 100 neuronas.
- modelo__learning_rate: Especifica cómo se actualizan los pesos durante el entrenamiento de la red neuronal. Puede ser 'constant', lo que significa que la tasa de aprendizaje se mantiene constante, o 'adaptive', lo que significa que la tasa de aprendizaje se ajusta en función de si la pérdida de validación no mejora.
- modelo__solver: Es el algoritmo de optimización utilizado para entrenar la red neuronal. En este caso, se está utilizando 'sgd', que se refiere al descenso de gradiente estocástico (Stochastic Gradient Descent).

U2) KERAS: The Sequential API

Quiz:

¿Qué es KERAS?

¿Qué es un modelo secuencial? Existen otras estructuras con bifurcaciones y demás.

¿Qué es una red Densa?

¿Cuál es el número adecuado de capas, el número de perceptrones por capa, función de activación, etc.?

¿Qué es la precisión como métrica de una clasificación?

¿Y el recall?

¿Qué es el termino bias en una neurona?

¿Un perceptrón se puede entender como una regresión lineal a la que le aplicamos una X? Explica qué es X.

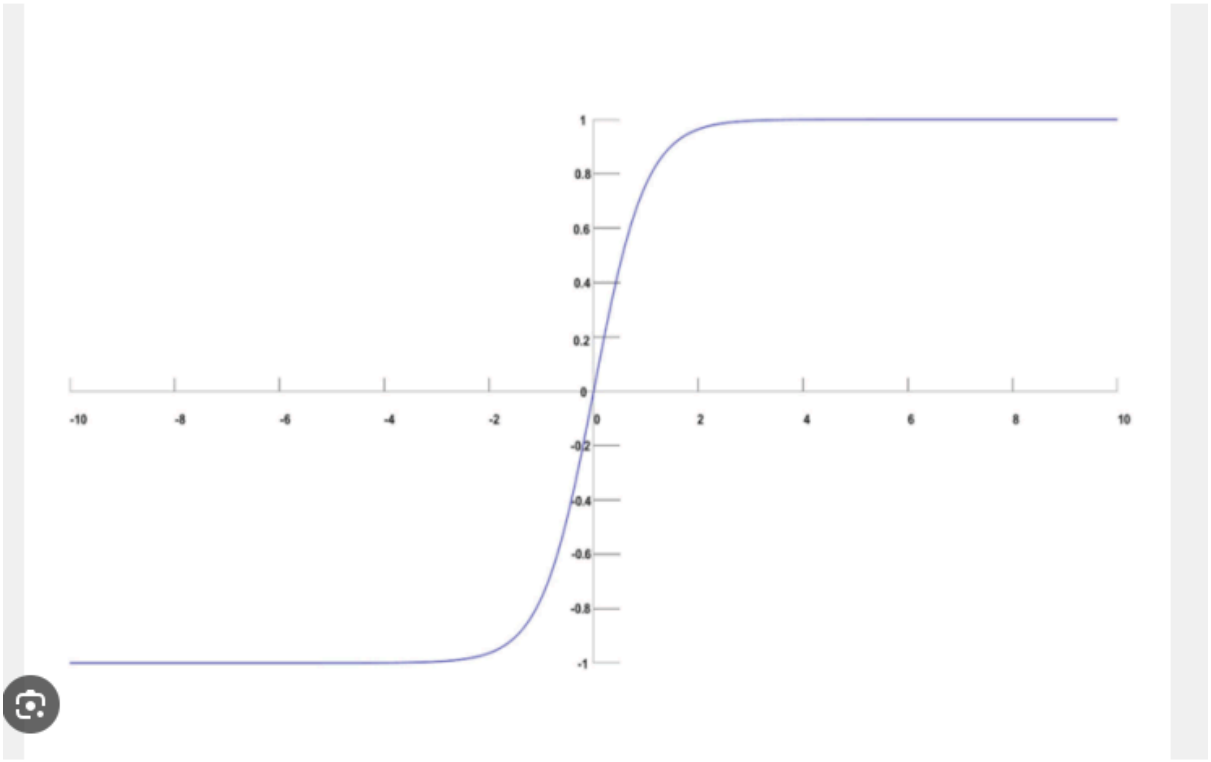
¿Qué quiere decir *nivel de confianza* en una clasificación? No te compliques.

¿Qué forma tiene una red con salida *"softmax"*? Recordatorio: salida entre -1 y 1.

¿Y una con salida de tipo "*sigmoid"*? Recordatorio: salida entre 0 y 1.

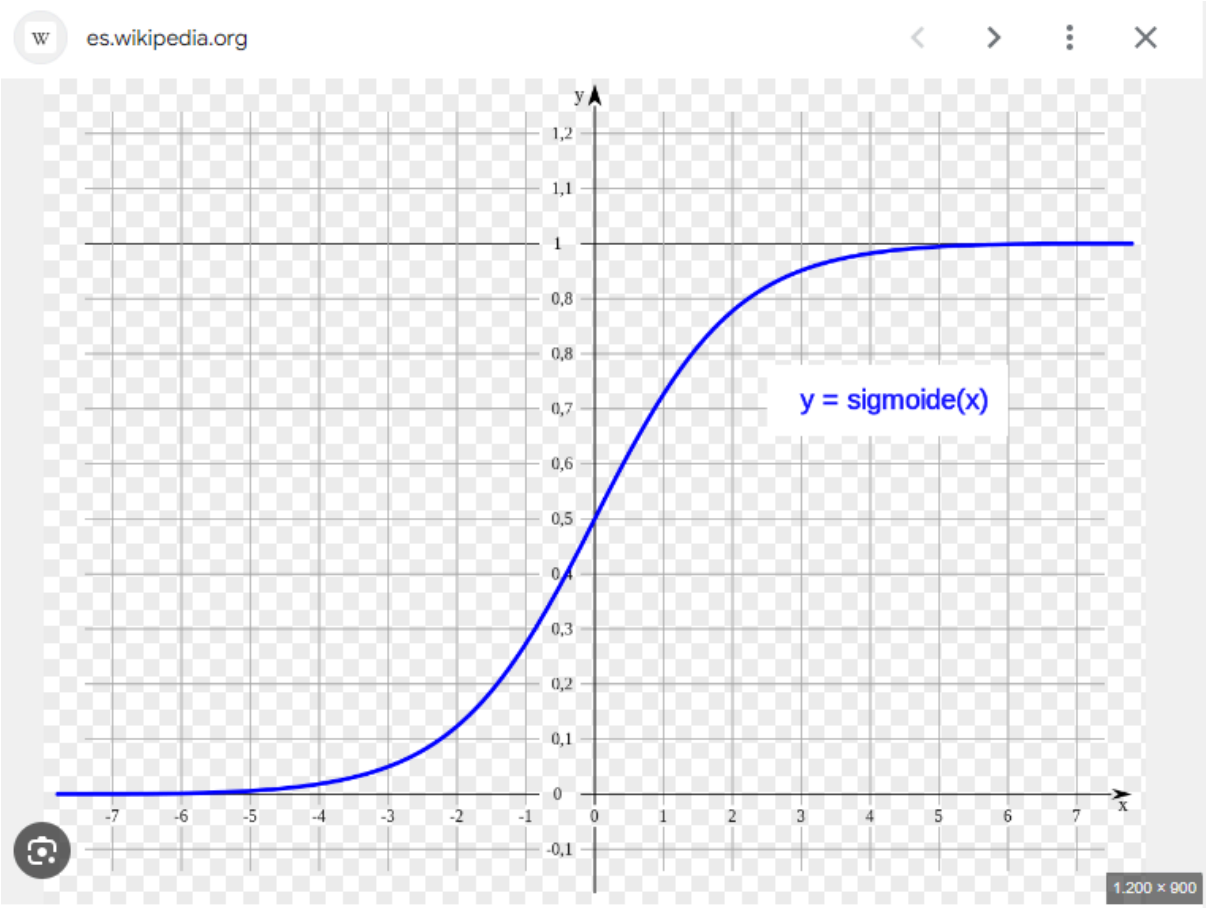
En una red neuronal con 5 entradas, una sola hidden layer de 10 neuronas y un output de 1 neurona, ¿cuantos términos de bias hay?

Jacar



Función SoftMax: Activación para la clasificación - Jacar

Visitar >



¿Qué tipos de datos de entrada esperan las redes neuronales?

¿Cuántas salidas va a tener un modelo Keras que clasifique entre 10 categorías?

¿Qué es categorical cross-entropy?

¿Qué neuronas tienen un término de bias en una red de arquitectura (input = 10), (hidden = 50 y 25) y (output = 1)?

¿Cuántos parámetros tendrá?

(10) + (50 * 25 + 25 de bias) + (25 + 1 de bias) = 1,311.

Introducción:

- Hoy resolvemos un problema de ML mediante un modelo de red neuronal (librería Keras).
- Vamos a clasificar las prendas del dataset MNIST.
    - Tiene 10 tipos de prendas así que tenemos que crear un **clasificador multiclase.**
    - Esto implica un modelo Keras con 10 salidas.
- Utilizamos distintas estructuras de redes intermedias para ver cual tiene la mejor performance.
- Vamos a profundizar en qué es lo que más falla de nuestro modelo.
    - Vamos a estudiar errores cruzados.

Enunciado de la práctica:

Fashion MNIST dataset.

- Tiene 70,000 imágenes de prendas de vestir.
- Se pide clasificar en 10 categorías las prendas.
- La resolución de las imágenes es baja 28x28.
- Clases:
    - T-shirt -> 0
    - Trouser -> 1
    - Pullover -> 2
    - Dress -> 3
    - Coat -> 4
    - Sandal -> 5
    - Shirt -> 6
    - Sneaker -> 7
    - Bag -> 8
    - Ankle-boot -> 9
- Se pide un clasificador basado en redes neuronales de Keras. No es Deep Learning a esta escala.
    - Se recomiendan 1 o 2 capas ocultas.

    - Necesitarás aplanar las imágenes de 28x28 a 784.
- Muestra la evolución del entrenamiento y la validación con un 20% de los datos.
    - Razona si conviene un callback de earlystopping.
    - Implementa uno de todas formas
- Compara el resultado con el test set
- Muestra qué clases se confunden con qué clases.
- EXTRA: obtén los errores de clasificación en los que la confianza del modelo sobre su clasificación errónea supere el 0.7 tanto por 1 (es decir los errores en los que la probabilidad de la clase elegida, equivocadamente, fuera igual o superior 0.7), de existir muéstranos y analiza si son de alguna clase específica.

Desarrollo de la práctica

- **Cargar los datos FashionMNIST**.
- Shape del train 60000x28x28, que ocupa 47 MB
    - ¿Son muchos datos, pocos, qué podemos decir?
    - Cada etiqueta es un escalar en el dataset: de 0 a 9.
    - Resolución baja.
    - Normalizamos dividiendo por 255 para tener tantos por 1s, necesario para las entradas de la red neuronal
- Usamos *show_images_batch* de un Sprint anterior para visualizar alguna instancia.
- **Mini-EDA y Feature Engineering**:
    - Target: balanceo perfecto
    - Por lo demás no hacemos bivariante por ser pixeles flattened.

    - Escalado de features: 1/255.
    - Es fundamental estandarizar los datos con StandardScaler, incluso si ya están en una escala de 0 a 1 (tanto por uno). Estandarizar los datos puede mejorar el rendimiento y la velocidad de convergencia de la red.
- **Modelado**: diseñamos redes neuronales con distintas estructuras:
    - **“Model”**
        - Flatten (28,28)
        - Dense(128,relu)
        - Dense(10,softmax)
    - **“Model_Vanessa”**:
        - Input (shape (28,28)) -> 784 entries
        - Flatten() -> 784 entries
        - Dense_1(300, relu) ampliamos.
        - Dense_2(300, relu) no aseguramos de tener suficiente.
        - Output (10,softmax) una salida por clase, entre 0 y 1
    - **“Samuel”**
        - Flatten(28x28,relu)
        - Dense(150,relu)
        - Dense(75,relu)
        - Dense(10,softmax)

```
model_vanessa.summary()
```

✓ 0.0s

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28)	0
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 300)	235,500
dense_3 (Dense)	(None, 100)	30,100
dense_4 (Dense)	(None, 10)	1,010

ChatGPT

Si deseas utilizar un modelo secuencial en Keras para optimizar la clasificación del dataset MNIST, puedes construir una red neuronal profunda simple. Aquí tienes un ejemplo de una arquitectura secuencial básica para este propósito:

python — Copiar código

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Crear el modelo secuencial
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Aplanar la imagen de 28x28 a un vector de 784
    Dense(128, activation='relu'),  # Capa densa con 128 unidades y activación ReLU
    Dense(64, activation='relu'),   # Capa densa con 64 unidades y activación ReLU
    Dense(10, activation='softmax') # Capa de salida con 10 unidades y activación softmax
])
```

NADIE SABE CUÁL ES LA ESTRUCTURA ÓPTIMA

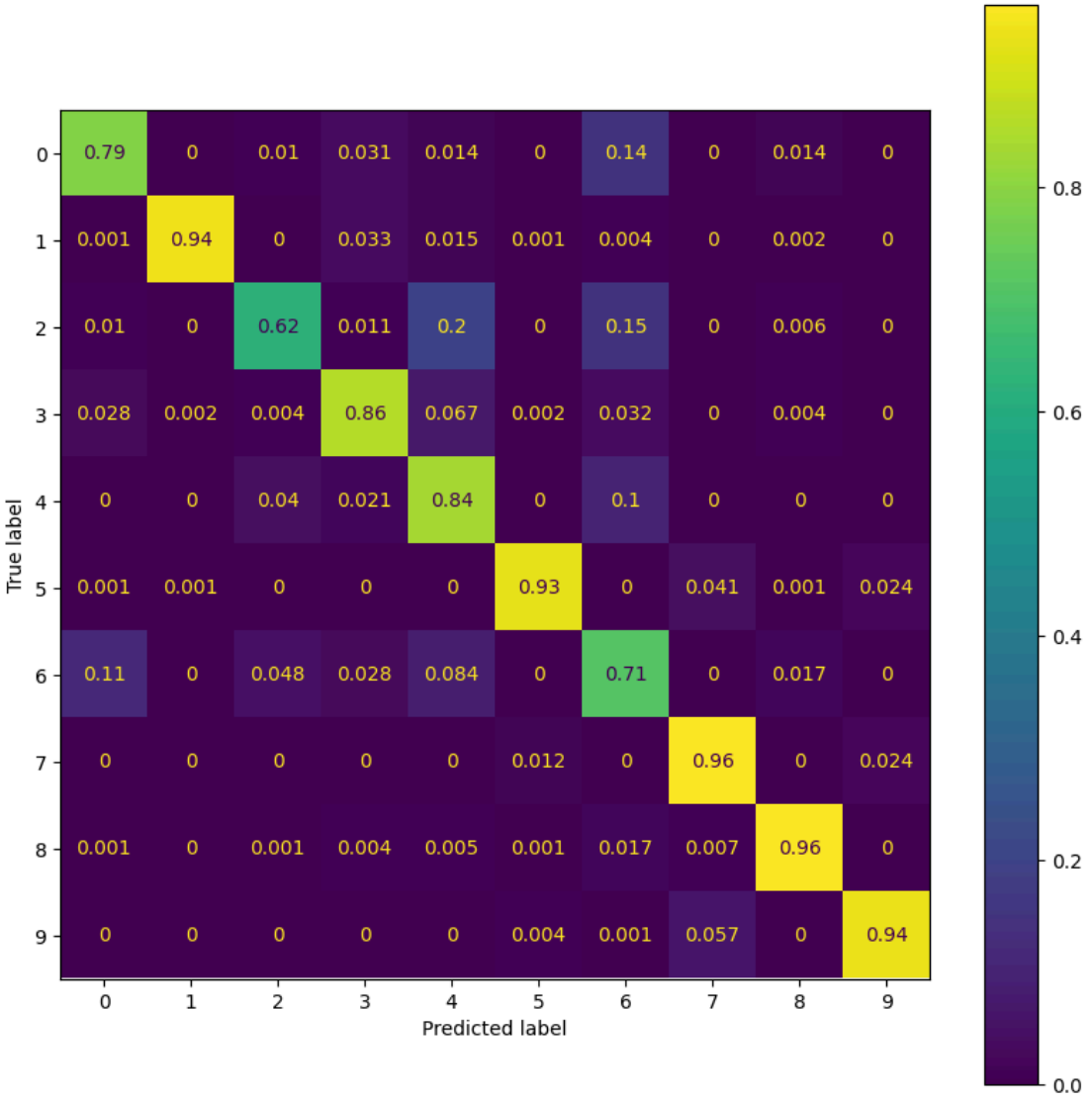
- **Más adelante lo complicamos más con:**
  - **Redes neuronales convolucionales CNN**

- **Autoencoders**

Así se calcula **y_pred** para **multiclase** con Keras: coge el número de celda del predictor que ha dado un número más alto.

```
y_pred = [np.argmax(prediction) for prediction
in model_vanessa.predict(test_images)]
```

- **Miramos la evolución del error con las epochs y vemos que no hay mejoría en el validation dataset a partir del epoch 3 o 4. Un poco de overfitting. Deberíamos hacer un callback para frenar a tiempo.**
- Ahora calculamos las métricas con **'evaluate'**
  - El performance es parecido entre los modelos pero, por poco, **Vanessa** gana con un **89 % de accuracy**.
  - Seleccionamos el modelo **Vanessa**.



- Miramos errores cruzados:
  - La clase 6 se confunde con la 0 y con la 2
  - La clase 4 se confunde con la 2.
  - Veamos unos ejemplos.
  - Hacemos una máscara y seleccionamos errores de predichas 6, reales 2, reales 0 y predichas 4.
  - Vemos que se confunden T-shirts con Shirt y que se confunden Pullover con Coat.

- Observamos las imágenes con confidence level alto pero mal clasificadas:
  - Creamos un DataFrame con True/Predicted/Confidence.
  - Hacemos una máscara con los fallos

    - y otra con las imágenes que tienen un confidence 0.7 o mayor.
    - Observamos las características de esas imágenes. e.g.: value_counts()
- Concluimos que el 6 es el número que más aparece seguido por el 4.

Conclusiones:

- Distintas topologías de redes tienen distinta performance.
- Las capas de entrada y de salida vienen determinadas por el problema técnico.
- No existe una receta para cuántas capas intermedias (hidden) ni cuántos perceptrones.
- Con el tiempo se desarrolla la intuición.
- No hacemos validación cruzada ni GridSearchCV dado el alto consumo de recursos de CPU y RAM.
- Sí que reservamos un 20% de los datos de entrenamiento aproximadamente para hacer una comparación, época a época, de los progresos de las métricas en cuestión.
- El estudio del performance es un poco más técnico ya que, para clasificadores multi categóricos, tenemos una red neuronal con 10 salidas, por ejemplo, que predicen valores entre 0 y 1. Lo ideal sería que, en todos los casos que tenga que clasificar, uno de los 10 predictores nos diera un valor cercano a un 0.9999 y que todos los demás 0.00001%. Y que además acertara, claro. Entonces estaría clarísimo qué valor predicho es la etiqueta del predictor que da el 0.9999. Eso puede ser un mono, una camisa, una leona.
- Las redes neuronales son algoritmos más complejos que los que hemos visto.

Quiz copia

¿Qué es KERAS?

Keras es una biblioteca de código abierto diseñada para construir y entrenar modelos de aprendizaje profundo (deep learning) de manera sencilla y rápida.

¿Qué es un modelo secuencial? Existen otras estructuras con bifurcaciones y demás.

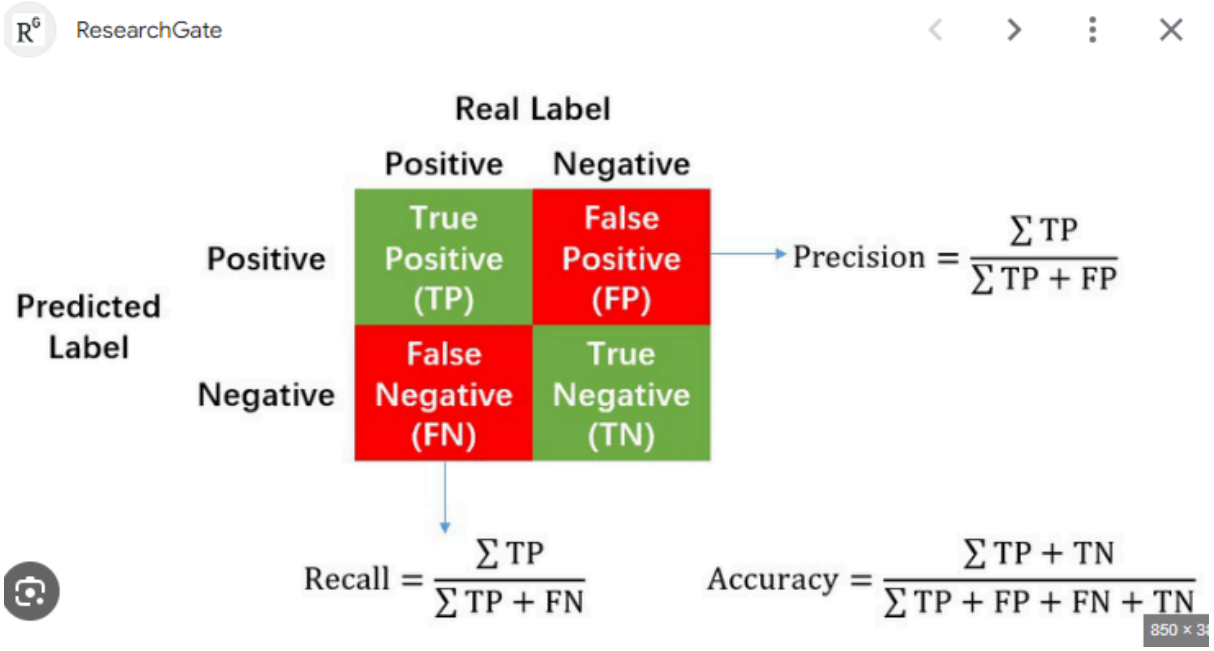
Un modelo secuencial es un tipo de arquitectura de red neuronal que organiza sus capas de manera lineal y secuencial. Es decir, las capas están dispuestas una tras otra, en una secuencia definida, donde la salida de una capa se convierte en la entrada de la siguiente.

¿Qué es una capa Densa?

Una capa densa, también conocida como una capa completamente conectada o fully connected layer en inglés, es un componente fundamental en las redes neuronales artificiales.

¿Cuál es el número adecuado de capas, el número de perceptrones por capa? ¿Función de activación?

¿Qué es la precisión como métrica de una clasificación?



¿Y el recall?

¿Qué hace el término bias en una neurona?

Se refiere a un parámetro adicional en el modelo que se utiliza para ajustar la salida de la neurona, permitiendo que el modelo se ajuste mejor a los datos. No influye el peso de ningun perceptrón anterior.

¿Un perceptrón se puede entender como una regresión lineal a la que le aplicamos una X? Explica qué es X.

¿Qué quiere decir *nivel de confianza* en una clasificación? No te compliques.

¿Qué forma tiene una red con salida *"softmax"*? Recordatorio: salida entre -1 y 1.

¿Y una con salida de tipo "*sigmoid*"? Recordatorio: salida entre 0 y 1.

****

En una red neuronal con 5 entradas, una sola hidden layer de 10 neuronas y un output de 1 neurona, ¿cuantos términos de bias hay?

Teoría:

Keras General:

Keras es una biblioteca de código abierto diseñada para construir y entrenar modelos de aprendizaje profundo (deep learning) de manera sencilla y rápida. Es conocida por su facilidad de uso, modularidad y extensibilidad. Originalmente, Keras fue una biblioteca independiente que podía usar múltiples backends (Theano, TensorFlow, CNTK). Sin embargo, desde 2017, Keras se ha integrado directamente en TensorFlow y se utiliza principalmente con este backend.

Características Principales de Keras:

1. **Facilidad de Uso**:
    - **API Intuitiva**: Keras está diseñada para ser fácil de usar y rápida para crear prototipos. Ofrece una API sencilla y coherente que permite construir modelos de forma rápida y sin complicaciones.
    - **Mínimo Código**: La cantidad de código necesaria para crear un modelo en Keras es significativamente menor comparada con otros frameworks de deep learning.
2. **Modularidad**:
    - **Capas como Bloques de Construcción**: En Keras, los modelos se construyen utilizando una serie de capas, lo que facilita la creación y modificación de modelos.
    - **Modelos Secuenciales y Funcionales**: Keras proporciona dos tipos principales de modelos: el modelo Secuencial (Sequential), que es una pila lineal de capas, y el modelo funcional (Functional), que permite crear gráficos de modelos más complejos.
3. Extensibilidad:
    - Fácilmente Personalizable: Si necesitas una capa, métrica, función de pérdida, optimizador o cualquier otro componente que no esté disponible en Keras, puedes escribirlo tú mismo y agregarlo sin problemas.
    - **Soporte para GPUs y TPUs**: Keras, a través de TensorFlow, puede aprovechar la potencia de cálculo de GPUs y TPUs para acelerar el entrenamiento de modelos.
4. **Amplia Comunidad y Documentación**:

- Documentación Completa: Keras tiene una documentación extensa y detallada que facilita a los desarrolladores aprender y utilizar la biblioteca.
- Comunidad Activa: Una gran comunidad de usuarios y desarrolladores contribuye con ejemplos, tutoriales y soluciones a problemas comunes.

Modelo Secuencial:

Un modelo secuencial es un tipo de arquitectura de red neuronal que organiza sus capas de manera lineal y secuencial. Es decir, las capas están dispuestas una tras otra, en una secuencia definida, donde la salida de una capa se convierte en la entrada de la siguiente. Esta estructura es simple y directa, lo que la hace adecuada para muchos tipos de problemas de aprendizaje automático. Aquí hay algunas características y detalles importantes sobre los modelos secuenciales:

1. **Estructura Lineal:** En un modelo secuencial, las capas están organizadas en una secuencia estricta, sin bifurcaciones ni conexiones múltiples. Cada capa recibe la salida de la capa anterior y proporciona su salida a la siguiente capa en la secuencia.
2. **Simplicidad**: Esta estructura es fácil de entender y de implementar. Es adecuada para modelos en los que se desea un flujo de datos sencillo y directo desde la entrada hasta la salida.

Capa Densa

Una capa densa, también conocida como una capa completamente conectada o fully connected layer en inglés, es un componente fundamental en las redes neuronales artificiales. A continuación, se describen sus características y funcionamiento:

1. **Conexiones Completas**: En una capa densa, cada neurona está conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa siguiente. Esto significa que cada neurona recibe entradas de todas las neuronas de la capa precedente y envía su salida a todas las neuronas de la capa subsiguiente.
2. **Pesos y Biases**: **Cada conexión entre neuronas tiene un peso asociado, y cada neurona tiene un término de bias**. Estos pesos y biases son los parámetros que la red ajusta durante el proceso de entrenamiento para aprender a mapear las entradas a las salidas correctas.
3. **Función de Activación**: Después de calcular la suma ponderada de las entradas más el bias, el resultado se pasa a través de una función de activación no lineal (como ReLU, sigmoide, tanh, etc.). La función de activación introduce no linealidades en la red, permitiendo que pueda aprender y representar relaciones complejas en los datos.
4. Cálculo Matemático: Matemáticamente, la salida de una capa densa se puede expresar como:
   $salida=activacio´n(entradas \cdot pesos+biases)$ $\text{salida} = \text{activación}(\text{entradas} \cdot \text{pesos} + \text{biases})$
   donde "·" representa la multiplicación de matrices, "entradas" es el vector de entradas, "pesos" es la matriz de pesos, "biases" es el vector de bias y "activación" es la función de activación.
5. Uso en Redes Neuronales: Las capas densas se utilizan principalmente en redes neuronales de tipo perceptrón multicapa (MLP, por sus siglas en inglés) y en las capas finales de otras arquitecturas de redes neuronales más complejas, como redes convolucionales (CNN) y redes recurrentes (RNN), donde se encargan de la clasificación o regresión final basada en las características extraídas por las capas anteriores.

En resumen, una capa densa es esencialmente una matriz de conexiones completas entre neuronas de diferentes capas, con ajustes de pesos y bias para optimizar la precisión de la red en tareas específicas.

EarlyStopping Callback:

Para evitar el sobreajuste (overfitting) y reducir el tiempo de entrenamiento innecesario.

El early stopping monitorea la métrica de evaluación de un modelo durante el entrenamiento y detiene el proceso cuando la métrica deja de mejorar.

¿Cómo funciona el Early Stopping?

El early stopping monitorea una métrica específica en un conjunto de datos de validación (por ejemplo, la pérdida de validación o la precisión de validación). Si esta métrica no mejora después de un número definido de épocas consecutivas (llamado `patience` o paciencia), el entrenamiento se detiene. Esto **permite que el modelo se entrene sólo hasta el punto en que su desempeño en los datos de validación no mejora más, evitando así el sobreajuste a los datos de entrenamiento**.

RMSpropOptimizer:

El optimizador RMSprop (Root Mean Square Propagation) es un algoritmo de optimización. RMSprop es una técnica de ajuste del paso de aprendizaje (learning rate) adaptativa, que

intenta mantener un paso de aprendizaje eficiente y estable durante el entrenamiento.

Este optimizador es particularmente útil para lidiar con problemas de desvanecimiento del gradiente y para estabilizar el entrenamiento en redes profundas.

Categorical Cross-Entropy

Esta función de pérdida mide la discrepancia entre la distribución de probabilidad predicha por un modelo y la distribución de probabilidad real de los datos.

- Cuando el modelo predice correctamente la clase correcta con alta probabilidad (alta para la clase correspondiente), la pérdida es baja.
- Cuando el modelo predice incorrectamente la clase con alta probabilidad, la pérdida es alta.

La entropía cruzada (cross-entropy) es una medida utilizada en teoría de la información y en aprendizaje automático para cuantificar la diferencia entre dos distribuciones de probabilidad. En el contexto del aprendizaje automático, especialmente en problemas de clasificación, la entropía cruzada se utiliza como función de pérdida para evaluar la discrepancia entre las probabilidades predichas por un modelo y las probabilidades reales (o etiquetas).

La entropía cruzada mide la "cantidad de sorpresa" en la predicción del modelo. Si el modelo predice correctamente con alta probabilidad, la entropía cruzada es baja, indicando poca sorpresa. Si el modelo predice incorrectamente o con baja probabilidad, la entropía cruzada es alta, indicando alta sorpresa.

En resumen, es una herramienta fundamental en la optimización de modelos de clasificación multiclase, ayudando a guiar el proceso de aprendizaje para que el modelo pueda hacer predicciones más precisas sobre la clase de cada dato de entrada.

The bias term:

Con un ejemplo, mejor.

En una red neuronal con 5 entradas, una sola hidden layer de 10 neuronas y un output de 1 neurona, cuantas términos de bias hay?

Los términos de bias se añaden en cada capa, incluyendo la capa oculta y la capa de salida. Aquí está el desglose de los términos de bias:

1. Capa de Entrada:
    - No hay términos de bias en la capa de entrada porque solo transmite las entradas sin aplicar ninguna operación que requiera bias.
2. Capa Oculta:
    - La capa oculta tiene 10 neuronas.

    - Por lo tanto, hay 10 términos de bias asociados a las 10 neuronas de la capa oculta.
3. Capa de Salida:
    - La capa de salida tiene 1 neurona.
    - Por lo tanto, hay 1 término de bias asociado a la neurona de salida.

De prácticas anteriores:

Sobre si afecta el orden de las columnas al resultado en redes neuronales con datos tabulares.

En general, el orden de las columnas en los datos tabulares no debería afectar los resultados de un modelo de red neuronal, siempre y cuando el modelo esté correctamente implementado y entrenado.

Ahora bien, si los pesos varían en su inicialización, todo puede cambiar.

Es importante recordar que aunque el orden de las columnas no debería afectar los resultados, otros factores como el preprocesamiento de datos, la normalización, la codificación de características categóricas y el manejo de valores faltantes sí pueden tener un impacto significativo en el rendimiento del modelo.

Por lo tanto, asegúrate de manejar estos aspectos de manera adecuada para obtener los mejores resultados posibles.