

Ajustes de un modelo de aprendizaje automático.

Caso práctico



[LookStudio \(https://www.freepik.es/foto-gratis/hermosa-mujer-independiente-rizada-linda-manicura-usando-laptop-sonriendo-retrato-interior-secretaria-rubia-sentada-junto-companero-trabajo-africano-camisa-azul_10483920.htm#query=lookstudio&position=18&from_view=search\)](https://www.freepik.es/foto-gratis/hermosa-mujer-independiente-rizada-linda-manicura-usando-laptop-sonriendo-retrato-interior-secretaria-rubia-sentada-junto-companero-trabajo-africano-camisa-azul_10483920.htm#query=lookstudio&position=18&from_view=search)
(CC BY-SA (<http://creativecommons.org/licenses/?lang=es>)))

En la empresa Pick&Deliver están embarcados en un proceso de transformación de sus procesos, adaptándose a la nueva era de automatización basada en inteligencia artificial que ya aprovechan miles de empresas.

Se ha formado un equipo de técnicos que están trabajando en ello. Lorena entró en el equipo gracias a un contrato de prácticas, pero tras terminar éstas, le han ofrecido un contrato indefinido y ella ha decidido quedarse, pues el ambiente y oportunidad de aprender hacen lo suficiente.

Lorena ha estado probando y desarrollando varios modelos de aprendizaje automático para adelantarse a las incidencias en la logística y envío de pedidos, así como modelos de detección de situaciones de riesgo en el almacén a través de imágenes. Pero todavía está en proceso de mejorarlos, y afinarlos para que sean más precisos.

En esta unidad repasaremos el post-tratamiento que se hace cuando ya hemos creado y entrenado un modelo de aprendizaje automático utilizando redes neuronales profundas. En concreto:

- ✓ Evaluación de un modelo de deep learning y preparación del modelo predictivo
- ✓ Overfitting o sobre entrenamiento
- ✓ Recursos de Tensorflow y keras para facilitar la monitorización y evaluación del entrenamiento.
- ✓ Técnicas avanzadas para mejorar el entrenamiento del modelo.

Cuando hayas terminado esta unidad, deberías saber:

- ✓ Utilizar las funciones de keras que permiten obtener la precisión del modelo y su representación gráfica.
- ✓ Utilizar las funciones de tensorflow y keras que te permiten generar predicciones con nuevos datos para la puesta en producción del modelo.
- ✓ Saber evitar la distorsión en las predicciones debido al sobre entrenamiento del modelo a través de diversas técnicas y decisiones
- ✓ Analizar el desempeño del modelo y aplicar correcciones o modificaciones que contribuyan a hacerlo más robusto y eficiente.



Ministerio de Educación y Formación Profesional
(<https://www.educacionyfp.gob.es/portada.html>) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](http://www.educacionyfp.gob.es/fpadistancia/comunes/aviso-legal.html) (<http://www.educacionyfp.gob.es/fpadistancia/comunes/aviso-legal.html>)

1.- Evaluación de un modelo de machine learning.

Caso práctico



LookStudio (https://www.freepik.es/foto-gratis/retrato-espalda-nina-africana-camisa-blanca-mochila-descansando-amigos-despues-lecciones-fotografia-interior-jovenes-discutiendo-algo-sala-conferencias-grandes-ventanales_22048433.htm#page=3&query=lookstudio&position=48&from_view=search)
(CC BY-SA (<http://creativecommons.org/licenses/?lang=es>)))

Lorena ya tiene entrenado su modelo. A priori, parece tener una tasa de acierto del 94%. "¡Qué bien me ha salido!" Piensa. Pero no está del todo tranquila. Se había reservado una cierta cantidad de datos etiquetados, para hacer pruebas, y decide utilizar la función "evaluate" para comprobar si la métrica de precisión se mantiene. Lamentablemente, con los nuevos datos, que no han intervenido en el entrenamiento, la tasa de acierto baja al 88%. ¿Qué puede estar pasando?

Como has visto en algunos ejemplos utilizados en la unidad anterior, una vez hecho el entrenamiento del modelo, recurrimos a la función "evaluate" para, con nuevos datos etiquetados, obtener el valor del error (Loss) y de la tasa de acierto (Accuracy). Es muy habitual que este valor sea mucho menor que el de la última iteración del entrenamiento.

Cuando esto ocurre, el modelo no está siendo del todo efectivo, pues tiene un buen desempeño con los datos con los que se entrenó y, sin embargo, al introducir nuevos valores de entrada, la salida no es la deseable.

Para poder hacer esta evaluación del modelo, necesitamos reservarnos parte de los datos etiquetados de los que disponemos. Por eso, antes de hacer el entrenamiento, se suele hacer una división del dataset entre datos para el entrenamiento (train) y datos para la evaluación (test). Para ello, tenemos en Keras la función `train_test_split`. Como parámetros, debemos indicar la porción de datos que queremos reservar para el test y el grado de aleatoriedad en el reparto entre ambos conjuntos:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

En este caso, se ha dedicado el 75% de los datos para entrenamiento y el 25% para el test.

Después del entrenamiento, utilizaremos esos datos de test con la función `evaluate`, como en el siguiente ejemplo:

▼ Entrenamiento del modelo

Utilizamos la función `fit` para el entrenamiento de la DNN. Tenemos que pasar como argumentos los valores de entrada, los de salida (las etiquetas) y el número de iteraciones o epochs.

```
▶ model.fit(train_images,train_labels,epochs=5)

[ ] Epoch 1/5
1875/1875 [=====] - 6s 3ms/step - loss: 0.5016 - accuracy: 0.8219
Epoch 2/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.3733 - accuracy: 0.8645
Epoch 3/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.3396 - accuracy: 0.8765
Epoch 4/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.3147 - accuracy: 0.8849
Epoch 5/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.2956 - accuracy: 0.8906
<keras.callbacks.History at 0x7f0956d78a50>
```

▼ Evaluación del modelo

Para evaluar el modelo, recurrimos a la función `evaluate` y guardamos los valores que nos devuelve, el error o loss, y la precisión o accuracy.

```
[ ] test_loss,test_acc = model.evaluate(test_images,test_labels)

print('Test accuracy: ', test_acc)

313/313 [=====] - 1s 2ms/step - loss: 0.3582 - accuracy: 0.8724
Test accuracy: 0.8723999857902527
```

Carmen Bartolomé ([CC BY-SA \(<http://creativecommons.org/licenses/?lang=es>\)](http://creativecommons.org/licenses/?lang=es))

En el ejemplo de la imagen, la precisión del modelo con datos nuevos, es del 87%, frente al 89% que habíamos alcanzado en el entrenamiento.

Autoevaluación

Para saber cómo de bueno es nuestro modelo, nos fijamos en la precisión (accuracy) del entrenamiento y nos quedamos con el valor más alto que se haya alcanzado.

Verdadero Falso

Falso

El valor del accuracy del entrenamiento puede ser engañoso por el overfitting, por lo que es necesario hallar el de los datos de test.

2.- Sobre entrenamiento u "Overfitting".

Caso práctico



[LookStudio \(https://www.freepik.es/search?
format=search&page=3&query=lookstudio\)](https://www.freepik.es/search?format=search&page=3&query=lookstudio) (CC BY-SA
(http://creativecommons.org/licenses/?lang=es))

Lorena ha comprobado que su modelo tiene sobre entrenamiento. La precisión del modelo para los datos de test es mucho menor que la que se había parecido alcanzar en las últimas iteraciones del entrenamiento. Cuando lo comenta con Miguel, éste le sugiere: "¿Por qué no pruebas a utilizar un set de datos de validación durante el entrenamiento? Es una buena forma de saber por dónde van los tiros y hacer ajustes rápidos"

En el aprendizaje automático, el objetivo es lograr que un modelo se adapte al patrón subyacente en un conjunto de datos lo máximo posible, para reproducir dicho patrón frente a un nuevo evento o entrada. Pero la teoría no siempre tiene en cuenta cómo es la realidad, con sus interferencias y anomalías.

A veces, entre los datos que se recogen de un sistema, se cuelan datos anómalos, casos

excepcionales o errores. El algoritmo de aprendizaje automático, a priori, no sabe que esos datos no son como los demás, no sabe que no reflejan el patrón general que queremos captar y reproducir. El entrenamiento del modelo, tratará de adaptarse a estos datos también, y el modelo reflejará un patrón de comportamiento que, en determinados casos, no será del todo acertado, porque las variables de entrada de dichos casos, se acercarán mucho a las que tenían los datos anómalos, y el modelo se confunde.

Este efecto se conoce como sobre entrenamiento u "overfitting". El problema es que es difícil de detectar hasta que se ha concluido el entrenamiento y se prueba con los datos de test. Hay entrenamientos que son largos y costosos, con muchas iteraciones, y nos gustaría poder entender lo que está pasando con una "foto" más completa de dicho proceso. Para ésto, se recomienda utilizar una nueva muestra de datos, que llamaremos set de validación. De nuevo, sepáramos un cierto conjunto de datos del dataset de entrenamiento, y los utilizamos para hacer la evaluación del modelo al final de cada iteración del entrenamiento. Así, podemos ir monitorizando cómo ha ido evolucionando el error de datos no implicados en el ajuste del modelo. Es como hacer evaluación con datos de test, pero en cada "epoch".

Aquí tienes un notebook de ejemplo (accede al [cuaderno original](#) (https://colab.research.google.com/drive/14APDqlCY_gQ3t7B-9H7NIj_C2pbDuznw?usp=sharing) si quieras trabajar sobre tu propia copia del mismo), en el que reservaremos datos para validación y veremos, tras el entrenamiento, que, a partir de la iteración 4 o 5, no merece la pena continuar, porque solo conseguiremos aumentar el error y bajar la tasa de acierto. Estamos ante un caso claro de overfitting. Una vez hecho este análisis, se procede a reiniciar el modelo con nuevos coeficientes aleatorios, y se vuelve a entrenar, esta vez, solo con 5 epochs).

Ejemplo de clasificación de valoraciones de películas con deep learning

Trabajamos con el dataset IMDB, que ya viene preprocesado y el texto se ha codificado como una secuencia de números enteros.

Viene precargado en la librería Keras: <https://keras.io/api/datasets/imdb/> (<https://keras.io/api/datasets/imdb/>)

In [1]:

```
from keras.datasets import imdb

(train_data, train_labels), (test_data,test_labels) = imdb.load_data(num_words=10000)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [=====] - 0s 0us/step
17473536/17464789 [=====] - 0s 0us/step
```

Vemos que train_data y test_data son listas de índices correspondientes a palabras presentes en una review de una película

In [4]:

```
print("Los 10 primeros tokens del comentario con índice 0 son: ")
print(train_data[0][:10])
```

Los 10 primeros tokens del comentario con índice 0 son:
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]

Por otro lado, train_labels y test_labels son listas de 0 y 1, correspondiendo el 0 a una valoración negativa y el 1 a una valoración positiva

In [5]:

```
train_labels[0]
```

Out[5]:

1

El módulo incluye la función get_word_index, que es la que codifica el texto a índices. El criterio de codificación asigna a cada palabra el número entero correspondiente al orden que ocupa dicha palabra en cuanto a frecuencia de aparición en el dataset. Para recuperar el texto literal, invertimos índices y valores tras aplicar la función.

In [6]:

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [=====] - 0s 0us/step
1654784/1641221 [=====] - 0s 0us/step

In [7]:

```
decoded_review
```

Out[7]:

"? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would recommend it to everyone to watch and the fishy fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left out of the ? list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

Preparamos los datos de entrada

No podemos introducir en la red neuronal listas de diferente longitud. Las redes neuronales solo aceptan tensores, y tenemos que "vectorizar" las listas de los comentarios haciendo un "one_hot encoding"

In [8]:

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [9]:

```
len(x_train)
```

Out[9]:

25000

In [10]:

```
x_train[0].shape
```

Out[10]:

(10000,)

Convertimos en array también las etiquetas y

In [11]:

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Construimos el modelo

Creamos un modelo Sequential con 3 capas neuronales, dos de 16 neuronas y la de salida con 1 neurona, pues es un problema de clasificación binario

In [12]:

```
from keras import models, layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Fijamos los parámetros del entrenamiento

Podemos utilizar el módulo optimizers para fijar parámetros internos del optimizador.

In [13]:

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
```

In []:

```
# from keras import optimizers
# model.compile(optimizer=optimizers.RMSprop(Lr=0.001),
#                 loss='binary_crossentropy',
#                 metrics=['accuracy'])
```

Reservamos datos para la validación.

In [14]:

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Entrenamos el modelo

In [15]:

```
training=model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val,y_val))
```

Epoch 1/20
30/30 [=====] - 2s 46ms/step - loss: 0.5102 - acc: 0.7912 - val_loss: 0.4192 - val_acc: 0.8361
Epoch 2/20
30/30 [=====] - 1s 32ms/step - loss: 0.3111 - acc: 0.9005 - val_loss: 0.3206 - val_acc: 0.8777
Epoch 3/20
30/30 [=====] - 1s 31ms/step - loss: 0.2278 - acc: 0.9277 - val_loss: 0.2929 - val_acc: 0.8834
Epoch 4/20
30/30 [=====] - 1s 31ms/step - loss: 0.1807 - acc: 0.9415 - val_loss: 0.2725 - val_acc: 0.8913
Epoch 5/20
30/30 [=====] - 1s 31ms/step - loss: 0.1489 - acc: 0.9532 - val_loss: 0.2877 - val_acc: 0.8867
Epoch 6/20
30/30 [=====] - 1s 32ms/step - loss: 0.1208 - acc: 0.9621 - val_loss: 0.3016 - val_acc: 0.8802
Epoch 7/20
30/30 [=====] - 1s 31ms/step - loss: 0.0988 - acc: 0.9713 - val_loss: 0.3122 - val_acc: 0.8837
Epoch 8/20
30/30 [=====] - 1s 33ms/step - loss: 0.0811 - acc: 0.9776 - val_loss: 0.3243 - val_acc: 0.8808
Epoch 9/20
30/30 [=====] - 1s 32ms/step - loss: 0.0660 - acc: 0.9823 - val_loss: 0.3517 - val_acc: 0.8766
Epoch 10/20
30/30 [=====] - 1s 31ms/step - loss: 0.0547 - acc: 0.9858 - val_loss: 0.3726 - val_acc: 0.8763
Epoch 11/20
30/30 [=====] - 1s 46ms/step - loss: 0.0436 - acc: 0.9896 - val_loss: 0.4059 - val_acc: 0.8718
Epoch 12/20
30/30 [=====] - 1s 32ms/step - loss: 0.0321 - acc: 0.9941 - val_loss: 0.4264 - val_acc: 0.8741
Epoch 13/20
30/30 [=====] - 1s 32ms/step - loss: 0.0277 - acc: 0.9943 - val_loss: 0.4599 - val_acc: 0.8728
Epoch 14/20
30/30 [=====] - 1s 32ms/step - loss: 0.0197 - acc: 0.9975 - val_loss: 0.4867 - val_acc: 0.8725
Epoch 15/20
30/30 [=====] - 1s 32ms/step - loss: 0.0152 - acc: 0.9979 - val_loss: 0.5210 - val_acc: 0.8720
Epoch 16/20
30/30 [=====] - 1s 32ms/step - loss: 0.0125 - acc: 0.9983 - val_loss: 0.5502 - val_acc: 0.8721
Epoch 17/20
30/30 [=====] - 1s 31ms/step - loss: 0.0074 - acc: 0.9997 - val_loss: 0.6256 - val_acc: 0.8593
Epoch 18/20
30/30 [=====] - 1s 33ms/step - loss: 0.0082 - acc: 0.9987 - val_loss: 0.6313 - val_acc: 0.8701
Epoch 19/20
30/30 [=====] - 1s 33ms/step - loss: 0.0036 - acc: 0.9999 - val_loss: 0.6683 - val_acc: 0.8676
Epoch 20/20
30/30 [=====] - 1s 32ms/step - loss: 0.0071 - acc: 0.9985 - val_loss: 0.6954 - val_acc: 0.8671

Representamos la evolución del error y la precisión en cada epoch

In [16]:

```
history_dict = training.history  
history_dict.keys()
```

Out[16]:

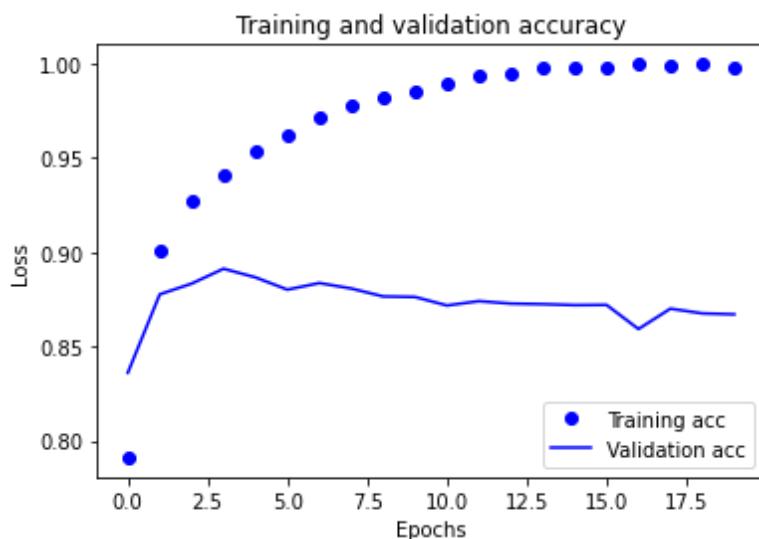
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

In [17]:

```
import matplotlib.pyplot as plt  
loss_values = history_dict['loss']  
val_loss_values = history_dict['val_loss']  
#epochs = range(1, len(history_dict['acc']) +1)  
epochs = training.epoch  
plt.plot(epochs, loss_values, 'bo', label = 'Training loss')  
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')  
plt. title('Training and validation loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()
```

In [18]:

```
plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Carmen Bartolomé - Elaboración propia (Dominio público)

En el momento en que la precisión del modelo con los datos de entrenamiento aumenta pero la de los datos de validación no, tenemos indicios de que se está dando lo que se denomina "overfitting". Volvemos a entrenar un nuevo modelo pero, esta vez, solo con 5 epochs

In [19]:

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

training=model.fit(partial_x_train,
                    partial_y_train,
                    epochs=5,
                    batch_size=512,
                    validation_data=(x_val,y_val))
```

Epoch 1/5

```
30/30 [=====] - 2s 45ms/step - loss: 0.5109 - acc: 0.7905 - val_loss: 0.3790 - val_acc: 0.8725
```

Epoch 2/5

```
30/30 [=====] - 1s 32ms/step - loss: 0.3043 - acc: 0.9006 - val_loss: 0.3135 - val_acc: 0.8777
```

Epoch 3/5

```
30/30 [=====] - 1s 36ms/step - loss: 0.2218 - acc: 0.9281 - val_loss: 0.2768 - val_acc: 0.8939
```

Epoch 4/5

```
30/30 [=====] - 1s 31ms/step - loss: 0.1755 - acc: 0.9429 - val_loss: 0.3125 - val_acc: 0.8751
```

Epoch 5/5

```
30/30 [=====] - 1s 34ms/step - loss: 0.1448 - acc: 0.9533 - val_loss: 0.2965 - val_acc: 0.8840
```

Probamos a hacer una predicción

In [20]:

```
prediction = model.predict(x_test)
```

In [21]:

```
prediction[0]
```

Out[21]:

```
array([0.30326813], dtype=float32)
```

In [22]:

```
y_test[0]
```

Out[22]:

```
0.0
```

Para saber más

Siquieres profundizar en esta parte, puedes revisar la [documentación sobre el método evaluate](https://keras.io/api/models/model_training_apis/#evaluate-method) (https://keras.io/api/models/model_training_apis/#evaluate-method) en la documentación de Keras. También puede venirte bien echar un vistazo a la [guía sobre entrenamiento y evaluación](https://www.tensorflow.org/guide/keras/train_and_evaluate) (https://www.tensorflow.org/guide/keras/train_and_evaluate) que tienes en la documentación de Tensorflow.

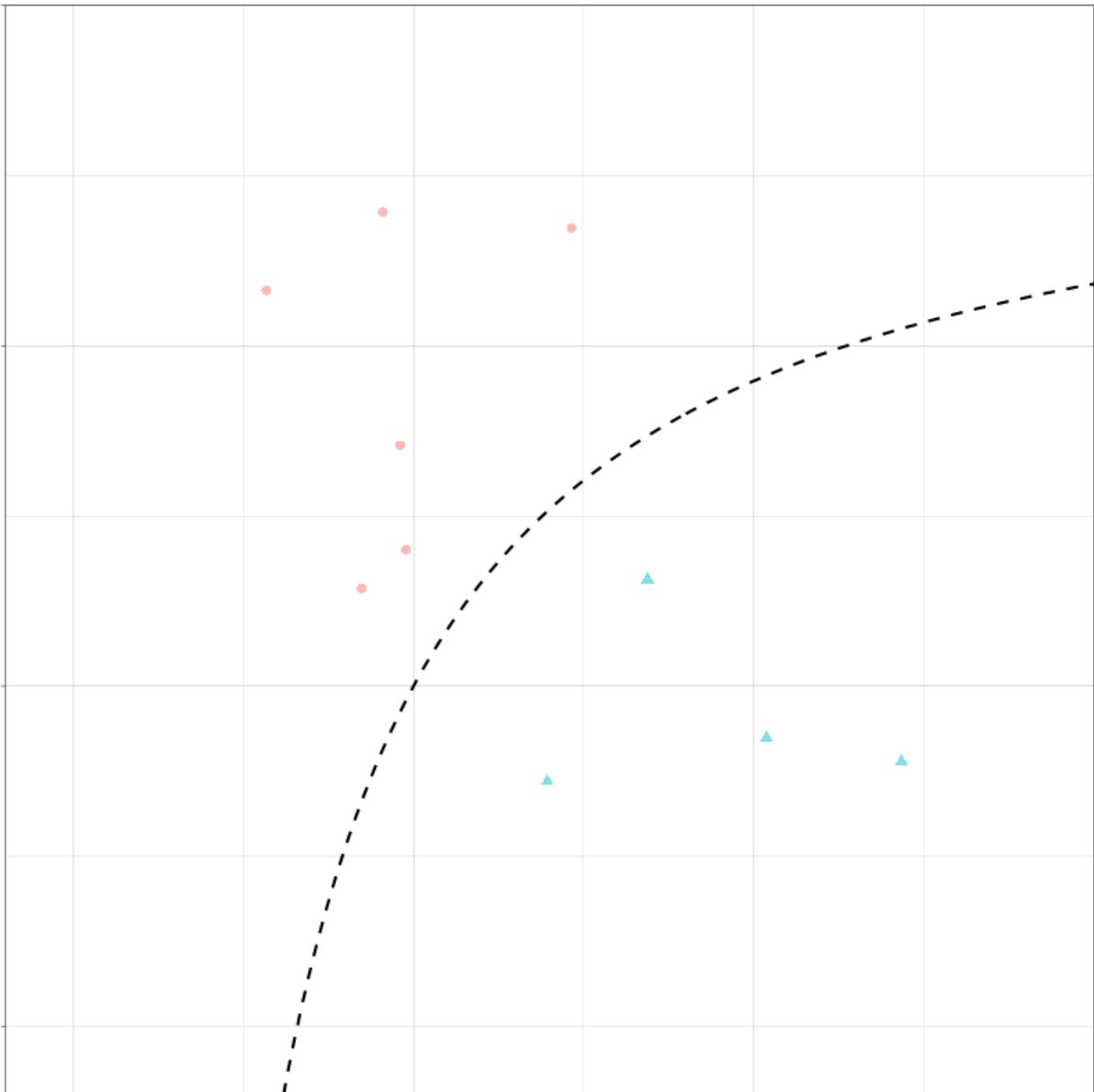
2.1.- Evitar el sobre entrenamiento.

En el caso anterior, hemos limitado el número de iteraciones para ahorrar en recursos y tiempo, pero esta no es una forma muy efectiva de evitar el overfitting. El modelo no generalizará bien y la precisión queda muy limitada. Para aumentar el desempeño del modelo y evitar este sobre entrenamiento, hay varias soluciones, que forman parte del campo de técnicas denominado "Regularización". Comentamos las más utilizadas:

- ✓ Contar con más datos: cuantos más casos pasen por el entrenamiento, más generalista conseguiremos que sea el modelo, y las anomalías tendrán poco impacto.
- ✓ Reducir el tamaño de la red: el sobre entrenamiento parece estar relacionado con la cantidad de información que es capaz de "almacenar" el modelo a través de los pesos. Reduciendo el número de parámetros del modelo significa reducir capas o neuronas. Como criterio general, la idea sería partir de un modelo lo más pequeño posible, e ir ajustando los parámetros en base a la evaluación que vamos haciendo de éste.
- ✓ Regularización de los pesos: esta técnica consiste en limitar el valor que pueden alcanzar los pesos, como una forma de limitar la capacidad de memoria del modelo.
- ✓ Descarte (dropout): si se hacen nulos, de forma arbitraria, ciertos pesos por capa, durante el entrenamiento, mejora bastante la capacidad de generalización del modelo.

Los dos últimos puntos, los analizamos con más detalle en las siguientes secciones.

El término "overfitting" también se traduce como "sobre ajuste", en su forma más literal. Al fin y al cabo, consiste en tratar de ajustar la superficie o hiperplano solución a la distribución de los datos, aunque éstos no siempre estén donde deberían estar.



Ryan Holbrook (<https://github.com/ryanholbrook/decision-boundaries-animations>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Autoevaluación

¿Cuál de estas características de un problema de aprendizaje automático contribuye a evitar el "overfitting"?

- Tener una gran cantidad de datos
- Tener un hardware muy potente
- Que los datos sean números enteros

Opción correcta

La capacidad de computación no influye en el sobre entrenamiento

A priori, el tipo de los datos no influye en el sobre entrenamiento

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

2.2.- Decaimiento de los pesos.

Dentro de las técnicas de Regularización, el decaimiento de los pesos es una forma bastante intuitiva de tratar el problema. Además, se aplica mediante parámetros muy concretos en la arquitectura de la red, de forma sencilla.

Como hemos comentado antes, el objetivo es simplificar o "aligerar" el modelo, y si limitamos el valor que podrían adquirir los pesos a valores bajos, la distribución de éstos es más regular, menos "entrópica", y el modelo está más acotado. Esto se consigue añadiendo a la función de coste un término que penaliza a los pesos si éstos adquieren valores elevados. Hay dos formas de hacer esta penalización, conocidas como L1 y L2.

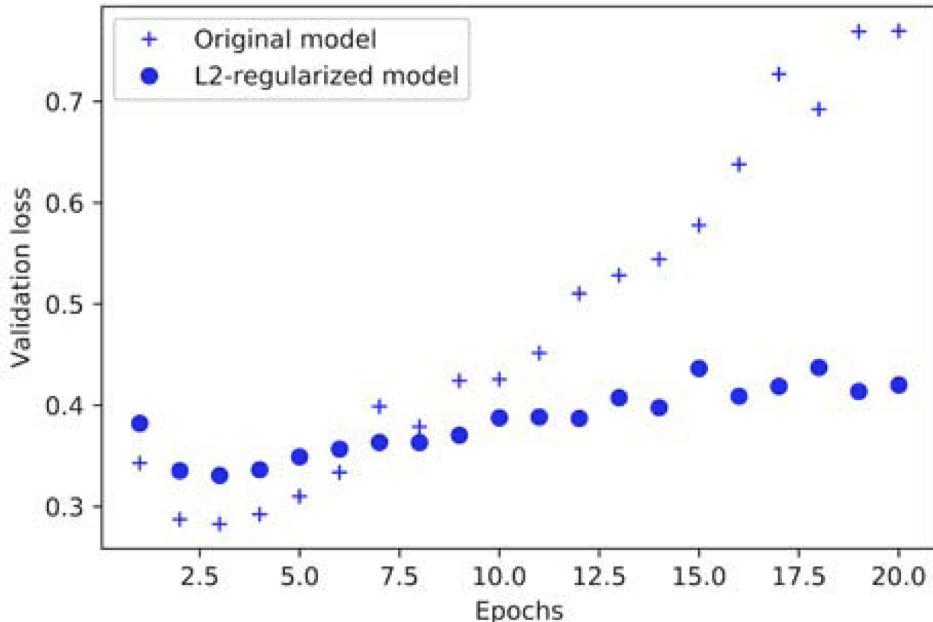
- ✓ Regularización L1: en esta técnica, la penalización es proporcional al valor absoluto de los pesos.
- ✓ Regularización L2 o decaimiento de los pesos: en este caso, la penalización es proporcional al cuadrado del valor de los pesos.

En Keras, si queremos aplicar una o las dos técnicas de regularización, lo pasaremos como argumento en la capa, para el parámetro *kernel_regularizer*.

En el ejemplo de la clasificación de las valoraciones de películas que hemos visto en esta unidad, podríamos aplicar la técnica L2 de la siguiente manera:

```
from keras import regularizers
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_shape=(2000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

De esta forma, al coste o error, se le añadirá, por cada peso de la red, el producto del coeficiente 0,001 por el cuadrado del valor del peso. En consecuencia, el error durante el entrenamiento será mucho mayor que en la validación o test.



Carmen Bartolomé ([CC BY-SA \(<http://creativecommons.org/licenses/?lang=es>\)](http://creativecommons.org/licenses/?lang=es))

2.3.- Descarte (Dropout).

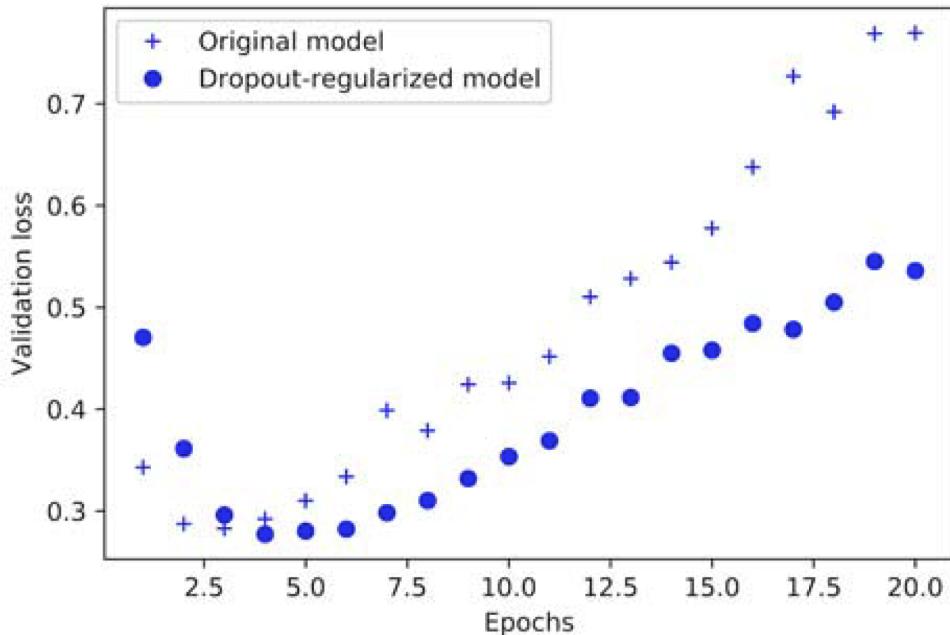
La técnica del "dropout" es la más efectiva y utilizada desde que Geoffry Hinton la sacó a la luz junto al resto de su trabajo en el desarrollo de redes neuronales profundas. Como anécdota curiosa, Hinton cuenta que esta técnica se le ocurrió al darse cuenta de que, en su banco, los empleados que estaban en caja, cambiaban con mucha frecuencia. Pensó que esa dinámica, realmente, les ponía difícil defraudar o engañar al banco, pues requería de que se pusiesen todos de acuerdo para que no les pillaran. ¿Y si se pudiera boicotear de alguna manera a los nodos para que la red no se acomode demasiado durante el entrenamiento? ¿Romper patrones espúreos ayudaría a hacer más robustos los patrones significativos?

Básicamente, esta técnica consiste en deshacerse de algunas salidas de cada capa durante el entrenamiento. Esto se hace dando valor cero a ciertos nodos, en su valor de salida, al azar. El dropout rate es la fracción de variables que se han anulado, y suele estar entre 0,2 y 0,5. En Keras, se aplica introduciendo una capa de tipo "Dropout".

Por ejemplo, para aplicar el dropout en el ejemplo de las valoraciones de películas, el modelo quedaría así:

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

Observando el efecto de haber aplicado las capas de dropout, vemos que el error de los datos de validación mejora significativamente respecto del que teníamos inicialmente.



Carmen Bartolomé ([CC BY-SA \(<http://creativecommons.org/licenses/?lang=es>\)](http://creativecommons.org/licenses/?lang=es))

Autoevaluación

El dropout es un parámetro que metemos dentro de la función Dense, después del parámetro "activation".
¿Verdadero o Falso?

Verdadero Falso

Falso

Dropout es una tipo de capa, y la añadimos con la clase del mismo nombre, después de una capa de tipo Dense.

3.- Tensorboard.

Caso práctico



[LookStudio](https://www.freepik.es/search?format=search&page=3&query=lookstudio) (<https://www.freepik.es/search?format=search&page=3&query=lookstudio>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Uno de los consejos que le da Miguel a Lorena es que utilice una herramienta que Google ha facilitado a la hora de trabajar con Tensorflow para monitorizar el entrenamiento de modelos de aprendizaje automático profundo: TensorBoard.

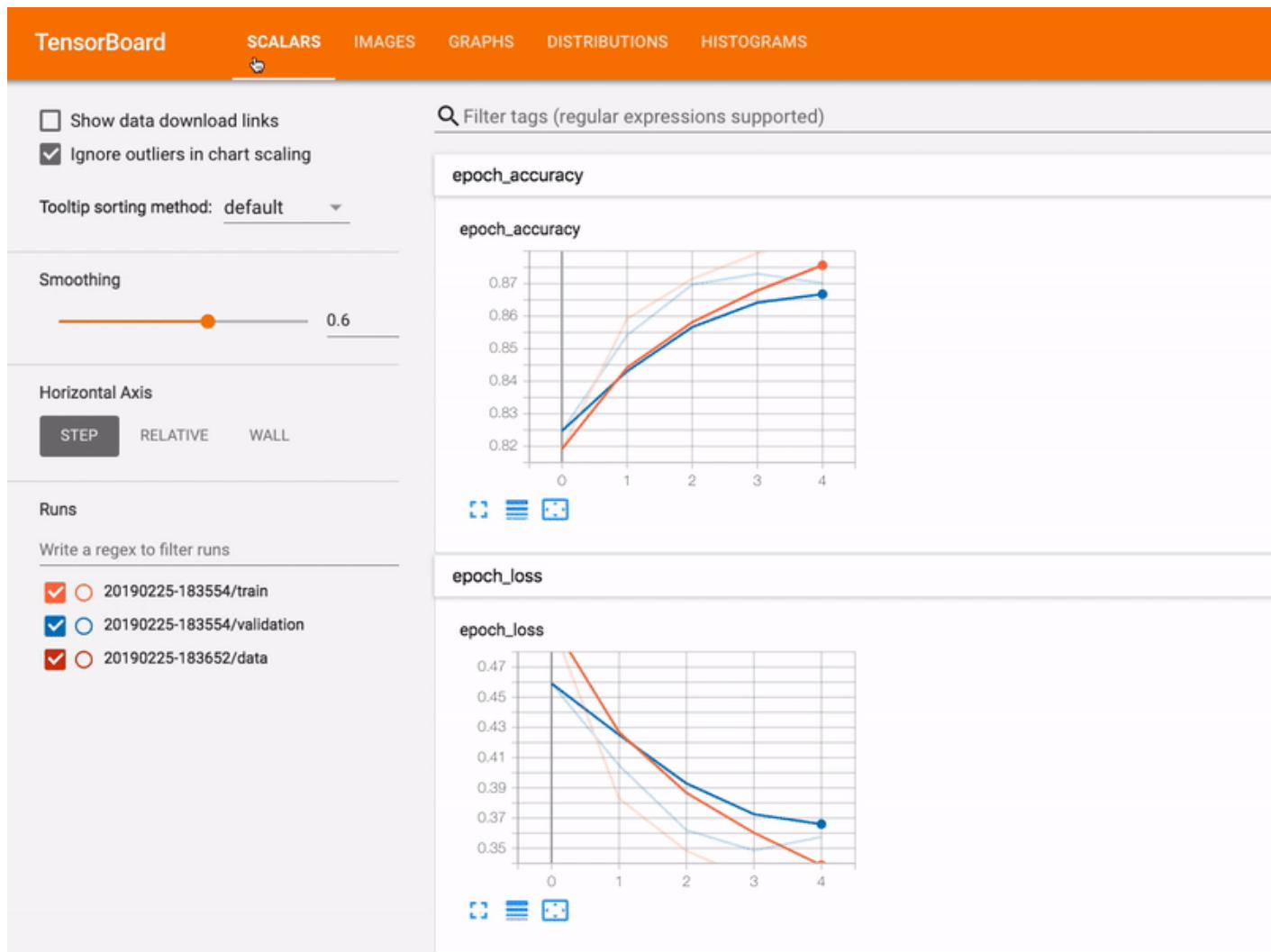
"¿Es como el listado de métricas que se muestra en consola con las epochs del entrenamiento?" le pregunta Lorena.

"No, no, que va. Es una interfaz de tipo gráfico que va dibujando literalmente esas y otras métricas para que luego puedas tomar decisiones respecto al entrenamiento y sus parámetros" le responde Miguel. "Prueba, haz que se ejecute la llamada a Tensorboard en el código y me cuentas si te es de ayuda"

De nuevo, Lorena se va a la documentación y consulta cómo incorporar Tensorboard a su proyecto.

Dentro de su estrategia de mejora y evolución de Tensorflow, Google, con ayuda de la comunidad, lanzó un kit de herramientas de visualización con una interfaz muy sencilla, denominado TensorBoard. Las principales funcionalidades disponibles son:

- ✓ Monitorizar de forma visual las métricas de "Loss" y la precisión.
- ✓ Contar con una representación de tipo grafo del modelo.
- ✓ Gráficas de histogramas de coeficientes del modelo.
- ✓ Hacer seguimiento del histórico de proyecto, creando perfiles.



[tensorflow.org/tensorboard](https://www.tensorflow.org/tensorboard) (Dominio público)

A continuación, tienes un notebook de ejemplo con el código que sería necesario para inicializar Tensorboard y poder utilizarlo de forma interactiva dentro del mismo cuaderno. Te recomendamos que acudas al [notebook original](https://colab.research.google.com/drive/1YPSZfcaqZl-CXpw2WVCveJIMANKk4JGg?usp=sharing) (<https://colab.research.google.com/drive/1YPSZfcaqZl-CXpw2WVCveJIMANKk4JGg?usp=sharing>) para poder copiarlo, editarlo y trabajar sobre él.

Visualización del entrenamiento con Tensorboard

En colab, empezamos cargando la extensión TensorBoard para cuadernos.

In [3]:

```
%load_ext tensorboard
```

The tensorboard extension is already loaded. To reload it, use:
`%reload_ext tensorboard`

Importamos tensorflow, datetime, y os

In [2]:

```
import tensorflow as tf
import datetime, os
```

Ejemplo Tensorboard con modelo entrenado con FashionMNIST

Descargamos el dataset [FashionMNIST](https://github.com/zalandoresearch/fashion-mnist) (<https://github.com/zalandoresearch/fashion-mnist>) y adimensionalizamos

In [4]:

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=====] - 0s 0us/step
40960/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [=====] - 0s 0us/step
26435584/26421880 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [=====]
===== - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [=====] - 0s 0us/step
4431872/4422102 [=====] - 0s 0us/step
```

Creamos una red neuronal profunda, incluyendo una capa dropout

In [5]:

```
def create_model():
    return tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

Al configurar el entrenamiento, añadimos el callback a Tensorboard

In [6]:

```
def train_model():

    model = create_model()
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
    tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

    model.fit(x=x_train,
              y=y_train,
              epochs=5,
              validation_data=(x_test, y_test),
              callbacks=[tensorboard_callback])

train_model()
```

Epoch 1/5

1875/1875 [=====] - 11s 5ms/step - loss: 0.4968 - accuracy: 0.8238 - val_loss: 0.4520 - val_accuracy: 0.8365

Epoch 2/5

1875/1875 [=====] - 9s 5ms/step - loss: 0.3830 - accuracy: 0.8594 - val_loss: 0.3815 - val_accuracy: 0.8630

Epoch 3/5

1875/1875 [=====] - 9s 5ms/step - loss: 0.3507 - accuracy: 0.8703 - val_loss: 0.3563 - val_accuracy: 0.8688

Epoch 4/5

1875/1875 [=====] - 10s 5ms/step - loss: 0.3293 - accuracy: 0.8782 - val_loss: 0.3663 - val_accuracy: 0.8686

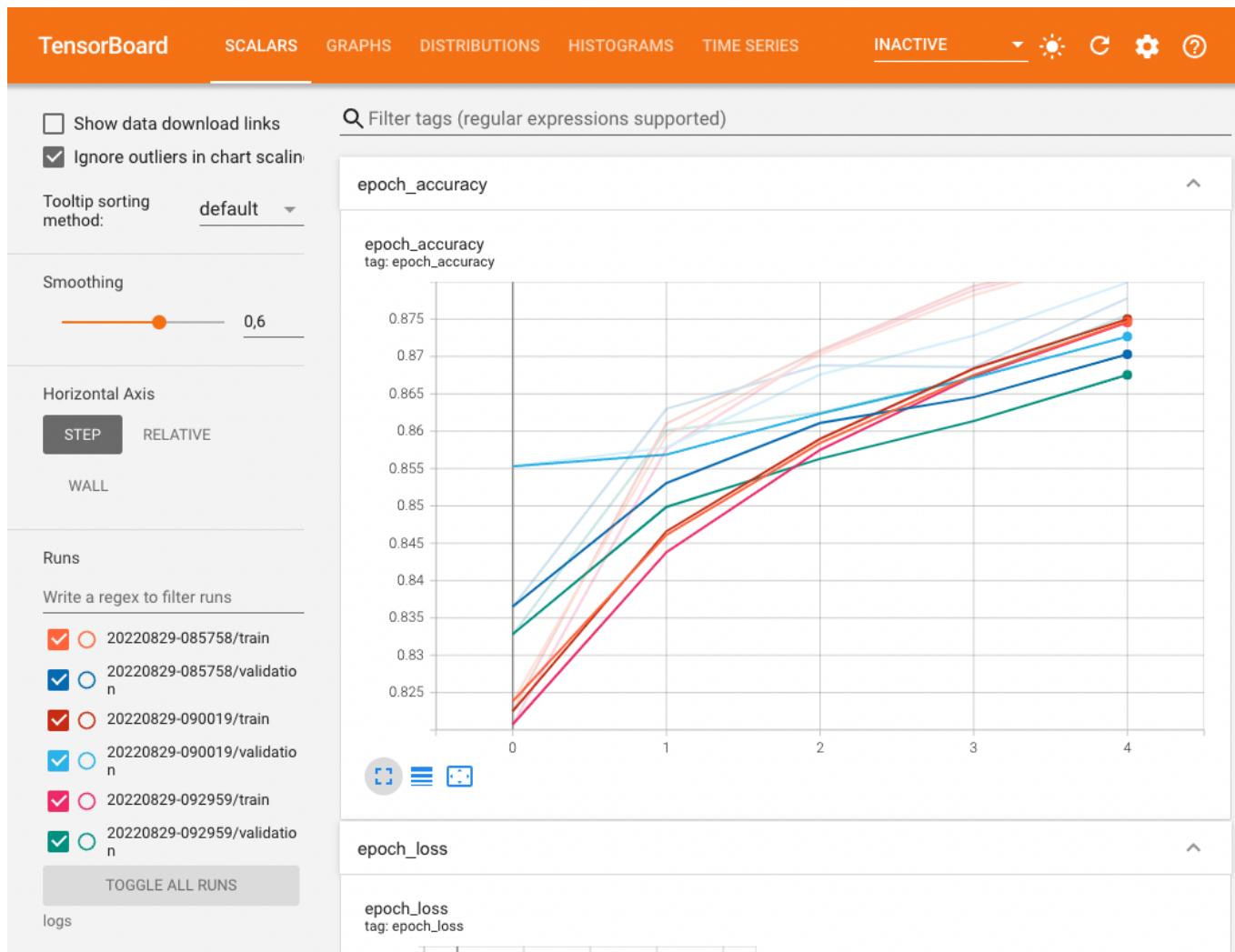
Epoch 5/5

1875/1875 [=====] - 10s 5ms/step - loss: 0.3125 - accuracy: 0.8840 - val_loss: 0.3426 - val_accuracy: 0.8778

Iniciamos la visualización con TensorBoard utilizando su función [mágica](#) (<https://ipython.readthedocs.io/en/stable/interactive/magics.html>):

In [7]:

```
%tensorboard --logdir logs
```



Carmen Bartolomé - Elaboración propia (Dominio público)

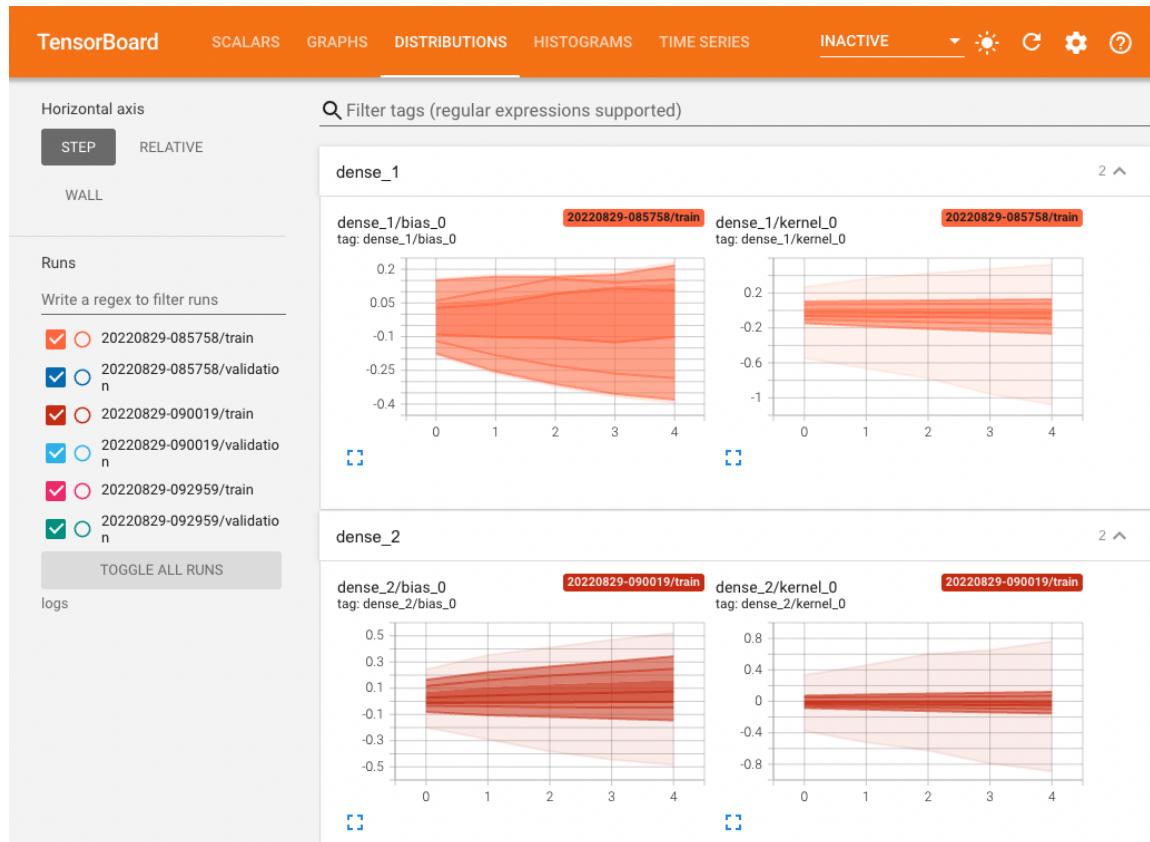
La interfaz de TensorBoard nos muestra representaciones interactivas de escalares y grafos.

Si iniciamos TensorBoard antes del entrenamiento, lo podremos monitorizar mientras éste va transcurriendo

In [8]:

```
%tensorboard --logdir logs
```

Reusing TensorBoard on port 6006 (pid 172), started 0:00:55 ago. (Use '!kill 1 172' to kill it.)



Carmen Bartolomé - Elaboración propia (Dominio público)

In [9]:

```
train_model()
```

```
Epoch 1/5
1875/1875 [=====] - 10s 5ms/step - loss: 0.4967 -
accuracy: 0.8225 - val_loss: 0.4118 - val_accuracy: 0.8553
Epoch 2/5
1875/1875 [=====] - 10s 6ms/step - loss: 0.3836 -
accuracy: 0.8610 - val_loss: 0.3911 - val_accuracy: 0.8578
Epoch 3/5
1875/1875 [=====] - 11s 6ms/step - loss: 0.3502 -
accuracy: 0.8708 - val_loss: 0.3587 - val_accuracy: 0.8676
Epoch 4/5
1875/1875 [=====] - 10s 5ms/step - loss: 0.3293 -
accuracy: 0.8795 - val_loss: 0.3534 - val_accuracy: 0.8728
Epoch 5/5
1875/1875 [=====] - 11s 6ms/step - loss: 0.3143 -
accuracy: 0.8837 - val_loss: 0.3404 - val_accuracy: 0.8799
```

Con la API (Application Programming Interfaces) especial para cuadernos
tensorboard.notebook algunas funcionalidades van mejor

In [10]:

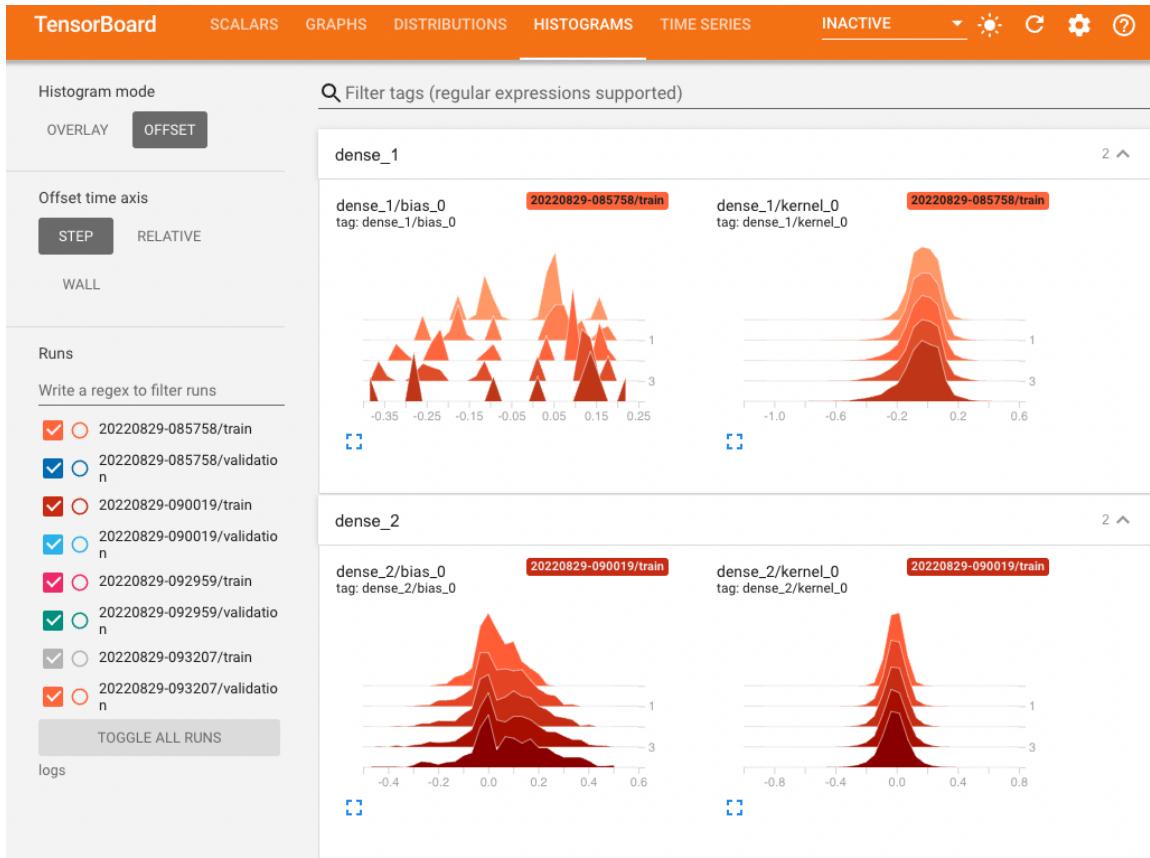
```
from tensorboard import notebook
notebook.list() # View open TensorBoard instances
```

```
Known TensorBoard instances:
- port 6006: logdir logs (started 0:02:12 ago; pid 172)
```

In [11]:

```
# Control TensorBoard display. If no port is provided,  
# the most recently launched TensorBoard is used  
notebook.display(port=6006, height=1000)
```

Selecting TensorBoard with logdir logs (started 0:02:28 ago; port 6006, pid 172).



Carmen Bartolomé - Elaboración propia (Dominio público)

Copyright 2019 The TensorFlow Authors.

In []:

```
#Licensed under the Apache License, Version 2.0 (the "License")  
  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
# https://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.
```

4.- Técnicas avanzadas de mejora en aprendizaje automático.

Caso práctico



[LookStudio](https://www.freepik.es/search?format=search&page=3&query=lookstudio) (<https://www.freepik.es/search?format=search&page=3&query=lookstudio>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Lorena ha conseguido avanzar mucho en entender bien el comportamiento del modelo en el entrenamiento, y ahora es capaz de detectar el grado de precisión real que se está alcanzando. La verdad es que si echa la vista atrás y recuerda el punto de partida de su modelo de red neuronal, se da cuenta de todo lo que ha ido aprendiendo y añadiendo al proyecto.

Repasando su lista de lectura de medios y blogs que tratan el tema de la programación de inteligencia artificial, da con un artículo en el que se comentan dos ideas muy interesantes para rematar el estudio de la evaluación y corrección de un modelo: la normalización por lotes y la optimización de hiperparámetros.

En su afán por controlar todo lo que afecta o puede beneficiar a su modelo, Lorena lee el artículo e investiga un poco más sobre cada uno de estos temas.

En general, tendrás la sensación de que el trabajo con modelos de machine learning es un poco del estilo "prueba-error" y que el MLOps debe trabajar de forma intuitiva, construyendo a ciegas y ajustando después a base de pruebas. Esto es así en cierta manera, pero hay una técnica y una metodología que suelen practicar los expertos en este área que pueden conducirte antes a la excelencia.

Patrones de diseño avanzados: "Batch normalization"

Los patrones de diseño son una de las técnicas que más han ayudado a mejorar el mundo del software, pues habilitan, desde el propio diseño de la arquitectura del código, un funcionamiento robusto.

En el ámbito del aprendizaje automático, entre varios patrones de diseño, está el de la normalización por lotes o "Batch Normalization". En el siguiente punto lo veremos en más profundidad.

Optimización de hiper-parámetros

Los modelos avanzados que nos ofrecen los mejores científicos de datos, se han creado siguiendo una metodología cuidadosa y ordenada. Y ésta técnica de la optimización de hiperparámetros consiste, precisamente, en ir explorando la combinación de parámetros del modelos que nos arroja mejores resultados, independientemente de otros factores.

4.1.- Normalización por lotes.

La normalización es una técnica que utilizamos cuando queremos hacer que distintas muestras puedan ser similares, comparables, para apreciar mejor las diferencias que merece la pena analizar. En base a esto, existe una técnica que contribuye a una mejora de la velocidad y efectividad del entrenamiento de un modelos de deep learning, especialmente para modelos complejos, con capas convolucionales.

La normalización se suele aplicar centrando los datos en 0, restando la media a los datos y dividiéndolos por la desviación estándar. Hemos visto algunos ejemplos en los que esto se hace antes de que los datos entrenen en el modelo para el entrenamiento.

Pero la normalización por lotes propone que se aplique esta normalización como una capa de transformación de los datos que salen de una capa y van a entrar en otra, dentro del modelo, de la red multicapa. y para poder aplicarla, existe la capa BatchNormalization en Keras. Su principal efecto es que beneficia al entrenamiento en redes con muchas capas, ayudando a propagar el gradiente.

Se efectividad ya ha sido comprobada ampliamente, y por eso va incluida en las arquitecturas de convnets ya conformadas que ofrece Keras: ResNet50, InceptionV3 y Xception.

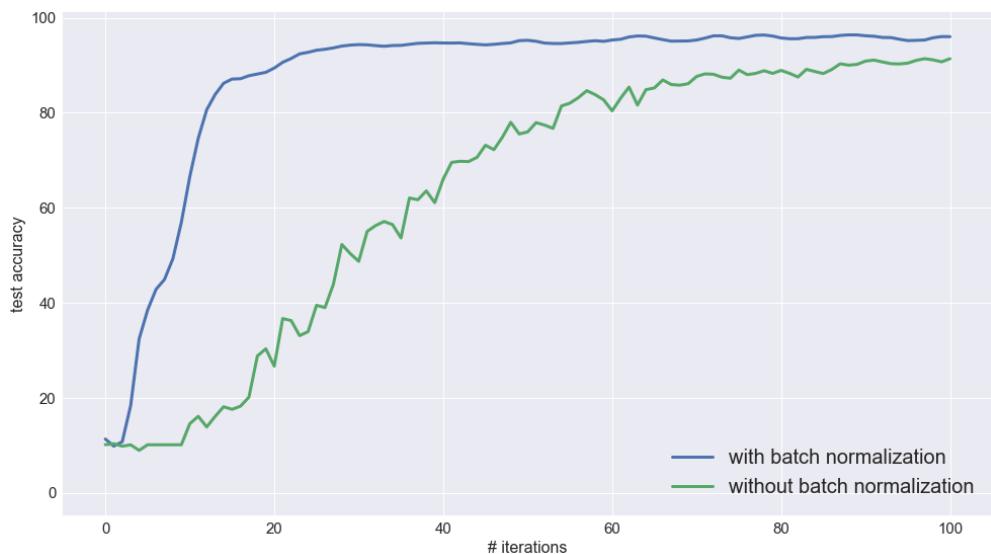
Se suele poner después de una capa de tipo Dense:

```
dense_model.add(layers.Dense(32, activation='relu'))  
dense_model.add(layers.BatchNormalization())
```

Y, sobre todo, después de una capa de tipo convolucional (este tipo de capa lo veremos en la siguiente unidad):

```
conv_model.add(layers.Conv2D(32, 3, activation='relu'))  
conv_model.add(layers.BatchNormalization())
```

En esta imagen, extraída del [artículo de Chris Rawles sobre normalización por lotes](https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73) (<https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73>), se puede apreciar el efecto de esta técnica sobre la rapidez y fluidez con la que converge el modelo:



[Chris Rawles](https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73) (<https://towardsdatascience.com/how-to-use-batch-normalization-with-tensorflow-and-tf-keras-to-train-deep-neural-networks-faster-60ba4d054b73>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Autoevaluación

¿Dónde suele ir la capa de BatchNormalization dentro del modelo de aprendizaje automático?

- Antes de la capa Flatten
- Detrás de una capa convolucional o neuronal.
- Es la capa de entrada del modelo.
- No se posiciona en ninguna parte, es una función de optimización en sí misma.

Su efecto de normalización debe aplicarse a la salida de capas neuronales o convolucionales

Opción correcta

Su efecto de normalización debe aplicarse a la salida de capas neuronales o convolucionales

La clase BatchNormalization crea una capa en la que se aplica la normalización a las variables de salida de la capa anterior antes de que entren en ella.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

4.2.- Optimización de hiperparámetros.

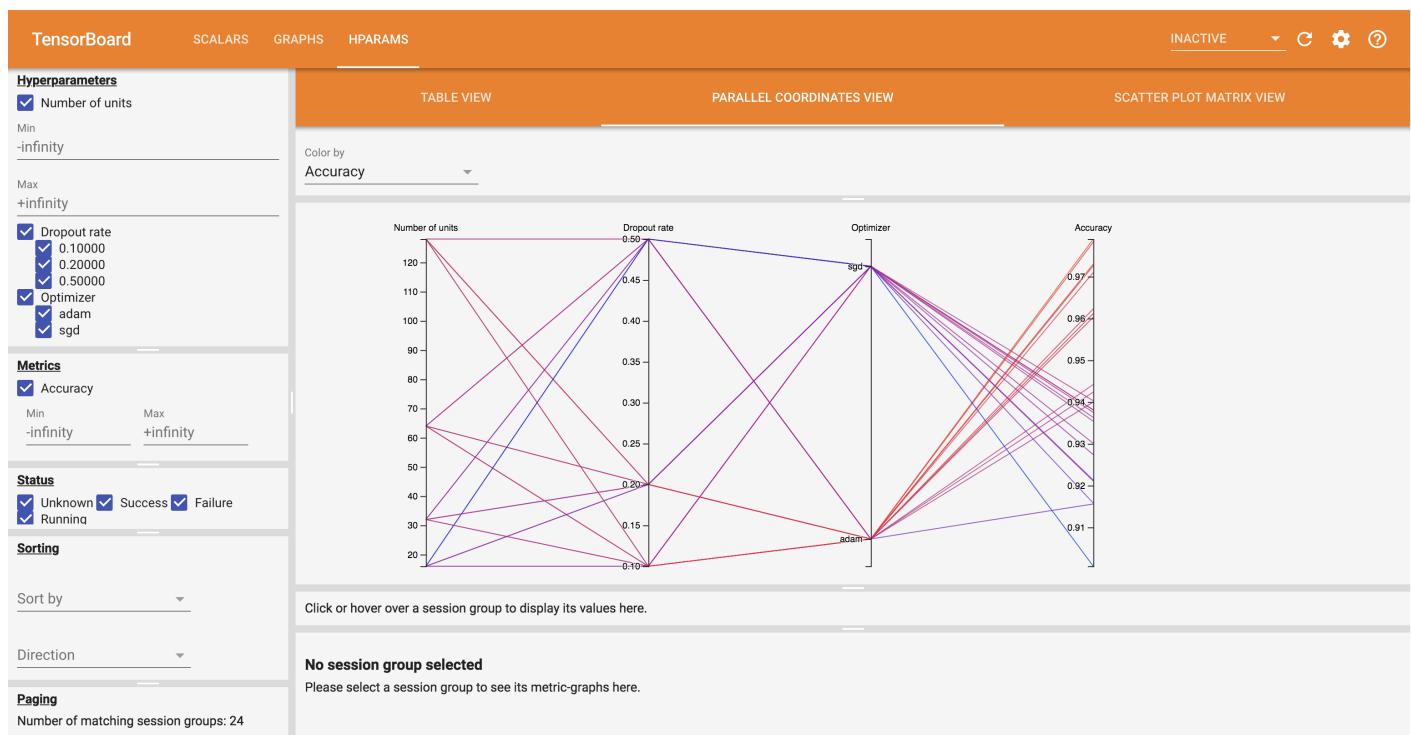
En el diseño de un modelo de deep learning, nos toca ir tomando decisiones sobre sus parámetros que nos parecen arbitrarias en su gran mayoría. En general, la mayoría de científicos de datos saben que hay una cierta dosis de intuición y creatividad, sumada a una cierta dosis de suerte. Como en cualquier proceso de barrido sobre diferentes opciones, el ser metódicos e ir haciendo variaciones según un patrón de variación ordenado, es lento y pesado, pero suele ayudar. Pero, en todo caso, no cabe duda de que la búsqueda de la mejor arquitectura para un modelo es un proceso eminentemente empírico.

El proceso suele seguir este esquema:

- 1.- Se elige un conjunto de hiperparámetros en base a la intuición
- 2.- Se programa el modelo
- 3.- Se entrena con un dataset de entrenamiento y validación, observando la métrica de validación.
- 4.- Se elige un parámetro que variar y se cambia solo ese
- 5.- Se repite el entrenamiento y validación
- 6.- Tras varias variaciones en un parámetro que hayan ido mejorando la métrica de validación, es conveniente utilizar también datos de test y evaluar el estado tras la evolución de dicho parámetro antes de empezar con pruebas de variación de otro parámetro.

Si vas guardando un histórico de las métricas de las variaciones de los distintos hiperparámetros, después, puedes aplicar, a su vez, alguna técnica de selección, como algoritmos genéticos u optimización basada en Bayes.

Una herramienta disponible en Tensorboard para esta evaluación de los hiperparámetros, es Hparams, que presenta de forma gráfica las relaciones entre estos parámetros en las distintas pruebas realizadas. Al observar el eje de precisión, se puede ver qué conjuntos de parámetros desembocan en los mejores valores de precisión. En esta imagen, vemos que los valores más altos de precisión se han conseguido con el optimizador Adam y los valores más altos de neuronas.



[Tensorboard Hparams \(\[https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams\]\(https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams\)\)](https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams) (Dominio público)

Hay algunas opciones para la selección automatizada de conjuntos de parámetros que puedes investigar:

- ✓ Hyperopt: <https://github.com/hyperopt/hyperopt> (<https://github.com/hyperopt/hyperopt>)
- ✓ Hyperas: <https://github.com/maxpumperla/hyperas> (<https://github.com/maxpumperla/hyperas>)