KERAS

Conceptos que sabemos hasta ahora
Conceptos que nos quedan por conocer

IES Ribera del Tajo
Enseñanza bilingüe

Tensorflow - Librería para el uso de tensores

Keras – Api incorporada por tensorflow para creación de RNAs y modelos profundos de aprendizaje (Deep Models)

Se pueden resolver típicos problemas de machine learning:

- **regresión**
- **clasificación**
  - **binaria**
  - **multiclase**
  - **multietiqueta**

Workflow

1. Get data ready (turn into tensors)

2. Build or pick a pretrained model (to suit your problem)

3. Fit the model to the data and make a prediction

4. Evaluate the model

5. Improve through experimentation

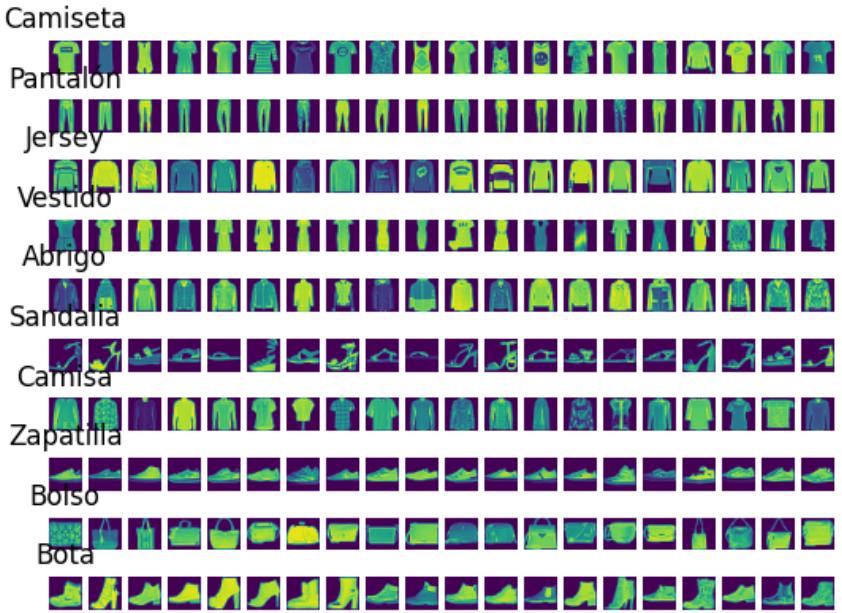
6. Save and reload your trained model

Visualizando los datos:

- permite identificar patrones, anomalías, tendencias y relaciones inherentes en los datos
- ayuda a tomar decisiones informadas sobre la preparación y limpieza de los datos, la selección de las características más relevantes y la configuración adecuada de la RNA

```
lista=np.array(lista)
fig,axs=plt.subplots(10,20)

for idx,i in enumerate(lista):
  for j in range(0,20):
      if j==0:
        axs[idx][j].set_title(class_names[idx])
      axs[idx][j].axis('off')
      axs[idx][j].imshow(lista[idx][j])
```



Crear DNNs con Keras:

Paso 1: Obtener X, y de un conjunto de datos y visualizarlos

Paso 2: Crear un modelo secuencial con:
- una capa de entrada
- una o varias capas ocultas
- una capa de salida

Paso 3: Compilar el modelo utilizando una función de pérdida, un optimizador y una serie de métricas

Paso 4: Entrenar el modelo

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Evaluar y Predecir:

```
model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.3297 - accuracy: 0.8891
```

```
classifications = model.predict(test_images)

print(classifications[34])
print(test_labels[34])
```

Mejorar modelos:
[hyperparameter tunning]

Añadir Capas
Incrementar número de neuronas
Cambiar funciones de activación
Cambiar la función de optimización
Cambiar el ratio de aprendizaje
Entrenar en más datos
Entrenar más tiempo (más épocas)

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Dependiendo de si es regresión o clasificación, cambiaremos los hiperparámetros de una manera o de otra

Regresión

Ejemplos:

- Predecir el precio de venta de las casas dada información sobre ellas (como el número de habitaciones, tamaño, número de baños).
- Predecir las coordenadas de un cuadro delimitador de un artículo en una imagen.
- Predecir el coste del seguro médico para un individuo dada su demografía (edad, sexo, género, raza).

Hyperparameter	Typical value
Input layer shape	Same shape as number of features (e.g. 3 for # bedrooms, # bathrooms, # car spaces in housing price prediction)
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited
Neurons per hidden layer	Problem specific, generally 10 to 100
Output layer shape	Same shape as desired prediction shape (e.g. 1 for house price)
Hidden activation	Usually ReLU (rectified linear unit)
Output activation	None, ReLU, logistic/tanh
Loss function	MSE (mean square error) or MAE (mean absolute error)/Huber (combination of MAE/MSE) if outliers
Optimizer	SGD (stochastic gradient descent), Adam

Clasificación

Ejemplos:

•Predecir si alguien tiene o no enfermedades del corazón basado en sus parámetros de salud. Esto se llama **clasificación binaria** ya que solo hay dos opciones.

•Decidir si una foto es de comida, una persona o un perro. Esto se llama **clasificación multiclase** ya que hay más de dos opciones.

•Predecir qué categorías deben asignarse a un artículo de Wikipedia. Esto se llama **clasificación multietiqueta** ya que un solo artículo podría tener más de una categoría asignada.

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 100	Same as binary classification
Output layer shape	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden activation	Usually ReLU (rectified linear unit)	Same as binary classification
Output activation	Sigmoid	Softmax
Loss function	Cross entropy (`tf.keras.losses.BinaryCrossentropy` in TensorFlow)	Cross entropy (`tf.keras.losses.CategoricalCrossentropy` in TensorFlow)
Optimizer	SGD (stochastic gradient descent), Adam	Same as binary classification

Visualizando el modelo

```
# Establecer semilla
tf.random.set_seed(42)

# Crear el modelo
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1, input_shape=[1])
])

# Compilar el modelo
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])
```

```
# Esto funcionará si la RNA ya conoce la
#forma de la capa de entrada (input_shape)
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
```

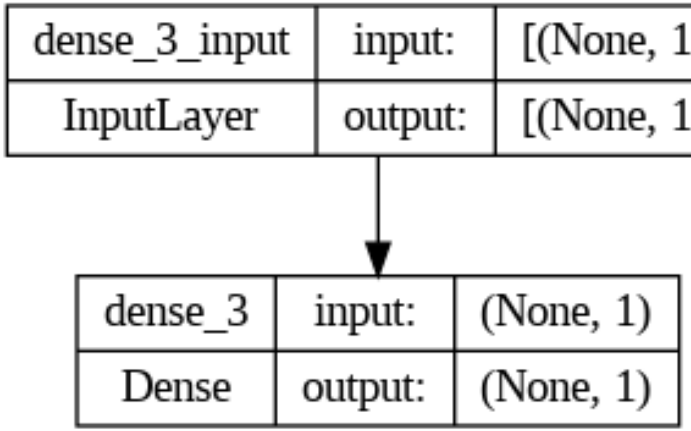
Llamar a **summary()** en nuestro modelo nos muestra las capas que contiene, la forma de salida y el número de parámetros.

- **Total params** - número total de parámetros en el modelo.
- **Parámetros entrenables** - estos son los parámetros (patrones) que el modelo puede actualizar a medida que se entrena.
- **Parámetros no entrenables** - estos parámetros no se actualizan durante el entrenamiento (esto es típico cuando se incorporan patrones ya aprendidos de otros modelos durante el aprendizaje por transferencia).

Visualizando el modelo utilizando keras.utils

```
from tensorflow.keras.utils import plot_model

plot_model(model, show_shapes=True)
```



Normalizar vs Escalar

$$X_{changed} = \frac{X - \mu}{\sigma}$$

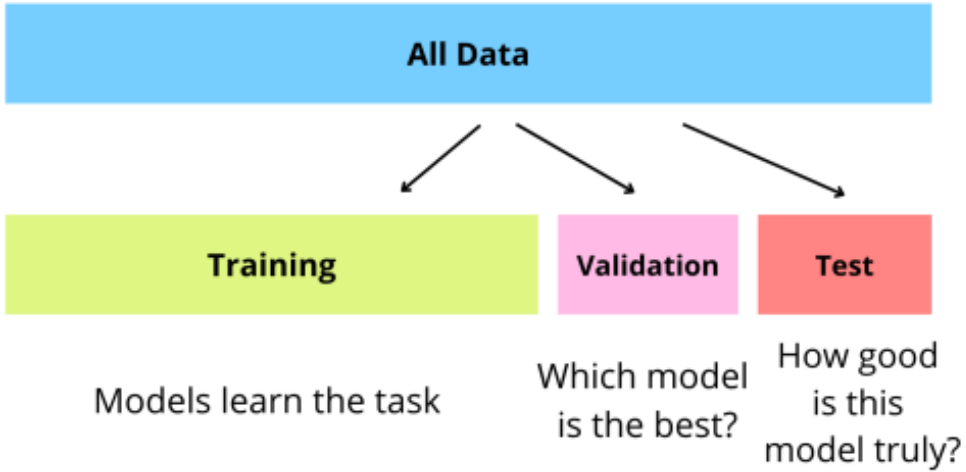
```
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
```

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

```
training_labels.size, test_labels.size
```

Dividir el conjunto de datos:



sklearn.model_selection.train_test_split ¶

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```
[source]

Calcular métricas

```
# Calcuate the MAE
mae = tf.metrics.mean_absolute_error(y_true=y_test,
                                     y_pred=y_preds.squeeze()) 
mae
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=30.48433>
```

```
# Calculate the MSE
mse = tf.metrics.mean_squared_error(y_true=y_test,
                                    y_pred=y_preds.squeeze())
mse
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=939.59827>
```

Callbacks

```
import tensorflow as tf
print(tf.__version__)

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.9):
      print("\Alcanzado el 90% de efectividad, cancelando entrenamiento!")
      self.model.stop_training = True

callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images/255.0
test_images=test_images/255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

Nuevos conceptos de Keras que aprenderemos en los próximos temas:

-Data Augmentation
- Guardar y cargar modelos
- Nuevas capas : DropOut, Poolings, Convolutions, Embedding, etc.

Transfer learning:
- reusar modelos: Feature Extraction+Fine Tuning
- Usar tensorflow Hub para, entre otras cosas, llevar el control y contabilidad de los modelos que entrenamos

Otros tipos alternativos a modelos sequential: Functional Keras API

Keras preprocessing: Tokenizer+PadSequences