Material de la charla

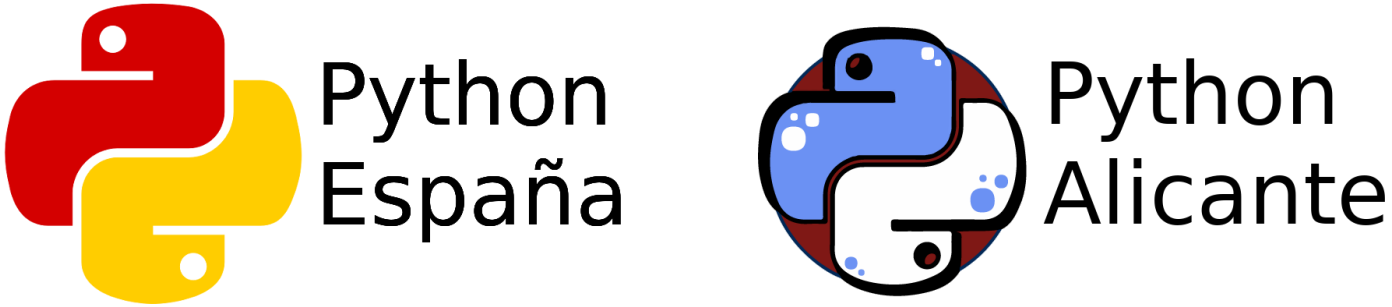
editado: borrado los parches para los ordenadores de la EPS para la instalación de las librerías en un entorno virtual (es una buena práctica, aunque no necesaria) ve a la sección "Instalación del entorno de trabajo"

```
git clone https://github.com/jaume-gasa/IntroduccionDeepLearningMesCultural2017ALC
cd IntroduccionDeepLearningMesCultural2017ALC
pip install -r requirements.txt
jupyter notebook
```

En caso de no funcionar, seguramente sea debido a la versión de python (aún no se ha llevado a cabo completamente la transición de python 2 a python 3)

Introducción al *deep learning* con Keras ¶

Jaume Gasa Gómez



Me presento

- Jaume Gasa
- Grado en Ingeniería Informática
- Actualmente en el Máster de Automática y Robótica
- Email: *ua.jaume@gmail.com*
- Python Alicante
  - Telegram y Twitter: @python_alc
- Python España
  - Telegram: @PythonEsp
  - Twitter: @python_es
  - Convención nacional de Python 2017, 22 al 24 de septiembre en Cáceres (http://2017.es.pycon.org/)

¿A quién va dirigida esta charla?

- ***Estudiantes*** de informática, multimedia y otras ingenierías que trabajen con datos
- Personas que desean dar sus ***primeros pasos*** en el mundo del *machine learning* o repasar los conceptos básicos
- Conocer el *framework* Keras

Lo que aprenderás:

- Qué es *machine learning* y *deep learning*
- Concepto de red neuronal
- Tipos de redes neuronales
- Encapsulación de datos con Pandas
- Uso de Keras para crear redes neuronales

Qué es lo que tratamos de resolver exactamente

Podemos separar los problemas de aprendizaje en:

- Supervisados: los datos están etiquetados y se corresponden a una salida.
  - Clasificación cuando el resultado el discreto (Reconocimiento de carácteres manuscritos)

    - Regresión si la salida es un valor contínuo (El precio de una casa en base a sus m²)
- No supervisados: muchos datos no etiquetados a los que se les busca una forma de agruparlos.

De qué hablamos cuando hablamos de ~~amor~~ *deep learning*

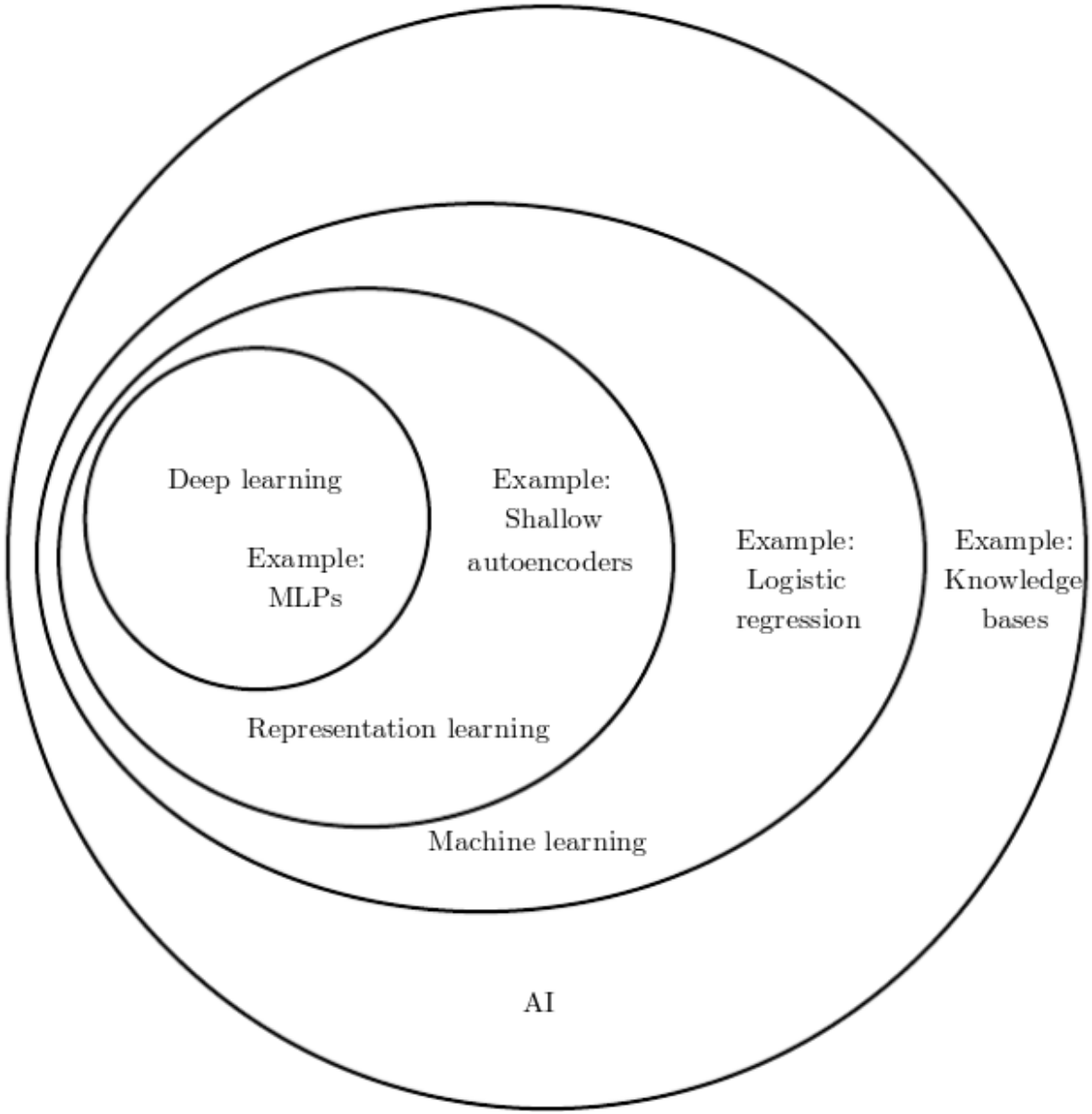


Figura (http://www.deeplearningbook.org) por Ian Goodfellow, Yoshua Bengio and Aaron Courville

Tipos de redes neuronales

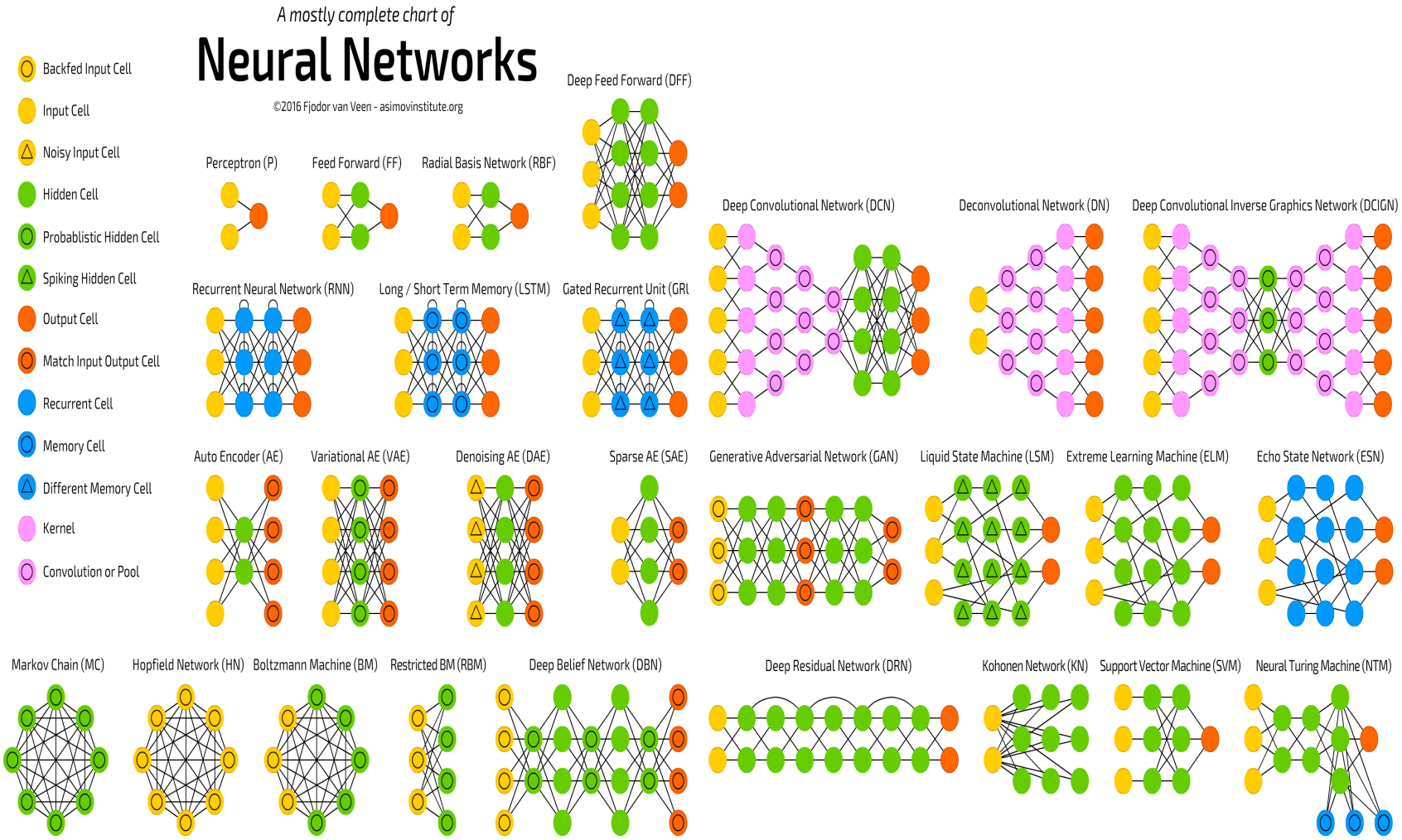


Figure (http://www.asimovinstitute.org/wp-content/uploads/2016/09/neuralnetworks.png) by Fjodor van Veen

Redes neuronales

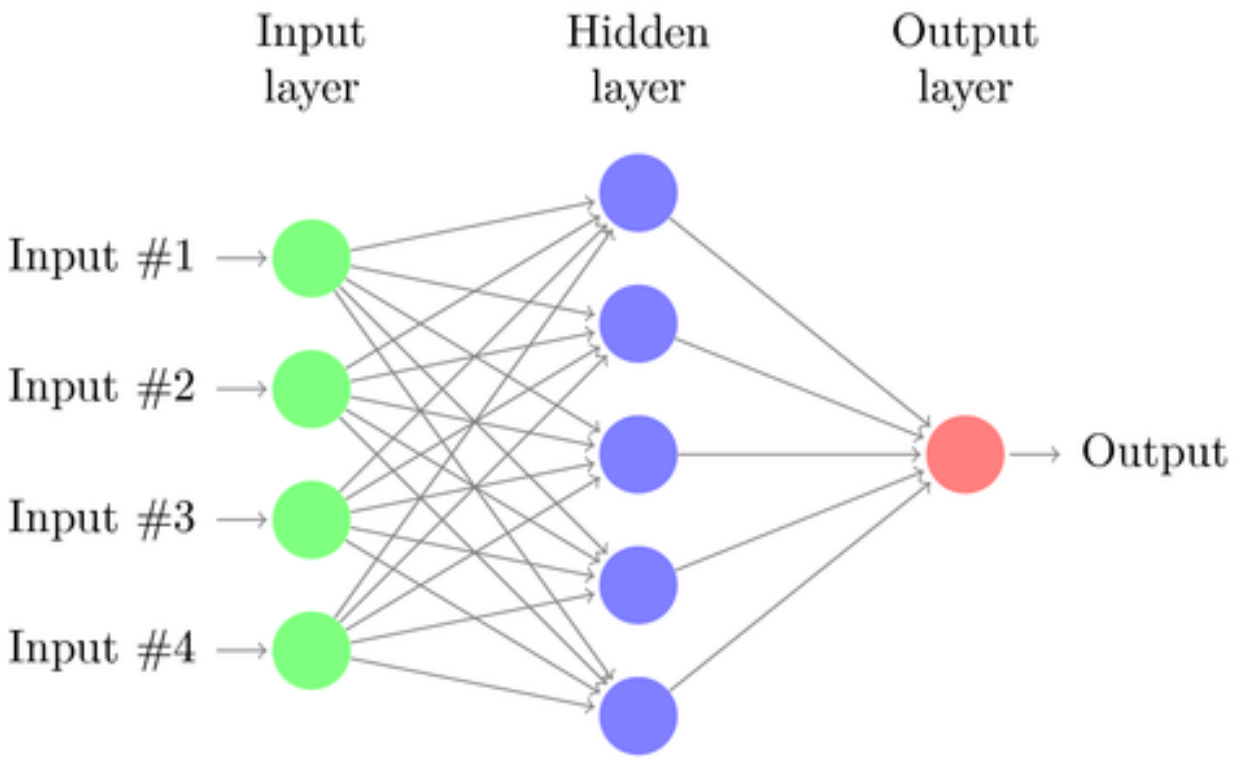


Figura (http://www.texample.net/media/tikz/examples/PDF/neural-network.pdf) por Kjell Magne Fauske / CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/)

- Entradas
- Salida(s)
- Pesos
- Funciones de activación

initializations

keras.io/inizializations (https://keras.io/initializations/)

Antes de entrenar, los pesos de la red han de ser inicializados, de no ser así, existe riesgo de saturación, lo que probocaría que la red no aprendiera por un aprendizaje muy lento o a estancarse en mínimos locales.

Los métodos más comunes son:

- uniform
- normal
- *glorot_uniform*
- he_normal

Activations

keras.io/activations (https://keras.io/activations/)

Permite a las redes neuronales aprender transformaciones no lineales complejas apartir de las entradas. Son las encargadas de *activar* las neuronas.

Las opciones más populares son:

- Sigmoid
- Tanh
- ReLu
- Softmax (*output layer*)

Regularizers

keras.io/regularizers (https://keras.io/regularizers/)

Métodos que se encargan de evitar el *overfitting*. Penaliza pesos excesivamente grandes.

Dropout

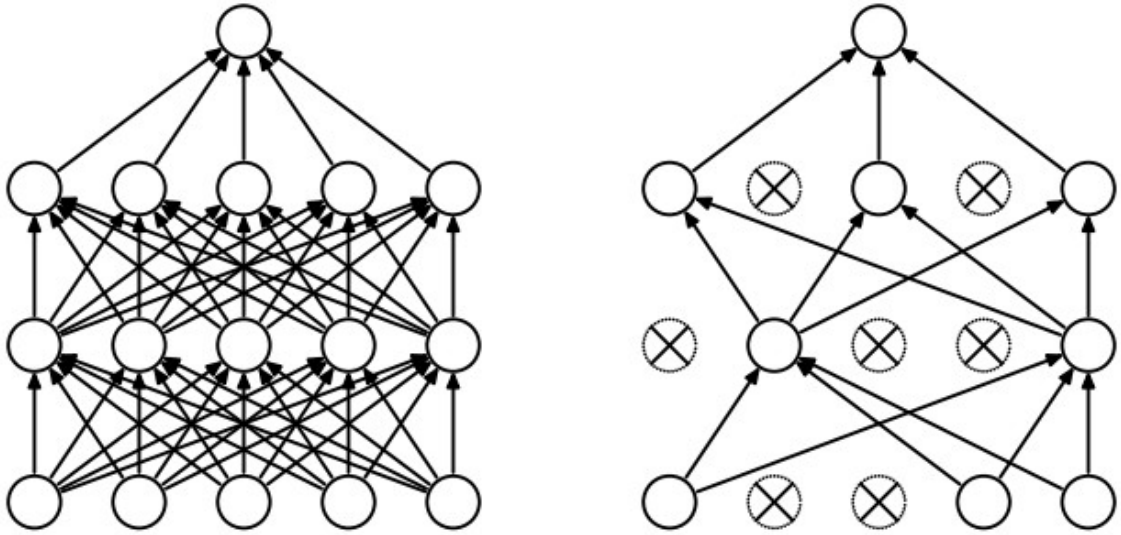


Figura (http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf) por N. Srivastava, G. E. Hinton,A. Krizhevsky,I. Sutskever,R. Salakhutdinov

Objectives

keras.io/objectives (https://keras.io/objectives/)

También llamado *loss function* o *optimization score function* es el primer páramatro de los dos necesarios para compilar una red. Es la manera matemática de cuantificar como dista la estimación actual de la deseada, es decir, permite medir el rendimiento de nuestra red y determina la penalización de los errores.

Algunas de las funciones más comunes son:

- mean_squared_error
- mean_squared_logarithmic_error
- categorical_crossentropy
- poisson

Optimizadors

keras.io/optimizers (https://keras.io/optimizers/)

Es el segundo parámetro necesario para compilar la red. El proceso de aprendizaje es un problema de optimización global en el cual los pesos de la red se van actualizando (aprendiendo) según se va minimizando la función de pérdida.

Ejemplos de optimizadores:

- *Stochastic gradient descent* (SGD)
- RMSprop
- Adam
- Adagrad
- Adamax

Instalación del entorno de trabajo

Instalación *virtualenv*

Los virtualenv son entornos virtuales en los que tenemos un mejor control de las versiones de las librerías que utilizamos

```
pip install virtualenvwrapper
mkdir ~/Virtualenvs
echo "export WORKON_HOME=~/Virtualenvs" >> ~/.bashrc
echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
source ~/.bashrc
```

Crear entorno virtual

```
mkvirtualenv nombre_que_quieras
```

Al terminar la creación del entorno, se entrará automáticamente. Verás que en la terminal se ha añadido (nombre_que_quieras)

Entrar y salir del entorno virutal

Para entrar (puedes autocompletar el nombre mediante la tecla tab):

```
workon nombre_que_le_hayas_puesto
```

Para salir:

```
deactivate
```

Instalación mediante requirements.txt

```
pip install -r requirements.txt
```

Abrir un *notebook*

`jupyter notebook` abrirá una instancia de tu navegador donde podrás interactuar con el *notebook*

```
cd directorio_del_notebook
jupyter notebook
```

Ejemplo práctico

¿Sobrevivirá al Titanic?

El 15 de Abril de 1912, el Titanic colisionó con un iceberg. En el accidente murieron 1502 de las 2224 personas abordo.

El dataset es un csv compuesto por información sobre los pasajeros del Titanic, por ejemplo, el nombre, sexo, edad, clase social, si sobrevivió o no etc.

Adquisición y tratamiento de datos

Pandas es una librería de alto rendimiento para encapsular datos.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

data_frame = pd.read_csv("./Data/train.csv")
train_data_frame, test_data_frame = train_test_split(data_frame, test_size = 0.2)
train_data_frame.info()
```

Limpieza y observación de los datos

A continuación nos desharemos de los datos que no nos interesen

In [ ]:
```
test_data_frame.info()
test_data_frame = test_data_frame.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
test_data_frame = test_data_frame.dropna()
test_data_frame.info()
```

In [ ]:
```
train_data_frame = train_data_frame.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)

train_data_frame.info()
```

Limpieza y observación de los datos

El atributo *age* tiene menos valores que el número de elementos, luego hay personas sin edad conocida.

Para inicializar esos valores, se usará la función ***fillna***

In [ ]:
```
age_mean = train_data_frame['Age'].mean()
# age_mean = train_data_frame['Age'].describe()['mean']
train_data_frame['Age'] = train_data_frame['Age'].fillna(age_mean)
test_data_frame['Age'] = test_data_frame['Age'].fillna(age_mean)
```

Limpieza y observación de los datos

Los atributos *Sex* y *Embarked* no son números, por lo que, para usarlo como entradas a la red neuronal, hay que transformarlos.

In [ ]:
```
from collections import Counter

Counter(train_data_frame['Embarked'])  # Más común
train_data_frame['Embarked'] = train_data_frame['Embarked'].fillna('S')
train_data_frame['Embarked'] = train_data_frame['Embarked'].map({'C':1, 'S':2, 'Q':3})

print(train_data_frame['Embarked'])

test_data_frame['Embarked'] = test_data_frame['Embarked'].fillna('S')
test_data_frame['Embarked'] = test_data_frame['Embarked'].map({'C':1, 'S':2, 'Q':3})

train_data_frame['Sex'] = train_data_frame['Sex'].map({'female':0, 'male':1})
test_data_frame['Sex'] = test_data_frame['Sex'].map({'female':0, 'male':1})
```

Limpieza y observación de los datos

Los atributos *Sex* y *Embarked* no son números, por lo que, para usarlo como entradas a la red neuronal, hay que transformarlos.

In [ ]:
```
num_features = train_data_frame[train_data_frame.columns.difference(['Survived'])].shape[1]
X_train = train_data_frame[train_data_frame.columns.difference(['Survived'])].values
y_train = pd.get_dummies(train_data_frame['Survived']).values

test_data_frame.info()

X_test = test_data_frame[test_data_frame.columns.difference(['Survived'])].values
y_test = pd.get_dummies(test_data_frame['Survived']).values
```

Creación de la red neuronal

Utilizando keras montaremos una red neuronal para que aprenda qué tipos de persona (en base a sus *features*) sobrevivieron y qué tipo, no.

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.layers.normalization import BatchNormalization
from keras.optimizers import *
from keras.callbacks import EarlyStopping, ModelCheckpoint

def make_neural_network():
    # instantiate model
    model = Sequential()

    # Construimos la input layer
    model.add(Dense(500, input_dim=7, init='glorot_uniform'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Primera hidden layer
    model.add(Dense(500, init='glorot_uniform'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # Segunda hidden layer
    model.add(Dense(500, init='glorot_uniform'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.3))

    # output layer
    model.add(Dense(output_dim=2, init='glorot_uniform'))
    model.add(BatchNormalization())
    model.add(Activation('softmax'))

    # setting up the optimization of our weights
    sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy',
                  optimizer=sgd,
                  metrics=['accuracy'])
    return model

model = make_neural_network()
```

Entrenamiento de la red neuronal

In [ ]:
```
def train(model):# running the fitting
    model.fit(X_train, y_train, batch_size=32, nb_epoch=500, validation_split=0.2, verbose = 1,
          callbacks=[
              EarlyStopping(verbose=True,patience=40,monitor='val_loss'),
              ModelCheckpoint('best-fit-model',monitor='val_loss',verbose=True,save_best_only=True)
          ])
train(model)
```

Clasificación del *dataset* de validación

In [ ]:
```
model.load_weights('best-fit-model')

model.fit(X_test, y_test)

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy = {:.2f}".format(accuracy))
```

Una vuelta de tuerca

Un 70% no nos parece un resultado excesivamente bueno.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# print(X_train)

X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

model_2 = make_neural_network()

# running the fitting
model_2.fit(X_train, y_train, batch_size=32, nb_epoch=500, validation_split=0.2, verbose = 1,
          callbacks=[
              EarlyStopping(verbose=True,patience=40,monitor='val_loss'),
              ModelCheckpoint('best-fit-model',monitor='val_loss',verbose=True,save_best_only=True)
          ])
```

Una vuelta de tuerca (2)

In [ ]:
```
model_2.load_weights('best-fit-model')

model_2.fit(X_test, y_test)


loss, accuracy = model_2.evaluate(X_test, y_test, verbose=0)
print("Accuracy = {:.2f}".format(accuracy))
```

Inconvenientes del *deep learning*

A pesar del cuento de que el *deep learning* aprende los features y los factores de manera automática, para problemas reales, sigue siendo necesario que los datos estén bien estructurados y relativamente adaptados al problema.

Problemas complejos requieren **grandes** cantidades de datos.

Conseguir resultados superiores al 98~99% de aciertos (lo que espera la gente de nuestro producto, sino "no funciona") requiere ajustes muchos hiperparámetros (*learning rate*, *loss function*, *mini-batch size*, número de iteraciones en la etapa de aprendizaje, *momentum*, selección de optimizadores...)

¡Las matemáticas son necesarias! (especialmente la estadística -> Statistics for hackers (https://www.youtube.com/watch?v=Iq9DzN6mvYA))

Recursos recomendados

- (Libro) Deep learning por Ian Goodfellow, Yoshua Bengio y Aaron Courville http://www.deeplearningbook.org/ (http://www.deeplearningbook.org/)
- (Curso *online*) Machine learning por Andrew Ng https://www.coursera.org/learn/machine-learning (https://www.coursera.org/learn/machine-learning)
- (Documentación) Scikit-learn http://scikit-learn.org/stable/documentation.html (http://scikit-learn.org/stable/documentation.html)
- (Curso *online*) Deep Learning with TensorFlow por Saeed Aghabozorgi https://bigdatauniversity.com/courses/deep-learning-tensorflow/ (https://bigdatauniversity.com/courses/deep-learning-tensorflow/)
- (Web) Kaggle, competiciones de ánalisis de datos y predicciones usando *machine learning* https://kaggle.com (https://kaggle.com)


RUMBABEAT
Thank you!