
UNIVERSITAT DE BARCELONA

**Inteligencia Artificial**

Curso 2020 - 2021

# Deep Learning usando GPT2 Aplicado a sinopsis

realizado por

Joan Domenech Puig

NIUB: 20206955

y

Estíbaliz Martínez Cano

NIUB: 20345592

Barcelona

22 de diciembre de 2020

# Índice

# 1. Introducción

Hemos hecho uso de la IA a través de un algoritmo basado en redes neuronales recurrentes, denominado **GPT-2**. De este modo, hemos sido capaces de generar nuevas sinopsis a partir de sinopsis ya existentes. En particular, hemos tomado como `input` 4832 sinopsis en inglés de series de animación japonesa, también conocidas como *animes*, y a partir de éstas hemos generado nuevas sinopsis con coherencia.

# 2. Redes neuronales

Comenzaremos dando una breve introducción a las **redes neuronales** o redes neuronales artificiales (ANN) puesto que serán la base para comprender la arquitectura de GPT-2. Una red neuronal es un conjunto de algoritmos que son diseñados para reconocer patrones o relaciones entre una gran amplia cantidad de datos."Aprende" cuáles son estos patrones mediante múltiples entrenamientos hasta que llega a ser capaz de interpretar los datos de entrada mediante agrupamientos o etiquetados. Básicamente, puede extraer las características de la entrada dadas y devolver los datos clasificados/agrupados.

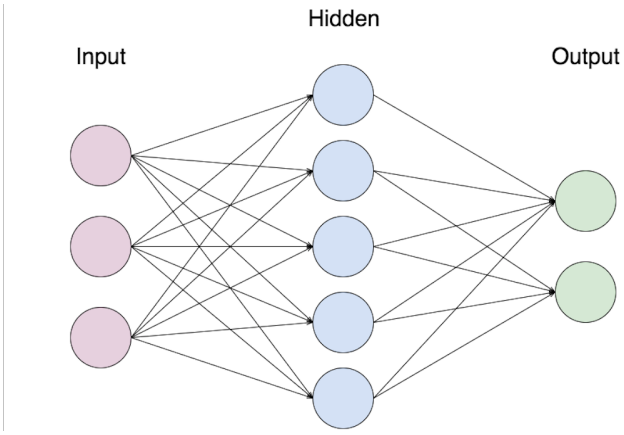


Figura 1: Arquitectura de una red neuronal simple

Para obtener los resultados deseados una red neuronal hace uso de un pro-

ceso de entrenamiento o aprendizaje, como si fuera un auténtico cerebro. Para ello la primera acción que lleva a cabo es dividir los `inputs` en tres conjuntos:

- **Training o entrenamiento**: Ayuda a la red neuronal a comprender el peso entre los nodos, la importancia de unos frente a otros. Por ejemplo, en nuestro caso aprendería a distinguir cuando una palabra es un mero conector y cuando se trata de un nombre.
- **Validación**: Se utiliza para ajustar el rendimiento de la red.
- **Test**: Determina la precisión o margen de error de la red neuronal. Sirve para conocer cuan buenos son los `outputs` que esta proporcionando

Como podemos apreciar en la **fig.1** su arquitectura se fundamenta en capas compuestas por nodos o neuronas interconectados, cada nodo se encuentra conectado a nodos de las siguiente capa. La red neuronal más básica contiene una capa de `input` o datos de entrada, un capa oculta y una capa de `output` o datos de salida.

La capa (o capas ocultas) será donde tenga lugar el cálculo principal de la red neuronal. Toma los `inputs` de la capa anterior y realiza las operaciones necesarias para generar un resultado que se reenvía a la capa de `outputs`.

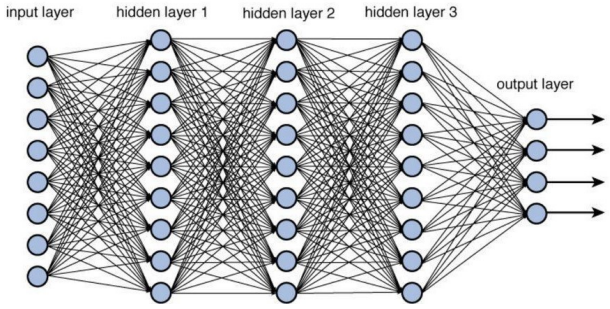


Figura 2: Arquitectura de una red neuronal profunda

Cuando una ANN tiene más de tres capas (incluidas la entrada y la salida), es decir tiene múltiples capas ocultas se denomina **red neuronal profunda** y

pasa a ser una arquitectura del conocido *deep learning* o aprendizaje profundo.

En particular, las redes que haremos servir nosotros también son consideradas parte del *deep learning*, se denominan **redes neuronales recurrentes** (RNNs). Este tipo de redes tienen bucles en ellas permitiendo que la información persista.

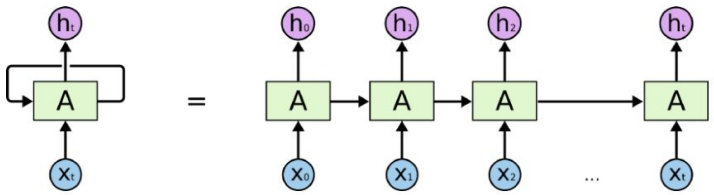


Figura 3: Arquitectura de una red neuronal recurrente

Los bucles pueden interpretarse también como múltiples copias de la misma red neuronal, en la que cada red pasa un mensaje a su sucesor **fig.3**. En la mencionada figura nuestra RNN sería $A$, $x_t$ representaría los `inputs` y $h_t$ los `outputs`.

# 3. GPT-2

El transformador generativo pre-entrenado 2 (GPT-2) es una inteligencia artificial de código abierto. GPT-2 utiliza deep learning para traducir texto, responder preguntas, resumir pasajes y generar resultados de texto legibles por humanos en un nivel que a menudo es indistinguible del de los humanos. Se trata de un modelo generativo que puede entrenarse previamente utilizando una enorme cantidad de texto, y su función consiste en generar texto mediante la predicción de la siguiente palabra a través de secuencia de tokens

# 4. Código

A continuación, haremos una explicación del código, así como todos los pasos previos necesarios para ejecutarlo.
Cabe destacar que este código y todas las librerías usadas han sido instaladas mediante la consola de Anaconda (Anaconda `prompt`). Todas las instrucciones, resultados y conclusiones se basarán sobre el trabajo en ese ambiente. También cabe señalar que el contenido de las librerías que usamos puede ser bastante elevado y que el uso de una **Nvidia GPU** es necesario para ejecutar una parte del código.

## 4.1. Librerías necesarias

### 4.1.1. PyTorch

Es un paquete informático científico basado en *Python*, este se puede usar como reemplazo de $NumPy$ para usar la potencia de las GPU de manera más flexible y más veloz. También incluye **CUDA** que es una plataforma de computación paralela y un modelo de programación que hace uso de una GPU. Desafortunadamente esta librería es la que fuerza el uso de una Nvidia GPU. Se instala con el comando siguiente:

```
conda install -c pytorch
```

### 4.1.2. Transformers

Esta librería es el principal punto de nuestro proyecto, contiene todos los elementos necesarios para llevar a cabo GPT-2 así como muchos otros generadores de lenguaje natural. Una de sus mayores ventajas es que contiene modelos preentrenados para hacer más simple el entrenamiento para casos específicos como el nuestro. Se puede instalar mediante:

```
conda install -c conda-forge transformers
```

### 4.1.3. Tqdm

Esta librería tiene un valor puramente estético ya que permite mostrar barras de progreso durante bucles e iteraciones, aun así, las librerías de $Transformers$ y $Torch$ la usan y es necesaria para el correcto funcionamiento del código. Se

puede instalar como:

```
conda install -c conda-forge tqdm
```

#### 4.1.4. Pandas y Numpy

Pese a usar *Torch*, las librerías *Numpy* son necesarias para algunas de las clases del código por lo que es necesario tener estas librerías instaladas, por otra parte, Pandas es necesarias para poder gestionar los datos para crear el entrenamiento. Se pueden instalar con los siguientes comandos:

```
conda install -c anaconda numpy
```

```
conda install -c anaconda pandas
```

#### 4.1.5. Librerías estándar

Las siguientes librerías también son usadas, aunque teóricamente, deberían estar instaladas de manera predeterminada con *Python*.

- Librería re
- Librería os
- Librería shutil

#### 4.1.6. Errores y posibles soluciones

Durante la instalación de las librerías, encontramos múltiples problemas para lograr que estas funcionasen, aun así, logramos solucionarlos todos.

Uno de los posibles errores a encontrar es que la versión de CUDA este obsoleta por defecto, para arreglarlo hemos instalado una versión de CUDA que sabemos que funciona mediante el comando siguiente:

```
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
```

Otro posible problema es que la librería *Numpy* este en una versión incompatible, nosotros solucionamos este problema usando la versión 1.16

```
conda install numpy=1.16
```

También cabe la posibilidad de que las librerías $Transformers$ o $Tqdm$, instaladas anteriormente, estén obsoletas, aunque desconocemos el motivo por el que las librerías se instalan en versiones antiguas, encontramos que usando pip se instalan con la versión más reciente.

```
pip install transformers
```

El ultimo problema mayor que se puede encontrar es un error con CUDA. Este error puede encontrarse en dispositivos que no tienen una GPU, con el mensaje

```
RuntimeError: Attempting to deserialize object on a CUDA device but
torch.cuda.is_available() is False. If you are running on a CPU-only
machine, please use torch.load with map_location='cpu' to map your
storages to the CPU.
```

La solución a este problema es simple y extraña. Se debe acceder al archivo `serialization.py` que se guarda en `./site-package/torch/serialization.py`, y modificar una función tal que:

```
def load(f, map_location=None, pickle_module=pickle, **pickle_load_args):
```

se convierte en

```
def load(f, map_location='cpu', pickle_module=pickle, **pickle_load_args):
```

Normalmente este problema implica que no se posee una Nvidia GPU, aunque como explicaremos más adelante, no todo el código es dependiente de ello.

## 4.2. Descripción de los datos

Con tal de entrenar la IA hemos generado un archivo `csv` con alrededor de 5000 sinopsis en inglés obtenidas de la web MyAnimeList. En nuestro caso hemos decidido que las sinopsis usadas sean de series de animación japonesa, aunque, si se quisiera, se podrían cambiar los datos por series de Netflix o similares y el programa funcionaria, aunque el estilo de las sinopsis generadas sería distinto, es decir, más similar a las nuevas muestras.

## 4.3. Explicación del código

A continuación, explicaremos el código por partes, es decir, explicaremos la función de cada uno de los ficheros que usamos en el proyecto.

#### 4.3.1. config.py

Este archivo contiene variables globales, estas variables son usadas universalmente por todo el código. Entre ellas, se encuentran el número máximo de caracteres de las sinopsis para entrenar, el nombre del archivo donde se guardará (o desde donde se leerá) el resultado del entrenamiento, los parámetros del entrenamiento, como el tamaño de las muestras (batch_size) o el número de veces que el dataset se pasará por la red neuronal (epoch), y para finalizar los modelos GPT-2 pre-entrenados que usaremos para el proyecto.

#### 4.3.2. utils.py

Este archivo es prácticamente una librería, su único uso es para AverageMeter(), que sirve para guardar datos. Esta librería fue obtenida a través de la página web oficial de PyTorch.

#### 4.3.3. clean.py

Este archivo se encarga de depurar las sinopsis que se le pasan, elimina aquellas que superen el número deseado de caracteres y elimina todos los símbolos que no nos interesen, así como aquellas sinopsis poco relevantes (como por ejemplo: Esta es una adaptación de la novela XXX).

#### 4.3.4. dataset.py

Se encarga de gestionar la tabla con las sinopsis. Entre sus funciones, llama a clean.py para depurar el contenido del csv y de igual manera posee un método para poder dividir las sinopsis mediante tokens y retornarlos como tablas de PyTorch.

#### 4.3.5. train_fn.py

Esta es la función encargada de generar el modelo. Primero toma los identificadores y las máscaras del cargador de datos y los pasa a través del modelo. A continuación, realiza la retro-propagación y actualiza los parámetros además de devolver la pérdida media del epoch.

#### 4.3.6. run.py

Este es el archivo principal por donde se ejecutará el entrenamiento del modelo, primero lee los datos usando dataset.py, a continuación, se define un optimizador. Los optimizadores dan forma y moldean el modelo en su forma más precisa posible comparando el peso de los tokens, en nuestro caso usamos AdamW.

El siguiente paso es definir un planificador, que es el encargado de hacer que la velocidad de aprendizaje varíe. De la manera que lo hemos programado, los primeros 10 pasos de entrenamiento, la tasa de aprendizaje aumentará linealmente y luego disminuirá linealmente.

Finalmente ejecutamos la función de entrenamiento por tantos epochs como hayamos definido, cuantos más se hagan, menor será la perdida de datos, y cuanto menor sea la perdida de datos, mejores resultados se obtendrán. En nuestro caso la perdida tras 6 epochs es de 1.05, un muy buen resultado, pero que suele tardar alrededor de 50 minutos en total.

### 4.3.7. generate.py

Este es el último archivo del proyecto y es el encargado de generar las sinopsis. Este archivo se divide en una función, y un código que hace la función de main. El código principal llama a la función mencionada anteriormente y le pasa por parámetro unos ejemplos para empezar a generar sinopsis, el número de sinopsis que genera depende del número de ejemplos y estos solo sirven de guía para empezar a generar. A continuación, imprime por pantalla los ejemplos generados y los guarda en un archivo llamado "Generated Examples.txt".

Por otra parte la función es la encargada de generar las sinopsis. La función simplemente genera predicciones de las palabras más probables después de los ejemplos introducidos, de esta manera es capaz de generar sinopsis coherentes.

## 4.4. Ejecución del código

Con tal de ejecutar el código se deben seguir los siguientes pasos (todos los ficheros deben estar en la misma carpeta de la misma manera que la consola):

- Instalar las librerías necesarias
- (Opcional) Ejecutar "python run.py", este paso es opcional debido a que el modelo ya esta adjunto junto con el código (archivo adjunto gpt2_model.bin). Además debido al tiempo que tarda en ejecutar y a la posibilidad de que el código no funcione, hemos adjuntado una grabación ejecutando este archivo (ver achivo adjunto run_execution.mp4).
- (Opcional) Modificar la variable input_texts. Si se modifica esta variable se pueden comprobar múltiples tipos de sinopsis (ver 4.3.7).
- Ejecutar el comando "python generate.py". El código imprimirá las sinopsis por pantalla y creara un archivo para poder leerlas posteriormente.

## 4.5. Resultados obtenidos

A continuación presentamos algunas de las sinopsis que hemos generado:

- *Spoon was a young boy who was sent to a boarding school in England to study at. He had no idea what to expect when he arrived at the school he had just moved into. There, he met a beautiful girl named Rita, who had come to stay with him after his parents had passed away. The two of them decided to start a new life together in London, where they met many interesting people along the way.*
- *In the year 2199, the Earth is in a state of war between two rival superpowers: the United Federation of Nations, and the Coordination Church of Humanity. The Coordinators, a group of people who have been artificially enhanced by genetically altering their bodies to be more like humans, have begun a campaign of mass murder and terror in order to bring about an end to the war.*

  *Seventeen-year-old Yuuichirou Inoue is a member of the 501st Joint Fighter Wing, an elite squadron that is tasked with defending Earth against these terrorist attacks. However, when his squadron is attacked by an organization calling themselves the Knights of St. Michael, he is forced to join their squadron as a means of protection against further alien threats that threaten the peace of all around the world.*
- *During the war against the Zentradi, the United Nations established a special task force called the Sibyl System to deal with the threat of extraterrestrials.*

*This specialized unit, known as the 501st Joint Fighter Wing, was tasked with intercept and eliminate any alien that came within the borders of the planet.*

*During one of its missions, a bomber carrying a humanoid robot fell into a waterlogged area. The bomber was able to escape, but was hit by an unknown object that crashed into the ocean. In the ensuing chaos, it was discovered that the robot had been artificially enhanced and was in fact a descendant of a race of superhumans that had descended to Earth from outer space and colonized it. To combat the alien threat, an international organization called SEED was formed, and together with a squadron of special pilots, they launched Operation Odessa to search for the lost robot.*

- *A young man, who has lost his mother in a traffic accident, finds himself stranded on a deserted island with only a few of his possessions. There, he is greeted by a young girl, whose name is Mari, and he quickly falls in love with her. However, Mari is not the type of girl you would expect from a girl his age: she is actually a vampire, born with the appearance of a human woman.*

  *Mari soon finds herself drawn into a world of vampires and other supernatural beings, where she discovers that she has the power to see into the hearts of other living creatures. As she learns more about the world around her, she also begins to realize that there is more to this world than meets the eye, as she begins searching for the reason for her existence.*