Generación procedural de contenido basado en aprendizaje automático

Procedural content generation via machine learning

TRABAJO DE FIN DE GRADO

Víctor Emiliano Fernández Rubio
Gonzalo Guzmán del Río
Carlos Llames Arribas
Tutor: Samir Genaim

Facultad de Informática
Universidad Complutense de Madrid

15 de junio 2020

Documento maquetado con TEXIS v.1.0+.

Agradecimientos

Nos gustaría agradecer la realización y el desarrollo de este proyecto, en especial a nuestro tutor de TFG, Samir Genaim, el cual nos ha ayudado en todo lo que le hemos ido preguntando, además de proponernos nuevas ideas sobre las que investigar. Además, agradecer también su predisposición a realizar reuniones virtuales en estos meses difíciles. También nos gustaría destacar a nuestros familiares, y compañeros y amigos que nos han ayudado a tomar ciertas decisiones, esenciales para poder desarrollar nuestro trabajo de manera correcta. Por último nos gustaría destacar nuestro trabajo tanto individual como colectivo, todas las charlas que hemos tenido y las propuestas de cada uno, lo que ha permitido llevar el trabajo siempre por el camino correcto.

Acknowledgment

We would like to thank the development and realization of this project to our TFG director, Samir Genaim, who has helped us on everything that has been asked to him, and also his proposals about new ideas to investigate on. Beside, we want to thank his predisposition to hold virtual meetings in all these difficult months. We would also like to highlight the support of all our families, colleagues and friends, who have helped us to take essential decisions in order to a correct development of the project. Finally, we want to stand out, our individual and collective work, all the meetings that we have had and each other's proposals, what have allowed us to go always the right way.

Resumen

Durante los últimos años, tanto la inteligencia artificial como el aprendizaje automático, se han convertido en un foco constante de investigación y enseñanza, así como de aprendizaje. Además cada vez más empresas, ven a estas técnicas como un punto de partida hacia su crecimiento tanto económico como tecnológico, permitiendo a estas, entrar en otros sectores. Por ejemplo Microsoft, empezó a adentrarse en el mundo de la inteligencia artificial y aprendizaje automático con Kinect, o Google desarrollando un algoritmo capaz de derrotar a los mejores jugadores del mundo de Dota. En otros sectores como en el de la agricultura, la inteligencia artificial está siendo utilizada, para mejorar la eficiencia en cuanto a producción, prediciendo los rendimientos de la cosecha. Además todo lo comentado anteriormente, nos lo encontramos hoy en día y vivimos con ello, destacando entre otros a asistentes personales como Alexa o Siri.

Debido a esto, hemos planteado nuestro trabajo de fin de tal manera, que se nos presenta una oportunidad única de aprender como funcionan los algoritmos de aprendizaje automático e inteligencia artificial, aplicándola al ámbito sobre el que hemos desarrollado nuestros estudios durante los últimos años, los videojuegos, y en concreto a la creación de mapas del videojuego SuperMario.

A lo largo de todo el proyecto, investigaremos acerca de cuales son las mejores técnicas tanto de aprendizaje automático como de inteligencia artificial, así como cuál es la forma más óptima de implementarlas. Desarrollaremos scripts para comprobar su funcionamiento, con la ayuda de diversas librerías entre las que se encuentran Tensorflow o NLTK, así como una aplicación en Unity, la cual nos servirá de base para poder mostrar los mapas que vayan siendo generados. Esta aplicación, permitirá la posibilidad de mostrar los 8 mapas originales de los que disponemos, los cuales han sido realizados a mano, así como crear nuevos mapas, con algunos de los algoritmos investigados e implementados. Estos mapas podrán ser monotema, que resultan a partir de un solo mapa, o multitema, que se crean a partir de la unión de diferentes mapas.

Palabras clave

Generación procedural de contenido, aprendizaje automático, inteligencia artificial, TensorFlow, nltk, Mario Bros, clasificación de texto, modelos del lenguaje, procesamiento de lenguajes naturales, n-gramas, redes neuronales.

Summary

During the last years, both the artificial intelligence and machine learning have become in a constant focus of research and teaching, and also of learning. Besides, more and more companies, see these techniques as an starting point to their economical and technological growing, by entering other sectors. For example Microsoft, stepped into in artificial intelligence and machine learning with Kinect, or Google developing an algorithm able to beat the best Dota players all over the world. In other sectors like the agricultural, the artificial intelligence is being used to improve the production efficiency by predicting the crop yields. Also, everything that has been commented above, can be found nowadays in our lifes, highlighting for example, voice assistants such as Alexa or Siri.

Because of all of this, we have raised our final degree project as a unique opportunity to learn how the machine learning and artificial intelligence algorithms work, by applying it on what we have been studying during the last years, the video games, and in particular the map generation based on SuperMario.

Throughout all the project, we will research about which are the best techniques of machine learning and artificial intelligence, and also which is the best way of implementing them. We will develop scripts in order to check the behaviour, with the help of some libraries such as Tensorflow or NLTK, and also a Unity project, which will serve us as a base to show all the maps that are being generated. This application will give the possibility of showing the first eight original maps, and also creating new ones, with some of the researched and implemented algorithms. These maps will be mono theme, from just one map, or multi theme, created from the join of different maps.

Keywords

Procedural content generation, machine learning, artificial intelligence, Tensorflow, nltk, Mario Bros, text classification, language models, natural language processing, n-grams, neural networks.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

En estos años, la inteligencia artificial y el aprendizaje automático se han convertido en una tecnología muy popular en todo el mundo, accesible por todos y con muchas proyecciones de cara al futuro. Además, grandes compañías como Google, Apple, Amazon o Microsoft están invirtiendo tiempo y dinero en proyectos de investigación y desarrollo de aplicaciones basadas en inteligencia artificial y aprendizaje automático. Por supuesto, además de las grandes compañías, muchas nuevas empresas aparecen y crecen centrándose únicamente en este tipo de tecnología. Esta tendencia solo aumentará de cara al futuro, ya que se estima que la industria de la inteligencia artificial alcanzará los 118 mil millones de dólares en 2025.

No obstante, no todo son beneficios, hay otra parte más oscura acerca del aprendizaje de las máquinas. Stephen Hawking dijo: "la inteligencia artificial probablemente sea lo mejor o lo peor que le puede pasar a la humanidad"[1]. De momento no hemos visto lo peor, y es muy difícil predecir cómo se desarrollará la inteligencia artificial en el futuro, y si en algún momento se volverá plenamente consciente. Sin embargo, ya se conocen muchos de los problemas que podrían ocasionar tomar decisiones sin razonamientos humanos, basándose solamente en datos.

Pero aparte de los peligros, la inteligencia artificial tiene una gran cantidad de ventajas que nos pueden facilitar la vida a todos. Además, es aplicable a todos los sectores económicos. Actualmente, sectores como el automovilístico o el sanitario se están beneficiando de estos algoritmos de aprendizaje automático, y los videojuegos no están excluidos. Muchos videojuegos utilizan el aprendizaje automático durante su desarrollo para facilitar y ahorrar tiempo y costes a los desarrolladores. Incluso durante el juego se utilizan inteligencias artificiales para dar una sensación más realista, consiguiendo sumergir al jugador más aún en el propio juego. Y es que las expectativas de la inteligencia artificial, como hemos visto antes, no son para nada malas. Cada vez se van a ir uniendo al uso de ella más y más sectores, y gracias a esto va a continuar evolucionando.

1.2. Objetivos

El objetivo general de este proyecto es la investigación y el análisis de las distintas formas de generación procedural de contenido y la comparación de algunos de estos

algoritmos. Para poder probar esto hemos considerado generar mapas o niveles y en última instancia, implementar una pequeña aplicación demo con la que probar los diferentes resultados de cada modelo estudiado. Para poder alcanzar estos objetivos se han establecido una serie de hitos[1] específicos:

1. Investigar sobre la clasificación de texto y sobre los modelos del lenguaje más usados, tales como Bolsa de Palabras (Bag of Words) o n-gramas[2].
2. Estudiar el modelo n-gramas e implementar nuestro propio modelo de este tipo para entender su funcionamiento en la práctica.
3. Comparar el comportamiento del modelo n-gramas en situaciones con distintos tamaños de corpus, es decir, datos para entrenar.
4. Comparar el rendimiento de n-grams con un tamaño variable respecto a n-grams con un tamaño fijo.
5. Estudiar los distintos tipos de redes neuronales.
6. Estudiar a fondo las redes neuronales recursivas y porque son óptimas para la creación de contenidos, sobretodo si son artísticos.
7. Comparar las distintas posibles redes neuronales recursivas, así como sus posibles parámetros.
8. Implementar los distintos modelos anteriormente estudiados, así como su adaptación para la generación de los niveles.
9. Comparar los modelos nombrados anteriormente.
10. Crear una aplicación demo para mostrar los diferentes resultados generados.

Para ello, utilizaremos como ejemplo el juego de Super Mario Bros, un juego de plataformas 2D horizontal, de la Nintendo Enetertaiment System, NES.

1.3. Metodología

Para realizar los objetivos citados anteriormente se investigarán fuentes en internet, artículos científicos, estudios previos y libros, todos ellos reflejados en la bibliografía y web grafía. Estos recursos ayudarán a la hora de investigar entre los diferentes modelos existentes, así como la comparación entre ellos y su implementación. Una vez completada la investigación teórica, se estudiarán las diferentes herramientas para el estudio de los modelos elegidos. Con el fin de ver los puntos fuertes y débiles de cada algoritmo en su implementación, ejecución y resultados obtenidos. A continuación, se pondrán a prueba dichos modelos. Se implementarán algoritmos para la generación de niveles y se compararán tanto los resultados como sus tiempos de ejecución o su complejidad de implementación.

[1] Acontecimiento puntual y significativo que marca un momento importante en el desarrollo de un proyecto

[2] n-grams o n-gramas es una secuencia de palabras continuas de tamaño n que aparecen en un texto

Por ultimo se desarrollará una aplicación donde poder explotar al máximo los modelos elegidos. De esta forma, obtener unos resultados y establecer unas conclusiones acerca de la generación procedural mediante aprendizaje automático respecto a la generación de mapas en los videojuegos. Para la creación de la aplicación, se utilizará como entorno de desarrollo Unity 2019.2.1f1, uno de los motores de videojuegos[3] más conocidos y punteros en la industria. Este motor está presente en la facultad de informática de la Universidad Complutense de Madrid para el estudio a lo largo del grado de desarrollo de videojuegos. Gracias a esto, nos resultará más cómodo el desarrollo de nuestra aplicación.

La implementación de los algoritmos de aprendizaje automático será en el lenguaje Python, escribiendo estos en unos *scripts* a parte, que posteriormente conectaremos con Unity. Para ello se utilizarán librerías en Python y el entorno de desarrollo de Jupyter NoteBook de Anaconda. Presenta una interfaz muy clara y sencilla de usar a la hora de probar. A la hora de implementar los *scripts* definitivos que se conectarán se utilizará el editor de texto preferido por cada integrante del grupo, Sublime, Atom y Notepad++. Para la construcción de los mapas modelo del videojuego se utilizará el programa Tiled. Muy sencillo de manejar, permitiendo exportar los mapas en varios formatos. El sistema de control de versiones para el proyecto será Github. Las herramientas para comunicación entre el equipo serán Discord y Google Meets.

1.4. Plan de trabajo

La primera parte del trabajo consistirá en investigar que algoritmos se han usado hasta ahora para la generación de niveles en 2D. Tras tener unos modelos de referencia, empezaremos por implementar algoritmos básicos para ver su comportamiento ante pequeños ejemplos genéricos. Los modelos escogidos tras el estudio son el algoritmo de predicción de texto de n-gramas y las redes neuronales recursivas.

Posteriormente se hará una implementación más específica a nuestro tema concreto. El modelo de n-gramas y la primera implementación de la red neuronal recursiva se basaban en la predicción de texto debido a como hemos decidido abordar el problema de la generación de niveles. Por ello, las implementaciones sencillas se basarán en simples predicciones de palabras y la generación de pequeños fragmentos de texto.

Seguidamente estas implementaciones irán evolucionando para adecuarse a la generación de niveles de videojuegos. En este proceso surgirán problemas e incompatibilidades que se irán resolviendo modificando los algoritmos para adaptarlos a la generación de niveles, manteniendo su esencia de predicción de texto.

Por último, una vez repetido este proceso por cada modelo estudiado, se compararán dichos modelos y se pondrán a prueba en una aplicación. Esta aplicación se basará en la generación de mapas y de una pequeña adaptación del juego elegido para el ensayo de los algoritmos implementados. Como hemos mencionado anteriormente el juego elegido es el Super Mario Bros de la consola NES.

[3]Motor de videojuegos o *game engine* es un software que proporciona un conjunto de herramientas para la creación de un videojuego

Capítulo 1

Introduction

1.1. Motivation

In these years, artificial intelligence and machine learning have become a very popular technology worldwide, accessible to everyone and with many projections for the future. In addition, large companies such as Google, Apple, Amazon or Microsoft are investing time and money in research and development projects of applications based on artificial intelligence and automatic learning. Of course, in addition to large companies, many new companies appear and grow by focusing solely on this type of technology. This trend will only increase in the future, as the artificial intelligence industry is estimated to reach 118 billion dollars by 2025.

However, it's not all benefits, there's another, darker side to learning about machines. Stephen Hawking said: Artificial intelligence is probably the best or worst thing that can happen to mankind[1]. So far we have not seen the worst, and it is very difficult to predict how artificial intelligence will develop in the future, and whether it will ever become fully conscious. However, many of the problems that could result in making decisions without human reasoning, based on data alone, are already known.

But apart from the dangers, artificial intelligence has a lot of advantages that can make life easier for all of us. Furthermore, it is applicable to all economic sectors. Currently, sectors such as the automotive or health are benefiting from these automatic learning algorithms, and video games are not excluded. Many video games use machine learning during their development to make it easier and save time and costs for developers. Even during the game, artificial intelligence is used to give a more realistic feel, immersing the player even more into the game itself. The fact is that the expectations of artificial intelligence, as we have seen before, are not bad at all. More and more sectors are going to join the use of it, and thanks to this it will continue to evolve.

1.2. Goals

The general objective of this project is the research and analysis of the different forms of procedural content generation and the comparison of some of these algorithms. In order to test this we have considered generating maps or levels and ultimately implementing a small demo application with which to test the different

results of each model studied.

In order to achieve these objectives, a series of specific and significant milestones [1] have been established, which mark an important moment in the development of a project

1. Research on text classification and on the most used language models, such as Bag of Words or n-grams [2]
2. Study the n-gram model and implement our own n-gram model to understand how it works in practice.
3. Compare the behaviour of the n-gram model in situations with different corpus sizes, i.e. data for training.
4. Compare the performance of n-grams with a variable size to n-grams with a fixed size.
5. Study the different types of neural networks.
6. Study in depth the recursive neuronal networks and why they are optimal for the creation of contents, especially if they are artistic.
7. Compare the different possible recursive neural networks and their possible parameters.
8. Implement the different models previously studied, as well as their adaptation for the generation of the levels.
9. Compare the models mentioned above.
10. Create a demo application to show the different results generated.

To do that, we'll use as an example the Super Mario Bros. game, a 2D horizontal platform game, from the Nintendo Entertainment System, NES.

1.3. Methodology

In order to achieve the objectives mentioned above, Internet sources, scientific articles, previous studies and books will be researched, all of which are reflected in the bibliography and webgraphy. These resources will help in researching between the different existing models, as well as comparing them and their implementation. Once the theoretical research is completed, the different tools for the study of the chosen models will be studied. In order to see the strengths and weaknesses of each algorithm in its implementation, execution and results obtained
These models will then be tested. Algorithms will be implemented for the generation of levels and both the results and their execution times or implementation complexity will be compared.

[1] Punctual and significant event that marks an important moment in the development of a project

[2] n-grams is a sequence of continuous n-size words that appear in a text

Finally, an application will be developed where the chosen models can be exploited to the full. In this way, obtain results and establish conclusions about the procedural generation through automatic learning with respect to the generation of maps in video games.

For the creation of the application, Unity 2019.2.1f1, one of the best known and leading video game engines [3] in the industry, will be used as the development environment. This engine is available in the faculty of computer science of the Universidad Complutense de Madrid for the study along the degree of development of video games. This will make the development of our application more convenient for us.

The implementation of the automatic learning algorithms will be in the Python language, writing these in a separate script, which will later be connected to Unity. This will be done using Python libraries and Anaconda's Jupyter NoteBook development environment. It has a very clear and easy to use interface when testing.
When implementing the final scripts to be connected, the text editor preferred by each member of the group, Sublime, Atom and Notepad++ will be used.
For the construction of the model maps of the video game, the program Tiled will be used. Very easy to use, allowing the export of maps in various formats.
The version control system for the project will be Github.
The tools for communication between the team will be Discord and Google Meets.

1.4. Working plan

The first part of the work will be to investigate what algorithms have been used so far for the generation of 2D levels. After having some reference models, we will start by implementing basic algorithms to see their behavior in small generic examples. The models chosen after the study are the n-gram text prediction algorithm and the recursive neural networks.

A more specific implementation will be made later on to our specific theme. The n-gram model and the first implementation of the recursive neural network were based on text prediction because of how we decided to address the problem of level generation. Therefore, simple implementations will be based on simple word predictions and the generation of small text fragments.

These implementations will then evolve to suit the generation of video game levels. In this process problems and incompatibilities will arise and will be solved by modifying the algorithms to adapt them to the generation of levels, maintaining their essence of text prediction.

Finally, once this process is repeated for each model studied, these models will be compared and tested in an application. This application will be based on the generation of maps and a small adaptation of the game chosen for testing the implemented algorithms. As we mentioned earlier, the game chosen is Super Mario Bros. for the NES system.

[3] Video Game engine is software that provides a set of tools for the creation of a video game

Capítulo 2

Estado del Arte

2.1. Historia

El primer paso a lo que hoy conocemos como aprendizaje automático fue dado por McCulloch, un neurofísico, y Walter Pitts, un matemático. Decidieron, en 1943, crear un modelo de aprendizaje automático basado en un circuito eléctrico a partir del cual nacieron las redes neuronales. Sin embargo, Alan Turing ya había comenzado a estudiar el cerebro como una forma de ver el mundo computacional en 1936.

En 1949 Donald Hebb escribió un libro basándose en el aprendizaje psicológico. Fue el primero en explicar los procesos del aprendizaje desde un punto de vista psicológico. Hoy día sigue siendo el fundamento de muchas funciones de las redes neuronales.

Tras esto Alan Turing publica en 1950 un artículo titulado Çomputación e Inteligencia", en donde planteaba lo que hoy conocemos como la Prueba de Turing. Esta prueba consiste en que una máquina consiga mostrar un comportamiento inteligente de la misma forma que un ser humano.

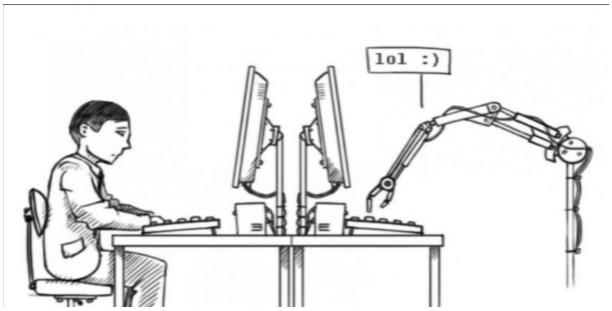

Figura 2.1: Test de turing [1]

En 1952, Arthur Samuel desarrolla un programa capaz de aprender. El programa simplemente jugaba a las damas y era capaz de aprender de los errores cometidos en la partida anterior. El programa poco a poco iba mejorando su habilidad para jugar a las damas gracias a la experiencia.

[1]Imagen sacada de `https://www.google.com/search?q=alan+turing+test&sxsrf=ALeKk0321GXw6mOF7h3IQ8j-uQn4nwQ39w:1592401178842&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjB5urU_IjqAhVSJBoKHSv4C6IQ_AUoAXoECBMQAw&biw=1920&bih=937#imgrc=OFx48S4x5QmKlM`

Pocos años más tarde Frank Rosenblatt comenzó a desarrollar en 1957 el perceptrón. Un perceptrón es la unidad más básica de una red neuronal, y podría llegar a considerarse una red neuronal como tal, es decir, una red de un solo perceptrón. Las redes neuronales de hoy día los siguen utilizando. Este modelo es capaz de reconocer caracteres y patrones. Sin embargo, tenía limitaciones como el problema de la función XOR, OR-exclusiva, incapaz de clasificar clases no separables linealmente.

En 1979, estudiantes de la diversidad de Stanford, diseñaron un carro capaz de cruzar una habitación llena de obstáculos moviéndose autónomamente. El carro tardó 5 horas, pero logró pasar la prueba con éxito.

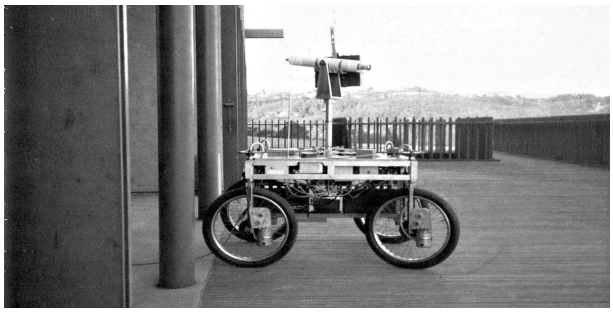

Figura 2.2: Carro de Stanford [2]

Gerald Dejong, en 1981, introduce el concepto de aprendizaje automático basado en la experiencia, haciendo que un computador analice información de entrenamiento y cree una regla general que le permita descartar información no importante. Pocos años después Terry Sejnowski inventa NetTalk, un programa capaz de aprender y pronunciar palabras como lo haría un niño en el año 1985.

A finales de 1980 y principios de 1990 apenas hubo grandes descubrimientos en el campo del aprendizaje automático. Sin embargo, en 1996, el ordenador Deep Blue, desarrollado por IBM consigue ganar a Gary Kaspárov una partida de ajedrez, aunque seguidamente Kásparov gano 3 partidas. En mayo de 1997, vuelven a enfrentarse Kaspárov y una versión mejorada del ordenador, el Deep Blue. Esta vez el ganador fue la computadora tras 6 partidas.

Figura 2.3: Kasparov jugando contra DeepBlue al ajedrez [3]

[2]Imagen sacada de `https://info.motorbit.com/mx/autos-autonomos-a-traves-del-tiempo/`

[3]Imagen sacada de `https://www.xataka.com/otros/deep-blue-el-ordenador-con-una-sola-mision-ganar-al-humano`

En el año 2006, Geoffrey Hinton presenta el concepto de *Deep Learning* o aprendizaje profundo. Gracias a esto se explicaron los algoritmos que permiten que los ordenadores distingan objetos y textos tanto en imágenes como en vídeos.

En 2010, el Kinect de Microsoft es capaz de reconocer 20 características corporales a una velocidad de 30 veces por segundo.

Figura 2.4: Kinect de Microsoft [4]

En 2011 el ordenador Watson de IBM vence a dos inteligentes concursantes en la tercera ronda del concurso estadounidense de preguntas y respuestas Jeopardy.

Entre 2011 y 2012 se crea Google Brain por Jeff Dean de Google y Andrew profesor de la Universidad de Stanford. El proyecto consistía en utilizar toda la infraestructura de Google para detectar patrones en vídeos e imágenes.

Los laboratorios Google X pasan a llamarse X a mediados de 2012. Estos desarrollaron un algoritmo capaz de buscar e identificar gatos en los vídeos de YouTube. También AlexNet gano la competición de ImageNet, lo que llevo al uso de GPUs y redes neuronales convolucionales en el aprendizaje automático. Además, crearon ReLU, una función de activación que mejoraba en gran medida las CNN.

Figura 2.5: Laboratorios Google X [5]

En 2014 un programa de ordenador logro convencer a más del 30 % de los jueces que era humano. Se trataba de un chatbot que respondía al nombre de Eugene

[4]Imagen sacada de `https://www.extremetech.com/extreme/160162-making-gesture-recognition-work-lessons-from-microsoft-kinect-and-leap/2`

[5]Imagen sacada de `https://www.google.com/search?q=google+x&tbm=isch&ved=2ahUKEwj9wdyDhYnqAhWv1uAKHSaeCG8Q2-cCegQIABAA&oq=google+x&gs_lcp=CgNpbWcQAzIECCMQJzICCAAyAggAMgIIADICCAAyAggAMgIIADICCAAyAggAMgIIADoFCAAQsQM6BwgAELEDEEM6BAgAEENQq7NCWIq-QmCAv0JoAHAAeACAAVOIAdcEkgEBOJgBAKABAaoBC2d3cy13aXotaW1n&sclient=img&ei=4CXqXv3RJ6-tgwemvKL4Bg&bih=937&biw=1920#imgrc=xHxk6uKAovXh4M`

Goostman, el programa fue capaz de convencer a los jueces que participaron en la prueba de que estaban chateando con un niño ucraniano de 13 años. Además, Facebook desarrolla DeepFace, un algoritmo de software que puede reconocer individuos en fotos al mismo nivel que los humanos.

En el 2015 ocurrieron una gran cantidad de avances en el campo del aprendizaje automático. Amazon lanza su propia plataforma de aprendizaje automático. Microsoft crea el kit de herramientas para el aprendizaje de máquinas, que permite la distribución eficiente de problemas de aprendizaje automático en múltiples computadoras. Google entrena un agente conversacional que no solo puede interactuar con humanos como un servicio de soporte técnico, sino también discutir la moralidad, expresar opiniones y responder preguntas generales basadas en hechos. Es fundada OpenAI, compañía de investigación de inteligencia artificial sin fines de lucro que tiene como objetivo promover y desarrollar inteligencia artificial de manera que beneficie a la humanidad. Entre sus fundadores se encuentra Elon Musk. Debido a estos grandes avances obtenidos en el área de la inteligencia artificial, más de 3000 investigadores de estas áreas firman una carta abierta advirtiendo del peligro de las armas autónomas.

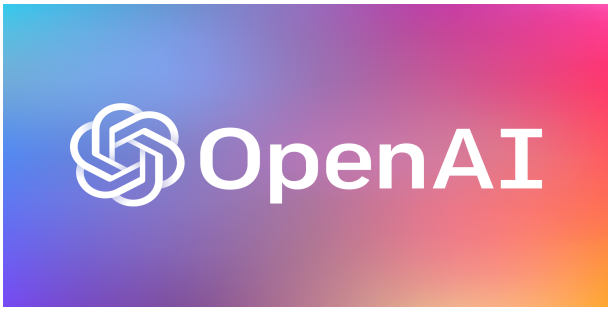

Figura 2.6: OpenAI [6]

En el año 2016 el algoritmo desarrollado por Google, DeepMind, logró ganar cinco juegos de cinco a un jugador profesional en el juego de mesa chino Go, que es considerado el juego de mesa más complejo del mundo.

Figura 2.7: DeepMind jugando al juego de mesa chino Go [7]

[6]Imagen sacada de `https://openai.com/`

[7]Imagen sacada de `https://primerfoton.wordpress.com/2016/03/15/alphago-la-inteligencia-artificial-que-derroto-al-campeon-de-go/`

Un algoritmo desarrollado por OpenAI en 2017 derrota a los mejores jugadores de Dota 2 [8] en partidos 1 contra 1.

Hoy día no hay prácticamente ningún sector que no se beneficie del aprendizaje automático por lo que existen una gran variedad de proyectos. Las tendencias en 2020 para proyectos de aprendizaje automático e inteligencia artificial son olvidar datos mediante el aprendizaje automático o "desaprendizaje automático", hay ocasiones en que es mucho más beneficioso que algunos datos sean olvidados por el sistema. Interpolación entre redes neuronales. La convergencia del internet de las cosas [9] y el aprendizaje automático.

La anterior sección de historia ha sido recopilada de la unión y resumen de los siguientes enlaces [2] [3] [4] [5] [6].

2.2. Actualidad

2.2.1. Herramientas para el desarrollo software

En la actualidad el desarrollo del aprendizaje automático en los distintos sectores económicos ha aumentado enormemente, así como la gran cantidad de herramientas para sus desarrolladores. Conocer las distintas herramientas y saber cuáles usar para el desarrollo puede ser de gran ayuda a la hora de desarrollar tu software. Algunas de las principales herramientas de aprendizaje automático hoy día son:

1. **Tensorflow**

   Desarrollada por el equipo de Google, Tensorflow dispone de un esquema flexible de herramientas, bibliotecas y recursos que permite a los investigadores y desarrolladores crear aplicaciones de aprendizaje automático. Entre sus características principales destaca la ayuda a la hora de construir y entrenar sus modelos. Ofrece un ciclo completo del aprendizaje automático en todas sus fases. Es un software de código abierto muy flexible que ofrece la posibilidad de ejecutarse tanto en CPU como GPU.

Figura 2.8: Tensorflow [10]

[8] Videojuego del género MOBA, en inglés *multiplayer online battle arena*
[9] *Internet of Things*, IoT, en inglés es la conexión de distintos objetos cotidianos con internet

2. **Google Cloud ML Engine**

Es una plataforma alojada donde los desarrolladores de aplicaciones de aprendizaje automático ejecutan modelos de aprendizaje automático de calidad óptima. Sus principales características son proporcionar modelos de entrenamiento, construcción, aprendizaje profundo y modelos predictivos. Los servicios de predicción y entrenamiento pueden ser usados independientemente uno del otro si así se desea. Es un software utilizado por las empresas, destacando su uso en la respuesta a los emails de clientes.

Figura 2.9: Google Cloud ML Engine [11]

3. **Amazon Machine Learning**

Es un software robusto de aprendizaje automático basado en la nube, que puede ser usado por cualquier desarrollador web o móvil. Sus principales características son los asistentes y herramientas visuales que proporciona. Admite tres tipos de modelos, es decir, clasificación multiclase, clasificación binaria y regresión. Permite a los usuarios crear objetos desde bases de datos MySQL y desde datos almacenados en Amazon Redshift.

Figura 2.10: AWS Machine Learning [12]

[10]Imagen sacada de `https://towardsdatascience.com/10-most-popular-machine-learning-software-tools-in-2019-678b80643ceb`

[11]Imagen sacada de `https://towardsdatascience.com/10-most-popular-machine-learning-software-tools-in-2019-678b80643ceb`

[12]Imagen sacada de `https://www.e4developer.com/2018/11/11/making-your-machine-learning-idea-real-with-aws/`

4. **Accord.Net**

   Aprendizaje automático .Net que se combina con bibliotecas de procesamiento de imágenes y audio escritas en C♯. Consiste en un conjunto de bibliotecas para el reconocimiento de patrones, procesamiento de datos estadísticos, álgebra lineal.

Aparte de estas existen muchas más herramientas para el desarrollo de aplicaciones de aprendizaje automático [7]. Estas herramientas de desarrollo suelen ir acompañadas de muchas otras para poder interpretar, analizar y comparar resultados. Algunos ejemplos de herramientas que ayudan al aprendizaje automático son Weka o matplotlib[8]. Weka es un software de aprendizaje automático en Java que tiene una amplia gama de algoritmos de aprendizaje automático para tareas de minería de datos. Matplotlib es una biblioteca es una biblioteca para crear visualizaciones estáticas, animadas e interactivas en Python. Es útil para una visualización de distintos tipos de gráficas ya sean en 2D o 3D.

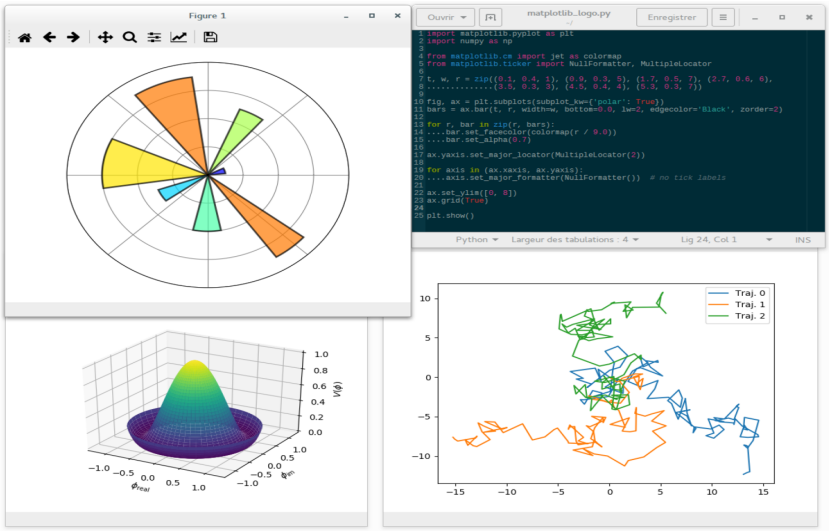

Figura 2.11: Matplotlib [13]

2.2.2. Aplicación en los sectores económicos

La mayoría de las industrias que trabajan con una gran cantidad de datos han reconocido el valor del aprendizaje automático. Coleccionando ideas de esta gran cantidad de datos, la industria es capaz de trabajar de una forma más eficaz para obtener ventaja ante sus competidores. Algunos de los sectores[9] que más se benefician del hecho de aplicar el aprendizaje automático a su sector son:

1. **La industria de la salud**

   Desde principios del 2013 el aprendizaje automático se ha utilizado en el mundo de la medicina. Un ejemplo fue Google DeepMind que después de ganar la serie de juegos contra el mejor jugador del mundo de GO, el equipo de Google decidió apoyar proyectos de medicina con su tecnología. Además, muchas de

[13] Imagen sacada de `https://en.wikipedia.org/wiki/Matplotlib`

las nuevas empresas de la industria del aprendizaje automático están ayudando a la atención médica. Algunos ejemplos de lo que se puede conseguir con el uso del aprendizaje automático en la medicina son:

a) **El diagnóstico a partir de imágenes médicas**. Gracias al aprendizaje profundo automático y al tratamiento de imágenes Microsoft ha desarrollado InnerEye, una herramienta que está actualmente trabajando en dar diagnósticos a partir de imágenes. Sin embargo, se sabe que las aplicaciones de aprendizaje profundo tienen una capacidad explicativa limitada. Es decir, este sistema de aprendizaje automático no puede explicar cómo llegó a sus predicciones, incluso cuando son correctas.

b) **Consultas de tratamiento y sugerencias**. El diagnóstico es un proceso muy complicado que involucra una gran cantidad de factores que las máquinas no pueden clasificar. Sin embargo, hay pocas dudas de que una máquina podría ayudar a los médicos a tomar las consideraciones correctas en el diagnóstico y el tratamiento de sus pacientes.

c) **El descubrimiento de medicamentos**. Si bien gran parte de la industria de la salud es un cúmulo de leyes de varias partes interesadas, el descubrimiento de medicamentos se destaca como algo relativamente sencillo para el aprendizaje automático[10].

Figura 2.12: InnerEye [14]

2. **La industria financiera**

El uso del aprendizaje automático en el mundo de las finanzas es una de las mejores cosas que le podría haber ocurrido a este sector. Y es que el uso de este tipo de algoritmos permite predecir como va a evolucionar el mercado lo que supone una gran ventaja en este ámbito. Y es que esta clase de algoritmos se está desarrollando rápidamente debido al interés que acarrean. Además, existen muchas opciones para su uso, algunas de las más relevantes son:

[14]Imagen sacada de `https://format-com-cld-res.cloudinary.com/image/private/s--2YZVQMUu--/c_crop,h_1066,w_1600,x_0,y_0/c_fill,g_center,w_900/fl_keep_iptc.progressive.apng/v1/e8ffbcac8dbd18b012e29f9655f2fc0b/UI_02_THUMB_innereye.png`

a) **Evaluación de solvencia de crédito**. La Inteligencia Artificial ayuda a los bancos a emitir créditos con mayor confianza a quienes aprueban las verificaciones del sistema. Para esto, los programas y algoritmos analizan toda la información disponible, estudian su historial de crédito, los cambios en su nivel de salarios y, sobre esta base, determinan la confiabilidad del cliente y la seguridad del préstamo.

b) **Toma de decisiones**. Esta es una tarea global que se resuelve con éxito mediante la introducción de inteligencia artificial y aprendizaje automático en los servicios financieros. Cuando un algoritmo puede analizar todos los datos estructurados y no estructurados disponibles, tanto internos como externos, solicitudes de clientes y sus acciones en las redes sociales, una institución financiera puede descubrir tendencias útiles y peligrosas. Ayuda a evaluar los niveles de riesgo y permite a las personas tomar las decisiones más informadas.

c) **Protección contra el fraude**. Los bancos y los sistemas de pago ya han desarrollado modelos para identificar y bloquear la mayoría de las transacciones fraudulentas. Estos modelos se basan en el historial de transacciones del cliente, así como en el comportamiento del cliente en Internet. Los sistemas basados en inteligencia artificial que detectan fraudes en línea se han desarrollado a partir de tecnologías de Big Data[11].

3. **La industria del transporte**

Los casos de uso de inteligencia artificial en el sector del transporte[12] justifican por qué el mercado está experimentando un aumento al alza y por qué las empresas deberían adoptar la tecnología. Algunos ejemplos de los proyectos en este sector son:

a) **Vehículos sin conductor**. Una de las aplicaciones más innovadoras y ambiciosas de la inteligencia artificial son los vehículos autónomos. Aunque las personas se mostraron escépticas sobre esta tecnología durante sus etapas de desarrollo, los vehículos sin conductor ya ingresaron al sector del transporte. Los taxis autónomos ya comenzaron a operar en Tokio. Sin embargo, por razones de seguridad el conductor se sienta en el automóvil para tomar el control del taxi en caso de emergencia. Del mismo modo, Estados Unidos está adoptando camiones autónomos para obtener beneficios. Por ahora, la mayoría de las compañías todavía están ejecutando sus proyectos piloto, esforzándose por hacer que los vehículos autónomos sean perfectos y seguros para los pasajeros. A medida que esta tecnología evoluciona, los vehículos autónomos ganarán una gran confianza y se convertirán en la corriente principal en el ámbito del consumidor.

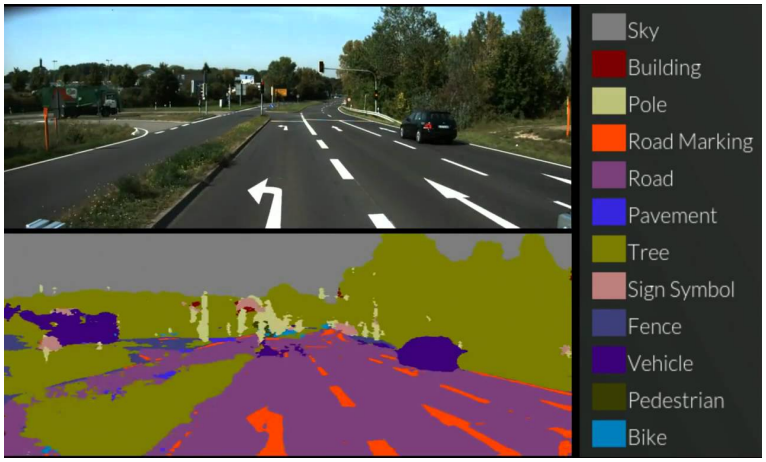

Figura 2.13: Aprendizaje automático de un coche autónomo [15]

b) **La gestión del tráfico**. Otro problema de transporte al que las personas enfrentan a diario es la congestión del tráfico. La inteligencia artificial puede ayudar a resolver esta clase de problemas. Los sensores y las cámaras integradas en todas partes en las carreteras recogen una gran cantidad de información acerca del tráfico. Estos datos se analizan y se obtiene un patrón del tráfico. De esta forma, la inteligencia artificial se puede utilizar para, no solo reducir el tráfico no deseado, sino también, para mejorar la seguridad vial y reducir los tiempos de espera.

c) **Predicciones de retraso en el transporte aéreo**. Otro problema hoy en día son los retrasos en los vuelos. Los costos estimados debido a retrasos en los vuelos son enormes. Junto con la pérdida financiera, el retraso de los vuelos tiene un impacto negativo en la experiencia de vuelo de los pasajeros. La experiencia negativa durante el vuelo puede dar como resultado una mayor tasa de rotación de clientes entre las distintas empresas de transporte aéreo. La inteligencia artificial puede predecir desde el mal tiempo hasta algún problema técnico que pueden retrasar los vuelos, es importante actualizar los detalles del vuelo a los pasajeros con anticipación para eliminar los tiempos de espera innecesarios. Con la ayuda de sistemas de inteligencia artificial y aprendizaje automático se procesarán datos de aviones en tiempo real, registros históricos y también la información meteorológica. El cálculo sobre el terreno ayudará a revelar patrones ocultos, lo que puede ayudar a la industria del transporte aéreo a obtener información útil sobre otras posibilidades que pueden causar retrasos y cancelaciones de vuelos.

[15] Imagen sacada de `https://www.youtube.com/watch?v=kMMbW96nMW8`

4. **La industria de agricultura**

   Gracias a la inteligencia artificial y el aprendizaje automático, los productores de este sector primario pueden acceder a datos y herramientas de análisis, lo que permite mejores decisiones, mejores eficiencias y menor desperdicio en la producción de alimentos y biocombustibles. Todo mientras se minimizan las consecuencias ambientales negativas.

   La agricultura digital aporta una mayor precisión a la producción de cultivos al apoyar decisiones clave de gestión con conocimientos basados en datos. Los agricultores toman cada año cientos de decisiones complejas que afectan a su sostenibilidad y rentabilidad empresarial y que implican un gran riesgo. Utilizando sensores en el campo junto con aplicaciones de aprendizaje automático se pueden predecir los rendimientos de cosecha, evaluar la calidad del cultivo o identificar especies de plantas.

   Además, gracias al procesamiento de imágenes basado en aprendizaje automático los agricultores pueden confiar en herramientas digitales para reconocer especies de malezas y determinar qué cultivos son saludables y cuáles están infestados con enfermedades. La capacidad de identificar estas plagas hace posible entrenar dispositivos, como robots, para extraer malezas de los campos, protegiendo el medio ambiente del daño causado por el uso de pesticidas. Incluso pueden evaluar los cultivos para detectar enfermedades, proporcionar un diagnóstico preciso de la enfermedad y recomendar un tratamiento óptimo[13].

5. **La industria del videojuego**

   El aprendizaje automático no solo se utiliza para el mundo empresarial como pueden ser las subidas o bajadas en bolsa de una determinada empresa, o la estimación de los posibles atrasos de un avión y los costes que ellos implica. El mundo del ocio también se aprovecha de esta clase de algoritmos para sacar rentabilidad a sus productos. Un claro ejemplo, es el mundo de los videojuegos[14][15]. Y es que esta industria explota al máximo el aprendizaje automático para toda clase de actividades. Desde publicidad y marketing de un videojuego en su campaña de salida, hasta las fases más tempranas de su desarrollo, incluyendo entre estas los algoritmos que le afectan in game . Algunos de los ejemplos más relevantes son:

   a) **Personajes interactivos**. Los personajes interactivos de los videojuegos están programados de tal forma que siguen un guion y responden a situaciones fijas, es decir, realmente está todo programado desde el principio y realizan unas acciones fijas. Sin embargo, con la incorporación del aprendizaje automático, estos personajes pueden adaptarse según el entorno y el estilo de juego del jugador. Por ejemplo, en el juego Metal Gear Solid 5, si un jugador usa continuamente la técnica de disparos a la cabeza en el juego, los personajes se adaptan a él y comienzan a usar

[15] Termino que hace referencia a algo que ocurre dentro del juego o algo en lo que tomará parte el jugador

cascos para evitar ser golpeados.

b) **Modelado de sistemas complejos**. Para crear un mundo más inmersivo, los desarrolladores usan algoritmos de aprendizaje automático para crear modelos predictivos.

c) **Interacciones realistas**. Los juegos de mundo abierto requieren que el jugador interactúe con su entorno. Con el auge del procesamiento del lenguaje natural, el jugador puede interactuar con otros personajes de una manera más realista. Por ejemplo, en Red Dead Redemption 2, los personajes del juego interactúan con el jugador de acuerdo con el nivel de honor de nuestro personaje.

d) **Creación dinámica del universo**. Otro caso de los juegos de mundo abierto es la creación de estos mismos. Esta creación del universo toma mucho tiempo para ser perfecta y consiste en tareas repetitivas y pequeñas. Con ayuda del aprendizaje automático el tiempo que lleva este proceso se ha reducido.

e) **Juegos móviles**. Los juegos móviles contribuyen con aproximadamente el 50 % de los ingresos generados por los videojuegos. El alcance de estos juegos es limitado debido al hardware de los teléfonos. Pero esta situación ha comenzado a cambiar a medida que ahora se están instalando componentes hardware más adecuados al procesamiento de los algoritmos de aprendizaje automático con relación al sector de los videojuegos. Este cambio supondrá una mejora en los juegos para móviles de cara al futuro.

Figura 2.14: Metal Gear Solid 5 y Red Dead Redemption 2 [16]

[16]Imagenes sacadas de `https://www.journaldugeek.com/2015/06/10/metal-gear-solid-v-5-phantom-pain-preview-apercu-ps4-xbox-one-pc/` y `https://www.mundoplayers.com/red-dead-redemption-2-honor-guide-como-adquirir-honor-positivo-o-negativo-beneficios-y-mas/`

2.2.3. Aplicaciones cotidianas

1. **Asistentes personales**

Siri, Alexa o Google Now son algunos de los ejemplos más populares de asistentes personales. Estos asistentes ayudan a encontrar información cuando se les pregunta por voz. Simplemente con realizar preguntas como por ejemplo "¿Qué tiempo hace hoy?", "¿Cómo tengo la agenda de mañana?". Para responder, su asistente personal busca la información, recuerda consultas relacionadas o usa otros recursos del móvil como aplicaciones para recopilar información. Incluso puede instruir a los asistentes para realizar ciertas tareas como configurar una alarma el día siguiente, añadir tareas o recordatorios en el calendario o la agenda e incluso pedir que llame a algún contacto.

El aprendizaje automático es una parte importante de estos asistentes personales, ya que recopilan la información de todas las anteriores veces que los has utilizado. Después, este conjunto de datos se utiliza para generar resultados que se adaptan a sus preferencias. Los asistentes virtuales están integrados en una variedad de plataformas como los smartphones, aplicaciones móviles, o los altavoces inteligentes.

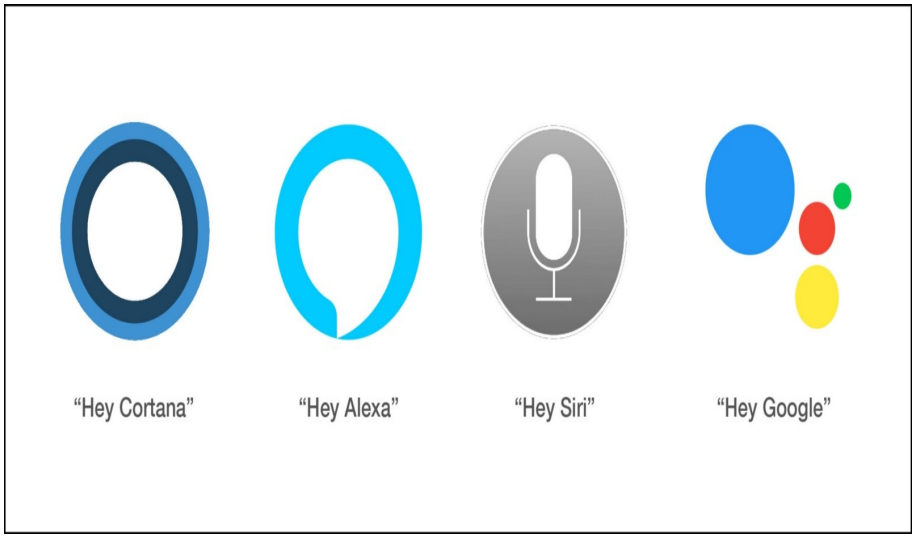

Figura 2.15: Aprendizaje automático en asistentes personales [17]

2. **Aplicaciones de transporte**

Cuando utilizamos servicios de navegación GPS, nuestras ubicaciones y velocidades actuales se guardan en un servidor central para administrar el tráfico. Estos datos se utilizan para construir un mapa del tráfico actual. Esto ayuda a prevenir el tráfico lento y hace un análisis de su congestión. Esta práctica es aprovechada por las redes de transporte y sus aplicaciones. Al reservar un taxi o un VTC, la aplicación calcula el precio del viaje.
Además, el aprendizaje automático ayuda a definir las horas de aumento de

[17] Imagen sacada de `https://elandroidelibre.elespanol.com/2019/07/el-gran-problema-al-que-se-enfrentan-los-asistentes-virtuales.html`

precios al predecir la demanda del piloto, a calcular la ruta más eficiente a un lugar para evitar un consumo de combustible excesivo o la perdida de tiempo en atascos. En todo estas operaciones, el aprendizaje automático está jugando un papel muy importante. Aplicaciones como Uber o Cabify son un ejemplo de esto.

3. **Redes sociales**

Tanto personalizar sus noticias como una mejor selección de anuncios que mostrarle, las redes sociales están utilizando el aprendizaje automático para sus propios beneficios. Este tipo de aplicaciones son de las que más datos recaudan de las personas, que cosas te gustan, cuales están de moda o que forma de vida llevas. Y esta información es muy relevante para toda clase de empresas. Algunos ejemplos de esto son:

a) **Las recomendaciones de personas que quizás conozcas**. Tanto Instagram, Twitter, Facebook y todas estas redes sociales están continuamente analizando los amigos con los que se conecta, los perfiles que visita con frecuencia, sus intereses, el lugar de trabajo, etc. En base a esta gran cantidad de información, se sugiere una lista de usuarios.

b) **Reconocimiento facial**. Subir una foto con un amigo a Facebook y este reconoce instantáneamente a ese amigo. Facebook verifica las poses y proyecciones en la imagen, observa las características únicas y luego las compara con las personas en tu lista de amigos. Todo este proceso lo realiza una aplicación de aprendizaje automático.

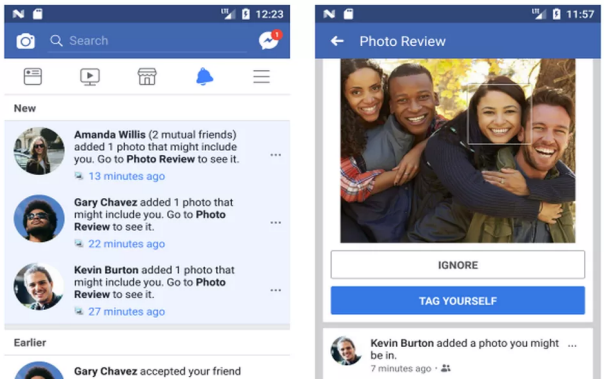

Figura 2.16: Reconocimiento facial de Facebook [18]

c) **Pines similares**. El análisis de imágenes y vídeos es un proceso basado en el aprendizaje automático. Pinterest utiliza esta forma de aprendizaje automático para identificar los objetos en las imágenes y recomendar similares.

[18]Imagen sacada de `https://www.lavanguardia.com/tecnologia/20171220/433775705940/reconocimiento-facial-facebook-fotos-etiquetas.html`

4. **Correos de spam y Malware**

   Existen distintos tipos de filtrado de spam que utilizan los correos electrónicos. Para asegurarse de que estos filtros de correo no deseado se actualicen continuamente, estos funcionan mediante aprendizaje automático. Ya que, si se realiza el filtrado de spam basado en reglas, este no puede rastrear los últimos trucos adoptados por los *spammers*. Las herramientas *antispam* vienen con las aplicaciones de correo electrónico por defecto.

   Una gran cantidad de *malwares* se detectan todos los días y muchas partes de código son entre un 90 y un 98 % similar a sus versiones anteriores. Los programas de seguridad que se ayudan del aprendizaje automático detectan fácilmente el nuevo programa maligno con una variación del 2 al 10 % y ofrecen protección contra ellos.

2.3. En desarrollo

Como hemos visto anteriormente la inteligencia artificial está presente hoy día en todas las industrias que nos rodean. De hecho, alrededor del 77 % de las personas en el mundo ya usan la inteligencia artificial de alguna forma. Además, la inteligencia artificial no solo afecta a la industria de la tecnología, sino a todos y cada uno de los sectores de la economía. Y con las principales compañías como Google, Facebook, Microsoft o Amazon trabajando en todas sus posibles aplicaciones, no hay duda de que habrá una gran evolución en los próximos años. Adobe incluso predice que el 20 % de todas las tecnologías emergentes tendrán algunas bases de inteligencia artificial para 2021[16]. Algunos ejemplos de proyectos y tendencias que ocurren actualmente son:

1. **Inteligencia artificial e internet de las cosas**

   Estas dos áreas de la tecnología se combinan bastante bien debido a que los dispositivos IoT crean una gran cantidad de información y, por otro lado, los algoritmos de inteligencia artificial requieren de datos antes de sacar conclusiones. Por lo tanto, los datos recopilados por IoT pueden ser utilizados por los algoritmos de aprendizaje automático para crear resultados.
   La mayoría de estos dispositivos son electrodomésticos del hogar. Estos electrodomésticos inteligentes para el hogar se están volviendo cada vez más populares. De hecho, un gran porcentaje de todos los hogares en los EE. UU. podrían convertirse en hogares inteligentes para 2021. No solo los hogares se benefician de estos aparatos, las empresas también están adoptando cada vez más dispositivos inteligentes, ya que reducen los costes. Nest , propiedad de Google, es una de las más conocidas en este mercado, ya que produce una gran variedad de productos inteligentes como termostatos, sistemas de alarma, timbres, etc.

[18]El internet de las cosas (*Internet of things*) consiste en la conexión a través de internet, de dispositivos tecnológicos que forman parte de nuestro día a día, para el envío y recepción de datos

[18]Google Nest es una marca de Google LLC que se utiliza para comercializar productos de domótica.

La integración de la inteligencia artificial e internet de las cosas también ha llevado a las ciudades como Nueva York a ser cada vez más inteligentes. Hay instalaciones para monitorizar el uso del agua y contenedores inteligentes que funcionan con energía solar que pueden controlar los niveles de basura y programar la recolección de desechos.

2. **Aprendizaje personalizado para los estudiantes**

Muchos de los objetivos del aprendizaje automático y de la inteligencia artificial es la personalización para cada persona. Desde recomendaciones de Netflix o YouTube, hasta la publicidad en muchas de nuestras aplicaciones, todas estas recomendaciones se basan en nuestras preferencias personales. Con esta misma idea, la inteligencia artificial podría ayudar a los profesores a abordar las necesidades individuales de cada estudiante desde cero, en lugar de identificarlos a lo largo de un semestre, un curso, o peor aún, al final de su etapa educativa. Esto les da a los profesores suficiente tiempo para ayudar a los estudiantes. Brightspace Insights predice y pronostica a los estudiantes en riesgo, para que los instructores puedan ayudarlos mientras aprenden. Al tener acceso a los datos de los estudiantes, los maestros pueden diseñar planes de aprendizaje personalizados para sus alumnos. En lugar de utilizar un enfoque único y común para toda una clase, de esta forma se trabajaría las fortalezas individuales de cada estudiante. El aprendizaje automático es capaz de identificar patrones que normalmente no reconocemos, esto podría ayudar a determinar qué estudiantes se sienten cómodos estudiando las distintas materias, así como su forma de aprenderlas, desde una forma puramente teórica a una más visual.

3. **Inteligencia artificial y computación en la nube**

La inteligencia artificial y la computación en la nube pueden revolucionar totalmente el mercado actual. Es obvio que la inteligencia artificial tiene una gran potencia, pero su integración también requiere empleados experimentados y una gran infraestructura. Cloud Computing puede proporcionar una ayuda inmensa. Incluso si las empresas no tienen una fuerte base informática o acceso a muchos datos, aún pueden aprovecharse de sus beneficios.
La inteligencia artificial también se puede usar para controlar y solucionar problemas en la nube. Se puede usar para automatizar el flujo de trabajo básico de los sistemas de computación en la nube pública o para crear formas de trabajo mucho más eficientes. Actualmente, los líderes del sector tecnológico incorporan inteligencia artificial en sus servicios en la nube como son Amazon Web Service (AWS), Google, IBM, Alibaba u Oracle. Se espera que crezcan aún más en el futuro gracias a la popularidad y el crecimiento que están teniendo estas áreas de la tecnología.

4. **Inteligencia artificial en ciberseguridad**

Hablamos de la mezcla de dos campos de la tecnología posiblemente en uno de sus mejores momentos respectivamente. Si bien la ciberseguridad es el dominio de la industria de tecnología, la inteligencia artificial tampoco se queda muy atrás. La adición de la inteligencia artificial a la ciberseguridad puede mejorar el análisis, la comprensión y la prevención de los ciberataques. También puede mejorar las medidas de seguridad cibernética de las empresas para que estén protegidas. Sin embargo, es muy difícil de implementar en todas las aplicaciones. Además, la inteligencia artificial es un arma de doble filo, ya que puede usarse para mejorar los ciberataques. A pesar de todo esto, la inteligencia artificial será un elemento crítico de ciberseguridad en el futuro.
Por lo tanto, las empresas pueden comenzar con la integración de algoritmos de aprendizaje automático en sus protocolos de Ciberseguridad existentes. Esto se puede hacer utilizando análisis de datos para detectar amenazas o actividades maliciosas.

5. **Conducción autónoma**

Como hemos mencionado anteriormente la industria del transporte y de la automoción apuesta por la conducción autónoma de los vehículos, además esta, se está popularizando en la sociedad. Algunos países de como EE. UU. o Tokio han puesto en marcha algunos modelos. Sin embargo, es una realidad que aún existen reticencias y con motivos justificados. Google está tratando de crear automóviles sin conductor con su proyecto Waymo , lo que podría reducir las muertes en la carretera, disminuir la congestión del tráfico y proporcionar una solución si no se puede conducir.

Los automóviles Waymo tienen LiDAR, radar y cámaras colocados a su alrededor que les permite detectar todos los objetos en 360 grados alrededor del automóvil, incluso si es de noche. Google comenzó el proyecto de conducción autónoma de Waymo en 2009 cuando modificaron un Toyota Prius y practicaron la conducción autónoma a más de 100000 millas en la vía pública. Después de eso, desarrollaron un prototipo de automóvil totalmente autónomo llamado Firefly que no tenía volante ni pedales. Actualmente, Waymo tiene un juicio público en curso en Phoenix con el objetivo final de hacer que la conducción sea segura con cero muertes en el futuro.

[18]Waymo, antes conocida como Google self-driving car project, es una empresa encargada del desarrollo de vehículos autónomos perteneciente al conglomerado Alphabet Inc.

Capítulo 3

Aprendizaje Automático

3.1. Definición

El aprendizaje automático o aprendizaje automatizado (*Machine Learning*)[17] es una disciplina del campo de la inteligencia artificial. Consiste en el desarrollo de algoritmos "inteligentes" capaces de .ªprender.ª partir de datos y gracias a estos poder hacer predicciones. Lo que se denomina aprendizaje no es más que la capacidad del sistema para identificar una serie de patrones dentro de los datos recibidos y gracias a esto adquirir unas mejoras. Cuanta más información más adecuadas al contexto serán las predicciones. Es decir, la máquina no aprende por si sola, sino gracias a sus algoritmos y heurísticas, que modifican los datos recibidos dando lugar a escenarios futuros o a decisiones.

En muchas ocasiones el aprendizaje automático se solapa con el campo de la estadística, ya que las dos disciplinas se basan en analizar datos. Esto puede ocasionar en algunas dudas de si una aplicación utiliza algoritmos basados en operaciones probabilísticas o si realmente hace uso del aprendizaje automático. Y es que como hemos dicho antes, muchos algoritmos de aprendizaje automático comparten aspectos con la estadística inferencial [1]. No obstante, el aprendizaje automático tiene en cuenta más factores, por ejemplo la complejidad computacional[2].

Los algoritmos usados en el aprendizaje automático realizan muchas tareas por su propia cuenta. Obtienen datos propios a partir de cálculos y cuantos más datos obtienen más "aprenden", es decir, más precisos serán sus resultados. Cada dato puede modificar en mayor o menor medida dicho algoritmo, por lo tanto, a mayor cantidad de datos, mayor complejidad y efectividad del cálculo. La clave reside en tomar decisiones en base a los datos recibidos. Un sistema de aprendizaje automático necesita contar con datos relevantes suficientes para poder suministrar repuestas coherentes y válidas. El aprendizaje automático tiene una amplia gama de aplicaciones en distintos campos del mundo como pueden ser diagnósticos médicos, detección de fraudes, clasificación de secuencias de ADN, estudios de mercado o videojuegos.

[1]Parte de la estadística que comprende los métodos que mediante la inducción determina propiedades de una población a partir de una parte de esta.

[2]Cantidad de recursos necesarios para ejecutar dicho algoritmo, principalmente enfocados al tiempo y la memoria utilizada.

El aprendizaje automático tiene como resultado la creación de un modelo[18] para resolver un problema. Estos modelos se pueden distinguir en:

- **Modelos lógicos**: usan una lógica para dividir una parte del espacio en segmentos y así construir grupos. Esta lógica es una función que devuelve un valor bool. Una vez usada esta lógica, los datos quedan divididos en distintos grupos y se intenta resolver el problema.

- **Modelos geométricos**: en los modelos lógicos el espacio se divide en distintos grupos, sin embargo, dos partes de dicha división pueden ser similares, por ello existen modelos que tienen en cuanta esta similitud considerando la geometría de una parte del espacio. Las características de los modelos pueden ser tratadas como puntos en un plano (X, Y) o incluso en un espacio tridimensional (X, Y, Z). Esto es lo que se conoce como modelos lineales. Por otra parte, podemos usar la geometría para representar la distancia entre la similitud, es decir, si dos puntos están muy cerca es que guardan valores muy similares, esto se conoce como modelos basados en la distancia.

- **Modelos probabilísticas**: usan las probabilidades para la clasificación de nuevas entidades. Los modelos probabilísticos ven a los datos y las variables resultantes como valores aleatorios. El proceso de manipular el nivel de incertidumbre de estos datos. Hay dos tipos de modelos probabilísticos. Los predictivos, que usan la idea de una probabilidad condicional P (Y|X) en la que Y puede ser predicha a partir de X. Y los generativos, que estiman una probabilidad conjunta P (Y, X). Una vez se conoce la distribución conjunta podemos derivar cualquier distribución condicional que involucre a las variables.

3.2. Tipos de Algoritmos

Los algoritmos de aprendizaje automático se dividen en tres categorías[19], aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. De las cuales las más comunes son la supervisada y la no supervisada.

3.2.1. Algoritmos supervisados

Los algoritmos supervisados cuentan con una información previa basada en etiquetas asociadas a unos datos. Se entrena al sistema proporcionándole estos datos. Una vez que se le ha entrenado lo suficiente, es decir, se le han proporcionado suficientes datos, podremos introducirle nuevos datos sin necesidad de etiquetas[20].

Este sistema se conoce como clasificación. Otro método es predecir un valor continuo, utilizando distintos parámetros que, combinados de distintas formas, predicen el resultado. Es lo que se conoce como regresión.

1. **Clasificación y Regresión**

Clasificación es la acción de identificar a que categoría de un conjunto discreto pertenece un elemento de un conjunto de ejemplos, mientras que la regresión predice un número. Un ejemplo de clasificación es el correo, los correos pueden clasificarse como "spam" o como "legítimos", mientras que un ejemplo de regresión puede ser el precio de una casa en función de sus características.

La regresión es un modelo estadístico que permite establecer un patrón entre los datos de entrada con los resultados. Hay algoritmos muy comunes asociados respectivamente a la clasificación y la regresión, la Regresión Logística y la Regresión Lineal.

En estadística la regresión lineal se utiliza para representar la relación que existe entre un conjunto de datos explicativos X (variables de entrada) con un conjunto de datos dependiente Y (resultados). En su versión más sencilla lo que se hará será dibujar una recta que nos indicará la tendencia de ese conjunto de datos.

Para obtener automáticamente esa recta de tendencia lo que hacemos es medir el error con respecto a los datos de entrada y el resultado. El algoritmo debe minimizar el coste de la función y los coeficientes corresponden a la recta óptima. Hay distintos métodos para minimizar el coste, el más común es la llamada Ecuación Normal, que nos dará un resultado directo.

Dependiendo de la cantidad de valores de entradas podemos hablar de regresión lineal simple, como hemos estado haciendo anteriormente, o de regresión lineal múltiple. La regresión lineal múltiple sigue los mismos patrones que la simple, buscar la tendencia de los valores. Sin embargo, al tener más cantidad de valores de entrada el resultado no será una recta. Por ejemplo, en caso de tener tres variables de entrada el resultado será un plano bidimensional.

Al igual que en la regresión lineal, en la regresión logística[21] podemos tener dos posibles estados, SI o NO, binario, o un número finito de etiquetas como pueden ser el 1, 2, 3, 4, ..., en el reconocimiento de números, múltiple. De hecho, la regresión logística es una red neuronal de una solo neurona.

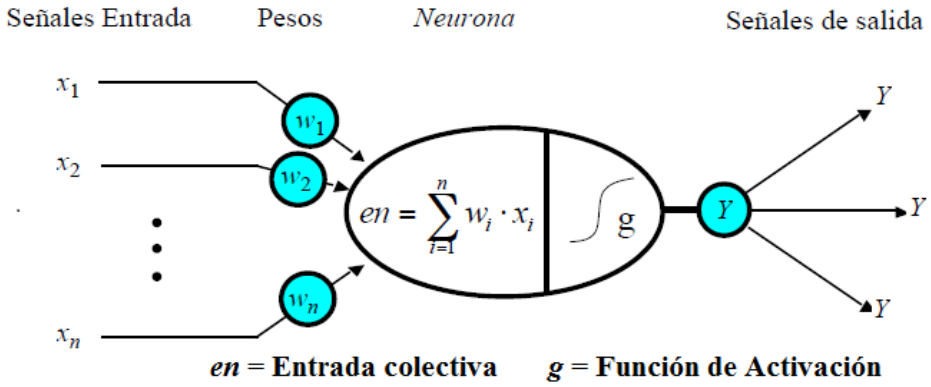

Figura 3.1: Regresión Logísitca perceptrón [3]

[3]Imagen sacada de `http://samuelabad1991.blogspot.com/2014/02/analisis-con-redes-neuronales-neural.html`

Tenemos distintos datos de entrada a nuestro problema y queremos ver la probabilidad de que ocurra una cosa u otra. Para ello la regresión logística tiene dos partes, una combinación lineal y después una aplicación de la función logística. De esta forma todas las entradas de datos se combinan linealmente y posteriormente se le aplica la función logística o sigmoide al resultado.

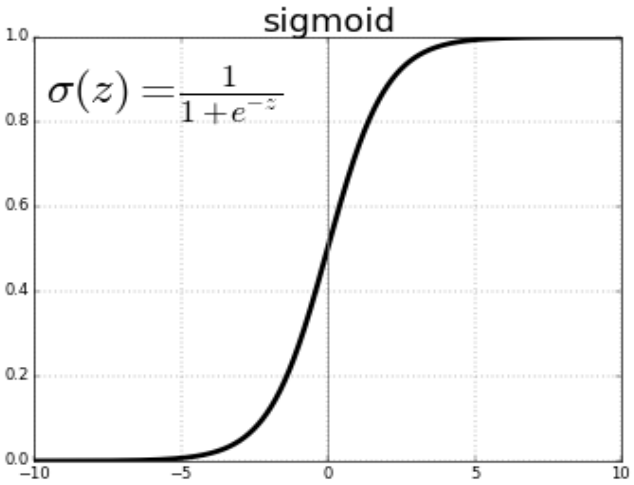

Figura 3.2: Sigmoide [4]

Las características de la función sigmoide son:

- Está acotada entre 0 y 1

- Podemos interpretar sus resultados como probabilidades

- Para problemas de clasificación binarios, podemos establecer un valor a partir del cual los resultados corresponde al 0 o al 1.

En el aprendizaje automático existen algoritmos clasificadores que pueden trabajar con múltiples clases, existen otros que no pueden como es el caso de la regresión logística. Sin embargo, existen técnicas que se pueden utilizar para la regresión logística múltiple[22], como, por ejemplo:

- **Uno contra todos**. En esta técnica se ha de entrenar tantos clasificadores binarios como clases existan. Cada modelo predice la probabilidad de que el resultado pertenezca a una clase o no. A la hora de la predicción se ejecutan todos los clasificadores y se elige el que tenga mayor probabilidad.
- **Uno contra uno**. En esta técnica se crean tantos modelos con pares de posibles resultados existan. Un clasificador decidirá solamente entre dos posibles resultados. Al igual que la técnica anterior se ejecutan todos los clasificadores y se selecciona el de mayor probabilidad.

2. **Evaluación de exactitud**

Normalmente los algoritmos de aprendizaje supervisando entrenan modelos de clasificación o de regresión usando ejemplos de entrenamiento. Estos ejemplos

[4]Imagen sacada de `https://ml4a.github.io/ml4a/es/neural_networks/`

de entrenamiento se basan en parejas de datos de entrada y resultados a esos datos de entrada. La evaluación de exactitud es una forma de medir la precisión que existe en el sistema que nosotros hemos entrenado. La exactitud de un sistema es la proximidad de los resultados de una medición al valor correcto. La precisión de un sistema es el grado en el que las mediciones repetidas en condiciones sin cambios muestran los mismos resultados[23].

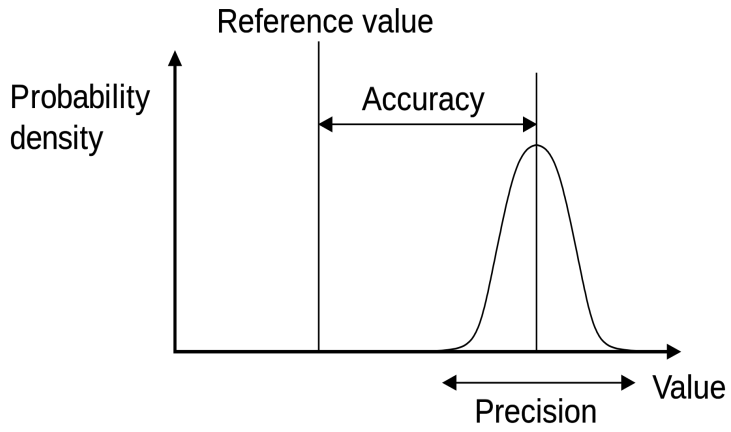

Figura 3.3: Exactitud y precisión [5]

Como hemos mencionado antes, el aprendizaje automático esté ligado al campo de la estadística en muchos casos. En este caso, la estadística se refiere a los términos de sesgo (*bias*) y varianza (*variability*) en vez de exactitud y precisión.

Un sistema puede combinar estas dos cualidades de sus cuatro formas posibles. Un sistema puede ser preciso e inexacto, impreciso pero exacto, ambas o ninguna.

Sin embargo, la evaluación de exactitud en muchos casos no se acerca al resultado real. En muchas ocasiones el modelo ha sido sobre entrenado con los casos de ejemplo. Esto significa que será capaz de evaluar correctamente los casos de ejemplo y casos similares a estos, pero nunca podrá ir más allá de estos casos y generalizar el proceso. Por ejemplo, si un conjunto de 100 datos de entrenamiento tiene 95 resultados A, y 5 resultados B, nuestro sistema clasificará con bastante precisión los ejemplos futuros A, pero apenas será fiable en los ejemplos B.

Para evitar esto nuestro conjunto de datos de entrenamiento debe ser lo más variado posible, incluyendo, en la mayor medida de lo posible, datos de todos los tipos. Sin embargo, esto no soluciona el problema del sobreentrenamiento. Una forma de solucionarlo es la división de nuestro conjunto de entrenamiento.

Existen varias formas de dividirlo, la más simple es repartir manualmente un 80 % para entrenamiento y un 20 % para validación. Sin embargo, si el conjunto de entrenamientos es lo suficientemente grande, se recomienda dividirlo en tres subconjuntos, uno de entrenamiento, uno de validación y uno final

[5]Imagen sacada de `https://commons.wikimedia.org/wiki/File:Accuracy_and_precision.svg`

de comprobación o testeo. Con un porcentaje de 60 %, 24 %, 16 % respectivamente del conjunto original(Los porcentajes no es obligatorio que sean estos, son orientativos). Para mayor fiabilidad en muchos casos se suele desordenar el conjunto original para evitar que todos los ejemplos con un mismo resultado caigan en el mismo subconjunto. Esto se conoce como validación cruzada.

3. **Redes neuronales**

Las redes neuronales artificiales es otro modelo de aprendizaje supervisado inspirado en el comportamiento de las neuronas biológicas. Sin embargo, las neuronas naturales son muchísimo más complejas que las neuronas artificiales que tenemos hoy en día. Y es por esto por lo que cuanto mejores sean nuestros conocimientos acerca de las neuronas mejores réplicas artificiales podremos hacer de ellas. Actualmente el éxito de las redes neuronales no es la recreación exacta de las naturales, sino que los investigadores han buscado la forma en la que las personas resuelven problemas que tradicionalmente no han podido resolverse mediante la informática. Para ello, una neurona artificial está formada por la simulación de cuatro funciones de las neuronas naturales[24].

Una neurona tiene una serie de datos de entrada, normalmente representados por x(n). Cada uno de estos datos de entrada son multiplicados por el peso de la conexión w(n)[25]. En el caso más simple de todos estos productos, simplemente son tratados por una función de suma para ver el dato de entrada total y posteriormente pasar por una función de transferencia generando un nuevo dato. Después, este dato pasará por una función de salida en la que se procesará su información para que pueda ser entendible.

En casos más complejos entre la función de transferencia y la función de salida puede haber una función intermedia de escalado y limitación (normalización). Este proceso de escalado simplemente multiplica un factor de escalado por el dato. El proceso de limitación sirve para asegurar que el resultado del producto anterior no exceda un límite, tanto superior como inferior.

En la mayoría de los sistemas, existe otra función que se encarga de calcular la diferencia entre el resultado actual y el deseado. Este error es transformado para la arquitectura de la red neuronal. La función de aprendizaje se centra en modificar los pesos de las variables de entrada en cada proceso según un algoritmo. Este algoritmo es el que clasifica a las redes neuronales dividiéndolas en si pertenecen al aprendizaje supervisado o no.

En su mayoría, las redes neuronales pertenecen al aprendizaje automático supervisado, sin embargo, existen algunos ejemplos que pertenecen al no supervisado. Estas neuronas están agrupadas en capas entre las que podemos distinguir la capa de entrada, la oculta y la de salida. Siguiendo esta estructura algunas neuronas están en contacto con los datos de entrada del mundo real. Otras suministran datos al mundo real. Todo el resto de las neuronas están en la capa oculta donde se procesa la información. Pero una red neuronal no es solamente una gran cantidad de neuronas artificiales conectadas aleatoriamente entre sí. Es aquí donde existen una gran cantidad de arquitecturas de

las redes neuronales, dependiendo de la forma en que se conectan las neuronas artificiales y todas las posibles funciones descritas anteriormente[26][27].

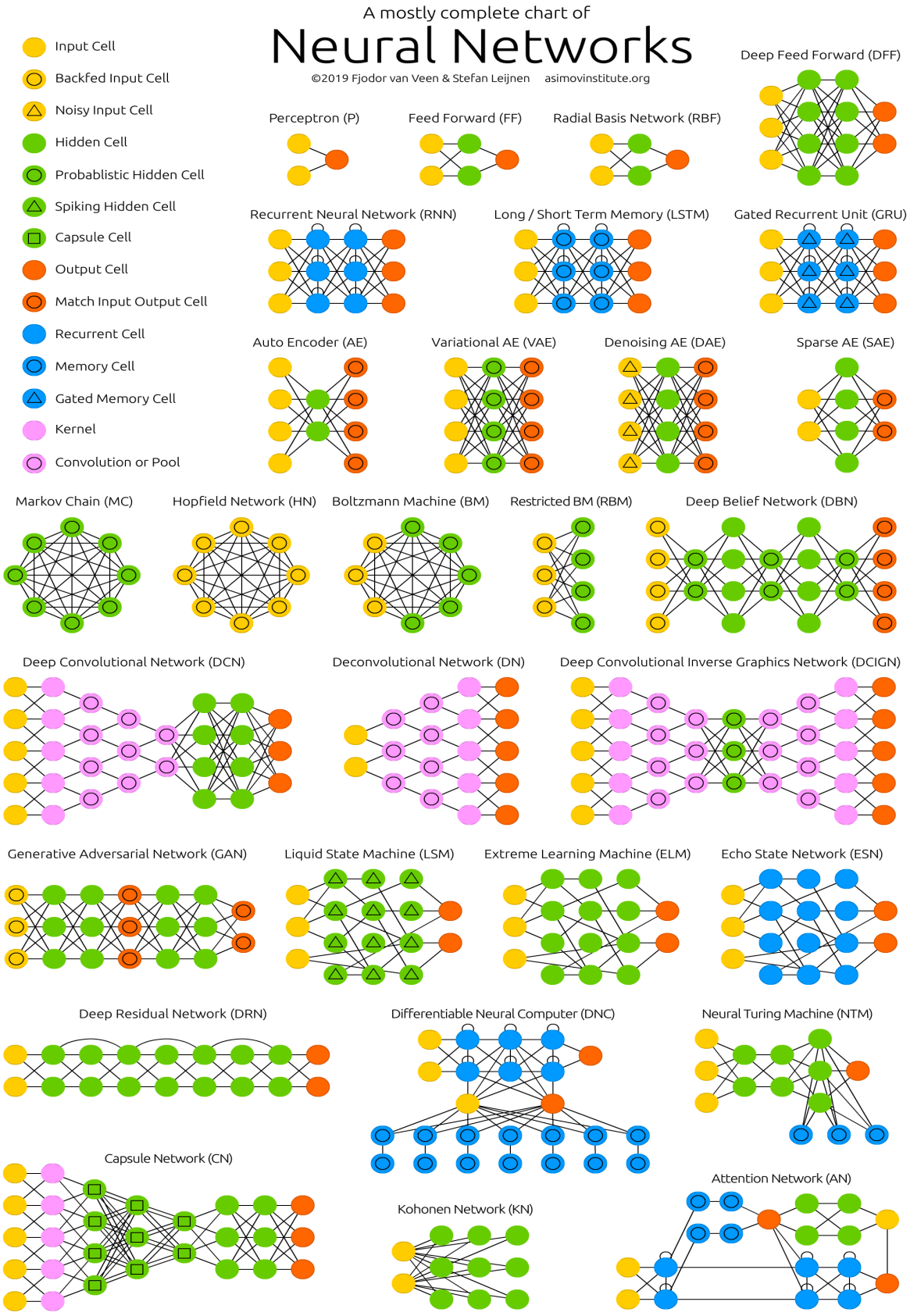

Figura 3.4: Redes Neuronales más comunes [6]

4. Conjuntos

Un conjunto es un algoritmo supervisado, ya que puede ser entrenado y, después, usado para realizar predicciones. El objetivo de los algoritmos de conjuntos es el de combinar las predicciones de varios modelos para mejorar la generalización y robustez respecto a un único modelo[28].
Este algoritmo utiliza distintos modelos para obtener mejores predicciones de las que se podrían obtener de los modelos que lo constituyen por separado, es decir, hace una predicción total basada en las predicciones de cada modelo. En muchas ocasiones existen muchas hipótesis que se adaptan muy bien a un problema o que todas ellas son muy prometedoras a la hora de resolver dicho problema. Los conjuntos nos permiten agrupar dichas hipótesis para poder generar una que resuma todas las anteriores.

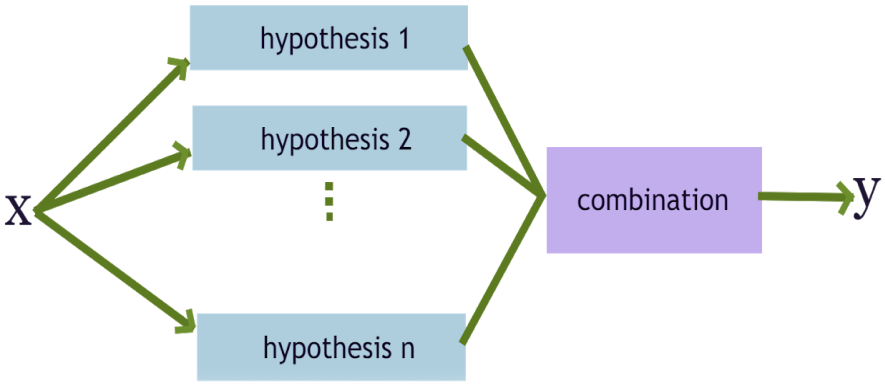

Figura 3.5: Arquitectura general de un algoritmo de conjuntos [7]

Existen dos tipos distintos de métodos de conjuntos, los promedio o *Averaging* y los que se van mejorando o *boosting*. Los métodos *averaging* se basan en la creación de varios modelos de predicción de manera independiente y, posteriormente, realizar una media de las predicciones de dichos modelos. La mayoría de las veces el conjunto de estos modelos es mejor que por separado, puesto que su varianza es menor. Ejemplos de estos tipos de métodos son *Bagging* y Bosques de árboles aleatorios.

[6]Imagen sacada de `https://www.asimovinstitute.org/neural-network-zoo/`
[7]Imagen sacada de: `https://analyticsindiamag.com/basics-of-ensemble-learning-in-classification-techniques-explained/`

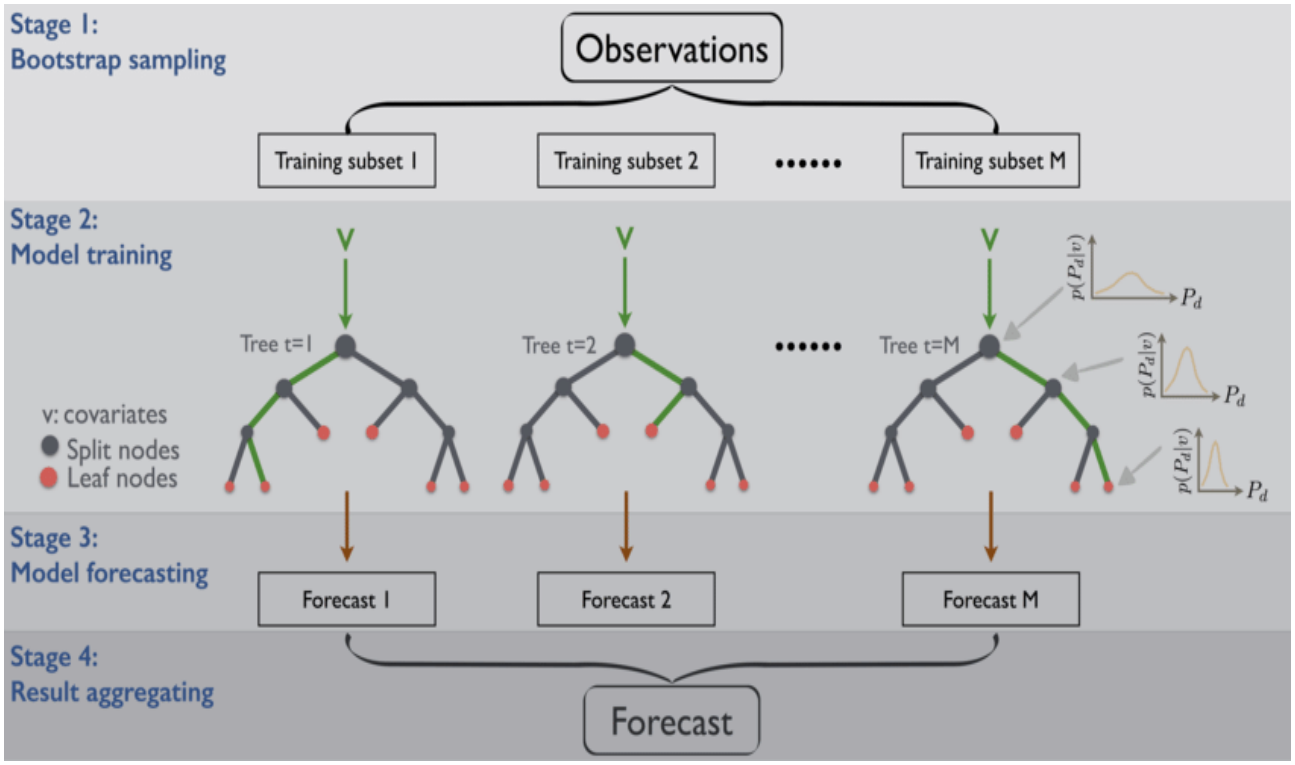

Figura 3.6: Esquema del método Averaging [8]

Por otro lado el objetivo de los métodos *Boosting* es la construcción de un conjunto de modelos construidos secuencialmente en el que cada modelo subsecuente realiza una nueva predicción tratando de solucionar los errores de su predecesor. De esta manera, se combinan modelos menos robustos para formar uno mucho más robusto. Ejemplos de estos métodos son AdaBoost, Gradient Tree Boosting, XGBoost y Light GBM.

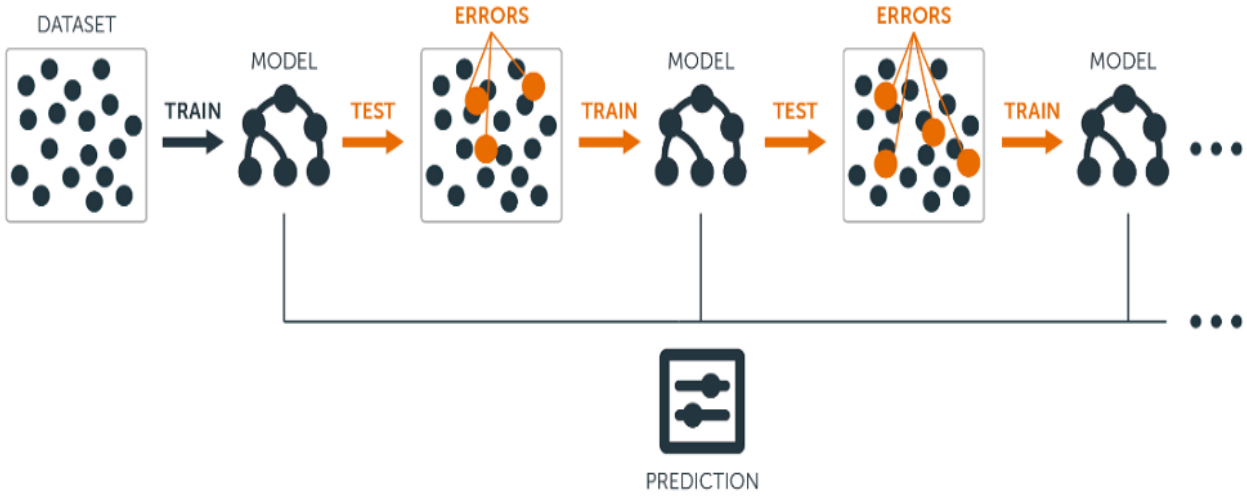

Figura 3.7: Esquema del método Boosting [9]

[8]Imagen sacada de `https://www.kdnuggets.com/2019/01/ensemble-learning-5-main-approaches.html`

[9]Imagen sacada de `https://www.kdnuggets.com/2019/01/ensemble-learning-5-main-approaches.html`

Este tipo de algoritmos cuenta con una serie de ventajas y desventajas.

Ventajas de los algoritmos de conjuntos:

- Mejoran la exactitud del modelo y funciona correctamente la gran mayoría de las veces.
- Hacen al modelo general más robusto y estable.

Desventajas de los algoritmos de conjuntos:

- Necesitan tiempo para desarrollarse.
- La creación del conjunto en cuanto a la elección de los modelos es muy difícil de realizar y se necesita experiencia en ello.

5. **Soportes de máquinas de vectores**

En el ámbito del aprendizaje automático, las *support vector machine* (SVM, también conocidas como *support vector networks*) son modelos de aprendizaje supervisado[29]. Estas están relacionadas con métodos de aprendizaje encargado de analizar datos utilizados para problemas de clasificación y análisis de la regresión. En términos formales, su objetivo es el de encontrar un hiperplano en un espacio N-dimensional[30] (N = número de características) que se encargue de clasificar los diferentes puntos de información. Dado un conjunto de ejemplos de entrenamiento, cada uno perteneciente a una clase etiquetada, un algoritmo de entrenamiento SVM construye un modelo que se encarga de predecir la clase de una nueva muestra. Un modelo SVM es la representación de los ejemplos de la muestra como puntos en el espacio, situados de tal forma que se distinga la separación de clases con un espacio intermedio que sea lo más ancho posible. Los nuevos ejemplos serán mapeados en ese mismo espacio y se les asignará una categoría basándose en el lado del espacio en el que hayan caído.

Refiriéndonos a los hiperplanos[30], éstos son áreas de decisión que nos ayudan a clasificar los puntos de información. La dimensión del mismo depende del número de características. Por ejemplo si tenemos 2 características, el hiperplano será una línea, si tenemos 3, será un plano tridimensional.

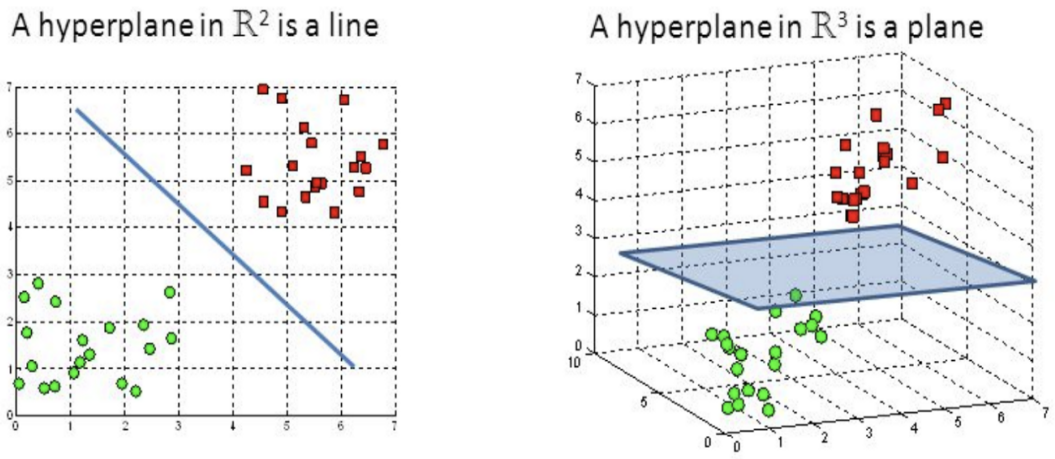

Figura 3.8: Representación hiperplano bidimensional y tridimensional [10]

Para separar dos clases, hay muchos hiperplanos que se pueden escoger. Lo más óptimo es encontrar un hiperplano con el margen máximo (máxima distancia entre puntos de información de ambas clases). Maximizar la distancia del margen nos puede ayudar en el futuro a clasificar con mayor seguridad nuevos ejemplos.

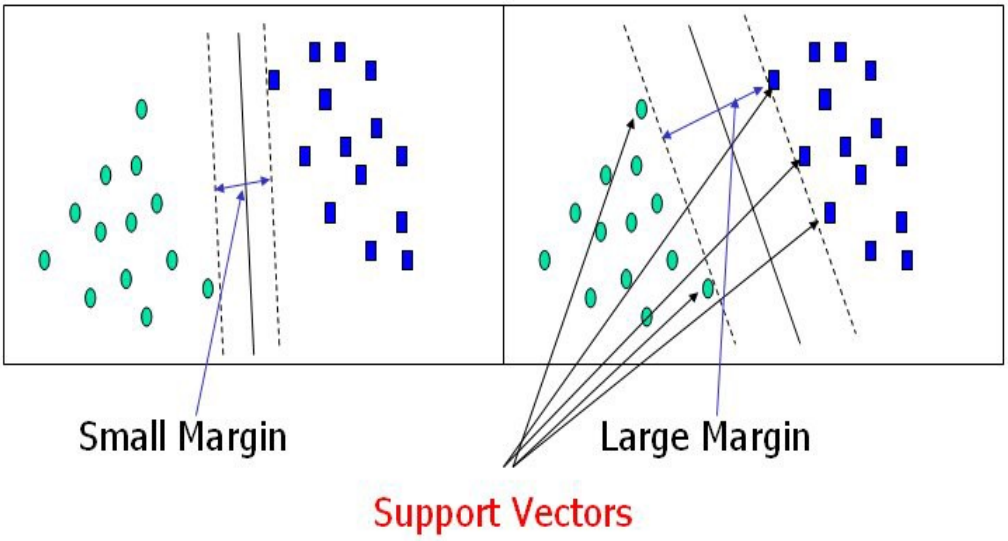

Figura 3.9: Márgenes [11]

En referencia a la imagen de arriba (Fig 3.4: Márgenes) los *support vectors* son los puntos de información más cercanos al hiperplano que influencian en la posición y orientación del mismo. Mediante estos *support vectors* conseguimos maximizar el margen del clasificador. Un algoritmo SVM no solo trabajará con planos bidimensionales. Tendrá que poder trabajar también con más de dos variables predictoras, conjuntos de datos en los que la separación entre clases no es tan clara, o clasificaciones con más categorías. Para ello utilizamos las funciones kernel.

Finalmente hay dos características claves, a parte del márgen, a la hora de ajustar el funcionamiento de una SVM[31]:

-**Regularización**: Especifíca a la SVM cuanto error se asume a la hora de clasificar cada ejemplo de entrenamiento

[10]Imágenes sacadas de `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`

[11]Imágenes sacadas de `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`

Figura 3.10: Representación de la regularización [12]

-Gamma: Define cuánto puede llegar a influenciar un único ejemplo de entrenamiento. Un valor bajo para gama indica que los puntos lejanos al margen también se tienen en cuenta para los cálculos, mientras que si este valor es alto, solo se tendrán en cuenta los valores más cercanos.

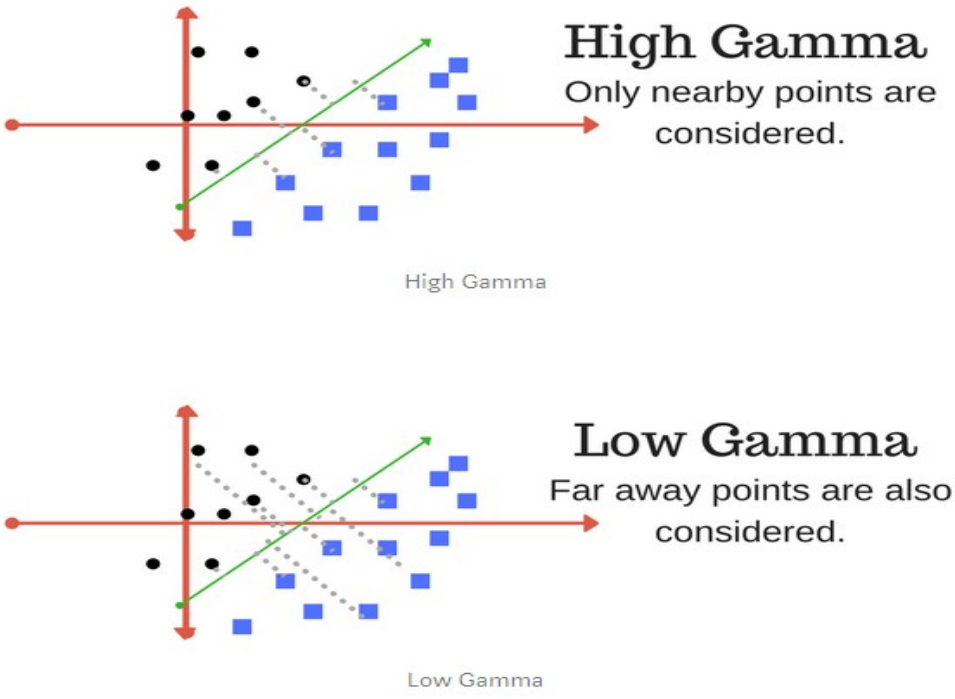

Figura 3.11: Representación de gamma [13]

[12] Imágenes sacadas de https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[13] Imágenes sacadas de https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

6. **Arboles de decisión**

Los árboles de decisión es un modelo de predicción utilizado tanto en el aprendizaje automático como en el mundo de la probabilidad, estadística y la economía. Este modelo de aprendizaje automático es la representación gráfica de posibles soluciones basándose en si los datos cumplen o no ciertas condiciones. Es uno de los algoritmos más utilizados del aprendizaje automático supervisado. Los árboles de decisión pueden ser de dos tipos principalmente, de regresión o de clasificación. Es decir, pueden realizar tareas de regresión o clasificación (Classification And Regression Tree (CART))[32].

Pueden tomar como resultado una clase discreta a la que pertenecen los datos o un número real. La comprensión de este tipo de algoritmos es muy simple, ya que utilizamos muy a menudo en nuestra vida cotidiana sin darnos cuenta y a la vez son muy potentes, como, por ejemplo: "¿Hace frío? Sí, me pongo un abrigo. No, no me pongo un abrigo."

Los árboles de decisión están formados por:

a) **Nodos**. Los nodos es el lugar en el que se toma una decisión entre varias posibles. Cuantos más nodos más posibilidades tienes a la hora de tomar una solución a la pregunta, es decir más finales a los que se pueden llegar.

b) **Vectores de números**. Los vectores de números serían la solución final a la que se llega a través de los distintos nodos visitados.

c) **Flechas**. Las flechas son las uniones ente nodos y representan la opción tomada tras evaluar la pregunta del nodo.

d) **Etiquetas**. Las etiquetas se encuentran en cada nodo y cada flecha y dan nombre a la acción.

Tratándose de árboles debemos tener en mente ciertos conceptos y reglas característicos de los árboles.

En cuanto a los criterios:

a) **Costo**. Se refiere al coste de determinadas propiedades como puede ser la inserción de un elemento, su borrado u operaciones como saber la profundidad del árbol.

b) **Poda**. Evitan el procesamiento de subárboles que no afectan a la decisión, consiste en eliminar una rama de un nodo transformándola en una hoja Terminal (Podas alfa y beta, usado en árboles de juego).

En cuanto a las reglas, los árboles de decisión han de cumplir:

a) Al comienzo del árbol se da un nodo inicial que no es apuntado por ningún otro y es el único con esta característica.

b) El resto de los nodos son apuntados por otro a través de una única flecha. Lo que se conocen como nodos padre e hijo.

c) Derivado de lo anterior se deduce que solo hay un camino para llegar del nodo inicial al resultado final.

Para obtener un árbol óptimo y valorar que subárbol escoger, el algoritmo deberá medir las predicciones para poder compararlas y obtener la mejor. Para ello utiliza diversas funciones basadas en la entropía. El concepto de entropía mide la impureza de un conjunto de entrada. En física y matemáticas es conocido como la aleatoriedad o impureza del sistema. Para entenderlo podemos imaginar un conjunto de 100 bolas verdes, se podría decir que es puro. En términos de entropía diríamos su entropía es 0. Imaginemos que 30 de esas bolas son azules y 20 rojas, ahora el conjunto será más impuro y por tanto su valor de entropía aumentará.

Las funciones más conocidas basadas en la entropía son: la ganancia de información, la impureza de Gini y la reducción de varianza[33][34].

3.2.2. Algoritmos no supervisados

Los algoritmos no supervisados a diferencia de los supervisados no cuentan con una información previa. Este tipo de algoritmos no usa etiquetas, su finalidad es la de encontrar patrones de todos los datos recibidos directamente.

Los algoritmos de aprendizaje no supervisado permiten trabajar con tareas mucho más complejas en comparación con los algoritmos supervisados. Aparte de para ejecutar tareas complejas, el aprendizaje no supervisado se puede usar también para encontrar todo tipo de patrones dentro de los datos o encontrar características útiles para categorización.

Sin embargo, los resultados serán menos consistentes e impredecibles debido a que los datos de entrada no están ni reconocidos ni etiquetados previamente. Además el usuario necesitará invertir más tiempo en interpretar y etiquetar las clases que necesitará en la posterior clasificación[35].

Hay 2 varios tipos de aprendizaje no supervisado agrupados en:

Clustering

Procedimiento de agrupación de una serie de vectores con un criterio, por lo general, distancia o similitud. Gracias a esto se forman grupos que comparten valores comunes o muy similares entre sí llamados *clusters*. El *clustering* puede ser utilizado para estudiar los terremotos. Basándose en áreas golpeadas por los mismos, puede ayudar a analizar cuál será la siguiente zona a la que podría afectar. Dentro del *clustering* encontramos *k-means*. Este algoritmo en cada iteración se encarga de asignar a cada ejemplo, el centroide [13] más cercano. Generando como salida un conjunto de etiquetas con los valores más óptimos. Dentro del *clustering* encontramos diferentes tipos de algoritmos[35]:

[13] Un centroide es el punto con el valor más alto, el centro de un *cluster*. El algoritmo *k-means* encuentra k centroides[36]

1. ***Clustering* jerárquico**. Es un algoritmo que se encarga de construir una jerarquía de *clusters*[37].

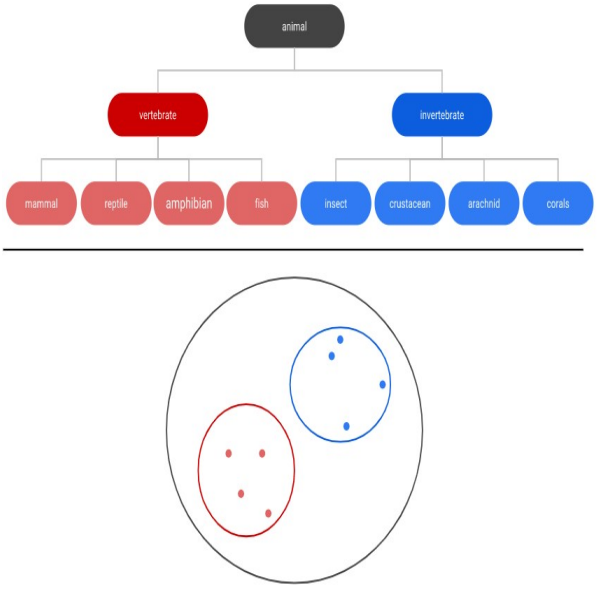

Figura 3.12: *Clustering* jerárquico [14]

2. **Basado en el centroide**. Organiza los datos en *clusters* no jerárquicos. Destaca *k-means*. Son algoritmos muy eficientes y efectivos pero sensibles a condiciones iniciales[37].

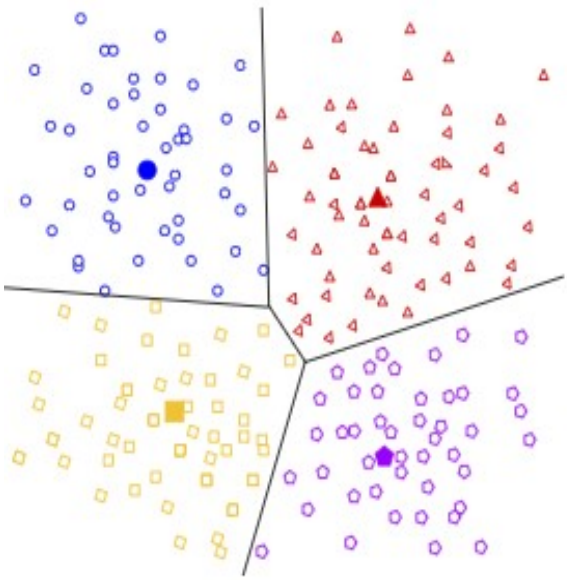

Figura 3.13: Basado en el centroide [15]

[14]Imagen sacada de `https://developers.google.com/machine-learning/clustering/clustering-algorithms`

[15]Imagen sacada de `https://developers.google.com/machine-learning/clustering/clustering-algorithms`

3. **Basado en la densidad**. Se encarga de conectar áreas de gran densidad dentro de *clusters*. Estos algoritmos tienen dificultades con datos en los que las densidades varían.y cuyas dimensiones son muy grandes[37].

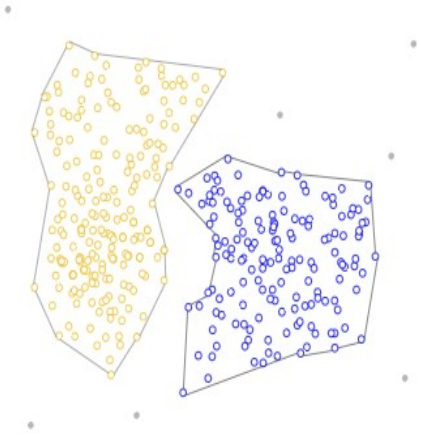

Figura 3.14: Basado en la densidad [16]

4. **Basado en la distribución**. Asume que el conjunto de datos está formado por distribuciones. En la imagen, podemos observar que el algoritmo ha dividido los datos en tres distribuciones gaussianas. Cuanto más lejos está un punto del centro de la distribución, su probabilidad de pertenecer a la misma disminuye[37].

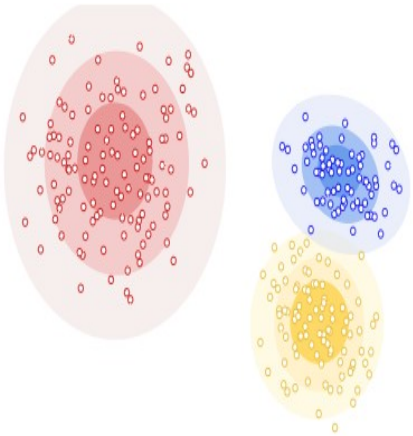

Figura 3.15: Basado en la distribución [17]

Asociación

Esta técnica permite descubrir relaciones de interés entre las diferentes variables de un conjunto de datos. Por ejemplo, si un cliente de una cadena de supermercados compra cebollas y patatas, muy probablemente comprará carne de hamburguesas[38].

[16]Imagen sacada de `https://developers.google.com/machine-learning/clustering/clustering-algorithms`

[17]Imagen sacada de `https://developers.google.com/machine-learning/clustering/clustering-algorithms`

1. **k-medias y k-medoides**

   K-means. Es un algoritmo iterativo que intenta dividir el conjunto de datos en k *clusters* (grupos) predefinidos de manera que cada punto de información pertenezca a un único grupo. Se encarga de asignar a cada *cluster* aquellos puntos de información en los que la suma de la distancia al cuadrado entre el *cluster* y el mismo es mínima. Cuanta menos variación tengamos, más homogeneidad habrá entre los puntos dentro de un mismo *cluster*. El algoritmo se compone de varios pasos[39]:

   a) Especifica el número de *clusters* K.

   b) Inicializa los centroides mezclando el conjunto de datos y seleccionando K puntos de información como centroides.

   c) Itera hasta que haya convergencia, es decir, no hay cambios en los centroides.

   En este último paso, calcula la suma de las distancias al cuadrado entre el centroide y los puntos de información, asigna cada punto al centroide más cercano y recalcula el centroide para cada *cluster* tomando la media de la distancia de cada punto de información que pertenecen a ese *cluster*.

   K-medoids. Este algoritmo es un acercamiento al *clustering* a través de *k-mean*, dividiendo el conjunto de datos en k *clusters*. Cada *cluster* es representado por uno de los puntos de información dentro del mismo. Estos puntos se llaman medioides. Un medioide se refiere a un objeto dentro del *cluster* cuya disimilaridad media a todos los objetos dentro del grupo es mínima. Es una alternativa fiable al clustering con k-mean. K-medioides es más robusta frente al ruido y los valores atípicos que *k-means*, ya que intenta minimizar la suma de las diferencias entre los puntos etiquetados dentro de un *cluster* y el mediode en vez de sumar todas las distancias euclídeas al cuadrado. La forma más común de llevar a cabo el *clustering* a través de k-medioides es con el algoritmo de Partición sobre los medioides (PAM)[40].

   a) Con él seleccionamos aleatoriamente k medioides de los n puntos de información.

   b) Asociamos cada punto al medioide más cercano.

   c) Para cada medioide "m" y cada punto de información "o" asociado a "m", intercambiamos ambos puntos y calculamos de nuevo el coste de esta nueva configuración (diferencia media desde "o" a todos los puntos "m" del *cluster*) y seleccionamos el medioide con el menor coste de configuración.

   Los pasos 2 y 3 se repetirán hasta que haya convergencia, es decir, que el resultado no cambie.

2. **Análisis de componentes principales**

   Es un método que permite simplificar la complejidad de espacios de muestra de grandes dimensiones, condensando la información en pocas componentes. Es una técnica utilizada para la extracción de características. Combina cada una

de las características de tal manera que nos podamos deshacer de las variables menos importantes, mientras que nos quedamos con las más relevantes. Esto es óptimo, ya que los supuestos de un modelo lineal requiere que nuestras variables sean independientes las unas de las otras. PCA debe utilizarse cuando queremos reducir el número de variables pero no somos capaces de identificar que variables borrar completamente y cuando nos queremos asegurar que nuestras variables son completamente independientes[41].

3.2.3. Aprendizaje por refuerzo

Los algoritmos por refuerzo se basan en el aprendizaje a partir de experiencias anteriores. La información que recibe el sistema son las respuestas que recibe del entorno que le rodea a sus acciones. Y poco a poco, mediante un sistema de premios y castigos, el sistema va aprendiendo a modificar sus acciones para acercarse más a la meta mediante dichos premios. Un ejemplo de esto es la clasificación de secuencias de ADN o los coches autónomos.
El aprendizaje por refuerzo[42] puede entenderse con los siguientes conceptos:

1. **Agente**. Es quien realiza la acción.

2. **Acción (A)**. Es el conjunto de todas las opciones posibles que un agente puede hacer.

3. **Factor de descuento**. Es el que se multiplica por recompensas futuras descubiertas por el agente para equilibrar el efecto de estas recompensas en la elección del agente, es decir que las posibles recompensas futuras tengan menos valor que las actuales.

4. **Medio ambiente**. Es el mundo en el que se mueve el agente y al que responde. El entorno toma el estado y la acción del agente como entrada y devuelve una recompensa como salida además de su siguiente estado.

5. **Estado (S)**. Es la situación concreta e inmediata en la que se encuentra el agente.

6. **Recompensa (R)**. Es la retroalimentación por la cual medimos el éxito o el fracaso de las acciones de un agente en un estado.

7. **Política (π)**. Es la estrategia a seguir que el agente emplea para determinar la siguiente acción basándose en el estado actual.

8. **Valor (V)**. Es el rendimiento esperado a largo plazo, en lugar de la recompensa a corto plazo. Vπ(s) es el rendimiento esperado según la política π.

9. **Valor Q o valor de acción**. Es similar al valor excepto porque requiere la acción actual. Qπ(s, a) es el retorno a largo plazo de una acción que toma una política π en el estado s.

10. **Trayectoria**. Es la secuencia de estados y acciones que originan esos estados.

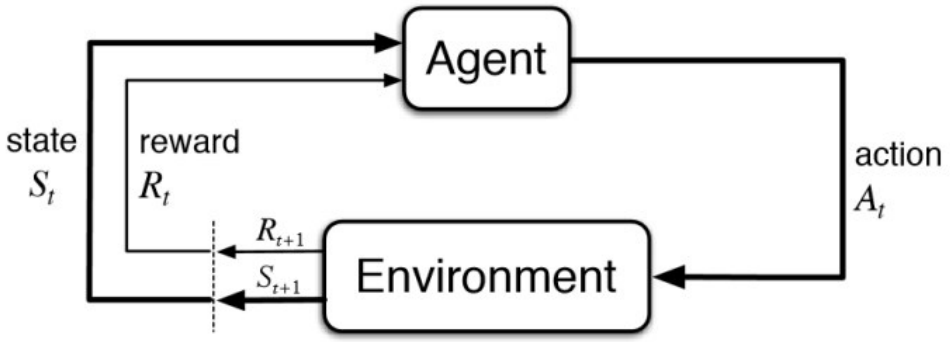

Figura 3.16: Aprendizaje por refuerzo [18]

Hay tres enfoques a la hora de implementar un algoritmo de aprendizaje por refuerzo.

1. **Basándose en el valor**. Un método basado en el valor debe intentar maximizar el valor devuelto a largo plazo siguiendo una política Vπ(s).

2. **Basándose en la política**. Un método basado en la política intenta llegar a una política que lo ayude a obtener la máxima recompensa en el futuro. Pueden ser deterministas, para cualquier estado la política pi produce la misma acción, o estocástica, cada acción tiene una probabilidad que viene determinada.

3. **Basado en el modelo**. Un método basado en el modelo debe crear un modelo virtual para cada entorno y el agente aprende a actuar en ese entorno específico.

Algunos de los algoritmos más representativos del aprendizaje automático por refuerzo son:

1. **Proceso de decisión de Markov**

   La propiedad de Markov establece que "El futuro es independiente del pasado dado el presente". Matemáticamente podemos establecer esta afirmación como:

   $$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \ldots, S_t] \tag{3.1}$$

   Donde S_t denota el estado actual del agente y S_{t+1} el siguiente. Lo que significa que una transición del estado t al estado t+1 es completamente independiente de las transiciones pasadas. Es decir, nuestro estado actual ya captura la información de los estados pasados.

   El proceso de decisión de Markov o cadenas de Markov es un proceso aleatorio con una secuencia de estados siguiendo la propiedad de Markov. Se puede definir con un conjunto de estados y una matriz de probabilidades de transición.

   Para maximizar la recompensa se sigue el proceso de recompensa de Markov en el que se obtiene un valor de cada estado en el que se encuentra nuestro agente.

   $$Rs = E[R_{t+1}|S_t] \tag{3.2}$$

   Cuanta recompensa Rs obtenemos de un estado particular St inmediatamente.

[18] Imagen sacada de `https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html`

Ahora podemos añadir una toma de decisiones a estas ecuaciones para obtener el proceso de decisión de Markov. Donde S es un conjunto de estados, A es el conjunto de acciones se pueden elegir, P es la matriz de probabilidad de transición, R es la recompensa acumulada por las acciones del agente y γ es el factor de descuento.

$$P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a] \tag{3.3}$$

Matriz de probabilidad de transición

$$R^a_s = E[R_{t+1} | S_t = s, A_t = a] \tag{3.4}$$

Función de recompensa

Hasta ahora obteníamos una recompensa (r) cuando pasábamos por un conjunto de estados siguiendo una política π. Realmente en el proceso de decisión de Markov la política es el mecanismo para realizar una acción. De esta forma ahora nosotros tenemos un mecanismo que tomará una acción[43].

2. **Aprendizaje Q**

El aprendizaje Q[44] es un método basado en valores Q, también llamados valores de acción, para mejorar iterativamente el comportamiento del agente de aprendizaje.

a) **Valores Q o valores de acción**. Los valores Q se definen para estados y acciones. Q(s, a) es una estimación de qué tan bueno es tomar la acción a en el estado s. Esta estimación se calculará de forma iterativa utilizando la regla de actualización TD.

b) **Recompensas y episodios**. El agente realiza una serie de transiciones desde su estado actual al siguiente dependiendo de la acción tomada y del entorno en el que el agente la realiza.

c) **Diferencia temporal**. La regla de diferencia temporal se puede representar de la siguiente manera:
Q(s, a) $\leftarrow$ Q(s, a) + α (R + γ Q(s', a') - Q(s, a))

Esta regla de actualización para estimar el valor de Q se aplica en cada paso de la interacción de los agentes con el entorno. Donde:

1) γ **(gt; 0 y <= 1)**. Factor de descuento para recompensas futuras. Las futuras recompensas son menos valiosas que las recompensas actuales, por lo que deben descontarse.
2) α. Longitud del paso tomada para actualizar la estimación de Q (s, a).

Elegir la acción a tomar usando la ϵ política de *greedy*. Consiste en elegir acciones utilizando las estimaciones actuales del valor Q. Va de la siguiente manera:

1) **Con probabilidad 1** $-\epsilon$. Elegir la acción que tenga el valor Q más alto.
2) **Con probabilidad** ϵ. Elegir cualquier acción al azar.

3.3. Conclusión

Después de ver el estado actual, y estudiar más a fondo el aprendizaje automático, sus distintos tipos de algoritmos y ejemplos de estos, podemos decir que es una técnica que replica muchas tareas que el cerebro humano puede hacer. En muchas ocasiones estas tareas son realizadas en menor tiempo y en muchas ocasiones con mejores resultados que el propio trabajo de un ser humano. Como hemos visto anteriormente las máquinas pueden vencer a los campeones de muchos juegos como el ajedrez o el Go. Además, hemos visto que las máquinas pueden aprender a realizar actividades en una gran cantidad de sectores, ayudando de esta forma a los seres humanos en sus trabajos, en sus casas o incluso a la hora de divertirse, es decir, en su vida diaria.

También existe una gran variedad de tipos de algoritmos de aprendizaje automático. Dentro de cada tipo existen a su vez diferentes algoritmos para usar. Cada uno de ellos se adapta mejor a un cierto tipo de problemas, por ello es necesario conocer el problema que se quiere resolver. Saber que se pide, pero también de que datos dispones.

Finalmente, cuando se trata del desarrollo de modelos propios de aprendizaje automático se estudia las distintas opciones de lenguajes de desarrollo, entornos de desarrollo y plataformas. Lo siguiente es conocer y saber aplicar cada técnica de aprendizaje automático a cada tipo de problema, es decir, por ejemplo si se necesita un algoritmo supervisado o no. Una vez elegido el tipo de algoritmo ver cual de todos ellos se adapta mejor a la resolución del problema.

El tema es muy amplio y ofrece muchas posibilidades, pero analizándolo detenidamente, cada técnica es diferente y específica para resolver un tipo de problema determinado.

Capítulo 4

Generación Procedural de Contenido

4.1. Definición

La generación procedural o generación por procedimientos de contenido es la producción de recursos de manera aleatoria en base a unos algoritmos preestablecidos por los desarrolladores, es decir, son recursos que no han sido creados de ante manos por personas. De esta forma, dichos recursos que se ven por pantalla son únicos, prácticamente irreproducibles otra vez.

Esto abre una gran cantidad de opciones y posibilidades creando factores impredecibles. No obstante, no todo son ventajas, esto implica un sacrificio en el diseño debido a la impredecibilidad del propio contenido creado.

4.2. Origen y evolución

El mundo de los videojuegos explota este concepto al máximo en todos los procesos de desarrollo del mismo. Además, al otorgar cierta aleatoriedad a los videojuegos, esto genera una sensación mucho más inmersiva en cuanto al juego se refiere. Es por estos motivos por lo que ha ganado estos últimos años mucha fama el uso de generación procedural de contenido a la hora de desarrollar un videojuego. Sin embargo, a pesar de que el término pueda parecer muy moderno, la realidad es que este método de creación de recursos se lleva usando bastantes años. Títulos famosos como 'Rogue' de 1980 padre del género *roguelike*, usaban estos mecanismos. *Rogue* utilizaba estos métodos para generar niveles aleatorios, si bien es cierto que no fue el primer juego en usar algoritmos semi aleatorios para la creación de contenido, sí que lo hizo con muy buenos resultados. Otros buenos ejemplos de que esta práctica se llevan usando durante bastante tiempo, el ejemplo de 'Elite' de 1984 o 'The Elder Scrolls II: Daggerfall' de 1996. También cabe destacar un juego algo más moderno, publicado en 2009, que ha tenido un gran impacto entre la comunidad de jugadores, el Minecraft, juego referente para este tipo de técnicas de creación de contenido[45].

Como hemos mencionado anteriormente, no es un descubrimiento actual, la cantidad de videojuegos que usan esta técnica de creación de contenido ha aumentado, y es un factor muy importante para los videojuegos producidos por pequeñas y medianas compañías. Algunos títulos posteriores son No Man's Sky de 2016 que implementaban generación procedural de galaxias, con planetas completamente explorables. Otro ejemplo es The Division, de 2016, que también utilizaba lo que podría

denominarse métodos semiprocedurales, ya que la disposición de muebles dentro de los edificios era generada proceduralmente, mientras que los edificios mantenían unas reglas fijas.

Algunos ejemplos más actuales de juegos que utilizan la generación procedural de contenido son, la saga Civilization a la hora de generar el mapa de juego o la saga Borderlands para la creación de armas y objetos que se pueden usar a lo largo del juego. Las estimaciones decían cerca de 17 millones de armas distintas, finalmente se llegó a 3 millones de combinaciones. El reciente Minecraft Dungeons que utiliza estos mecanismos para la generación de mazmorras 3D llenas de monstruos, trampas y tesoros. La nueva expansión del World of Warcraft, Shadowlands, para crear Torghast, una mazmorra con *spawners*[1] generados proceduralmente.

Sin embargo, la tecnología ha evolucionado, y gracias a esto los algoritmos han conseguido llegar más allá de simplemente un par de procesos semi aleatorios y basarse incluso en experiencias de jugadores para la creación de contenido. La generación procedural se presenta como una solución a la producción de contenido para videojuegos, mediante la ayuda o automatización del proceso de generación. Haciendo uso de estas técnicas, grandes cantidades de contenido se pueden generar de forma algorítmica sin la necesidad de trabajar en su creación. En otras palabras, la idea de la generación procedural de contenido es que parte del videojuego se genere computacionalmente a través de un procedimiento y algoritmos bien definidos, en lugar de ser creado a mano[46].

4.3. Procedural y aleatorio

Hay que diferenciar entre generación procedural y aleatoriedad. Por mucho que a simple vista puedan parecer sinónimos o incluso iguales, realmente guardan bastantes diferencias. La generación aleatoria es algo basado en al azar, en probabilidades y estadísticas.

La generación procedural se basa en procedimientos y seguir unos algoritmos que usualmente tienen en cuenta una gran variedad de factores. Estos factores pueden ser externos a la propia generación o pueden ser resultados anteriores, por tanto, los contenidos generados están supeditados a un mayor número de datos a evaluar[47].

Pongamos un ejemplo de un videojuego RPG [2] en el que derrotas a un monstruo y este deja caer algunos objetos. Dentro de la lista de objetos que puede soltar hay oro, un arma cuerpo a cuerpo y un arma a distancia. Si la generación fuese aleatoria lo más rápido seria asignar una probabilidad a cada elemento, por ejemplo, repartiendo una cantidad de números y escogiendo uno al azar. Las probabilidades de cada objeto irían en función de la cantidad de números que tengan asignados.
En cambio, si la generación fuese procedural, el algoritmo podría tener en cuenta factores como: el enemigo ha sido derrotado por un arma, ¿Qué clase de arma era?, ¿Cuerpo a cuerpo o a distancia?, ¿Tiene mucho oro el personaje? Y en función de estos factores alterar la generación de uno u otro objeto. Elementos propios de un

[1] Lugar donde se recrea un personaje, un enemigo o un ítem después de su muerte o destrucción

[2] *role-playing game*, es un género de videojuegos donde el jugador controla las acciones de un personaje.

juego como la dificultad elegida puede influir en este tipo de situaciones, tanto procedural como aleatoriamente.

Ahora surge la pregunta de cual es mejor. En cuanto a diseño y calidad, decidir entre contenido aleatorio o procedimental basándose en la cantidad de tiempo de que se dispone. Crear un algoritmo para generar contenido otorgará al videojuego la posibilidad de jugarlo una y otra vez, sin embargo, hacerlo bien lleva tiempo. Con contenido aleatorio, puedes tener resultados en poco tiempo gracias a que habrá elementos codificados que luego se barajan para crear una experiencia aleatoria.

4.4. Clasificación y taxonomía

La generación procedural de contenido no supone un problema trivial, ya que estas técnicas se pueden aplicar a muchos de los factores que componen un videojuego desde un nivel muy bajo cómo sonidos o texturas, hasta un nivel más abstracto como sus propias reglas. Existen una gran variedad de algoritmos de PCG para realizar esto. Dentro de esta gran cantidad de algoritmos hay algunos que comparten ciertas características en algunos aspectos. Por lo que agruparlos en distintos grupos según estas características comunes es algo interesante. Hay dos formas distintas de agrupar estos algoritmos, según el contenido que generan y otra referida a aspectos más técnicos. Dentro de la clasificación según el contenido que generan podemos diferenciar cinco clases principales dedicadas exclusivamente al videojuego, y una más orientada a atraer nuevos jugadores[48][49].

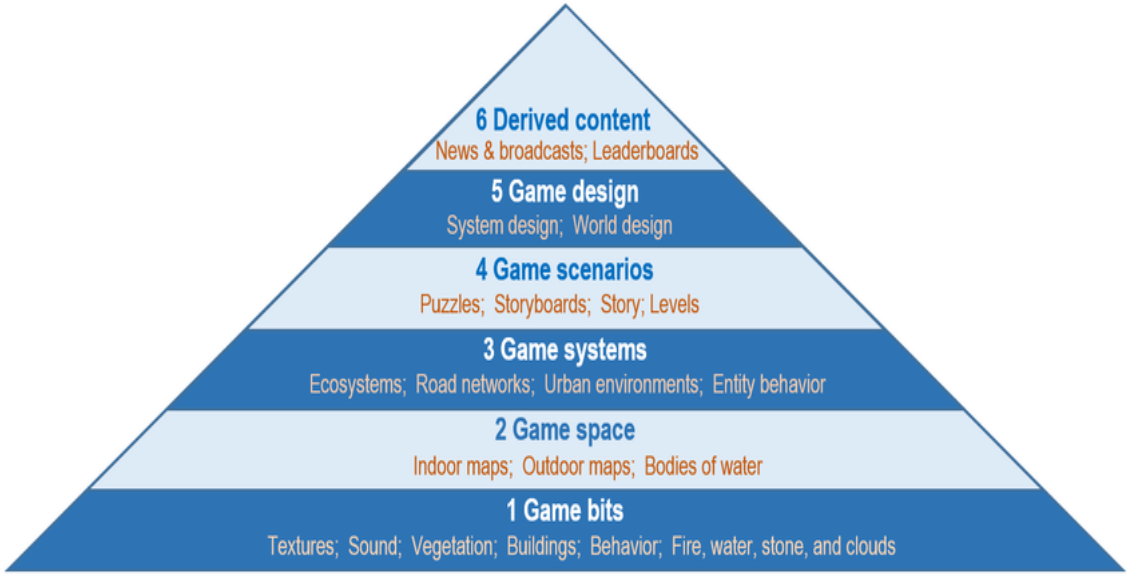

Figura 4.1: Taxonomía según el contenido [3]

1. **Capa 1**. Podemos encontrar lo que se conoce como *game bits*, son los componentes de más bajo nivel de un videojuego como pueden ser texturas, sonidos u objetos que componen la escena.

2. **Capa 2**. Vemos el espacio del videojuego como podría ser un nivel de juego o el entorno que puede rodear un juego de mundo libre.

[3]Imagen sacada de `https://core.ac.uk/download/pdf/156902177.pdf`

3. **Capa 3**. Esta presentes los sistemas de juego. Estos se refieren a esos componentes que hacen que el juego parezca más realista. Aquellos detalles que hacen que te puedas sumergir en su entorno. Un ejemplo de estos pueden ser los personajes no jugables y sus comportamientos.

4. **Capa 4**. Vemos los escenarios o fases del juego. Básicamente está compuesta por el orden en el que ocurren los eventos dentro del juego como puede ser la narrativa de este mismo. El reto es generar nuevas historias automáticamente, cuya progresión se base en las decisiones del jugador. Otros ejemplos son rompecabezas, guiones gráficos, historias y niveles.

5. **Capa 5**. Encontramos el diseño del juego. Se componen básicamente de las reglas y patrones para jugar al juego.

6. **Capa 6**. Se encuentra lo que se conoce como contenido derivado o contenido que se crea a parte del videojuego en sí. Incluye imágenes y noticias sobre el nuevo juego. Interacción entre distintos jugadores si es multijugador el videojuego. Publicaciones en medios sobre las noticias recientes del juego.

Por otro lado, la clasificación más técnica[50] es la siguiente:

1. **Online u Offline**. Diferencia ente el contenido generado en tiempo de ejecución del juego contra el contenido preestablecido anteriormente que se crea antes del comienzo de un juego. Por ejemplo, un juego en el que los niveles se van creando conforme se van superando, de esta forma el estilo de juego varia siendo el mismo juego. Por otro lado, por ejemplo, un entorno generado proceduralmente que se utiliza como escenario constante del juego.

2. **Necesario u Opcional**. Distingue el contenido que es necesario u obligatorio para alcanzar un objetivo frente a un contenido que puede ser simplemente decorativo. Por ejemplo, la llave que conduce a un enemigo de una mazmorra frente a una textura de una pared de la mazmorra.

3. **Grado y dimensiones del control**. Añade el control sobre el nivel de generación de contenido a través de unos parámetros modificables por el usuario. Un ejemplo de esto puede ser la temperatura de una red neuronal, variable utilizada para dar aleatoriedad a los resultados generados.

4. **Genérico o Adaptativo**. El contenido genérico es aquel que siempre va a ser generado de la misma forma, mientras que en el adaptable alguno de sus parámetros se va modificando conforme ocurren diferentes hechos. Por ejemplo, los niveles de los personajes de dos jugadores pueden ser distintos en una misma zona del juego debido a las decisiones tomadas anteriormente.

5. **Estocástico o Determinístico**. Diferencia entre el contenido creado mediante algoritmos deterministas que producen el mismo resultado, siempre que se den los mismos parámetros frente a algoritmos estocásticos que crean un contenido diferente cada vez.

6. **Constructivo o Generar y probar**. Crear contenido en una sola pasada frente a generar un contenido y probarlo continuamente de tal forma que se va mejorando.

7. **Generación automática o autoría mixta**. Generación de contenido totalmente producido por un algoritmo frente a un algoritmo que tenga en cuenta ciertos parámetros de entrada elegidos por el jugador. De esta forma puede cambiar el comportamiento del proceso de diseño.

Además de estas taxonomías, se pueden agrupar según el tipo de algoritmo usado para la generación de dicho contenido.

1. Generación de números pseudoaleatorios.
2. Gramáticas generativas.
3. Filtrado de imágenes, morfología binaria y filtros de convolución.
4. Algoritmos espaciales, mosaico y estratificación, subdivisión de cuadrícula, vectorización, fractales y diagramas de Voronoi.
5. Modelado y simulación de sistemas complejos.
6. Inteligencia artificial como algoritmos genéticos o redes neuronales artificiales.

4.5. PCG y aprendizaje automatico

En los tipos de algoritmos que hay para la generación procedural de contenido, encontramos uno, que es el que nos lleva a nuestro trabajo. Es el uso de aprendizaje automático para la generación de contenido, PCGML o *procedural content generation via machine learning*.

Es una técnica relativamente nueva a la hora de generar distintos tipos de contenido para videojuegos. La mayoría se basan en replicar los diseños ya existentes para que el jugador obtenga niveles infinitos de un juego con variaciones únicas entre ellos, pero manteniendo la impresión de que han sido creados por una persona. Otro enfoque que tiene la generación procedural mediante aprendizaje automático es la creación de mecánicas para un juego haciendo que el propio sistema PCGML sea un adversario contra el que jugar o una ayuda en la que apoyarse.

Como hemos mencionado, el PCGML puede ofrecer una gran cantidad de contenido tanto en entidades del juego como en las propias mecánicas de este. Generalmente, el uso de PCG ofrece una repetición del juego. Sin embargo, esto va a variar significativamente con la ayuda del aprendizaje automático, por ejemplo, un jugador podría volver a jugar un juego con un comportamiento diferente debido a que los eventos del juego han variado y por tanto sus acciones han sido distintas. Otra ventaja del aprendizaje automático es la adaptación de dificultad entre un nivel y el jugador.

Un ejemplo de este caso sería que el sistema podría funcionar a favor o en contra del jugador, utilizando un tipo u otro de arma en función de cómo evolucione el mismo y por tanto poder aumentar o disminuir la dificultad al momento. Otra ventaja es la conexión emocional con el propio jugador. Un ejemplo de esto es el efecto Tamagotchi que surgió del famoso juego de incubación de mascotas. Básicamente un sistema PCGML que necesita ser atendido a lo largo del juego generando así recuerdos positivos en los jugadores.

4.6. Desarrollo

Es recomendable atenerse a los objetivos que se quieren lograr, así como a sus propiedades requeridas cuando se desarrollan algoritmos sobre todo si están destinados a la generación de contenido. Se puede perder fácilmente el objetivo y los factores cruciales durante el desarrollo[50] y esto puede provocar una mala experiencia en los jugadores.
Algunos de los factores más críticos son:

1. **Velocidad**. Tanto si un algoritmo PCG produce contenido durante el juego como si lo generó antes del propio juego, nunca debe excederse con la cantidad de tiempo que necesita para la generación del contenido, ya que podría afectar a la experiencia del jugador.

2. **Fiabilidad**. Algunos algoritmos crean contenido desde cero sin saber lo que están creando mientras que otros son capaces de crear y evaluar su contenido. Esto es muy crucial sobre todo si lo que se está generando es el nivel que se va a jugar. Quizás la fiabilidad no es tan necesaria si el contenido a generar es meramente estético.

3. **Controlabilidad**. Es una propiedad básica en cuanto al PCG, ya que otorga una gran ventaja, tener el control para poder definir ciertos aspectos del contenido generado. Si que es cierto que esta propiedad puede interferir en ciertas ocasiones con la fiabilidad, ya que el usuario está modificando a su gusto ciertos parámetros que a lo mejor no deberían distar de los valores por defecto.

4. **Expresividad y diversidad**. Es necesario desarrollar algoritmos que creen contenido con expresividad y diversidad tal y como lo haría una persona.

5. **Creatividad y credibilidad**: Al igual que la expresividad y diversidad, es necesario que el algoritmo produzca un contenido creíble que parezca diseñado por el hombre. El objetivo es que los jugadores no puedan distinguir entre un contenido generado por algoritmos y uno completamente diseñado por personas. Al fin y al cabo, es lo que se quiere conseguir, imitar el proceso creativo humano, pero a partir de un algoritmo.

Además de las anteriores características es recomendable que los algoritmos de PCG sean simples y estén centrados en generar un contenido específico. No tratar de, con un único algoritmo generar elementos muy dispares. La idea es tener distintos algoritmos, una para cada contenido que se desee, y después mezclar los resultados. Tampoco es conveniente que los jugadores sean abrumados por muchos algoritmos PCG interactivos.

4.7. Conclusión

Como hemos podido observar a lo largo del desarrollo de este tema, cuando pensamos en generación procedural, nos referimos casi siempre al mundo de los videojuegos, ya que es algo muy visual y atractivo para el usuario. Sin embargo, la generación procedural puede aplicarse en otros campos como el cine[51], donde

se utiliza para crear rápidamente espacios atractivos y precisos. A parte del cine, también nos encontramos un acercamiento a la generación procedural en el mundo de la música, con la denominada música generativa[52]. Este término fue popularizado por Brian Eno, y se refiere a ese tipo de música que está en constante cambio y es siempre diferente, la cual es creada por un sistema. Sin embargo, a parte de todo esto, la generación procedural podría ser un término que podría ser aplicable a cualquier ámbito. Por ejemplo, en el ámbito de la arquitectura[53], un arquitecto podría servirse de la generación procedural, para crear edificios, y concluir sobre ciertas ideas para el posterior diseño de una nueva estructura, o para la planificación de una ciudad, etc. El uso de contenido generado proceduralmente en un juego ofrece muchas posibilidades, como hemos estado viendo. El hecho es que el uso de modelos de PCG en los juegos es una forma de que los jugadores puedan experimentar el juego de una forma nueva cada vez que se juega. Además, tiene el potencial de reducir significativamente los costes y tiempo de desarrollo, ya sea ayudando a los diseñadores o generando automáticamente el contenido. Esto es particularmente marcado en el desarrollo de secuelas o *reskins*. Un juego *reskin* es básicamente un juego donde el desarrollador ha cogido un juego anterior y le ha cambiado aspectos como los gráficos o la temática, pero el juego sigue siendo el mismo. Las mecánicas y el núcleo del videojuego es el mismo que el anterior. Es una práctica muy habitual en el desarrollo de juegos para móviles, donde la compañía lanzará varios juegos que parecen diferentes pero que comparten el mismo código. Por tanto, la parte que cambia como pueden ser texturas, mapas o niveles se generan automáticamente lo que supone la producción de un nuevo juego con costes prácticamente nulos. Además, el valor del nuevo producto se incrementa debido a tener un volumen de contenido mayor. Una habilidad que nunca puede ser imitada por los diseñadores humanos, es la capacidad de generar contenido adaptado a cada jugador específico, y hacerlo en tiempo de juego. El contenido personalizado puede llegar a ser un cambio en la industria del videojuego como lo fue en su momento los juegos en línea o como esta siendo actualmente la realidad virtual. Por tanto, la generación procedural de contenido para el mundo de los videojuegos presenta muchas ventajas siempre y cuando se realice de forma correcta. Además, exige una gran cantidad de conocimientos y tener en cuenta numerosos factores para poder lograr una buena experiencia. En caso contrario, los resultados pueden ser desastrosos.

Capítulo 5

Aplicación Práctica

5.1. Introducción

En el capítulo anterior, hemos estado hablando sobre qué es la generación procedural de contenido, qué técnicas utiliza y que aplicaciones tiene en un mundo tan tecnológico y avanzado como en el que vivimos. Debido a todo ello, consideramos que este concepto era algo muy interesante sobre lo que investigar y trabajar. En un primer momento tuvimos la idea clara de que queríamos relacionarlo con el mundo de los videojuegos, y sobre todo con el Super Mario, uno de los juegos más aclamados de toda la historia. Actualmente existen herramientas como MarioMaker para la creación de mapas basados en este juego. Sin embargo, nosotros quisimos realizar una aplicación en Unity con la que el ordenador aprendiera como eran los mapas originales de Mario para así poder crear modelos o estructuras a partir de las cuales generar nuevos mapas del Mario tanto monotemáticos, como multitemáticos, consiguiendo así la mezcla de artes entre los diferentes mapas.

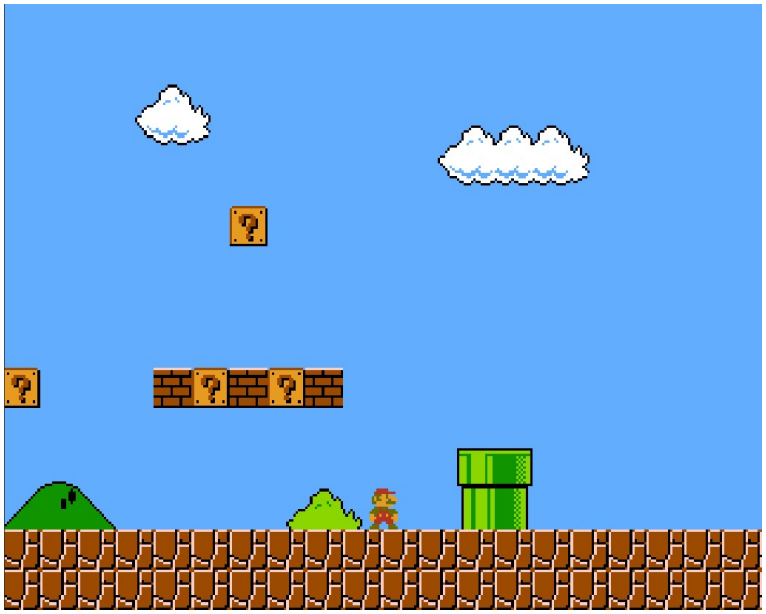

Figura 5.1: Fragmento del mapa 1 original del primer mundo

En la sección 5.2 explicaremos más a fondo el juego Super Mario, su historia, personajes jugables, *items* o transformaciones. En la sección 5.3 hablaremos sobre los retos que lleva consigo haber realizado esta aplicación y cuales son los resultados

esperados. Finalmente, en la sección 5.4 comentaremos como hemos interpretado los datos para poder representar cada *tile* del mapa en Unity y para su uso en los algoritmos de aprendizaje. Finalmente, la sección 5.5 corresponde a los algoritmos utilizados para la generación de mapas: n-gramas y RNN(redes neuronales recurrentes).

5.2. Nuestro Juego: Super Mario Bros

Resumen

Super Mario Bros es un videojuego publicado para la Nintendo Entertainment System (NES) en 1985. Este juego cambió la forma de jugar de todos sus juegos *arcade* predecesores, e instauró los juegos de plataformas laterales. No es el primer juego de la franquicia de Mario, sin embargo, es el más icónico. Introdujo una gran diversidad de elementos, desde *power-ups* hasta diversos tipos de enemigos con la premisa de rescatar a la Princesa Toadstool (Peach) del rey Koopa (Bowser). En nuestro proyecto hemos recreado un prototipo del Super Mario Bros, no todas las características del juego original han sido implementadas. Esta implementación nos ayuda a ver los mapas creados y realizar una pequeña interacción con ellos, pero en ningún caso nuestra intención es la de implementar un videojuego como tal. Todos los *sprites* han sido obtenidos del siguiente enlace [54].

Historia

Un día el Reino Champiñón fue invadido por los Koopa, una tribu de tortugas que podían utilizar magia. Utilizaban su magia para transformar a las personas del Reino Champiñón en objetos inanimados como consecuencia el reino cayó. Solamente la princesa Peach puede deshacer la maldición. Y restaurar la normalidad pero Bowser la mantiene cautiva. Mario escucha de la situación de la princesa y se embarca en una aventura para derrotar a la tribu Tortuga y devolver la paz al reino.

Descripción

Nuestro Super Mario Bros contiene los dos primeros mundos originales, cada uno de ellos contiene cuatro niveles. Sin embargo, puedes generar infinitos niveles a partir de los ocho originales.

Mario tiene que conseguir llegar hasta el final del nivel. También hay tuberías a lo largo del camino, en alguna de las cuales puedes entrar y visitar varias habitaciones secretas con monedas y volver más adelantado al nivel. Mario puede obtener *power-ups* como setas mágicas o monedas de los bloques mágicos. Con la seta mágica Mario se transforma en Super Mario, con lo que puede destruir los ladrillos.

Los ocho niveles originales a partir de los cuales puedes generar infinitos mundos son, tres principalmente sobre el suelo, uno subterráneo, dos de castillo, uno submarino, y uno que mezcla el suelo, la zona subterránea y el cielo.

Controles-Mecánicas

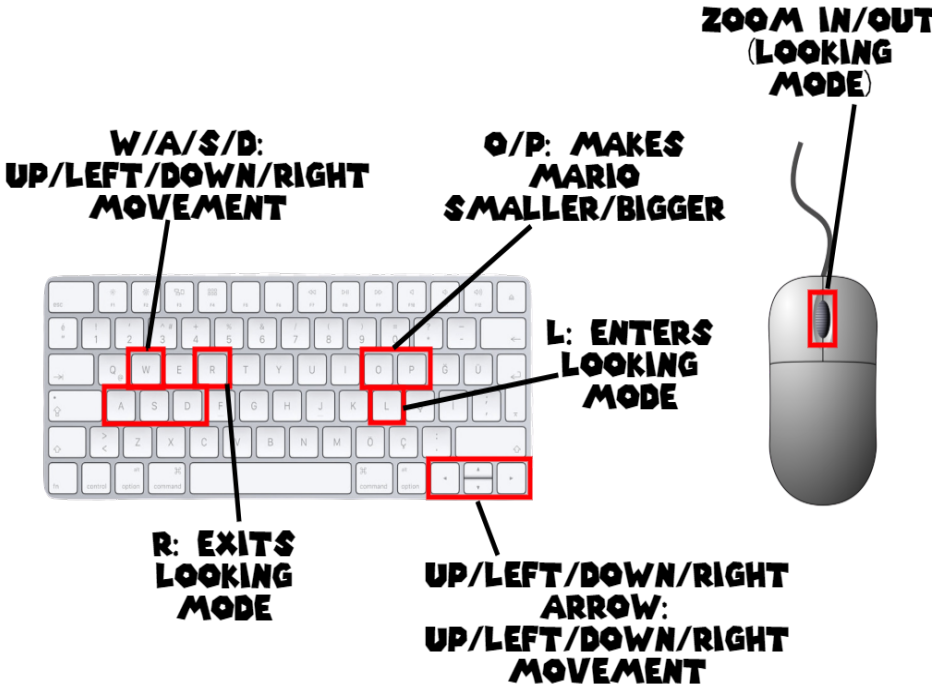

Figura 5.2: Controles del juego

El paso y la salida de las zonas secretas también se realizará a través de las flechas de movimiento.

Características

Género	Plataformas 2D lateral
Modos	Modo 1 jugador
Plataformas	PC: Aplicación
Publico Objetivo	Cualquier edad: PEGI3
Personajes Jugables	**Mario**: Es el personaje principal del videojuego y la mascota de Nintendo 
Objetos	**Monedas**: Coleccionables  **Seta Mágica**: Otorga la transformación de Super Mario 
Transformaciones	**Mario**: Forma inicial del juego. Es la forma más débil.  **Super Mario**: Requiere de una seta mágica. 
Enemigos	**Goomba**: Enemigo más débil del juego.  **Green Koopa Troopa**: Criatura parecida a una tortuga, las hay de diferentes colores, rojas y verdes. 

Tabla 5.1: Tabla de características de Super Mario

5.3. Retos

El objetivo principal como hemos comentado anteriormente en la introducción, es el de generar mapas del SMB para un diseñador y no para un jugador casual de videojuegos, ya que incluye una jugabilidad muy limitada. La aplicación permite por un lado mostrar los mapas originales del mundo 1, del mundo 2 y todos los generados por el usuario. Por otro lado, permite generar nuevos modelos de entrenamientos para la creación de nuevos mapas, o utilizar modelos ya existentes, para la generación también de nuevos mapas. Estos entrenamientos, pueden estar basados en un solo mapa, o en varios mapas, lo que implicará que los mapas generados a partir de ese *training*, serán una combinación de los mapas seleccionados en el primer paso. Por

todo ello, pensamos que nuestra aplicación podría ser utilizada en juegos basados en el Super Mario, cuyo propósito fuera la creación automática de un nuevo mapa cuando el jugador superara el nivel actual, o para diseñadores que quisieran adoptar ideas para la generación de nuevos niveles.

Hacer esto ha supuesto un gran reto, ya que hemos tenido que generar de cero esos 8 mapas originales del SuperMario a través de Tiled, colocando cada *tile* a mano. Tras ello, otro reto fue la lectura de cada mapa para poder procesar su información y su posterior guardado. Otro problema surgió, cuando quisimos conectar los *scripts* de Python, los cuales contienen los algoritmos de *machine learning* comentados en la sección 5.5. Para ello, finalmente, conseguimos conectar los *scripts* de Python a través de la creación de procesos en Unity. Estos procesos se basan en lanzar un terminal y ejecutar los *scripts* con los algoritmos de aprendizaje automático en un hilo a parte. Sin embargo, estos problemas fueron algo menor comparados con los ocurridos durante el desarrollo de los algoritmos de aprendizaje automático, los cuales serán vistos más adelante.

En cuanto a la jugabilidad, dentro de la existente en la aplicación, los dos retos más destacables fueron el de conseguir que Mario interaccionara con las tuberías para poder acceder a las zonas subterráneas/acuáticas. Para ello, lo que hicimos fue fijarnos en aquellas tuberías que en su vertical tenían una tubería de salida de una zona secreta, asignando a la inmediatamente anterior, la función de acceso a la zona secreta. Por otro lado, la interacción con las enredaderas o *beanstalk* para acceder a las zonas de cielo. Además, se pueden romper bloques, coger monedas o enfrentarte a dos tipos enemigos.

Finalmente, en lo que respecta a los retos planteados y cumplidos con los algoritmos de aprendizaje automático, nos surgieron sobre todo en cuanto a esos mismos algoritmos. Esto será explicado a fondo en la sección 5.5.

5.4. Tipos de datos

En nuestro proyecto creamos niveles para el juego Super Mario Bros de la NES. Para lograr esto seguimos un proceso en el que utilizamos varios tipos de datos a nuestro favor a partir de imágenes de los niveles originales.

A partir de una imagen completa del nivel original sacamos un CSV con los distintos bloques. Para ello cargamos la imagen en Tiled a modo de plantilla y una hoja de *sprites* para su creación y construimos el nivel deseado. Una vez completado el mapa *tile* a *tile*, se exporta en formato CSV para que sea leído por el algoritmo elegido por el usuario.

Tanto el algoritmo n-gramas, de predicción de texto, como las redes neuronales recursivas, tratan el CSV de la misma manera. Cada columna de *sprites* del nivel es lo que más tarde en el apartado de algoritmos trataremos como una palabra, y a su vez el nivel como una frase o texto completo. Con esta interpretación se puede establecer una relación directa entre texto y mapa. La relación es que cada *slice* del juego, conjunto de bloques en vertical, es una palabra. Al igual que las palabras están formadas por letras, los *slices* están formados por bloques. Un nivel está formado

por un conjunto de *slices* uno detrás de otro. En el caso de un texto, este estaría formado por la sucesión de palabras. De esta forma se establece la relación entre texto y mapa.

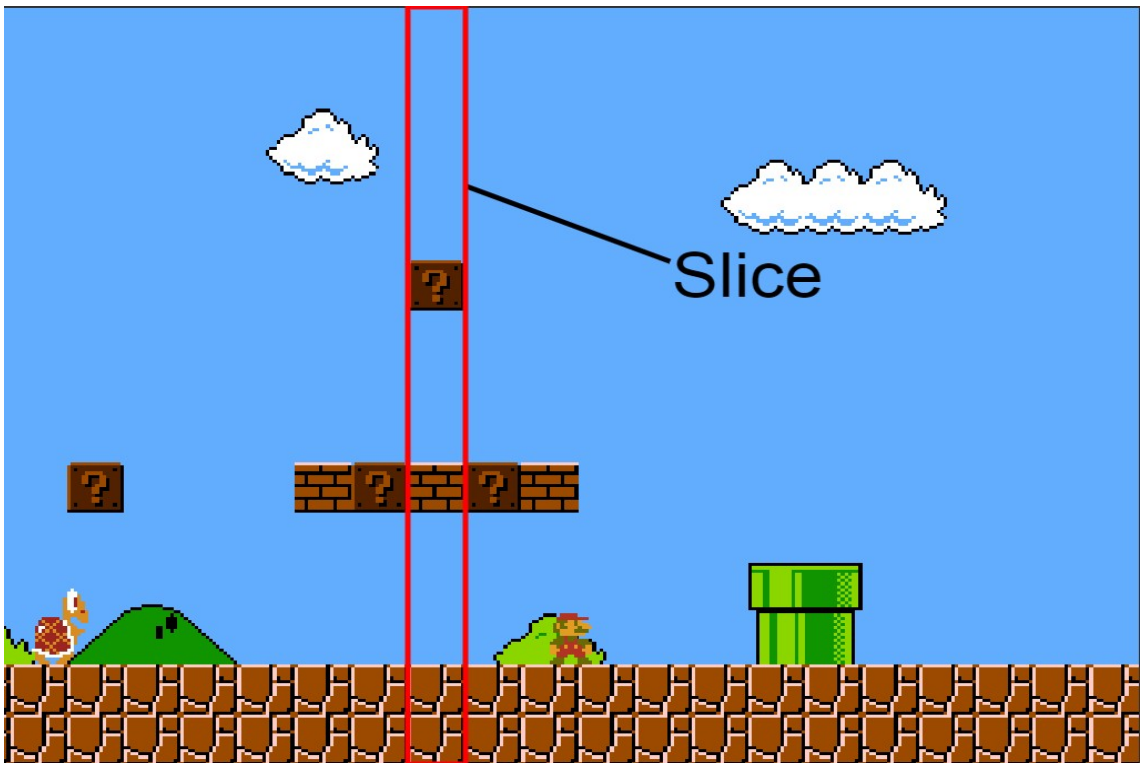

Figura 5.3: *Slice* de un nivel

Al principio los mapas eran de 15 bloques de altura por el largo que tuviesen según el mapa original. Sin embargo, tras aplicar la idea que definimos como multitema, los mapas pasan a tener 60 bloques de altura por la longitud correspondiente de ancho. Esto es debido a la oportunidad de incluir distintos temas en distintas alturas del mapa para conseguir su mezcla. Este concepto será explicado más detenidamente en los siguientes apartados.

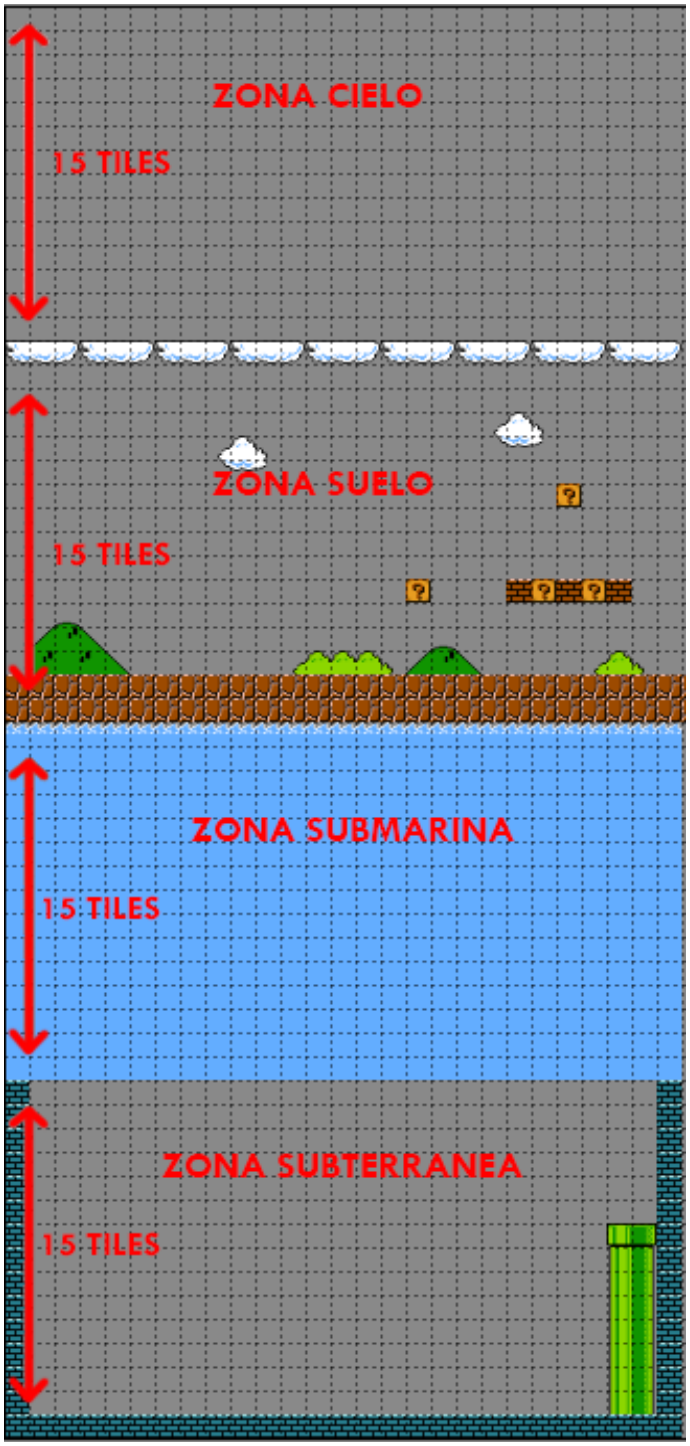

Figura 5.4: División de las zonas de los niveles en Tiled

Además, al cargar el CSV, este es transformado a una matriz de números y esta a su vez, es traspuesta para mayor facilidad de obtención de las palabras, *slices*, del nivel.

Tras la aplicación del algoritmo se crea una nueva matriz con el nuevo nivel generado. Esta matriz deberá de ser traspuesta de nuevo para generar un nuevo fichero CSV que mantenga las mismas características que el original, salvo por longitud en anchura que es la especificada por el usuario y el nombre del fichero. Tras esto, el fichero se deja en un directorio de Unity donde están el resto de archivos de mapas.

Por otro lado, otros tipos de datos que merece la pena nombrar son los *prefab*[1], una representación de los *tiles* que utilizamos para crear esos mapas en Unity. Los cuales dependiendo del tipo de bloque tienen unos u otros componentes.

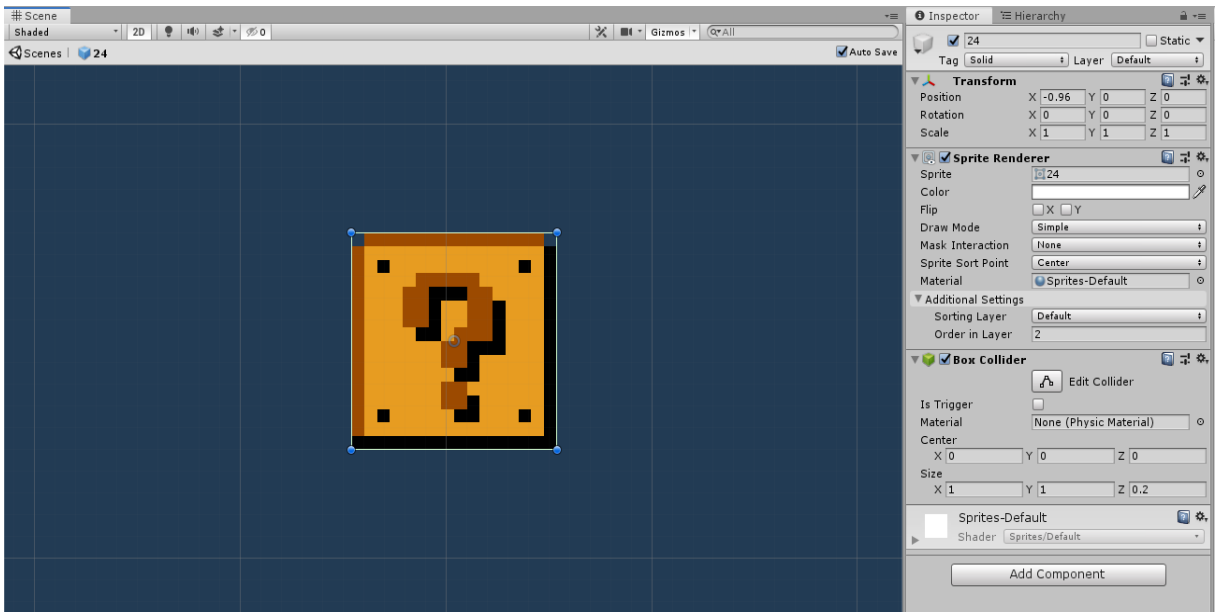

Figura 5.5: *Prefab* de un *tile*

5.5. Algoritmos

A continuación, desarrollaremos como hemos conseguido la generación de mapas de manera automática a partir de una serie de algoritmos.
Los algoritmos utilizados principalmente en nuestro proyecto han sido dos: Los n-gramas y las redes neuronales recursivas. Como breve introducción de ambos, podríamos decir que los n-gramas, nos permiten saber cuál es la siguiente subsecuencia a elegir dentro de una secuencia de n datos usando probabilidades. Por otro lado, las redes neuronales recursivas, permiten un análisis muy potente de secuencias, con una serie de conexiones arbitrarias entre las neuronas, lo cual permite crear ciclos entre las propias neuronas, dotando así a toda la red de "memoria".

5.5.1. N-gramas

Parte de este capítulo es desarrollado gracias a información obtenida de las siquientes fuentes[55] [56] [57].

Introducción

Predecir unas pocas palabras es mucho más fácil que predecir un texto entero. Por ello ir prediciendo pequeñas secuencias de palabras progresivamente ayuda a obtener el texto final. Esta tarea se puede realizar utilizando modelos que asignan probabilidades a cada posible siguiente palabra, también pueden servir para asignar probabilidades a frases enteras. Las probabilidades son esenciales en tareas en

[1]Objeto de Unity con una serie de componentes y características prestablecidas para ser reutilizado

las cuales hay que identificar palabras en textos. Estas probabilidades sirven para identificar si una combinación de palabras es más probable de aparecer que otra combinación. La asignación de probabilidades es fundamental en tareas como el reconocimiento de voz, corrección ortográfica o la traducción automática.
Aquellos modelos que asignan probabilidades a las secuencias de palabras son llamados modelos del lenguaje o LMs (su acrónimo en inglés). El LM más simple es el n-grama.

Un n-grama es una subsecuencia de n elementos escogidos a partir de un texto dado. Esta subsecuencia puede estar formada por caracteres o palabras. Debido a la ambigüedad que pueda surgir, hay que considerar que podemos usar el término n-grama para referirnos al LM o la propia secuencia de palabras.

Se puede decir que un n-grama es un *token* formado a partir del movimiento de una "ventana" a través de un texto dado, o bien llamado "corpus". La ventana se mueve creando subsecuencias dependiendo del tamaño N del *token*, de este modo, por ejemplo, si N=2 se van formando *tokens* de 2 palabras o caracteres. Dependiendo del movimiento de la ventana, los n-gramas pueden clasificarse en dos tipos: los basados en caracteres y los basados en palabras. También pueden considerarse las frases, como nosotros hicimos en una primera implementación para obtener los mapas del Super Mario Bros. Sin embargo, este último tipo carece de sentido y coherencia según nuestra experiencia. Más adelante, en el apartado 5.6.1 se expondrán estos resultados y la correspondiente explicación.

Daniel Jurasfsky y James H.Martin definen, en su libro *Speech and Language Processing*, a los n-gramas como una secuencia de N palabras. De este modo, se puede obtener un 1-grama (unigrama) como "hola", un 2-grama (bigrama) que es una secuencia de dos palabras como "hola qué", un 3-grama (trigrama) que es una secuencia de tres palabras como "hola qué tal", etc.

Historia de los n-gramas

El modelo n-grama es un modelo probabilístico ideado por Andrey Markov y más tarde fue introducido por Claude Elwood Shannon en 1948, en su teoría de la comunicación (*Information Theory*). En ella, expresaba su preocupación por el conocimiento estadístico sobre la fuente de la información. Por ejemplo, en telegrafía, se transmiten mensajes que consisten en secuencias de palabras que no son aleatorias. El conjunto de estas palabras forma frases y mantienen un orden estadístico. Esto se puede demostrar observando que hay letras que son más frecuentes que otras. En español, la letra E tiene un porcentaje de aparición del 13,68 %, mientras que la Z tiene un 0,52 %. También hay secuencias de letras que son más frecuentes que otras. Estas condiciones ofrecen la posibilidad de ahorrar tiempo si se codifican los mensajes adecuadamente en señales. En 1838, Samuel Morse optimizó, usando esta estructura estadística de letras y palabras, el lenguaje máquina que desarrolló en 1830. El lenguaje desarrollado por Morse consistía en la codificación de letras y números como la combinación de puntos, guiones y espacios. La optimización que llevó a cabo se basó en la frecuencia de las letras en inglés, es decir, las letras más frecuentes se representan con combinaciones más cortas de esos símbolos.

Los modelos que producen secuencias de caracteres estructuradas por probabilidades son llamados procesos estocásticos. En teoría de la probabilidad un proceso estocástico o proceso aleatorio es un objeto matemático definido normalmente como una familia de variables aleatorias. En estos procesos se consideran los casos en los que caracteres seguidos secuencialmente de otros son elegidos por probabilidades dependientes en los caracteres anteriores. Claude Elwood Shannon usó los n-gramas para analizar y predecir texto en inglés y sirvió de precedente para su uso en otros temas.

Google ofrece una gran cantidad de información acerca de los n-gramas más frecuentes. Esta información ha sido obtenida a través del análisis de millones de libros, más de cinco millones, publicados desde hace quinientos años. Esa información está disponible públicamente en Google Ngram Viewer[58]. Este consiste en la búsqueda de la aparición de frases, elegidas por el usuario, en un corpus de libros (por ejemplo, "Inglés Británico", "Francés") durante los años elegidos. Una vez elegidos estos parámetros se muestra un gráfico con los resultados. El gráfico representa el porcentaje de aparición de un n-grama elegido por el usuario respecto a todos los n-gramas contenidos en los libros de ejemplo. La información recolectada por Google es usada para implementar su sistema de recomendación de consultas. Se puede visitar la página web de Google Ngram Viewer: `https://books.google.com/ngrams/info` para obtener más información acerca de esta aplicación.

En esta página `https://books.google.com/ngrams/` se puede probar de manera práctica la aplicación y puede resultar interesante para analizar el uso del lenguaje durante el avance de los años.

Uso de los n-gramas

Los n-gramas ofrecen una clasificación de texto eficiente y efectiva que puede ser usada en aplicaciones de corrección o detección de errores ortográficos, identificación de idioma, etc. El uso de modelos n-grama es una idea simple, pero de gran utilidad y efectiva en muchas ocasiones.

1. **Identificación de lenguajes de programación**

   Así como los n-gramas sirven para identificar idiomas en textos, también pueden ser usados para identificar lenguajes de programación. A pesar de que los n-gramas no son un concepto nuevo, como ya se ha visto en el apartado sobre su historia, estos han surgido como una alternativa viable frente a las librerías para la detección de lenguaje natural. Este tipo de librerías están implementadas sobre una gran cantidad de extensos diccionarios en los que realizan comprobaciones de coincidencias. Debido a esta implementación, necesitan de un mayor tiempo para compilar. Además, cuanto más extensos sean estos diccionarios, mayor será el tamaño de la librería.

   También puede darse la posibilidad que se reciban palabras que no están incluidas en la librería. Por otro lado, los n-gramas son más eficaces, además de poder usar archivos más pequeños en tamaño como entrenamiento.

2. **Traducción automática**

   La traducción de texto de un idioma [59] a otro es uno de los principales objetivos del procesamiento del lenguaje natural. Sin embargo, esta tarea no es tan sencilla como asumir que una palabra se puede traducir tal cual a otra de un idioma diferente. No siempre se corresponde una palabra de un idioma con una de otro idioma, por lo que la traducción automática no se puede conseguir únicamente con el análisis sintáctico y léxico. También hay que tener en cuenta la ambigüedad del texto y de los mensajes a transmitir.
   El proceso de traducción puede definirse en dos pasos. Estos son la decodificación del significado del texto y la codificación de este significado en el idioma deseado.

3. **Filtrado de contenido**

   El filtrado de contenido es un método muy útil para extraer características estructuradas de texto desestructurado. En esencia, permite la obtención de palabras que nos aportan una gran información y contexto. Para ello, este proceso se puede llevar a cabo mediante diferentes subprocesos, tales como la eliminación de palabras vacías, la eliminación de palabras raras y la reducción de una palabra a su raíz (en inglés este método es conocido como *stemming*).
   Las palabras vacías o *stop words* son aquellas palabras que apenas aportan nada al contexto.
   Las palabras raras o *rare words* son palabras que no aparecen con gran frecuencia en un texto, por lo que pueden ser palabras mal escritas o palabras inventadas. Aportan ruido al texto.
   Muchas veces tenemos palabras relacionadas que se escriben de forma diferente, pero mantienen la misma raíz y expresan un mismo significado. Es decir, aquellas pertenecientes a la misma familia de palabras. Un ejemplo sería el siguiente: flor, flores, florecilla, floricultura, florería, florista, florecita, floreado... Todas se escriben de una manera distinta, pero están relacionadas con las propias flores. La reducción a su raíz o *stemming* consigue cortar las palabras y obtener su raíz.
   La bolsa de palabras o *Bag of Words* (BOW)[60] es el método más simple para convertir un texto en características estructuradas.

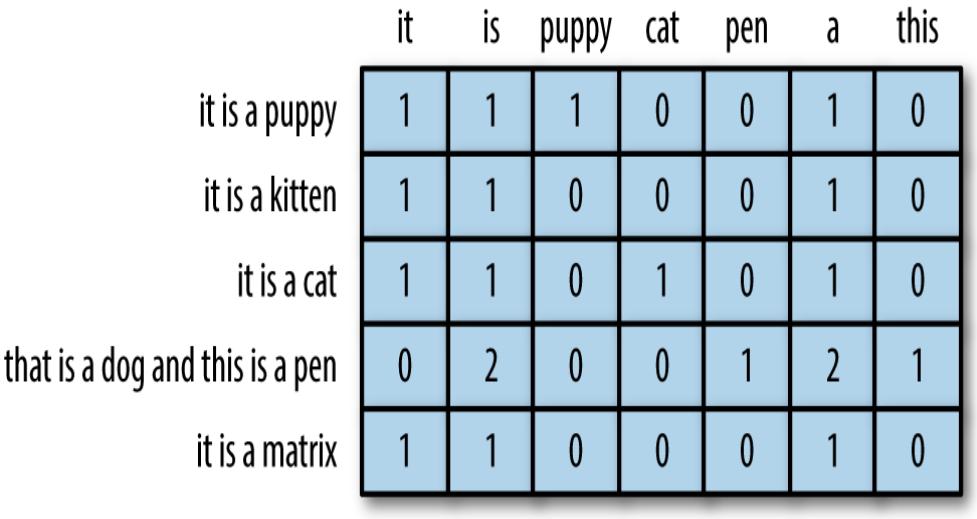

Figura 5.6: Conversión de texto a su representación en bolsas de palabras

4. **Detección de plagio**

   Los n-gramas también sirven para detectar un posible plagio realizado sobre un texto. De este modo, el texto sospechoso es comparado con ciertos textos usados como corpus. Según un estudio realizado por Alberto Barrón-Cedeño y Paolo Rosso (Universidad Politécnica de Valencia) se determina que los mejores resultados para detectar casos de duplicados son obtenidos mediante bigramas y trigramas.

   Esta tarea es más sencilla de llevar a cabo cuando el plagio se realiza de manera exacta, sin embargo, resulta más compleja cuando se producen cambios en el texto original. Por ello, se lleva a cabo una comparación mediante n-gramas. Textos independientes tienen una pequeña cantidad de n-gramas en común, lo que facilita la detección de plagios, puesto que esta cantidad de n-gramas comunes será mayor. La probabilidad de encontrar n-gramas comunes en textos diferentes disminuye cuando la n aumenta (Alberto Barrón-Cedeño y Paolo Rosso).

5. **Corrector automático**

   El modelo n-grama puede ser empleado para la construcción de aplicaciones de detección y corrección de errores ortográficos, para el autocompletado de palabras y para ofrecer sugerencias como continuación a una palabra.

   El proceso para realizar estas tareas se denomina corrección ortográfica o *spell checker*. Se detectan y proporcionan sugerencias para palabras incorrectas. Los algoritmos para implementar estos procesos se basan en diccionarios que contienen vocabulario con el que comprobar las palabras a corregir. Cuanto mayor sea este vocabulario mayor será el porcentaje de detección de errores. Sin embargo, un problema común es la falta de vocabulario y datos que contienen los diccionarios usados por estos algoritmos.

Modelo del lenguaje n-grama

El n-grama es el modelo del lenguaje más simple. Para recordar, se denomina modelo del lenguaje a aquellos modelos que asignan probabilidades a secuencias de palabras o caracteres de tamaño n. Permite clasificar texto desconocido, es decir, no usado para entrenar previamente, con gran efectividad y certeza.

Este modelo es capaz de predecir la siguiente palabra a una secuencia de palabras previas. Para lograr predecir palabras se debe calcular la probabilidad de una palabra p dada una secuencia de palabras anteriores, es decir, dada una historia. La manera de conseguir estimar esta probabilidad es mediante el conteo de frecuencias relativas que consiste en contar las veces que una palabra u otras (historia), dependiendo del valor de n para el n-grama, es seguida por otra. Un ejemplo, sería contar en un corpus las veces que la frase "me gusta el" es seguida por "café". La ecuación para esta probabilidad es:

$$P(\text{cafe}|\text{me gusta el}) = \frac{C(\text{me gusta el cafe})}{C(\text{me gusta el})} \tag{5.1}$$

Probabilidad condicional a la historia

Esta forma de estimar probabilidades a través del conteo de las apariciones de una palabra respecto a otras es eficaz en muchas ocasiones, sin embargo, no contamos con corpus lo suficientemente grandes como para estimar de manera correcta la gran mayoría de las veces. También se podría descomponer la probabilidad de una secuencia de palabras usando la regla de la cadena de probabilidad:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)...P(w_n|w_1^{n-1}) \tag{5.2}$$

Regla de la cadena de la probabilidad

Se multiplican las probabilidades condicionales de cada palabra de la secuencia, pero por la misma razón anterior no se puede calcular la probabilidad exacta de una palabra respecto a una secuencia. Por ello, los n-gramas calculan la probabilidad aproximada de una palabra respecto a unas pocas palabras anteriores. De este modo, siguiendo el ejemplo anterior en lugar de calcular la probabilidad de que a "me gusta el" le siga "café", un bigram calcularía la probabilidad de que a "el" le siga "café": P(café | el)

Las probabilidades se estiman usando el método *máximum likelihood estimation* (MLE), por el cual se realiza el conteo C de todos los n-gramas (secuencias de palabras de tamaño n) y se normaliza por la suma de todos los n-gramas que comienzan por esa secuencia de longitud n-1. La ecuación es la siguiente:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}{}^{w_n})}{C(w_{n-N+1}^{n-1})} \tag{5.3}$$

Maximum likelihood estimation (MLE)

Debido a que los datos de entrenamiento no comprenden a todos los n-gramas posibles en un idioma, puede producirse el hecho de encontrar n-gramas que no han aparecido previamente en los datos de entrenamiento, asignándoles entonces una probabilidad de cero. Una probabilidad de cero en un n-grama produce que toda la frase tenga una probabilidad de cero. Para solucionar este problema y ser capaz de calcular una probabilidad realista se usa el *smoothing*. Existen diversos métodos de suavizado como el *add-1*, *add-k*, *stupid backoff* y *Kneser-Ney*. Estos modelos tienen que ser entrenados con grandes corpus para permitir realizar una mejor predicción.

Proceso de desarrollo

En primer lugar, empezamos investigando acerca de este modelo del lenguaje, en qué consistía, su utilidad, modo de empleo para una posible implementación, características, etc. Posteriormente descubrimos que este algoritmo ya había sido utilizado para la creación de niveles de SMB en el artículo *Linear levels through n-grams*[2]. Esto nos impulsó a, para su entendimiento, realizar nuestra propia implementación e inicio del estudio de la generación procedural. Acto seguido estudiamos la teoría sobre los n-gramas con el objetivo de obtener el conocimiento necesario para poder

[2] `http://julian.togelius.com/Dahlskog2014Linear.pdf`

desarrollar el modelo.

Utilizamos como base de estudio el capítulo *N-gram Language Model* del libro *Speech and Language Processing* de Dan Jurafsky y James H.Martin, profesores de la Universidad de Stanford y de la Universidad de Colorado. Este capítulo explica de una manera amplia el modelo n-grama, así como métodos de mejora, métodos de evaluación de la validez del modelo y posibles problemas. Además, plantea una serie de ejercicios bastante útiles para poner en práctica los conceptos adquiridos.
A partir de la teoría comenzamos a realizar casos prácticos para entender completamente el modelo de n-gramas. Para ello a partir de la ecuación general calculamos las ecuaciones para calcular las probabilidades de los distintos n-gramas dependiendo de su tamaño n. De esta manera conseguimos entender los distintos parámetros de la ecuación, así como la forma en la que influyen estos a la misma.

Una vez hecho esto, nos planteamos a partir de un pequeño texto de ejemplo calcular las probabilidades de cada bigram extraído del texto.

Para obtener la probabilidad de un n-grama se realiza un conteo de la frecuencia con la que aparece el n-grama en un corpus y se divide por las veces que se repite la secuencia, de tamaño n-1, previa de ese n-grama. Para el caso general, como ya se ha mostrado previamente, la ecuación es la (5.3).

Más tarde, extrajimos la ecuación para los trigramas y obtuvimos las probabilidades para los trigramas distintos de cero en un corpus de ejemplo. El corpus en ambos casos era el siguiente:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Las etiquetas *<s>* y *</s>* se emplean para indicar el inicio y fin de una frase. También sirve para cuando se empiezan a construir los n-gramas desde el principio de un texto, ya que no se puede recorrer atrás en la historia. Así, como no existen más palabras hacia atrás en el inicio del texto, pues, como es lógico, antes de la primera no puede haber más palabras, se rellenan con las etiquetas de inicio *<s>*. Por ejemplo, el cálculo para un trigrama al principio del texto anterior sería P(I|<s></s>), el aumento de n supone la adición de más etiquetas.

Otro ejercicio realizado con el objetivo de dominar el modelo de n-gramas fue el de calcular la probabilidad de la frase "I want chinese food" de dos formas distintas, una de ellas de forma estándar, la otra, añadiendo un método de suavizado.

1. Usando las probabilidades útiles, sin modificaciones:

$$\begin{gathered} \text{P(<s>i want chinese food </s>)} = \\ \text{P(i|<s>)P(want|i)P(chinese|want)P(food|chinese)P(</s>|food)} = \\ 0.25 \text{ x } 0.33 \text{ x } 0.0065 \text{ x } 0.52 \text{ x } 0.68 = 0.0002 \end{gathered}$$

2. Usando el método *add-1* de suavizado: este método determina que se debe sumar 1 al conteo de todos los n-gramas para evitar probabilidades de 0 y

conseguir unas probabilidades más realistas más allá del corpus de entrenamiento. La fórmula sería:

$$P_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \tag{5.4}$$

Fórmula del suavizado add-k

donde V es el vocabulario del corpus, es decir, el número de palabras únicas que aparecen, sin contar repeticiones. Para el caso *add-1* (*Laplace smoothing*) se sustituiría k por 1. De este modo la probabilidad suavizada usando el método *add-1* sería:

P(<s>i want chinese food </s>=
P(i|<s>)P(want|i)P(chinese|want)P(food|chinese)P(</s>|food) =
0.19 x 0.21 x 0.0029 x 0.052 x 0.4 = 0.000024

Como se puede observar, las probabilidades sin suavizar son mayores, puesto que para suavizar se da cierto porcentaje a aquellos bigramas que tenían una probabilidad de cero lo que produce que las probabilidades restantes se reduzcan. Sin embargo, de esta forma las probabilidades obtenidas pueden ser consideradas más realistas, ya que el resultado obtenido no está tan ajustado al ejemplo de entrenamiento.
Los datos utilizados para la realización del ejercicio son proporcionados en el PDF del libro comentado anteriormente, *N-gram Language Model* del libro *Speech and Language Processing* de Dan Jurafsky y James H.Martin, asumiendo algunas probabilidades tal como se indican en el ejercicio.

Tras afianzar nuestro conocimiento sobre el modelo de n-gramas con pequeños ejercicios sobre un corpus reducido, pasamos a la implementación de nuestro primer programa para calcular unigramas y bigramas, en el cual leemos un archivo de texto y guardamos en un diccionario pares que contienen una clave, la palabra, y el valor, la frecuencia de aparición de esa palabra. Además, añadimos las etiquetas de inicio (*<s>*) y final de frase (*</s>*). Después, extraemos los n-gramas y calculamos una matriz de probabilidades a la que accedemos usando índices que se corresponden con cada palabra del bigrama. Rellenamos un diccionario con los n-gramas y sus probabilidades siendo la clave la secuencia de tamaño n-1 y valor la probabilidad calculada previamente y guardada en la matriz de probabilidades. Las columnas y filas de la matriz de probabilidades están formadas por las palabras que componen los bigramas y en cada celda, el valor de la probabilidad del bigrama formado por dichas palabras. En un primer lugar, sacamos las probabilidades sin suavizar, es decir, pueden aparecer bigrams en la matriz con probabilidad 0. Más tarde, añadimos la posibilidad de usar el método de suavizado *add-k*. Con esta primera implementación generamos una serie de textos.

Seguidamente, adaptamos esta implementación a la generación de mapas de SMB. Para ello, los datos de entrenamiento se cambiaron siendo ahora utilizados

archivos CSV. Los creamos mediante Tiled, que nos permite colocar *tiles* a nuestro gusto y exportar los niveles creados en formato CSV. Este archivo CSV representa cada *tile* como un número real. Gracias a esta conversión de *tile* a número podemos obtener nuestros datos con los que trabajamos. Los niveles usados como corpus de entrenamiento son los niveles originales de SMB del mundo 1 y 2. En la siguiente imagen, se puede observar la representación del nivel de la imagen troceada anterior.

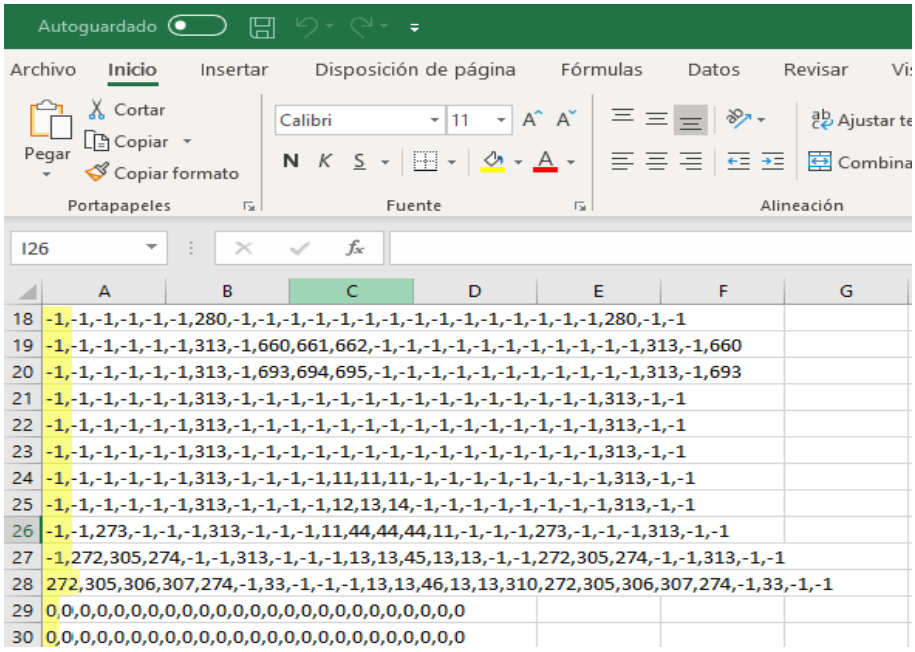

Figura 5.7: Excel de un mapa.

Durante este proceso comenzamos a tener problemas a la hora de generar secuencias. Además de problemas de rendimiento debido a la matriz de probbilidades, ya que había aumentado considerablemente al pasar del texto de ejemplo a un mapa de Super Mario Bros.

La siguiente gran modificación que hicimos fue usar la librería NLTK (*Natural Language Toolkit*) que nos permite crear las secuencias de n-gramas y nos evita crearlas nosotros mismos. En lugar de calcular todas las probabilidades en una matriz, generamos un diccionario donde la clave es la secuencia de palabras de tamaño n y el valor es una lista de las siguientes posibles palabras, de tal forma que la probabilidad está implícita en la lista, es decir si las siguientes posibles palabras son [azul, rojo, azul, azul, verde, rojo] al hacer un *random* sobre esa lista de palabras, directamente se elige la palabra resultante basándonos en las probabilidades implícitas según el número de veces que aparece una palabra en la lista. Además, escogemos la primera secuencia a partir de la cual generar los mapas y de este modo no usamos las etiquetas $<s>$ y $</s>$.

Gracias a este cambio pudimos generar de una manera más fácil y eficiente cualquier n-grama, no únicamente bigramas, pues no dependemos de la matriz de probabilidades que para cuatrigramas sería gigante, por ejemplo. Y es que el problema en espacio era importante, pero era aún más el problema del tiempo que consumía el rellenar dicha matriz de probabilidades. Con el cambio al diccionario, el tiempo de ejecución del algoritmo se redujo drásticamente. Este cambio supuso una nueva implementación desde cero.

Tras añadir la libreria NLTK, debíamos pensar cómo tratar los niveles de SMB para obtener una representación de los datos que pueda ser usable por los n-gramas. Los n-gramas clasifican el lenguaje mediante letras o caracteres o palabras, por lo que necesitamos representar de manera equivalente los niveles de SMB. Esto puede ser pensado como la creación de un nuevo idioma con nuevas letras y palabras. Este nuevo "idioma" es la base para el funcionamiento de los n-gramas. Una vez definido el objetivo al que llegar (crear el nuevo "idioma"), pensamos en las posibilidades para alcanzarlo:

1. Dividir los niveles horizontalmente en secuencias de bloques (*tiles*), de ancho la longitud del nivel y de alto un bloque. El nivel se iría recorriendo por secuencias horizontales de arriba a abajo.

2. Dividir los niveles verticalmente en secuencias de bloques, de ancho un bloque y alto la altura del nivel.

Finalmente, decidimos usar la segunda opción, pues imita mejor al lenguaje mediante la lectura de izquierda a derecha y aporta un mayor vocabulario y más enriquecido porque el ancho de los niveles es mucho mayor que el alto y existirían más secuencias de bloques. Es decir, de esta manera existirían más unidades de datos lo que ofrece una mayor variedad al lenguaje.

Al principio, tuvimos en cuenta cada bloque como si fuese una palabra, por lo que cada *slice* sería una frase. De esta forma al aplicar el algoritmo lo que se iba generando eran bloque tras bloque hasta completar un *slice* y pasar al siguiente. Esta representación no era buena y los resultados obtenidos muy malos, aunque en ellos se apreciaba un cierto orden entre los bloques, pero únicamente a nivel interno de las *slices*. Por ejemplo, se puede ver el poste típico del final de los niveles, ya que esos bloques que lo forman, son la única opción que dan los n-gramas a la secuencia anterior. Es decir, no existen más combinaciones de esos bloques en los niveles de SMB y las posibilidades de formarlo son muy altas. Sin embargo, con otros bloques puede haber muchas más combinaciones por lo que realizar los n-gramas a nivel de los bloques puede resultar muy arbitrario.

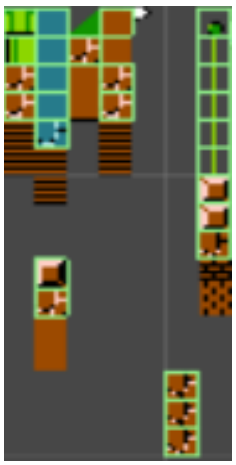

Figura 5.8: Bloques como palabras y *slices* como frases.

Las siguientes imágenes serían las combinaciones del bloque suelo con otro bloque en el nivel 1-1, es decir, para la obtención de bigramas:

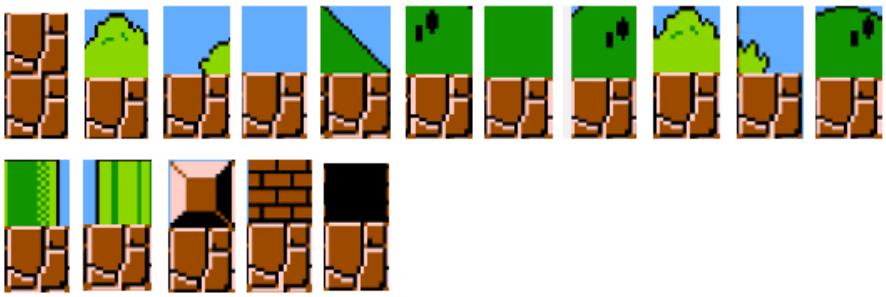

Figura 5.9: Bloques como palabras y *slices* como frases.

Este problema es equivalente a si cogiésemos de un texto, por ejemplo, bigramas formados por letras. Al final, contaríamos con muchísimos bigramas compuestos por casi todas las combinaciones de letras posibles y sería extremadamente difícil formar nuevas palabras, que de verdad existan. De esta forma se dieron malos resultados, ya que cualquier n-grama de *tiles* podía estar seguido de otro, en caso de una n pequeña. Carecía de coherencia y consistencia.

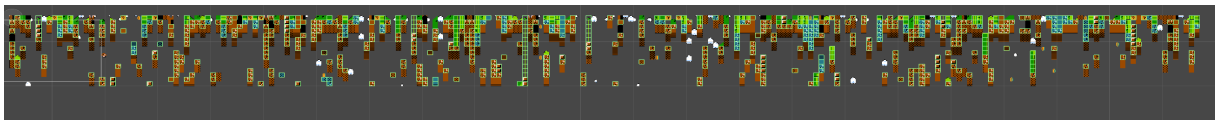

Figura 5.10: Cada *tile* (bloque) considerado como una palabra

Para solucionar esto tratar los bloques como letras, las columnas de bloques o *slices* verticales como palabras (son un conjunto de *tiles*, es decir, letras) y el nivel entero será el conjunto de palabras (frases o texto).
En nuestro caso las palabras son *slices* verticales de un bloque de ancho. Esta es la unidad de datos con la que trabajamos. Los *slices* forman los niveles, ya que estos son un conjunto de secuencias de *slices*. Esto se puede ver mejor pensando en un texto en español cualquiera. Por ejemplo, si tenemos el siguiente fragmento de *El Quijote*:

En esto descubrieron treinta o cuarenta molinos de viento que hay en aquel campo y así como don Quijote los vio dijo a su escudero.

En = slice 1
esto = slice 2
descubrieron = slice 3
...
escudero = slice 25

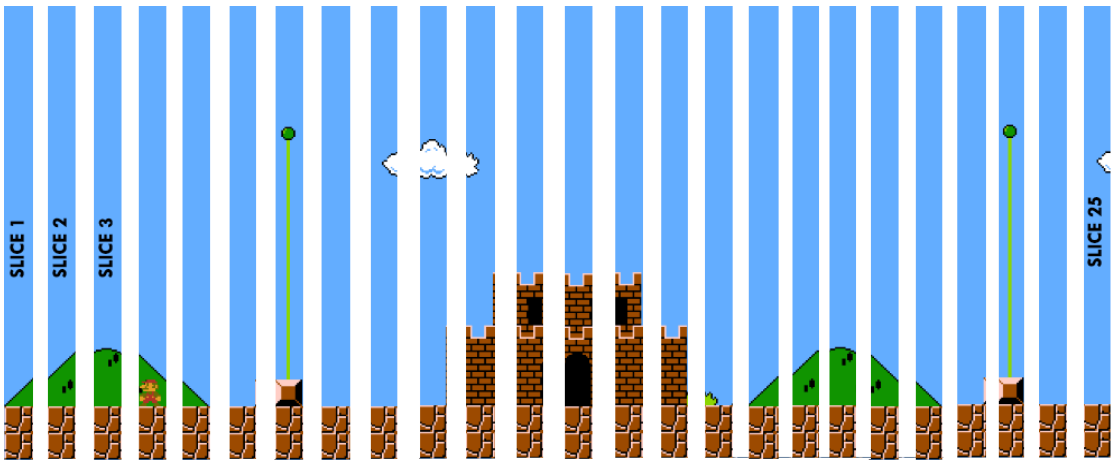

Figura 5.11: Mapa dividido en *slices*.

La frase está compuesta de 25 palabras, cada una de ellas sería la homóloga de nuestro *slices* en SMB. Por lo que, si las palabras de esta frase de El Quijote fuesen *slices*, la frase se correspondería al nivel entero, ambos de longitud 25 (la frase tiene 25 palabras y el nivel tiene 25 *slices*). De este modo, ya tenemos una unidad con la que formar secuencias y realizar un procesamiento de los distintos textos. Tratamos los *slices* como si fueran palabras y, así poder procesar los niveles como si fuesen frases. Los niveles están definidos de izquierda a derecha lo que facilita la interpretación de estos como secuencias de elementos.

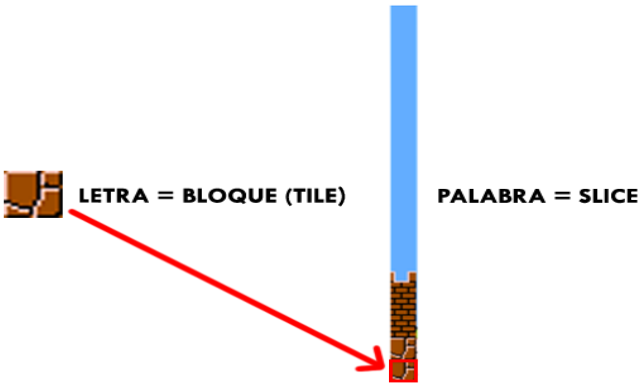

Figura 5.12: Representación gráfica de los datos

Una diferencia entre nuestros *slices* y las palabras de un idioma cualquiera, es que las palabras tienen una longitud variable de letras y nuestros *slices* una longitud fija de bloques. Esto es debido a que los niveles tienen una altura fija y si no existe un *tile* en alguna zona del nivel se rellena con -1, lo que sería el cielo. Así todos los *slices* tienen la misma longitud. Esta diferencia no cambia en nada el uso de los *slices* respecto a las palabras mediante los n-gramas, pues simplemente se toman secuencias de tamaño n y el tamaño de las palabras o *slices* no afecta a esta manera de procesar el texto.

A continuación, realizamos el proceso de clasificación de los niveles. La clasificación de los niveles consiste en la división de los textos de entrada en n-gramas. En base al valor n se irán creando secuencias de *slices* de tamaño n-1 que se guardan en un diccionario (como clave) y se añade a una lista (valor) el siguiente *slice* a esa secuencia. En caso de repetición de alguna secuencias, se añade el siguiente *slice* a la lista correspondiente de siguientes secuencias.

En la siguiente imagen se muestra cómo se clasifican las secuencias dependiendo del valor de n. Los unigramas no miran en la historia para estimar el próximo *slice*, únicamente usan la probabilidad de que salga un *slice* de entre todos los posibles *slices* del corpus. A partir de los bigramas (n gt;= 2) se estudia los *slices* previos para realizar la estimación.

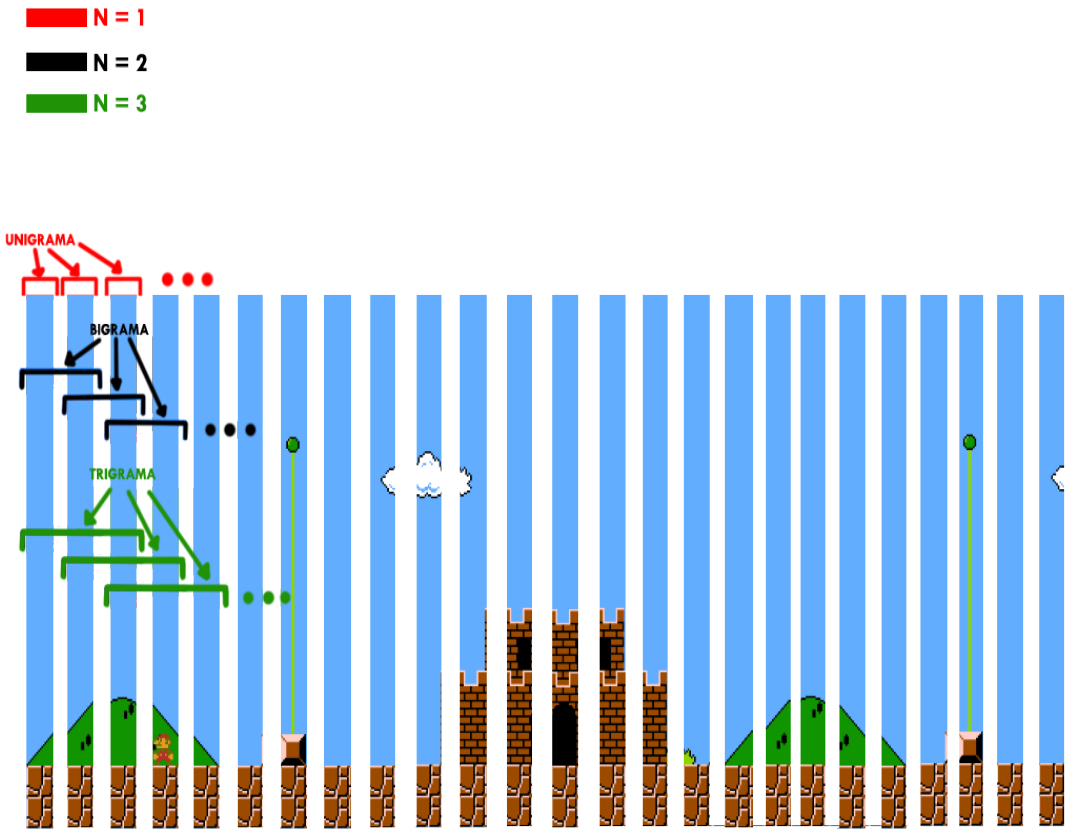

Figura 5.13: Clasificación nivel mediante n-gramas

Por ejemplo, suponemos que n = 2, entonces para cada secuencia de tamaño 1 (n-1), empezando por el principio se añade el siguiente *slice* en la lista de *slices* que siguen a la secuencia de tamaño n-1. En la siguiente imagen se puede ver este proceso respecto al nivel de la foto troceada de arriba. En la lista de secuencias que siguen al *slice* 1 se encontrarían estos dos *slices* (que son los mismos), puesto que en todo el nivel esos *slices* son los dos únicos que lo siguen. En cuanto al nombre *slice* 1 nos referimos a que es el *slice* en la posición 1, y más adelante (posición 17) se repite el mismo.

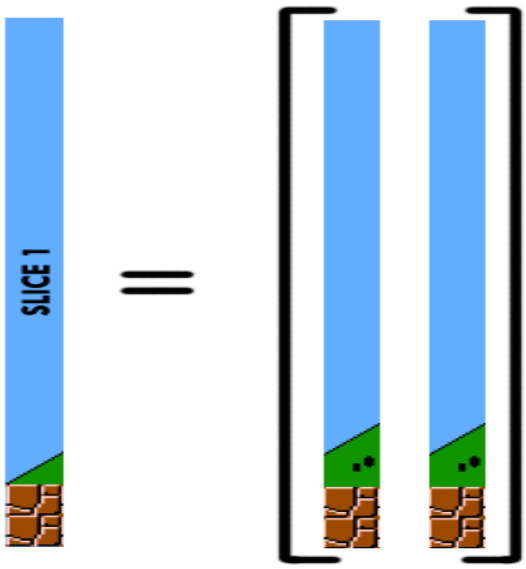

Figura 5.14: *Slices* que pueden seguir a una secuencia de un *slice*

Para n = 3, se formarían secuencias de dos *slices* y se añadirían en la lista correspondiente aquellas que las sigan. En el siguiente ejemplo pasa los mismo que con bigramas, la primera secuencia se repite más tarde y estás seguida las dos veces por el mismo *slice*.

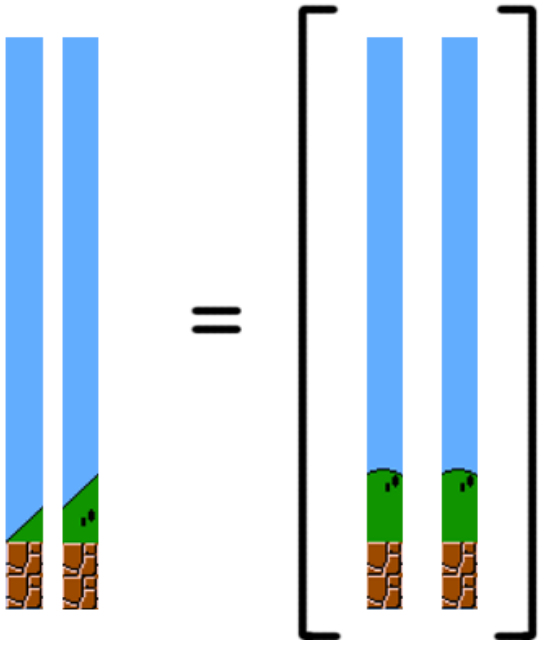

Figura 5.15: *Slices* que pueden seguir a una secuencia de dos *slices*

Decidimos usar una lista para guardar los *slices* que siguen a una secuencia porque es la manera más fácil de calcular la probabilidad condicional, puesto que se debe dividir las veces que aparece un n-grama (los *slices* de la lista representan esto) entre las veces que aparece la secuencia de tamaño n-1 para ese n-grama (la longitud de la lista es las veces que aparece esta secuencia). La clave del diccionario representa la secuencia de tamaño n-1, es decir, los *slices* previos, y la lista las veces que aparece esa secuencia formando los distintos n-gramas. Después simplemente se usa un *random* entre todos los *slices* de la lista para estimar el siguiente *slice*

(recordar que un *slice* era una palabra). Dado que la frecuencia de los *slices* en la lista van a determinar la probabilidad de que salgan no hay que calcular nada. Si en el ejemplo de la imagen de arriba se añadiese a la lista un *slice* porque suponga-mos que aparece en el nivel junto a la secuencia previa (la clave del diccionario), la probabilidad de que saliese el *slice* con la parte del arbusto sería del 66.67 % y que saliese el *slice* nuevo que hemos añadido sería del 33.33 %. Para los unigramas, los *slices* son calculados en base a la probabilidad que es simplemente la frecuencia con la que aparecen en el corpus. Para los bigramas, su probabilidad es la probabilidad condicional respecto al *slice* previo y, así sucesivamente para cada n-grama con su probabilidad condicional en base a los n-1 *slices* previos.

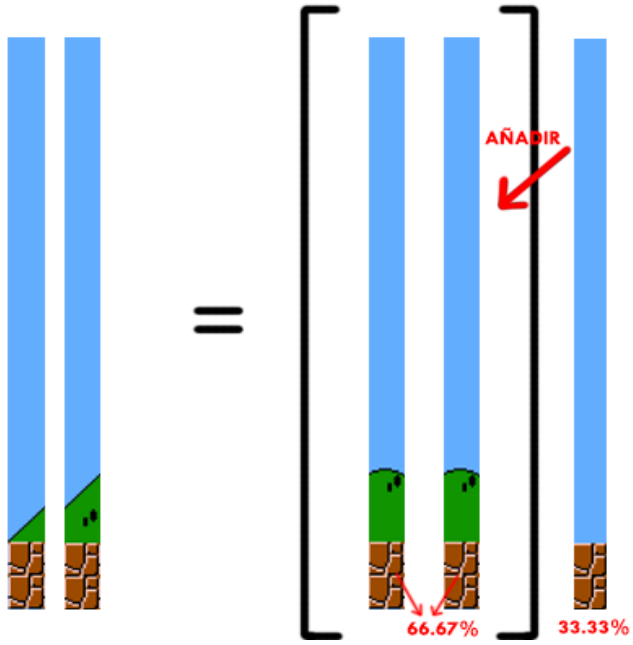

Figura 5.16: Porcentajes de cada posible *slice*

Los niveles del videojuego original tienen una longitud variable dependiendo de cada uno. Nosotros podemos generar también niveles con una longitud cualquiera, pero para mejor comparar resultados usaremos 300 bloques. Cuando generamos un nuevo nivel, la primera secuencia de este será la primera secuencia de tamaño n-1 del primer nivel que hemos pasado para el entrenamiento. Esto es debido a que no usamos etiquetas de inicio o final.

Una vez planteada la forma de generar los *slices*, nos surgió un problema por el cual muchos resultados no tenían la longitud esperada. Esto era debido a que se llegaba antes de tiempo a una secuencia final y el modelo ya no podía continuar. A sique de forma automática el CSV era completado con ceros.

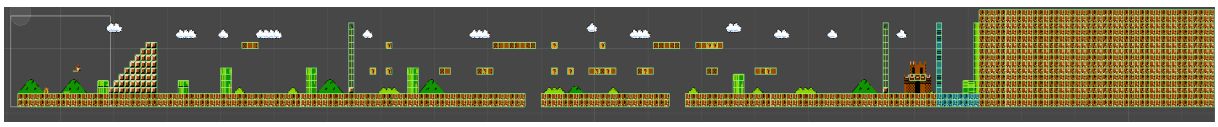

Figura 5.17: Mapa completado automáticamente con ceros

La solución a este problema fue resetear la secuencia actual por la secuencia con

la que se inició la generación del texto. Aunque, seguimos teniendo problemas en los resultados de los niveles debido a errores en el código. Los niveles continuaban con las mismas *slices*.

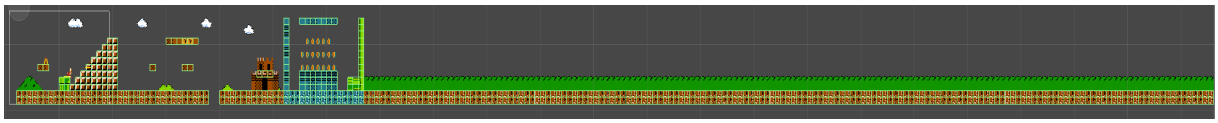

Figura 5.18: Mapa que se resetaba mal con la primera secuencia

Definitivamente arreglamos el problema y al llegar a la secuencia final antes de tiempo se reiniciaba con la primera secuencia.

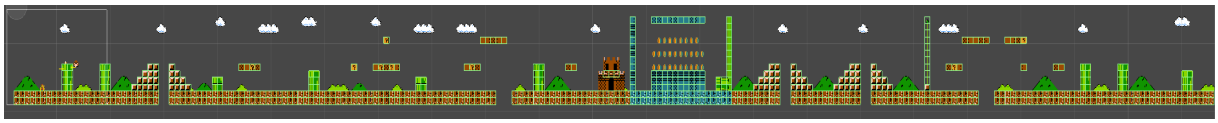

Figura 5.19: Mapa bien generado

Como se puede ver en la imagen la zona secreta aparecía al final del nivel, debido a que en Tiled la imagen de referencia para pintar el nivel original era de esta forma. En este momento, la idea de hacer mapas mezclando temáticas empezó a surgir. En el artículo que teníamos como referencia se trataban todos los niveles con la misma temática dando un resultado siempre sobre el suelo pero con los patrones de las otras temáticas. Juntaban los distintos mundos originando un corpus muy largo pero con la misma temática. En nuestro caso lo que buscábamos con la idea de multitema era poder generar mapas con las distintas zonas propuestas por el juego original y su mezcla. No la copia de su patrón. Por tanto, la idea de multitema se resume en conseguir mezclar todas las zonas que ofrece el juego original en nuevos mapas generados proceduralmente. Por tanto el siguiente paso fue la generación de niveles multitemas.

Para conseguir este objetivo, se aumentó el tamaño de los *slices*. De 15 bloques de altura se pasó a 60 bloques (15 para el cielo, 15 para el terreno normal, 15 submarino y 15 subterráneo, en este orden de arriba abajo). En Tiled creamos los mapas de entrenamiento con la nueva altura 60 y se rellenan los 15 *tiles* de altura correspondientes a cada temática. Dando lugar al tipo de dato explicado algunos apartados anteriores. Si que es cierto que no fue un proceso directo. Primero se probó con un mapa de 30 (15 para el suelo y 15 para zona subterránea) y, tras comprobar los exitosos resultados, probamos a ir añadiendo más sectores. En cuanto a la implementación, había que tener en cuenta los otros mapas elegidos para la generación multitema. Es decir, a la lista de n-gramas del primer mapa había que añadir las del resto de mapas seleccionados. Por tanto había que extraer los n-gramas del primero, y de los siguientes para luego poder mezclar todos esos datos. De este modo, en el momento de llegar a una secuencia apareciesen tanto las probabilidades del primer nivel como las del segundo y, así lograr el objetivo de generar mapas multitemáticos.

En nuestro caso, respecto a la generación de los niveles no tenemos el problema de la aparición de palabras desconocidas o *unknown words*. Las *unknown words* o palabras desconocidas son palabras que con las que hay que tratar sin haberse visto

antes. Esto es propio de cuando tenemos un vocabulario abierto y en ese caso las palabras desconocidas serán tratadas con la etiqueta <UNKgt;. Sin embargo, en nuestro caso el vocabulario es conocido como vocabulario cerrado o *closed vocabulary*, pues el modelo solo trabaja con las palabras utilizadas en los entrenamientos, ya que entrenamos el modelo con los diferentes mapas originales del videojuego y, posteriormente, se genera el mapa escogiendo una primera secuencia n y a partir de ahí seleccionando palabras de los datos entrenados.

A la hora de la generación de los mapas no empleamos métodos de suavizado, puesto que al utilizar un vocabulario cerrado no nos vamos a encontrar con probabilidades de cero. No tiene sentido suavizar el texto cuando conoces todo el vocabulario que vas a manipular. Algoritmos de suavizado como el de Laplace (conocido previamente también como *add-1*), *add-k*, *stupid backoff* o *Kneser-Ney* no tenía sentido usarlos para la generación de niveles.

En cuanto a la perplejidad, utilizada para evaluar el modelo, no la tenemos en cuenta. Esto es debido a que si se puede evaluar la coherencia que presenta un texto, sin embargo, en cuanto a un nivel es un proceso bastante más complejo. Se puede decir si está bien porque sigue los patrones de los *tiles* o su jugabilidad, o no, porque por ejemplo aparece una tubería incompleta o presenta zonas inalcanzables. Pero no se puede evaluar la calidad del resultado, ya que depende completamente de la percepción del usuario.

Seguidamente adaptamos el *script* para recibir parámetros del usuario de tal forma que este eligiese como generar los nuevos mapas. Esto supuso el poder conectar los *scripts* de Python con Unity sin tener que *harcodear*[3] los valores en el algoritmo. Además en los parámetros del *script* incluimos una opción de debug en la que se creaba un fichero donde se guardaba la información del modelo, así como las posibles secuencias y la secuencia escogida al generar texto. Datos bastante interesantes para poder realizar posteriormente un estudio técnico más profundo, en caso de ser necesario.

Tras tener definitivamente niveles completos decidimos añadirles algo de interactividad. Para ello añadimos el poder utilizar las tuberías y enredaderas para el paso entre zonas del mapa, la interacción con los distintos tipos de bloques o la interacción con algunos objetos y enemigos del nivel.
Para lograr la conexión de las tuberías con las zonas secretas se comprueba si una tubería tiene una tubería de una zona secreta exactamente abajo y se otorga el comportamiento de entrada a la zona a la tubería inmediatamente anterior. Para realizar el teletransporte de Mario a la zona secreta empleamos un *tile* invisible, que no tiene representación gráfica, pero que si cuenta con representación en el archivo CSV del nivel. De este modo, con este tipo *tile* facilitamos en la implementación del juego el comportamiento de teletransporte a las zonas secretas. Estos *tiles* invisibles también se tienen en cuenta, obviamente, para el procesamiento de los niveles en n-gramas.

[3]Introducir valores a la fuerza en el código fuente del programa

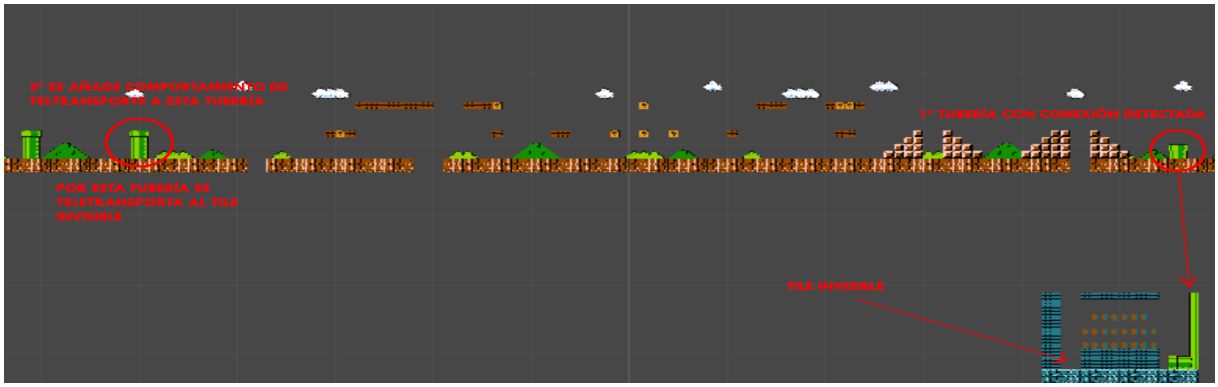

Figura 5.20: Explicación de la conexión de las tuberías

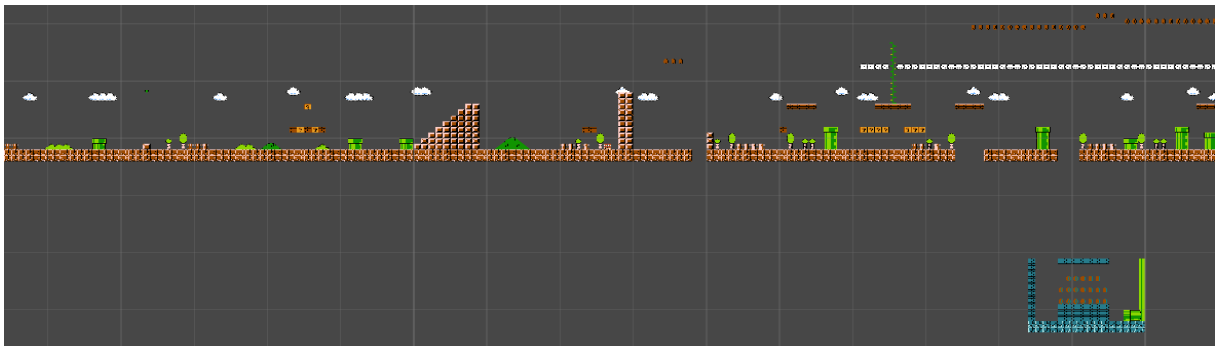

Figura 5.21: Mezcla de los mapas 1-1 y 2-1

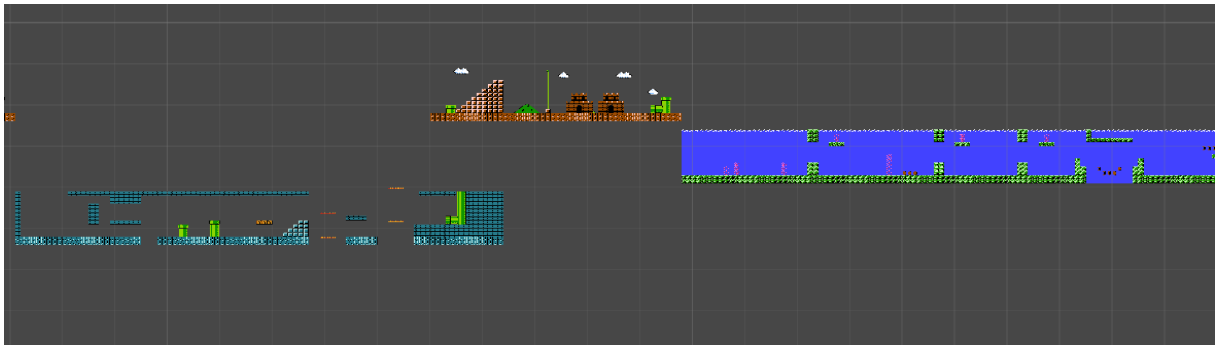

Figura 5.22: Mezcla de los mapas 1-2 y 2-2

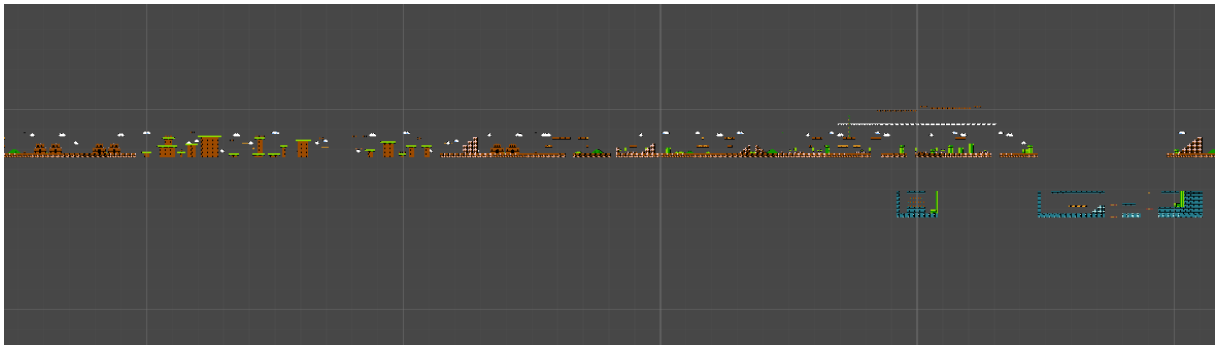

Figura 5.23: Mezcla de los mapas 1-1, 1-2, 1-3 y 2-1

Por último revisando el artículo de referencia para el estudio de n-gramas vimos un método de suavizado que podía parecer interesante. El método era la interpolación de n-gramas. Y es que este método de suavizado a parte de tener la finalidad de resolver parte del problema de las palabras desconocidas, tenía otra función bastante interesante y completamente aplicable a la generación de niveles. El método se basa en mezclar las siguientes palabras posibles de distintos n-gramas a una secuencia combinando estas según unos pesos establecidos. Es decir, dar más importancia a un n-grama que a otro. Con esto consigues salir de la regularidad de que otorgan los n-gramas más altos. Ya que estableciendo, poniendo un ejemplo, 0,8 0,1 y 0,1 a los trigramas, bigramas y unigramas respectivamente. Los resultados prácticamente son como los de trigramas pero puede ocurrir que se escojan en algún momento alguna palabra del bigramas o de unigramas, otorgando al mapa más variedad y alejándose del original. En muchas ocasiones esto puede generar secuencias que no pertenezcan ni al bigramas ni al trigramas, ya que si por casualidad se elige una palabra del unigramas, esta puede formar una secuencia desconocida. En esta situación se plantearon dos soluciones o bien seguir generando el nivel con unigramas, lo que supondría pasar de un nivel medianamente acorde al original a de repente un nivel muy aleatorio. O la opción por la que nos decantamos, resetear a la primera secuencia.

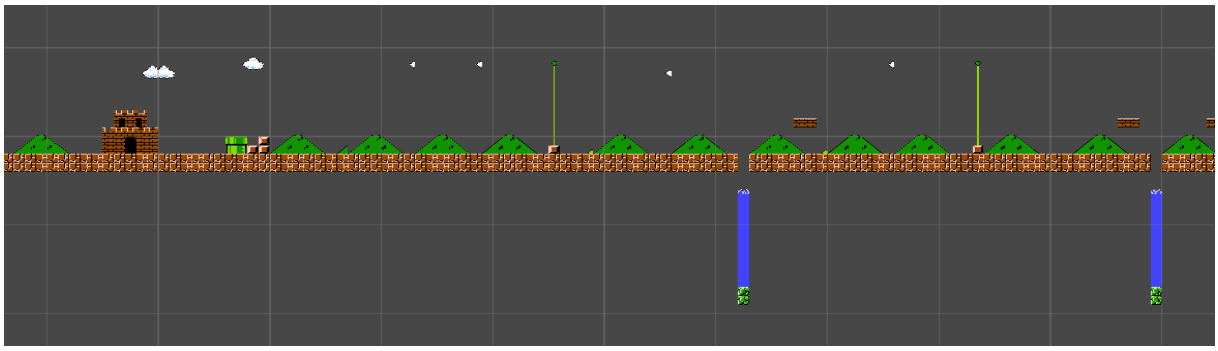

Figura 5.24: Generación del mapa 1-1 con interpolación y máxima prioridad a unigramas

Para la imagen de arriba se ha utilizado el método de interpolación, en concreto se ha utilizado un valor N=3 para los n-gramas, mientras que para las respectivas probabilidades se han utilizado:

trigramas –>0.2 %

bigramas –>0.2 %

unigramas –>0.6 %

Como podemos observar, en el mapa podemos apreciar numerosas *slices* que no tienen sentido del contexto del propio mapa, ya que los unigramas predicen muchas *slices* diferentes que no se adaptan al contexto del mapa como podría hacer un trigrama.

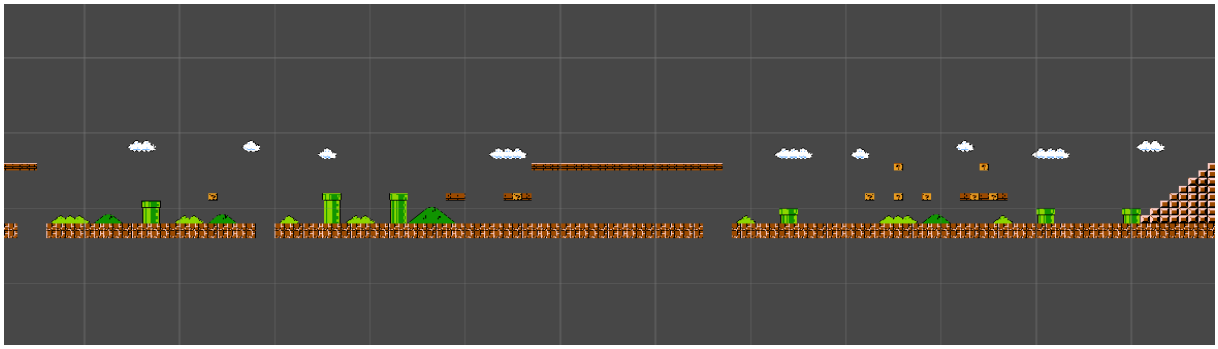

Figura 5.25: Generación del mapa 1-1 con interpolación y máxima prioridad a los trigramas

trigramas –>0.7 %

bigramas –>0.2 %

unigramas –>0.1 %

Después de este último paso decidimos separar el algoritmo en dos partes, una para entrenar los n-gramas y otra para generar mapas. Es ilógico mantener todo junto puesto que cada vez que se quisiese generar un mapa habría que entrenar el modelo. Si se quieren generar 4 mapas con el mismo modelo habría que repetir el mismo proceso de entrenamiento del modelo. Esto supone un desperdicio de tiempo y de recursos. Por tanto lo que hacíamos al entrenar es guardar los datos necesarios para la generación de texto en un fichero, como el diccionario de n-gramas con las probabilidades, el n elegido por el usuario o si se ha elegido interpolación o no y sus respectivos porcentajes.

5.5.2. Redes neuronales recursivas

Introducción

Las redes neuronales recursivas son un tipo de red neuronal cuya aplicación principal es el análisis de secuencias. Son modelos muy populares y utilizados en numerosas tareas relacionadas con el procesamiento de lenguaje natural.

La principal idea detrás de las RNN es el uso de información secuencial. En una red neuronal tradicional, se asume que cada uno de los inputs y outputs es independiente del resto. Por ejemplo, si queremos predecir la siguiente palabra de una secuencia, en nuestro caso lo que estamos estudiando, sería una mala idea, ya que nosotros lo que queremos es que cada output dependa del resto, no de sí mismo. Por tanto, podríamos considerar que una red neuronal recurrente tiene “memoria”, capturando así información sobre lo que se ha calculado anteriormente.

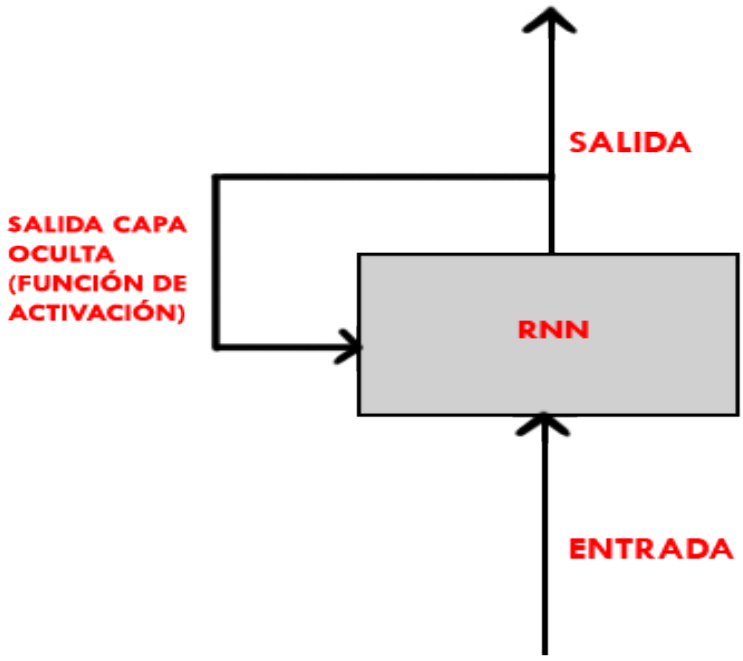

Figura 5.26: Esquema de una RNN simple

Las RNN pueden aparecer de diferentes formas como pueden ser [61]:

Completamente recurrentes. Los nodos están organizados en diferentes capas. Cada nodo de una capa dada, es conectado directamente con otro nodo de la siguiente capa. Además cada nodo guarda un factor de activación y un peso dentro de la red.

Redes recurrentes simples. Son redes de únicamente tres capas, con la suma de un conjunto de unidades de contexto. La capa de en medio, está conectada con estas unidades a través de sus respectivos pesos. En cada actualización, el input se transfiere hacia delante, y se crea una regla de aprendizaje.

Redes Bi-direccionales. Las RNN bidireccionales utilizan una secuencia finita para predecir o etiquetar cada elemento de la secuencia, basándose en los elementos pasados y futuros. Esto se consigue concatenando el input de dos RNN, en donde una se encarga de procesar la secuencia de izquierda a derecha, y la otra, de derecha a izquierda. Esta técnica es especialmente útil cuando se combinan LSTM y RNN.

Entre otras muchas como redes jerárquicas, de segundo orden, independientes o multicapa.

Historia

Las redes neuronales recursivas se basaron en el trabajo llevado a cabo por David Rumelhart en 1986. Las redes Hopfield, comentadas anteriormente fueron descubiertas por John Hopfield en 1982. En 1993, un sistema de compresor de historia neural, fue capaz de resolver una tarea que requería de un gran aprendizaje, formada por más de 1000 capas en una RNN desdoblada durante "ejecución".

En 1997, Hochreiter y Schmidhuber inventaron las redes LSTM (*Long short-term memory*) y establecieron récords de precisión, en aplicaciones de múltiples ámbitos. Alrededor de 2007, las redes LSTM empezaron a revolucionar el reconocimiento de voz, superando a modelos tradicionales en ciertas aplicaciones de reconocimiento de voz.

En torno a 2009, una CTC-trained LSTM (*Connectionist Temporal Classification*) fue la primera red neuronal recursiva en ganar concursos de reconocimiento de patrones, en pruebas como el reconocimiento de escritura. En 2014, el gigante Chino, Baidu, un navegador, utilizó CTC-trained RNN para romper el punto de referencia de reconocimiento de voz del Switchboard Hub5'00 sin usar ningún método tradicional de procesamiento de voz.

Las redes LSTM también mejoraron para el reconocimiento de grandes vocabularios así como la síntesis de texto a voz, siendo así utilizada por Android. En el año 2015, el método de Google para el reconocimiento de voz, experimentó una caída de rendimiento del 49 % a través de las redes CTC-trained LSTM.

Finalmente, las redes LSTM rompieron récords, una vez mejoraron para la traducción de texto, el modelado de lenguaje, y el procesamiento de lenguaje multilingüistico. Las redes LSTM combinadas con redes neuronales convolucionales, permitieron mejorar la captura automática de imágenes [61].

Uso de las RNN

Generar texto con redes neuronales recurrentes es quizá la forma más directa de aplicar las redes neuronales recursivas.

Desde el punto de vista del negocio, la generación de texto se utiliza como un método para agilizar el proceso del trabajo, y minimizar la rutina.
La generación de lenguaje natural se apoya en algoritmos de predicción de redes neuronales recurrentes. Dado que el lenguaje está organizado secuencialmente, es relativamente fácil entrenar un modelo para la generación de texto.
Las formas más comunes son [62]:

Resumen de textos: El proceso consiste en condensar el texto original. El resumen se utiliza en la gestión de proyectos para incorporar rápidamente a los trabajadores al flujo de trabajo. También se puede utilizar para resumir las noticias y agilizar la generación de artículos de noticias.

Generación de documentos: Usado en banca y seguros para crear cuestionarios adaptados a las necesidades de un cliente concreto.

Generación de reportes: En este caso, la generación de texto se utiliza como una forma para la visualización de datos.

Otra forma habitual son las interfaces de conversación y los chatbots.

Las redes neuronales recurrentes se pueden aplicar también en la traducción automática debido a su capacidad para determinar el contexto del mensaje, o en la localización de contenido. Desde un punto de vista técnico, la traducción automática no es más que una simple sustitución de palabras que representan ciertos conceptos, con sus equivalentes en otro idioma. Hoy en día la aplicación más utilizada es Google Translate. También podemos encontrar numerosas aplicaciones de las RNN en la localización de contenido, por ejemplo, en el comercio electrónico como Amazon o AliExpress. Usan la traducción automática para adaptar el contenido, como las fichas de producto, y mejorar la eficiencia de los resultados de la búsqueda.

Algunos asistentes virtuales como Alexa o Siri, se están volviendo algo habitual en nuestro día a día, ayudando a los usuarios en sus hábitos cotidianos, a partir de frases previamente formuladas. La tecnología que permite esto, es el reconocimiento de voz con las redes neuronales recurrentes. Sin embargo, para poder reconocer voz, se utiliza una capa de rendimiento extra. El reconocimiento de voz, se utiliza sobre todo en interfaces conversacionales (atención al cliente), chatbots, o aplicaciones que convierten audio en texto (micrófono de búsqueda de Google).

Por último, las redes neuronales recurrentes, pueden ser una herramienta muy útil frente a la detección de fraudes, spam o bots. Sobre todo, la prevención de fraude se apoya en algoritmos predictivos que se encargan de exponer las actividades ilegales. En el caso de los fraudes publicitarios, las RNN se usan para determinar patrones sospechosos o anormales. También pueden ser utilizadas para la detección de spam, aplicando NLP (*Natural Language Processing*), para exponer patrones y seguidamente bloquear el mensaje.

Finalmente, aparte de todo lo comentado, las RNN también pueden ser utilizadas para análisis predictivos (predecir la caída de la bolsa) o para el análisis del *feedback* de los clientes desde el punto de vista de los negocios.

Modelo del lenguaje neuronal

Las redes neuronales están construidas con unidades neuronales, inspiradas en las neuronas humanas. Cada unidad neuronal multiplica los valores introducidos por un vector de pesos, añade el término bias y aplica una función de activación no lineal como las funciones sigmoides, tanh o ReLU (lineal rectificada). La ecuación para el resultado de una unidad neuronal sería:

$$z = w \cdot x + b \tag{5.5}$$

Ecuación de la salida de una unidad neuronal

Donde x es el vector de entrada, w el vector de pesos por los que se multiplica la entrada y b el término escalar bias. Por lo que z será un número. Sin embargo, las unidades neuronales aplican una función no lineal a z, llamada función de activación. Como se ha mencionado arriba hay varias funciones no lineales de activación como el sigmoide, la tanh o RELU, las cuales son de las más usadas. Cada una presenta diferentes propiedades por lo que cada una es más útil en diferentes aplicaciones del lenguaje. Estas tres funciones de activación serán explicadas más adelante, puesto que fueron usadas a lo largo del proceso de desarrollo.

En una red totalmente conectada de tipo *feedforward*, es decir, aquellas redes neuronales en las que el output se usa como input en las siguientes capas, cada unidad en la capa i está conectada con cada unidad en la capa i+1 y, además no existen ciclos. Se usa una matriz de pesos para cada capa oculta para hacer los cálculos de cada capa más eficientes. Se emplea una función para normalizar los outputs de vectores de números en vectores de probabilidades. Esta función es llamada softmax:

$$\text{softmax}(\text{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} 1 \leq \text{i} \leq \text{d} \tag{5.6}$$

Ecuación de la salida de una unidad neuronal

La utilidad de las redes neuronales reside en que gracias a las representaciones aprendidas en capas previas, estas pueden ser utilizadas por capas más profundas en la red. El objetivo del entrenamiento es aprender los parámetros W y b, es decir, los pesos y el término bias para cada capa, para que el resultado de la red sea lo más aproximado al valor esperado.
Para lograr este objetivo, se usa la función de pérdida que mide la diferencia entre el resultado esperado y el obtenido.

Las redes neuronales se entrenan mediante algoritmos de optimización como el gradiente descendiente que se usa para minimizar la función de pérdida, mediante la búsqueda del parámetro W que lo cumple. En definitiva, consiste en buscar el mejor coeficiente.

El método de la propagación del error hacia atrás es usado para calcular los gradientes de la función de pérdida de una red neuronal.

Los modelos del lenguaje neuronales usan redes neuronales como clasificadores probabilísticos para calcular la probabilidad de la siguiente palabra dadas n palabras previas.
Estos modelos del lenguaje basados en redes neuronales tienen ciertas ventajas sobre los de modelos del lenguaje de n-gramas como, por ejemplo, no necesitan realizar la técnica del *smoothing*, ya que pueden generalizar datos en contextos con palabras similares. Sin embargo, su mayor desventaja es que son mucho más lentas entrenando.

Los modelos neuronales del lenguaje pueden usar incrustaciones o *embeddings* ya entrenados o pueden aprenderlos desde cero en el proceso de modelado del lenguaje. En estos modelos neuronales la historia se representa por *embeddings* de las palabras anteriores. Los *embeddings* hacen referencia a la vinculación de palabras o frases

de un vocabulario en vectores de números reales. Esta representación de la historia en lugar del uso de las palabras exactas permite generalizar mejor los datos no vistos.

Cuando comprendemos y producimos el lenguaje hablado, estamos procesando flujos continuos de entrada de indefinida longitud. E incluso cuando se trata de un texto escrito, normalmente lo procesamos secuencialmente, aunque en principio tenemos acceso arbitrario a todos los elementos a la vez. Esta naturaleza temporal es también reflejada en los algoritmos que usamos para procesar el lenguaje (Dan Jurafsky y James H.Martin). Entre ellos se encuentran las redes neuronales recursivas que permiten una mejor estimación de datos que están fuera del contexto de los datos entrenados.

Proceso de desarrollo

Comenzamos estudiando las redes neuronales, qué son, para qué se utilizan y cómo funcionan. Ya teníamos varias ideas sobre esto, pero continuamos investigando acerca de esta rama del aprendizaje automático. En concreto nos centramos en las redes neuronales recursivas, debido a que son utilizadas, entre otras muchas cosas, para la generación de textos, como hemos explicado al comienzo de esta sección. Esto nos daba pie para dos cosas, la primera poder reutilizar muchas cosas aprendidas a lo largo de los n-gramas, y dos, el poder comparar de primera mano dos algoritmos completamente distinto, pero que buscan el mismo fin, el de procesar secuencias de datos para generar unos nuevos. Durante toda esta investigación descubrimos que existen cientos de arquitecturas distintas de redes neuronales.

El siguiente paso era el de implementar la propia red neuronal. Para ello estuvimos barajando distintas opciones. La primera de ellas era la de utilizar la libreria de Keras para la implementación y manejo de la red neuronal. Comenzamos mirando documentación y todas las opciones que nos daba esta librería. Por otro lado, empezamos a investigar TensorFlow y todo lo que nos ofrecía, y tras mucho investigar descubrimos que Keras era la API de alto nivel de TensorFlow. Esto añadido a la buena documentación que ofrecía TensorFlow, además, de unos muy buenos ejemplos y tutoriales, fue lo que hizo que eligiésemos TensorFlow como librería a usar. Dentro de estos tutoriales había uno en el que se generaba texto con una red neuronal recursiva, lo cual nos ayudó mucho a la hora de implementar todo, entender los datos y porque son de una forma y no de otra. Por tanto comenzamos con replicar el ejemplo planteado por TensorFlow y generar texto a partir de un corpus de entrenamiento para la red neuronal. Esto fue relativamente sencillo en cuanto a la implementación, sin embargo a pesar de toda la información, había ciertos parámetros para la configuración de la red neuronal que no eran del todo entendibles. No obstante, obtuvimos los resultados esperados.

Sin embargo el ejemplo generaba texto a partir de letras y en nuestro caso, nosotros utilizábamos secuencias de números obtenidos de un CSV. Debido a esto hubo que realizar las primeras modificaciones e ir generado nuestro propio modelo tomando como ejemplo el modelo de predicción de texto. Para poder utilizar como entrada un texto, la red neuronal utilizaba un proceso de vectorización de datos, por lo que investigamos distintas formas de vectorización de texto. Entre ellas encontramos [63]:

1. **One-hot encoding**. Un primer acercamiento a la vectorización sería el *one-hot encoding* que consiste en codificar directamente cada palabra en nuestro propio vocabulario. Para representar cada palabra del vocabulario, se creará un vector de tamaño igual al vocabulario, relleno de ceros, colocando un 1 en el *index* que se corresponde con la palabra.

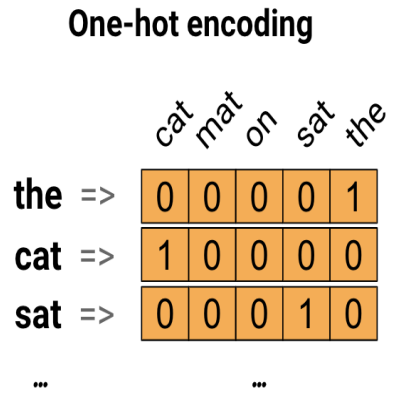

Figura 5.27: *One-hot encoding*. `https://www.tensorflow.org/tutorials/text/word_embeddings`

2. **Codificar cada palabra a un identificador único**. Otra manera de vectorizar los datos es la de codificar cada palabra del vocabulario con un valor único, en este caso un número. Con el ejemplo anterior, podríamos dar a cat el 1, mat el 2, etc. Podríamos vectorizar de esta manera la frase *The cat sat on the mat* de la siguiente manera [5,1,4,3,5,2]. Con este método tenemos un vector cuyos elementos tienen valor.

3. **Incrustaciones de palabras**. En cuanto a la incrustación de palabras (*word embedding*), es la manera más eficiente de vectorizar. Un *embedding* es un vector de valores en coma flotante. Estos valores son parámetros entrenables a partir de los pesos aprendidos por el modelo entrenado. Para conjuntos de datos pequeños, la mejor opción es utilizar una dimensión de incrustación (*embedding*) reducida, mientras que si tenemos una gran cantidad de datos, debemos utilizar una dimensión mayor.

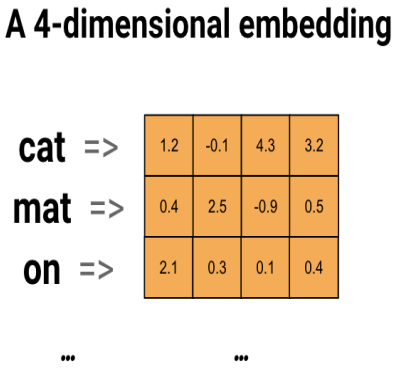

Figura 5.28: *Word embeddings*. `https://www.tensorflow.org/tutorials/text/word_embeddings`

Tras estudiarlas decidimos implementar la de otorgar a cada *slice* un id único para su identificación. Para ello creamos un mapa y una lista que nos servían de traductores entre el id y la secuencia de una forma muy eficaz. Después de esto comenzamos a estudiar a fondo la API que nos ofrece TensorFlow y estudiar los distintos parámetros. Aprendimos a generar modelos de redes neuronales, esto es hace añadiendo capas a un modelo vacío, especificando las características de cada capa. Dentro de estas características las más importantes son:

1. **Embedding dim o las dimensiones de las incrustaciones de palabras**. Como hemos explicado más arriba la dimensión hace referencia al tamaño que tendrá el vector o la matriz en la que incrustaremos las palabras del vocabulario.

2. **Rnn units o el número de neuronas recursivas**. Número de unidades de memoria a cada entrada de la secuencia colocadas de manera vertical y enlazadas entre sí. Cada una de estas células se encargan de pasar la información a las siguientes células con las que están unidas

3. **Batch size**. El *batch size* hace referencia al número de muestras o ejemplos que se propagarán a través de toda la red, en un filtrado de información hacia delante o hacia atrás. A mayor tamaño de batch, más espacio en memoria se necesitará para entrenar al modelo.

4. **Buffer size o tamaño del buffer**: Tamaño del espacio donde los valores se mezclan.

5. **Recurrent initializer**. Un inicializador de capa se encarga de repartir pesos iniciales completamente aleatorios a cada una de las capas disponibles.

6. **Activation function**. Las funciones de activación determinan la salida de un modelo de aprendizaje automático, su precisión respecto a las muestras del que aprende, así como la eficiencia del modelo. Además la función de activación se encarga de normalizar la salida de cada neurona a un rango entre 1 a 0, o -1 a 1, para evitar que cada output tenga un valor muy dispar del resto.

7. **Training epochs o entrenamientos**: Número de entrenamientos que llevará a cabo la red para producir un resultado definitivo.

8. **Temperature o temperatura**: Aleatoriedad del modelo entrenado.

A lo largo del proyecto, hemos utilizado diferentes funciones de activación como pueden ser la función Sigmoide, Tanh o ReLU [64]:

Sigmoide: Transforma los valores introducidos a la función de activación a una escala(0,1). Ofrece buen rendimiento en la última capa, la de densidad, sin embargo es lenta a la hora de encontrar el valor convergente u óptimo.

$$f(x) = \frac{1}{1-e^{-x}} \tag{5.7}$$

Función sigmoide

Tanh o Tangente hiperbólica: Transforma los valores introducidos a una escala (-1, 1). Su función de activación es muy similar a la sigmoide, por lo que es lenta también a la hora de encontrar valores óptimos. Es de las opciones más eficientes para usar en redes neuronales recurrentes.

$$f(x) = \frac{1}{1-e^{-2x}} - 1 \tag{5.8}$$

Función Tanh

ReLU o Unidad Lineal Rectificada: Transforma los valores introducidos anulando los valores negativos y dejando los valores positivos sin modificar, por tanto es una función sin acotar, por lo que muchas neuronas pueden quedar "inactivas". Es útil en redes convoluciones para la detección o reconocimiento de imágenes.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{5.9}$$

Función ReLU

Dentro de los tipos de capas usamos las **SimpleRNN**, las **LSTM** y las **GRU**.

La verdad es que el proceso de añadir la capa que se quiera a un modelo vacío es una forma muy cómoda, visual y sencilla de construir una red neuronal. Nuestro primer modelo presentaba la siguiente estructura. Primero una capa de entrada o *Embedding layer*, seguidamente dos capas ocultas de LSTM y por último una capa de output Dense. El modelo quedó de la siguiente forma:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (13, None, 256)           21504
_________________________________________________________________
lstm (LSTM)                  (13, None, 1024)          5246976
_________________________________________________________________
lstm_1 (LSTM)                (13, None, 1024)          8392704
_________________________________________________________________
dense (Dense)                (13, None, 84)            86100
=================================================================
Total params: 13,747,284
Trainable params: 13,747,284
Non-trainable params: 0
_________________________________________________________________
```

Figura 5.29: Modelo con 2 capas LSTM

La capa de *embedding* simplemente se encarga de en primer lugar inicializar el vector *embedding* con valores aleatorios y usa ciertos optimizadores, como puede ser Adam, para actualizarlo. Por otro lado podemos observar 2 capas LSTM que se encargarán de transportar los inputs y outputs de una capa a otra. Finalmente nos encontramos con la capa de densidad, que se encarga de realizar la función de activación. Tras este primer modelo fuimos probando numerosas combinaciones de capas. Algunos ejemplos de modelos creados contenían 4 capas ocultas LSTM. Otros

de 2 capas ocultas, una LSTM y otra GRU y viceversa. Los más simples de una sola capa oculta. Se nos abría un mundo de infinitas posibles combinaciones, sin embargo no todas generaban buenos resultados, con secuencias repetidas e incompletas como podemos observar en la imagen de abajo.

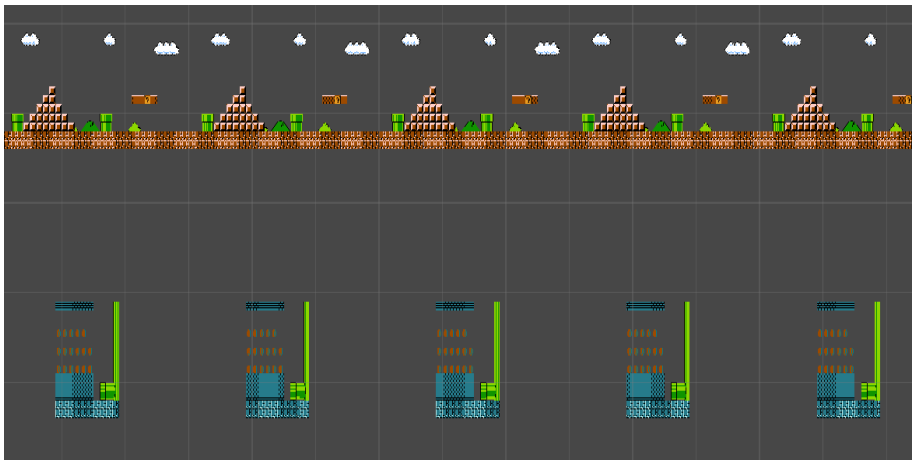

Figura 5.30: Mapa fallido con RNN

Estos resultados eran bastante aleatorios, ya que no conocíamos exactamente los valores con los que configurar la red neuronal. El proceso para mejorar los mismos fue lento, puesto que debíamos documentarnos acerca de lo que significaba cada parámetro y su contexto dentro del modelo. De esta forma, teniendo claro su función dentro de la red nos resultó más sencillo encontrar los valores óptimos para estos parámetros. Este proceso resultó algo tedioso, puesto que no hay documentación que estableciese como configurar estos parámetros para cada tipo de red. En muchas ocasiones las pruebas consistieron en ensayo y error. Por todo esto, decidimos realizar una serie de tablas de pruebas en la que cada uno de nosotros se encargaría de generar redes con parámetros específicos, para luego poder comparar esos mismos resultados y ver qué valor para cada parámetro era el más adecuado y poder así generar mapas considerados como "buenos", tanto con RNN simples, LSTM y GRU.

	A	B	C	D	E	F	G	H	I	J	K	L	M
		RNN											
		Test 1	Test 2	Test 3	Test 4	Test 5, RNN ==	Test 6	Test 7	Test 8	Test 9	Test 10	Test 11	Test 12
	SeqLength	10	10	10	10	20	20	20	20	30	30	30	30
	BATCH_SIZE	19	19	19	19	10	10	10	10	6	6	6	6
	BUFFER_SIZE	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
	embedding_dim	256	256	256	256	256	256	256	256	256	256	256	256
	rnn_units	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
	N° Layers	2	2	2	2	2	2	2	2	2	2	2	2
	EPOCH	50	50	50	50	50	50	50	50	50	50	50	50
	temperature	1	0.5	0.25	1.5	1	0.5	0.25	1.5	1	0.5	0.25	1.5

Figura 5.31: Tabla de tests para RNN simples

Sin embargo, estas tablas no nos terminaron de convencer, ya que prácticamente en todas las pruebas, los resultados no eran los esperados. Tras todo ello nos propusimos entrenar a los modelos de redes neuronales a partir de mapas generados

con n-gramas, ya que considerábamos que con los mapas originales apenas teníamos un conjunto de datos lo suficientemente grande como para que el modelo de RNN pudiera trabajar de manera correcta, provocando así la aparición de patrones incongruentes. Tras probar esta idea, observamos una mejora sobresaliente de los resultados con mapas generados que se acercaban a lo esperado desde un primer momento cuando empezamos a trabajar con las redes.

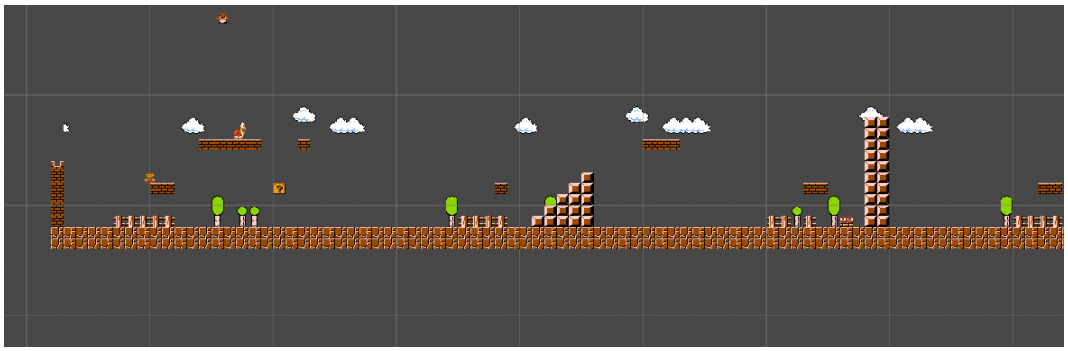

Figura 5.32: Mapa 2-1 con redes neuronales recurrentes

Además de todas estas variables que podemos observar en las tablas y con las que el usuario puede trabajar en la aplicación de Unity, también podrá elegir con que tipo de capas quiere entrenar el modelo: RNN Simple, LSTM o GRU. Al igual que en n-gramas, el usuario también puede elegir el tamaño del mapa a generar, así como el nombre del mapa. El proceso para la manipulación de estos parámetros dentro del *script* de redes neuronales recurrentes es el mismo que el utilizado en n-gramas. Uno de los problemas que nos surgieron mientras entrenábamos la red neuronal fue la generación de puntos de guardado o *checkpoints*. Cada *checkpoint* era creado por cada época utilizada para el entrenamiento, por lo que al realizar varios entrenamientos aumentaba considerablemente el espacio en memoria. La solución a este problema fue quedarnos con el último *checkpoint* que tiene la información más óptima para todos los entrenamientos y, borrar todos los demás *checkpoints* del sistema.

Tras las tablas que realizamos cada uno, continuamos modificando parámetros para obtener mejores resultados. Entre estas modificaciones se encontraba la reducción del parámetro de *embedding dim* y un aumento de la cantidad de neuronas por capa. Además otro parámetro conflictivo que nos encontramos fue el *batch size*. Este parámetro dependía completamente de una variable interna del *script* que se encarga de establecer una relación directa entre la longitud del texto y la longitud de la secuencia elegida. Por tanto, debido a esto, decidimos omitir la opción de ponerlo a disposición del usuario y limitarlo al númerod de examplesPerEpoch[4].

Por otro lado, probamos directamente el multitema. Lo que había que hacer para lograr que la red neuronal tuviese en cuenta ambos mapas era simplemente entrenar la red con cada uno de los niveles. De esta manera la red tenía en cuenta ambas temáticas. Tras hacer este análisis y estudiar la idea planteada, probamos por primera vez la generación de mapas multitema con redes neuronales recursivas, y pensamos que los resultados fueron bastante buenos a la primera. Sin embargo, tras numerosas pruebas, esto no fue como creíamos, puesto que la mezcla de aquellos mapas cuya

[4]Valor que establece una relación directa entre la longitud del primer input que reciben las redes, y la longitud del texto o vocabulario

longitud era distinta causó problemas en el algoritmo, ya que este se esperaba un conjunto de datos para ambos mapas de igual tamaño. El problema principal es que las redes neuronales operan con matrices por debajo, entonces al tener mapas con longitudes muy diferentes, y un corpus que se corresponde con el mapa entero, si se mezclan este tipo de mapas, las matrices internas de la red, al transformar el input a datos manejables por la propia red, no van a ser compatibles entre sí, debido a la diferencia de dimensiones. Una posible solución a esto es la generación de un mapa multitema mediante redes neuronales a partir de un mapa multitema generado por n-gramas, poniendo así en funcionamiento las dos técnicas que hemos desarrollado a lo largo del proyecto. Otra solución sería la de utilizar el multitema en aquellos mapas que tengan una longitud similar.

Tras esto, estuvimos mirando los posibles optimizadores para los algoritmos del modelo creado. Existen una gran cantidad de optimizadores y TensorFlow ofrece una gran variedad de estos para optimizar tu modelo de aprendizaje automático. Hay que señalar que los resultados de probar uno u otro optimizador depende del modelo y su conjunto de datos. Los parámetros ideales para un modelo pueden ser completamente diferentes al de otro. Esto supone un problema para nosotros, puesto que damos la oportunidad de modificar el modelo al usuario, por tanto el optimizador podría variar para cada modelo. Es por esto por el que hemos elegido el optimizador que se adapta mejor a casi todas las situaciones que es el Adam. Esto fue decidido tras estudiar la documentación y ejemplos de TensorFlow.

Además de todo esto, dividimos el *script* en dos partes para poder entrenar por un lado el modelo, y poder generar mapas por otro. De esta forma si el usuario quiere generar varios mapas en función de una misma red neuronal no tiene que estar entrenándola con los mismos parámetros cada vez que se quiera generar un mapa.

Finalmente, el proceso de desarrollo del algoritmo para la implementación del aprendizaje mediante redes neuronales recurrentes, no fue tan "descriptivo" como pudo ser el desarrollo del *script* de n-gramas. Al utilizar directamente una librería tan desarrollada como Tensorflow, nuestra tarea fue entenderla a fondo, saber toda su configuración además de qué métodos son aquellos que nos ayudan a la hora de organizar y tratar nuestros datos de manera correcta, para así poder adaptarla a nuestra conveniencia. Además de esto tuvimos que analizar los diferentes parámetros que podíamos dar al usuario para que pudiera trabajar con ellos dentro de la aplicación de Unity.

5.6. Resultados

5.6.1. N-gramas

Los resultados finales de n-gramas fueron bastante buenos mezclando temáticas e incluso interpolando distintos n-gramas.

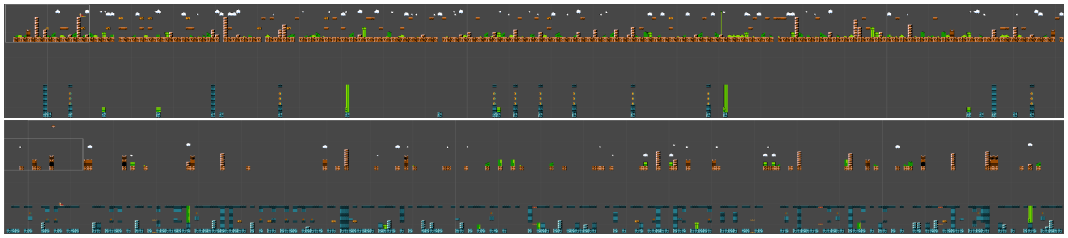

Figura 5.33: Niveles generados por unigramas con corpus de entrenamiento los niveles 1-1 y 1-2 respectivamente

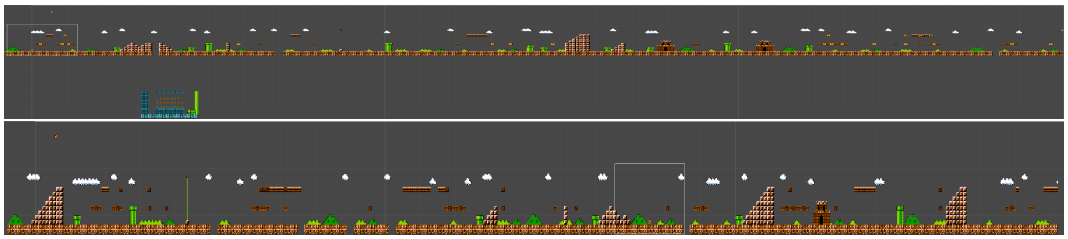

Figura 5.34: Niveles generados por bigramas con corpus de entrenamiento el nivel 1-1

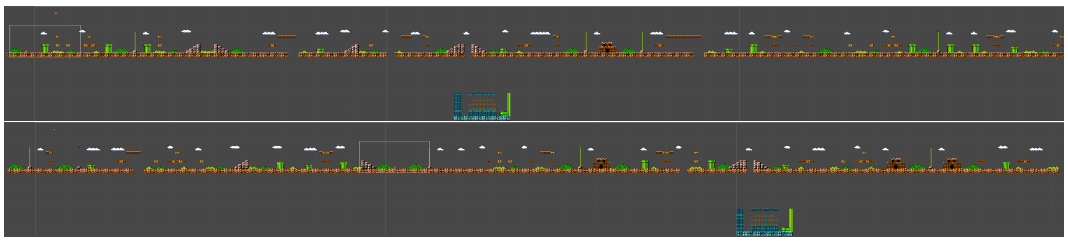

Figura 5.35: Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1

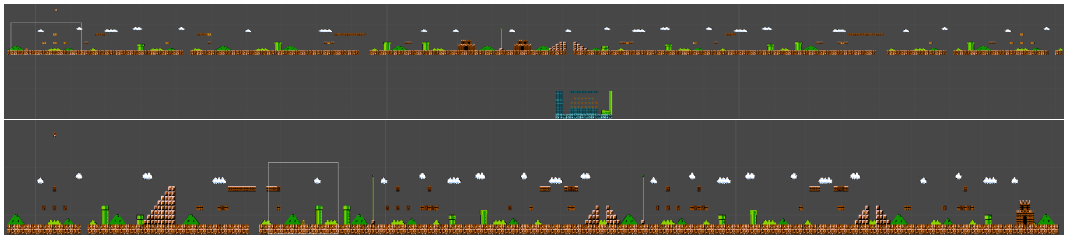

Figura 5.36: Niveles generados por cuatrigramas con corpus de entrenamiento el nivel 1-1

Damos la opción de entrenar el modelo con múltiples corpus, es decir, niveles. Leemos cada nivel y vamos ampliando el diccionario que contiene los n-gramas con los n-gramas del nuevo nivel leído. El mapa para generar comienza con la primera secuencia del primer mapa usado como entrenamiento, por lo que el mapa resultante es muy posible que se asemeje más al primero y, más si no comparten *slices* comunes. Esto se muestra en las siguientes imágenes:

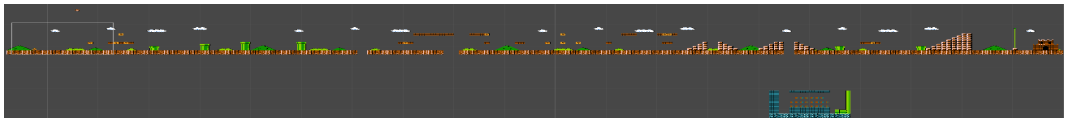

Figura 5.37: Nivel 1-1

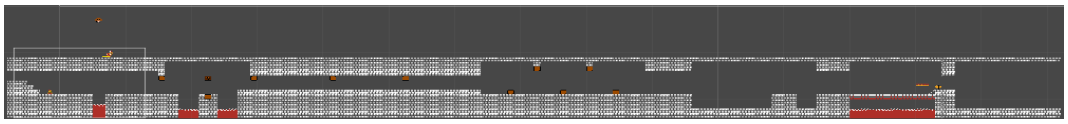

Figura 5.38: Nivel 1-4

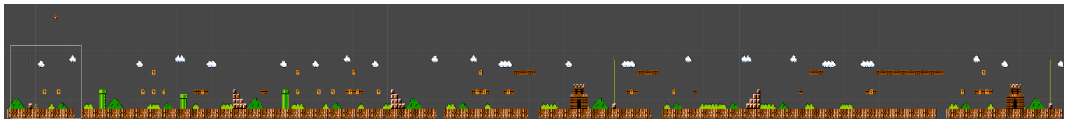

Figura 5.39: Nivel generado por bigramas con corpus de entrenamiento el nivel 1-1 y 1-4

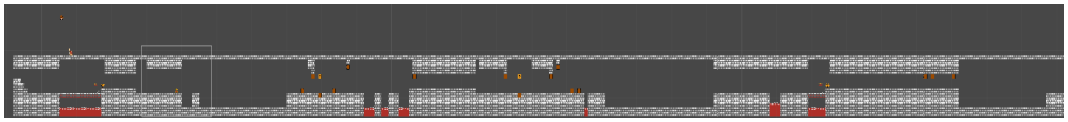

Figura 5.40: Nivel generado por bigramas con corpus de entrenamiento el nivel 1-4 y 1-1

A partir de aquí se muestran imágenes de ejemplos, de buenos mapas multitema generados:

Figura 5.41: Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1 y múltiples niveles generados con n-gramas a partir del nivel 1-1

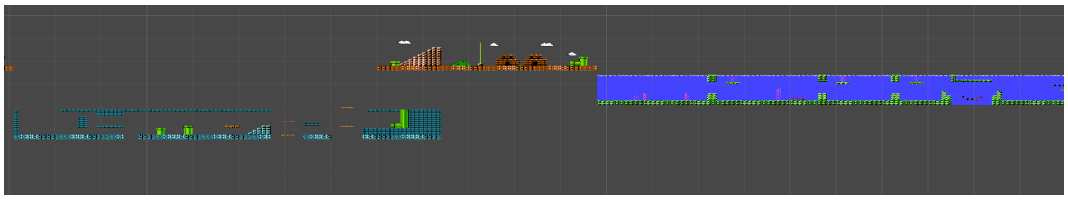

Figura 5.42: Niveles generados por trigramas con corpus de entrenamiento el nivel 2-2 y 1-2

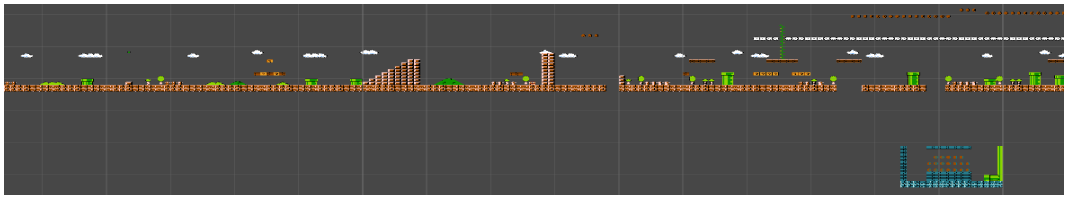

Figura 5.43: Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1 y 2-1

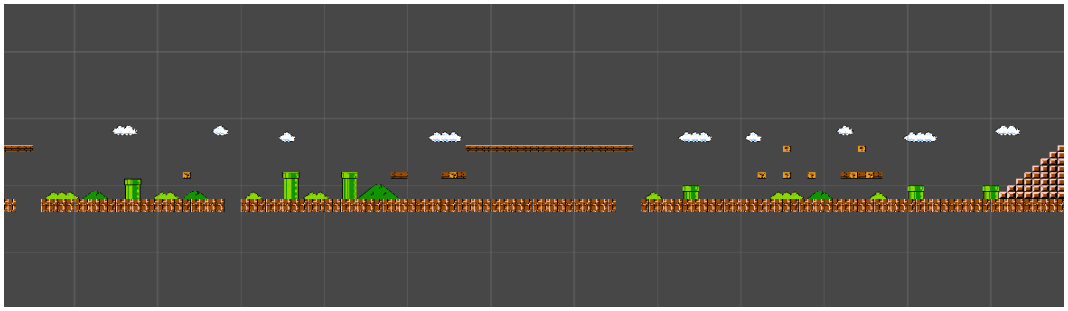

Figura 5.44: Niveles generados por trigramas usando interpolación sobre el nivel 1-1. Con peso 0.7 a trigramas, 0.2 a bigramas y 0.1 a unigramas

Una ventaja observable de los n-gramas es que no descarta palabras o *slices* poco frecuentes, como se hace en otros modelos de clasificación. Esto ofrece unos resultados con una cierta mayor variedad y no ajusta los niveles únicamente aquellas *slices* más repetidas.

Como se ha podido observar en los resultados, con el empleo de trigramas se producen mejores niveles y más similares a los del corpus de entrenamiento que usando bigramas y, bastante mejores que los resultados de los unigramas. La mezcla de varios niveles en el corpus de entrenamiento produce que los niveles generados presenten una gran variedad y un cambio en el estilo respecto a los originales. Esto se puede observar mejor en la figura 5.43 en la cual se mezclan de una manera suavizada los estilos de ambos niveles, pues comparten *slices* comunes. Sin embargo, con la figura 5.42 en la cual la mezcla de los niveles se realiza gracias a las *slices* comunes de la zona del suelo, pero más allá de eso no se realizan mezclas. Esto es normal, pues la zona subterránea no mantiene *slices* comunes con la zona de agua.

En general los resultados son buenos y podrían usarse para el juego. Tanto como para tener un mundo más amplio y añadir, de por sí, más horas de jugabilidad a Super Mario Bros, como una mecánica del propio juego mediante la cual al morir el jugador se reinicia el nivel, pero generando otro distinto en base al original. Esto permite que el jugador no vaya aprendiendo de memoria con la práctica los sitios o situaciones en las que muere y aumentar la dificultad para aquellos jugadores más experimentados. También observamos que los niveles generados en base a un solo nivel son bastante parecidos y mantienen los patrones del nivel original. Esto puede ser modificado con el valor de la temperatura, cuyo valor óptimo consideramos que se sitúa entre 0 y 1. Sin embargo, este parámetro es algo sensible, ya que si el valor es muy bajo dentro del rango 0-1, encontraremos patrones repetidos, pero si el valor es muy alto nos encontraremos con secuencias completamente aleatorias.

El modelo de n-gramas puede ser extendido a muchos otros juegos con cierta facilidad, otorgando una gran utilidad a este modelo como herramienta para crear niveles en videojuegos. El único factor a tener en cuenta sería la división del mapa a elaborar y establecer una relación entre esa división y el texto. Está bastante claro que en juegos similares al Super Mario Bros, es decir juegos de plataformas 2D, la división del mapa en *slices* verticales y esta implementación de n-gramas puede funcionar igual de bien que ha funcionado con el Super Mario Bros.

5.6.2. Redes neuronales recursivas

Los resultados de redes neuronales han sido buenos, sin embargo en muchas ocasiones surgen pequeños fallos en algunos mapas.

Figura 5.45: Mapa generado con una red GRU de 3 capas ocultas fallido

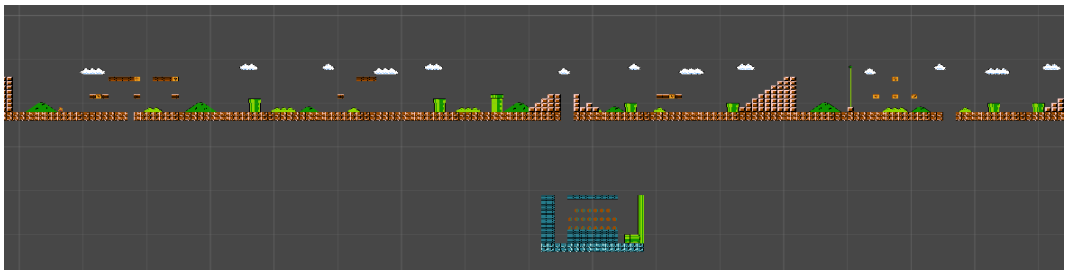

Figura 5.46: Mapa generado con una red GRU de 3 capas ocultas

Damos la opción de entrenar el modelo con múltiples corpus, es decir, niveles. Leemos cada nivel y entrenamos la red neuronal tantas veces como niveles dados. El mapa para generar comienza con la primera secuencia del primer mapa usado como entrenamiento, pero las rede neuronales intentan mezclar siempre los mapas aunque no compartan *slices*:

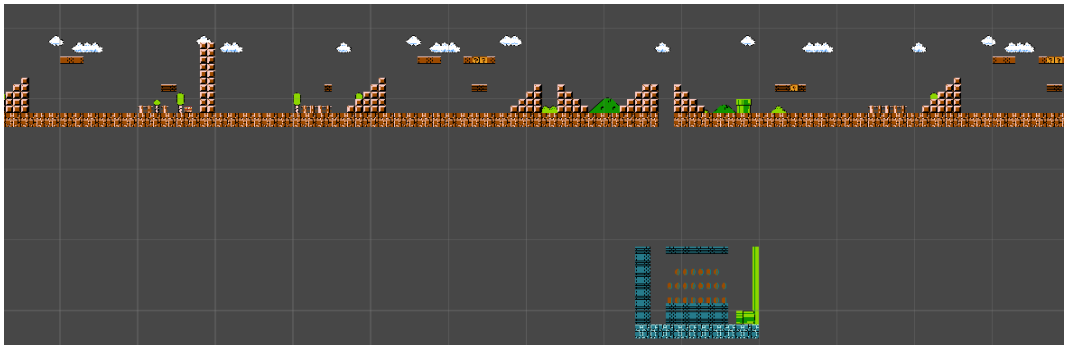

Figura 5.47: Mapa multitema 1-1 y 2-1 con redes neuronales recurrentes

Una ventaja de las redes neuronales es que pueden adaptarse al mapa original o diferir mucho de este. Como hemos visto anteriormente, conseguir el valor óptimo del parámetro de la temperatura es algo complicado, ya que números muy grandes suponen la creación de secuencias muy aleatorias. Mientras que valores muy bajos suponen prácticamente la copia del nivel original. Esto ofrece unos resultados muy originales pero que pueden ocasionar muchos fallos.

El modelo de redes puede ser aplicado a muchos otros juegos con cierta facilidad, no obstante, no podemos hacer una estimación de los resultados que podrían aparecer, ya que habría que probarlo. Podría darse la casualidad de que una configuración mala para nuestro juego fuese una buena para otro.

5.6.3. Comparativa

Con lo visto tanto en resultados de n-gramas como en resultados de redes neuronales recurrentes, podemos concluir que los resultados obtenidos son más parecidos a los niveles originales con los n-gramas. En los mapas generados por este modelo podemos comprobar que existen patrones similares o muy parecidas a los del mapa original con la que el modelo se entrenó. Sin embargo, los resultados obtenidos con redes neuronales proporcionan una mayor variedad a la hora de generar la secuencias, pero manteniendo el estilo del nivel con el que se entrenó la red.

En cuanto al tiempo que consumen ambos algoritmos, en el caso de n-gramas es un proceso muy rápido y podría decirse que no existe una tarea de "aprendizaje" como tal. En los n-gramas se estudian y calculan las probabilidades que puede tener una secuencia después de otra. Además, en el caso de n-gramas si bien es cierto que cuantos más datos de entrenamiento más opciones a generar, no es del todo necesario, ya que se adapta muy bien a tener escasos datos de entrenamiento.

Frente a los n-gramas, el modelo de red neuronal recursiva necesita un tiempo muchísimo mayor para realizar el entrenamiento. Además, para poder generar unos resultados buenos depende también del tamaño de los datos de entrenamiento. Con un corpus pequeño los niveles generados están bastante de lejos de imitar la labor de un diseñador y proporcionar una jugabilidad aceptable.
Si se proporciona una cantidad de datos aceptable, se puede decir que las redes neuronales resultan en niveles muy buenos con los que se mantiene un estilo similar al original, pero que añade diversidad entre los patrones. Por otro, lado los n-gramas ofrecen unos niveles bastante buenos también, pero con la necesidad de una menor cantidad de datos y un menor tiempo de entrenamiento.

Podemos concluir que los n-gramas son suficientes para conseguir el objetivo de crear nuevos niveles para SMB, debido a su eficacia, menor dependencia del tamaño del corpus y su mayor rapidez en el proceso de entrenamiento. Las redes neuronales pueden llegar a otorgar niveles mucho mejores que los generados por n-gramas pero requieren un muchísimo más tiempo. Aun así, se podrían combinar ambos modelos para conseguir resultados perfectos, es decir, usar el modelo de n-gramas para generar niveles en base a los originales del juego y, emplear estos como corpus para el modelo neuronal.

Capítulo 6

Conclusiones

6.1. Conclusiones

Hemos estudiado el aprendizaje automático y muchos de sus tipos, algoritmos, posibilidades y aplicaciones. Hemos visto aprendizaje supervisado, no supervisado y por refuerzo. Dentro de estos distintos tipos de algoritmos, algunos más comunes y prácticos y otros menos. También hemos investigado la generación procedural de contenido, su clasificación desde distintos puntos de vista y según los algoritmos a utilizar. Además, a lo largo de esto hemos ido viendo las diferencias entre algunos conceptos que a priori pueden parecer que son sinónimos, pero que no lo son, como podría ser el caso de aleatorio y procedural. Una vez vistos muchos de los caminos que existen planteamos el estudio de aplicar la generación procedural de contenido en el mundo de los videojuegos, un campo que lo explota bastante bien, en concreto al juego SMB[1].
Además, decidimos comparar dos de sus vertientes, una de ellas orientada a modelos probabilísticos basados en probabilidades, estadísticas y valores pseudoaleatorios. Por otro lado, un modelo más complejo usando el aprendizaje automático.

Como hemos visto en capítulos anteriores, los modelos elegidos han sido n-gramas y redes neuronales recursivas. Ambos recibían la misma forma de *input*, sin embargo, este es tratado de distinta forma antes de su procesamiento. En ambos casos los mapas eran tratados como palabras y la función de ambos algoritmos era predecir las siguientes, es decir, generar texto o en nuestro caso el nivel.

Como hemos podido observar y concluir, n-gramas es utilizado en labores referidas al texto, sirve tanto para su predicción como para la detección de plagio. Se ha demostrado que los n-gramas pueden ser usados para generar correctamente niveles parecidos a los usados en el entrenamiento. Es un modelo rápido y eficaz, con el cual, una vez tienes la información de los niveles, obtiene las posibles siguientes secuencias dependiendo del valor de n. Este proceso es eficiente y proporciona unos resultados confiables a nivel de jugabilidad y estética. Otra de sus grandes ventajas es que apenas necesita información de entrada, puesto que con un solo nivel usado para su entrenamiento es capaz de generar niveles muy similares. Como hemos podido observar con mapas de entre 150 a 250 *slices* de longitud los resultados son buenos. El uso de trigramas proporcionan unos mejores resultados, pues se aprecia mejor la aparición de patrones que resultan en un mejor aspecto visual y, muchas

[1]Super Mario Bros

veces a nivel de jugabilidad. Esto ocurre porque la generación de patrones facilita la aparición de plataformas, es decir bloques unidos, bloques con premio, precipicios o montañitas, por ejemplo. A medida que se entrena el modelo con un corpus mayor, combinado con los distintos niveles, se favorece la mezcla de estilos de los niveles. Por lo que, cuanto mayor sea el corpus de entrenamiento y su variedad del vocabulario, mayor será la diversidad de los niveles.

Por otro lado, las redes neuronales recursivas. Existen distintos tipos, las simples, las LSTM y las GRU. Cada una de ellas con sus propiedades. Es un modelo lento, necesita de un proceso de aprendizaje que requiere un esfuerzo y un tiempo. No es realizar una serie de operaciones sobre unos números para sacar una secuencia u otra. Requiere de saber que tipo de red es más idónea para cada problema. Existen una gran cantidad de parámetros para configurar una red neuronal. Algunos de estos pueden ser tanto el número como el orden de sus capas, el número de neuronas de la capa de entrada, de las capas ocultas o de la capa de salida, o distintos algoritmos optimizadores. Además de algunos parámetros muy parecidos como la longitud de las secuencias a tratar. Incluso parámetros más complejos como pueden ser la representación de las palabras a la hora de procesarlas, estas transformaciones que recibe el *input* se llama vectorización. En cuanto al tema de información necesaria, necesitan una gran cantidad de datos para poder trabajar de forma correcta. Como hemos dicho, el uso de redes neuronales recursivas requiere de un esfuerzo, pero además requiere de un tiempo de entrenamiento. Obviamente no es lo mismo utilizar dos épocas de entrenamiento que cien. Sin embargo, una vez escogidos buenos parámetros y entrenado el tiempo necesario, las redes neuronales ofrecen muy buenos resultados.

En comparación con n-gramas, se puede apreciar como estos, aumentando mucho la n, por ejemplo, cuatrigramas, ofrecían resultados con patrones del nivel original copiados y pegados con pequeñas uniones entre ellos. Dichas conexiones eran también patrones del mapa original. El resultado final es que conseguimos mapas muy parecidos a los originales, pero con secuencias colocadas en distinto orden. Si nos vamos al otro extremo con valores de n bajos, como unigramas, se aprecian resultados muy arbitrarios. Sin embargo, en los resultados de redes, se ve como se mantiene la idea del mapa sin llegar a notarse una gran cantidad de patrones copiados. Se podría decir que la red se acerca a seguir el mismo proceso creativo que siguió el diseñador en su momento.

Esto no quiere decir que los resultados de uno sean mejores o peores que los del otro. Simplemente cada uno cumple unas características y ofrece unas cosas a cambio de otras.

6.2. Posibilidades

Las posibilidades de la generación procedural de contenido en el mundo de los videojuegos es algo bastante relevante y con mucho futuro, pues se puede extender a muchas áreas de los videojuegos. Esto es algo de lo que las grandes empresas son conscientes e intentan aprovechar sobre todo por el beneficio económico. Esta técnica es cada vez más usada a la hora de crear grandes títulos. Las compañías están aprendiendo que con un buen modelo pueden ahorrarse mucho tiempo, esfuerzo y dinero obteniendo resultados lo suficientemente buenos que pueden imitar la labor

de un diseñador.

Tras haberlo estudiado bastante más a fondo, podemos ver como esta práctica va a ir cada vez a más y más, sobre todo gracias a su mezcla con el aprendizaje automático. Algo sorprendente es que el aprendizaje automático se utilice en todos los sectores del mundo, desde ayudar en granjas a producir óptimamente hasta conducir vehículos. Sin embargo, la generación procedural apenas tiene el público de los videojuegos y algún pequeño impacto en el mundo del cine y de la música. Se podría decir que es explotada por los sectores más artísticos de la sociedad, casi en su 100 % por el sector de los videojuegos.

6.3. Limitaciones

Las limitaciones de la generación procedural vienen dadas por el tipo de algoritmo utilizado. No es lo mismo basarte en un modelo que genera valores pseudoaleatorios cuyas limitaciones son muchas más que las de un modelo basado en algoritmos de aprendizaje automático. Como hemos visto el aprendizaje automático se utiliza en una gran cantidad de aplicaciones distintas, lo que se traduce en que sus limitaciones no son algo tan significativo. No obstante, también las tiene, no porque sean difíciles de alcanzar signifique que no las tenga. Y es que uno de los mayores problemas del aprendizaje automático es una de sus ventajas, la información y los datos. Hay una gran cantidad de estudios que demuestran que todavía se puede engañar con relativa facilidad a las máquinas.

En uno de estos experimentos trataron con una de las mejores redes neuronales, VGG-19. Mostraron a esta red imágenes en color de animales y objetos modificadas. Por ejemplo, una mostraba una tetera, pero con textura de pelota de golf, otra, por ejemplo, las rayas de las cebras, pero en un camello. VGG-19 solo puedo hallar 5 de 40 objetos en la primera ronda. Básicamente sobre entrenaban la red con unos datos sin enseñarle posibles modificaciones, aunque no reales, de esos datos. De esta forma algo que el ser humano distinguiría rápidamente, la máquina no pudo.

Otro experimento fue una inteligencia artificial basada en datos de pacientes de un hospital. Esta decidía que pacientes podían entrar al hospital para ser atendidos y cuales no dependiendo de la enfermedad y distintas características como edad, peso, etc. El algoritmo evaluaba muy bien los casos que oscilaban entre las edades más comunes de la población. Sin embargo, en edades como por ejemplo los 100 años no te debían atender en el hospital tuvieses lo que tuvieses, aunque fuese un simple constipado. Esto era debido a la poca información acerca de ese tipo de casos.

6.4. Ampliaciones

Se pueden probar los algoritmos desarrollados en muchos juegos y ver los resultados. Nosotros los hemos aplicado al juego Super Mario Bros, no obstante, estaría bien estudiar sus posibles aplicaciones en otros juegos del mismo estilo. Juegos de plataformas 2D como pueden ser el Donkey Kong, el Castlevania o el Metroid. Incluso el algoritmo de n-gramas puede ser aplicable a juegos con vista *top down* en 2D, solamente sería encontrar la división óptima del mapa para ese tipo de juegos.

Como hemos dicho anteriormente, en juegos de plataformas 2D puede mantenerse la división en *slices* verticales y probar esta implementación del algoritmo. Los resultados a priori no deberían de ser malos, ya que aparentemente se plantea el mismo problema. Un ejemplo de juegos *top down* pueden ser los Zelda o los escenarios de los Pokémon. En cuanto a las redes neuronales habría que probar y ver los resultados. No obstante nos parecería muy interesante el hecho de aplicar este mecanismo y más concretamente, esta implementación en distintos tipos de juegos, tanto los que comparten género, como los que no.

Otra posible ampliación sería el uso de redes neuronales convolucionales para la creación de niveles. Utilizar imágenes de los mapas generados para crear nuevos mapas. O para mezclar los ya existentes.

Otra posible ampliación sería la implementación de redes neuronales convolucionales para el análisis de la situación dentro del juego y desarrollar un algoritmo que juegue al juego. Para ello la red neuronal analizaría la situación dentro del juego, estudiando el mapa y posibles enemigos y gracias a un proceso de aprendizaje automático aprender como moverse dentro del propio nivel. Estas mismas redes convolucionales podrían servir para estudiar los mapas generados y si existen diferencias, por ejemplo, en el número de bloques normales generados por un modelo como n-gramas y un modelo de red neuronal o si existe algún patrón que siga una red neuronal LSTM, GRU o simple.

A la hora de entrenar la IA que juega los niveles de SMB probar distintos tipos de aprendizajes por refuerzo. Por ejemplo, uno de los más famosos es el proceso de decisión de Markov, del que hablamos un poco en el capítulo 3.

Otra idea sería la de añadir enemigos mediante aprendizaje automático, tanto si es como hemos hecho con los mapas, a través de la posición de los enemigos originales, como si es en función del estilo de juego del jugador.

Las aplicaciones de este proyecto son muchas y todas ellas pueden intentar aplicarse a otros videojuegos de las mismas características para ver su reacción. Esto es gracias a la gran cantidad de posibilidades que presenta el aprendizaje automático y la facilidad que tiene para mezclarse con cualquier tipo de tecnología.

Capítulo 6

Conclusions

6.1. Conclusions

We have studied machine learning and many of its types, algorithms, possibilities and applications. We have seen supervised, unsupervised and reinforcement learning. Within these different types of algorithms, some are more common and practical and others less so. We have also investigated the procedural generation of content, its classification from different points of view and according to the algorithms to be used. In addition, throughout this we have seen the differences between some concepts that may seem a priori synonymous, but are not, as might be the case of random and procedural. Once we have seen many of the existing ways, we propose to study the application of the procedural generation of content in the world of video games, a field that exploits it quite well, specifically to the game SMB[1].
In addition, we decided to compare two of its aspects, one of them oriented to probabilistic models based on probabilities, statistics and pseudo-random values. On the other hand, a more complex model using automatic learning.

As we have seen in previous chapters, the models chosen have been n-grams and recursive neural networks. Both received the same form of 'text', however this is treated differently before processing. In both cases the maps were treated as words and the function of both algorithms was to predict the next ones, that is, to generate text or in our case the level.

As we have been able to observe and conclude, n-grams are used in work referring to the text, serving both for its prediction and for the detection of plagiarism. It has been demonstrated that n-grams can be used to correctly generate levels similar to those used in training. It is a fast and effective model, with which, once you have the information of the levels, you obtain the possible following sequences depending on the value of n. This process is efficient and provides reliable results at the level of playability and aesthetics. Another great advantage is that it hardly needs any input information, since with only one level used for training it is able to generate very similar levels. As we have seen with maps of between 150 to 250 slices in length the results are good. The use of trigrams provides better results, since the appearance of patterns is better appreciated, resulting in a better visual aspect and, many times, at the level of playability. This happens because the generation of patterns facilitates the appearance of platforms, i.e. joined blocks, prize blocks, cliffs or mountains, for

[1]Super Mario Bros

example. As the pattern is trained with a larger corpus, combined with the different levels, the mix of styles of the levels is favored. Therefore, the larger the training corpus and its variety of vocabulary, the greater the diversity of the levels.

We have studied machine learning and many of its types, algorithms, possibilities and applications. We have seen supervised, unsupervised and reinforcement learning. Within these different types of algorithms, some are more common and practical and others less so. We have also investigated the procedural generation of content, its classification from different points of view and according to the algorithms to be used. In addition, throughout this we have seen the differences between some concepts that may seem a priori synonymous, but are not, as might be the case of random and procedural. Once we have seen many of the existing ways, we propose to study the application of the procedural generation of content in the world of video games, a field that exploits it quite well, specifically to the game SMB[2].
In addition, we decided to compare two of its aspects, one of them oriented to probabilistic models based on probabilities, statistics and pseudo-random values. On the other hand, a more complex model using automatic learning.

As we have seen in previous chapters, the models chosen have been n-grams and recursive neural networks. Both received the same form of 'text', however this is treated differently before processing. In both cases the maps were treated as words and the function of both algorithms was to predict the next ones, that is, to generate text or in our case the level.

On the other hand, recursive neural networks. There are different types, the simple ones, the LSTM and the GRU. Each with its own properties. It is a slow model, it needs a learning process that requires effort and time. It is not to carry out a series of operations on some numbers to get one sequence or another. It requires to know what type of network is more suitable for each problem. There are a lot of parameters to configure a neural network. Some of these can be both the number and order of its layers, the number of neurons in the input layer, hidden layers or the output layer, or different optimizing algorithms. In addition to some very similar parameters such as the length of the sequences to be treated. Even more complex parameters such as the representation of the words when processing them, these transformations received by the text input are called vectorization. As for the information needed, they need a large amount of data to be able to work correctly. As we have said, the use of recursive neural networks requires an effort, but it also requires a time of training. Obviously, it is not the same to use two training periods than one hundred. However, once good parameters have been chosen and the necessary time trained, neuronal networks offer very good results.

Compared to n-grams, it can be seen how these, by greatly increasing the n, for example, quadrigrams, offered results with patterns from the original level copied and pasted with small joints between them. These connections were also patterns from the original map. The final result is that we get maps very similar to the original ones, but with sequences placed in different order. If we go to the other extreme with low n values, like unigrams, we can see very arbitrary results.

[2]Super Mario Bros

However, in the network results, you can see how the idea of the map is maintained without a large number of copied patterns being noticed. You could say that the network is close to following the same creative process that the designer followed at the time. This does not mean that the results of one are better or worse than the other. Each one simply fulfills some characteristics and offers some things in exchange for others.

6.2. Possibilities

The possibilities of procedural generation of content in the world of video games is something quite relevant and with a lot of future, since it can be extended to many areas of video games. This is something that large companies are aware of and try to take advantage of above all for economic benefit.

This technique is increasingly used when creating great titles. Companies are learning that with a good model they can save a lot of time, effort and money by getting results that are good enough to imitate the work of a designer.

After having studied it a lot more thoroughly, we can see how this practice will go more and more, especially thanks to its mix with automatic learning. What is surprising is that machine learning is used in all sectors of the world, from helping on farms to produce optimally to driving vehicles. However, the procedural generation hardly has an audience for video games and some small impact on the world of film and music. It could be said that it is exploited by the more artistic sectors of society, almost 100 % by the video game sector.

6.3. Limitations

The limitations of the procedural generation are given by the type of algorithm used. It is not the same to base yourself on a model that generates pseudo-random values whose limitations are much more than those of a model based on automatic learning algorithms. As we have seen, machine learning is used in many different applications, which means that its limitations are not so significant. However, it also has them, not because they are difficult to achieve does not mean that it does not have them. One of the biggest problems with machine learning is one of its advantages, information and data. There are a lot of studies that show that machines can still be fooled relatively easily.

In one of these experiments they treated one of the best neural networks, VGG-19. They showed this network color images of animals and modified objects. For example, one showed a teapot, but with the texture of a golf ball, another, for example, the stripes of zebras, but on a camel. VGG-19 can only find 5 out of 40 objects in the first round. They basically overworked the network with some data without showing possible, though not real, modifications of that data. In this way something that humans would quickly distinguish, the machine could not.

Another experiment was an artificial intelligence based on patient data from a hospital. It decided which patients could enter the hospital to be treated and

which not depending on the disease and different characteristics such as age, weight, etc. The algorithm evaluated very well the cases that oscillated between the most common ages of the population. However, in ages such as 100 years old you should not be treated in the hospital whatever you have, even if it is a simple constipation. This was due to the lack of information about this type of case.

6.4. Extensions

You can test the algorithms developed in many games and see the results. We applied them to the Super Mario Bros. game, but it would be good to study their possible applications in other games of the same style. 2D platform games like Donkey Kong, Castlevania, and Metroid. Even the n-gram algorithm can be applied to games with a 2D top-down view, it would only be to find the optimal map division for that type of game. As we said before, in 2D platform games you can keep the division in vertical slices and try this implementation of the algorithm. The results a priori should not be bad since apparently the same problem arises. An example of top down games can be Zelda or Pokémon scenarios. As for the neural networks should be tested and see the results. However, we would find it very interesting to apply this mechanism and more specifically, this implementation in different types of games, both those that share gender, and those that do not.

Another possible extension would be the use of convolutional neural networks to create levels. Use images of the generated maps to create new maps. Or to mix up existing ones.

Another possible extension would be the implementation of convolutional neural networks for the analysis of the situation within the game and to develop an algorithm that plays the game. To do this, the neural network would analyze the situation within the game, studying the map and possible enemies and thanks to an automatic learning process learn how to move within the level itself. These same convolutional networks could be used to study the maps generated and whether there are differences, for example, in the number of normal blocks generated by a model such as n-gram and a neural network model or whether there is a pattern that follows an LSTM, GRU or simple neural network.

When training AI playing SMB levels try different types of learning by reinforcement. For example, one of the most famous is Markov's decision process, which we talked about a bit in chapter 3. Another idea would be to add enemies by automatic learning, either as we have done with the maps, through the position of the original enemies, or depending on the player's game style. The applications of this project are many and all of them can try to be applied to other video games of the same characteristics to see their reaction. This is thanks to the great amount of possibilities that automatic learning presents and the ease it has to mix with any type of technology.

Capítulo 7

Contribuciones

7.1. Víctor Emiliano Fernández Rubio

La primera tarea al comenzar el proyecto fue la investigación del tema que nos estábamos proponiendo estudiar. Por ello realice una búsqueda sobre las características y los usos del aprendizaje automático y de la generación procedural de contenidos. Tras buscar información más genérica pasé a buscar información más relativa al mundo de los videojuegos. Tras reunir información y hablar con nuestro tutor continuamos con un libro de la universidad de Stanfor. Este libro trataba todo el tema de procesamiento de texto y lenguajes, desde normalización de texto hasta analizar el sentido lógico de frases y expresiones, pasando por distintos algoritmos. Un libro muy interesante que nos ha ayudado a resolver bastantes dudas a lo largo de todo el proyecto y con el que se podrían realizar muchas ampliaciones. En este libro había mucha información sobre n-gramas y redes neuronales recursivas. Tras leernos los capítulos del libro que nos afectaban comenzamos a realizar una serie de ejercicios planteados a lo largo de estos capítulos para confirmar que habíamos entendido lo estudiado. Tras esto decidimos realizar las primeras implmentaciones de n-gramas.

Al mismo tiempo que se realizaron las primeras implementaciones de n-gramas, se comenzó el proyecto de Unity, la aplicación donde posteriormente se mostrarían los mapas generados. En Unity comencé ayudando con las primeras implementaciones del movimiento, la búsqueda de recursos, y la creación de los "prefabs" para los bloques para poder generar más tarde los mapas. También ayudé con la interacción de algunos elementos de la escena como romper ladrillos o crecer al coger los champiñones.

Tras realizar las primeras implementaciones de n-gramas generando pequeños fragmentos de texto empecé a crear mapas con Tiled. Descargué las imágenes originales de los primeros mundos y construí los mapas orinales de Super Mario Bros. Para ello busque imágenes de los sprites y las cargue en Tiled como hoja de patrones y cree los 8 primeros niveles originales de juego Super Mario Bros de la NES. Los mapas fueron evolucionando debido a distintos cambios en el proyecto, por lo que estuve cambiandolos durante todo el proyecto.

Tras una serie de cambios hubo que hacer una segunda implementación de n-gramas en la que en una primera versión participamos los tres integrantes del grupo a partes iguales y, más tarde, las últimas implementaciones, detalles y retoques fue-

ron divididos entre Carlos y yo. Tras esto empecé a desarrollar la idea de multitema. Multitema es la idea de mezclar las distintas zonas que ofrece el juego de Mario en un solo mapa generado proceduralmente. Para ello estuve desarrollando formas para poder hacerlo. Para ello estuve investigando más profundamente como funcionaban los n-gramas y si mi idea realmente iba a funcionar. Tras tenerlo claro empecé a desarrollarlo de forma bastante exitosa. Una vez mezcladas dos temáticas distintas fui añadiendo más. Tras esto les expuse lo conseguido a mis compañeros y al tutor para añadirlo al proyecto. Otra ampliación más fue la interpolación en n-gramas. Estuvimos estudiando entre los tres la teoría que nos ofrecía el libro de Standford, y de si era algo interesante o no para la generación de mapas. Tras llegar a la conclusión de que era necesario añadirlo, implementé esta ampliación en el código de n-gramas. Para ello tuve que mirar más a fondo el apartado y las distintas fórmulas que planteaba. Además de las implementaciones me dedique a generar distintos mapas par poder porbar a fondo las ideas de multitema, para comporbar que funcionaba bien, y de interpolaridad.

Tras cerrar prácticamente n-gramas, comenzamos de nuevo documentándonos todos acerca de redes neuronales recursivas. Distintas librerías y entornos que utilizar. Mientras buscábamos información, estructuré el índice de la memoria. Este fue algo modificado posteriormente, pero si que es cierto que nos ayudo para tener las ideas mas claras y ordenadas.

Después hice la primera implementación del algoritmo de redes neuronales basándome en los tutoriales de TensorFlow y con ayuda de su documentación. Después de la primera implementación, participé en la segunda implementación de redes neuronales más orientada a generación de niveles. Para ello ditrubuimos en tareas esta segunda implementación.

Una vez acabados los dos algoritmos principales, decidimos dividirlos en dos partes, una de entrenamiento y otra de generación. Para lo cual volvimos a repartir esta tarea en subtareas más pequeñas y las dividimos en partes iguales. Por ultimo, en cuanto a la escritura de la memoria, escribí los capítulos del uno al cuatro en español y el seis, en cuanto al 5 relice modificaciones, revisión y algo de organización de algún apartado.

7.2. Gonzalo Guzmán del Rio

Como primera tarea, al empezar el desarrollo e investigación del TFG, y con el objetivo final ya fijado, fue empezar a investigar sobre qué técnicas de aprendizaje automático podrán ser más útiles o más "cómodas" para la generación de mapas. Para ello empecé leyendo diferentes tesis y ensayos de la universidad de Stanford que trataban el tema de la propia generación de mapas con n-gramas interpretando los datos de una manera concreta (por *slices*, como hemos hecho finalmente) y sobre el procesado de secuencias con redes neuronales recursivas. Una vez que cada uno de nosotros terminamos de leernos estos ensayos, nos encargamos de realizar de manera individual una serie de ejercicios que venían adjuntados para así poder comprender de manera más concreta todo lo que habíamos leído anteriormente. Tras ello, un primer acercamiento, fue el de realizar un pequeño prototipo para la generación de texto a través de n-gramas.

Durante este proceso, de manera paralela, fuimos realizando individualmente diversas tareas dentro de la aplicación de Unity, como podían ser el movimiento básico de Mario, así como el salto. En mi caso, además de lo anterior, yo me encargué de realizar el seguimiento de la cámara estilo Super Mario. Para complementar a la cámara, también me encargué de realizar un modo vista. Este modo vista, al cual se accede con la tecla L, permite ver el mapa generado o cargado de manera libre, sin tener que mover al jugador, moviéndote por él con las teclas de dirección y haciendo zoom con la rueda del ratón.

Tras ello, entre los tres nos encargamos de seguir intentando implementar de manera correcta el algoritmo de n-gramas. Tras tener una versión previa del algoritmo, Víctor y Carlos siguieron trabajando en ello. Yo me encargué de realizar entonces de manera más concreta todo lo relacionado con la aplicación de Unity. En un primer momento, cree un menú inicial que permitiría elegir entre todos los mapas disponibles para cargarlos individualmente o generarlos (pero sin añadir la funcionalidad). Una vez tenía todo esto funcionando me encargué de, con ayuda de Carlos y Víctor, pues cada uno buscamos diferentes maneras de conectar Unity con Python, de la ejecución del proceso para lanzar el *script* de Python. Para lanzar este proceso se crearon diversos *InputFields*, cuyos valores eran recogidos para luego ser plasmados en el comando requerido por el *script* de n-gramas. Este menú permitiría la generación de un único mapa (monotemático). Además de todo ello, una vez Víctor había terminado de implementar el multitema en n-gramas, con la ayuda de Carlos me encargué de hacer la entrada y salida de las zonas subterráneas. Tras ello, y tras la implementación de Víctor, me encargué de permitir que Unity cambiara los parámetros para poder introducir las nuevas variables requeridas por los procesos. Tras ello, los 3 empezamos a leer acerca de TensorFlow y de redes neuronales recursivas. Una vez tuvimos una primera versión me encargué de simular todo lo relacionado con n-gramas en la aplicación de Unity, pero esta vez con redes neuronales recursivas. Con un primer *script* de RNN, cada uno, nos encargamos individualmente, de generar múltiples mapas con diferentes parámetros para tener una primera impresión de RNN. La mayor parte del tiempo de RNN lo invertimos en realizar el *script*, con la ayuda de TensorFlow, que se adaptase a nuestro conjunto de datos. Además gran parte del tiempo lo ocupó también el hecho de estudiar e investigar cuál era el significado real de los parámetros con los que teníamos que trabajar, así como afectaban al resto de la red.

Con todo ello realizado, continúe con la aplicación de Unity, implementando nuevas mecánicas como la de nadar, o mejorando ciertos aspectos que no estaban bien realizados como la entrada a tuberías. También me encargué de añadir comportamiento a los *tiles* del mapa que lo requerían. Explicándolo grosso modo, me encargué de seguir mejorando todo lo relacionado con Unity, haciendo la aplicación más interactiva y visual, y arreglando jugabilidad y más mecánicas que eran incorrectas. Aparte, me encargué de hacer una nueva escena con todo lo necesario para poder ejecutar la aplicación de manera correcta con imágenes descriptivas.

Además de todo lo comentado anteriormente, me he estado documentando acerca de otros tipos de redes neuronales como las convolucionales y de ciertos detalles relacionados con n-gramas como el *smoothing* o interpolación.

Relacionado con la memoria, me he encargado de escribir la parte de aprendizaje no supervisado del capítulo 3 y conclusiones del capítulo 4. Del capítulo 5 me he encargado de realizar la sección de mecánicas y características, y en conjunto con Víctor, los puntos 5.3 y 5.4. En cuanto a la sección 5.5, en lo que se refiere al apartado de RNN, me he encargado de escribir gran parte de todo lo relacionado con ello. Además de todo esto, revisar cambios necesarios dentro de la memoria, así como de leer la memoria varias veces para corregir todas las posibles faltas de ortografía que pudiera haber o frases mal estructuradas.

7.3. Carlos Llames Arribas

Una vez decidimos el tema del proyecto, empecé buscando bibliografía e información disponible relacionada con la generación automática de niveles/mapas en videojuegos.

En una de las primeras reuniones con nuestro director Samir, nos indicó un artículo respecto a la generación procedural de contenido a través del *Machine Learning* (PCGML), mediante el cual estudiamos los distintos métodos y soluciones para generar contenido de videojuegos usando modelos entrenados mediante contenido existente. También estudiamos los problemas más comunes en esta clase de procesos de generación, tales como la falta en muchas ocasiones de datos de entrenamiento, corpus de entrenamiento pequeños, ajuste de parámetros...

Inmediatamente, después acordamos emplear el modelo n-grama como primer método a desarrollar para la generación de nuestros niveles para el juego. Decidimos utilizar este modelo debido a que es un modelo más o menos simple, pero que resulta muy útil y eficaz. Para conseguir este objetivo, realicé de manera paralela a mis compañeros una investigación acerca de este método, tanto para aprender en qué consiste, sus posibles usos (que son bastantes y comunes como se ha explicado en el capítulo correspondiente a n-gramas), el empleo y procesamiento que realiza sobre los datos de entrenamiento, etc.

Paralelamente a este proceso de investigación y estudio, fuimos creando la aplicación demo destinada a probar los niveles resultantes generados por los modelos a implementar. Así, como en mi caso, la implementación del movimiento, salto, algunos enemigos (que, aunque no se usan, podríamos ampliar fácilmente la generación de los niveles con los enemigos), la entrada y salida de tuberías. Muchas de estas tareas las realizamos conjuntamente.

Retomando tema del modelo n-grama, hice, como mis compañeros, de manera individual unos ejercicios prácticos sobre la teoría de los n-gramas para afianzar el conocimiento en esta área. Seguidamente, comenzamos a realizar la implementación de este método. Para ello, realizamos implementaciones individuales y, más tarde, pusimos en común. Y tras una versión inicial, Víctor y yo seguimos trabajando en la mejora de la implementación de este modelo. Tanto buscando librerías útiles, como seguir investigando en métodos de mejora y refinamiento para el modelo (como el *smoothing*). Primero para textos y después adaptado al tipo de datos usado por nosotros para representar los niveles. Más tarde, me encargué de realizar pruebas y obtener resultados de niveles. A continuación, cada uno buscó formas de conectar

Python con Unity y, ejecutar el *script* de Python.

Por último, comenzamos la investigación y estudio de las redes neuronales como modelo para la clasificación de texto, aprendizaje y generación. Sobre todo Víctor trabajó en la implementación del modelo, sin embargo, yo seguí investigando sobre la teoría de las redes neuronales, tales como sus tipos, funcionamiento, problemas, etc., así como sobre la API de TensorFlow. Por ejemplo, sobre las distintas funciones de activación para las redes neuronales, sus ventajas y desventajas. De este modo, tener una idea del ajuste de los distintos parámetros para mejorar la generación de los resultados. Para acabar realicé pruebas variando parámetros para ver los distintos resultados.

Respecto a la memoria del trabajo, me he encargado de escribir los objetivos en el apartado 1.2, algún apartado de los algoritmos supervisados en el 3.2.1, todo respecto a los n-gramas en el apartado 5, así como una parte de redes neuronales recursivas y el apartado de comparativa 5.6.3. Además, de todo esto he revisado la memoria y realizado los cambios necesarios para corregirla.

En resumen, me he encargado sobre todo en la investigación y estudio para tener una base y, en el desarrollo de los modelos. También, he trabajado en la demo de Unity, pero en menor medida.

7.4. Repositorios

Repositorio principal: https://github.com/PCGML-TFG

Repositorio de la aplicación de Unity: https://github.com/PCGML-TFG/SuperMarioBros

Repositorio de la memoria: https://github.com/PCGML-TFG/Memory

Repositorio de los scripts de Python: https://github.com/PCGML-TFG/PythonScripts

Repositorio del ejecutable: https://github.com/PCGML-TFG/Executable

Bibliografía

[1] Harkiran Kaur. *Top 5 Artificial Intelligence(AI) Predictions in 2020*. URL: https://www.geeksforgeeks.org/top-5-artificial-intelligenceai-predictions-in-2020/. (accessed: 04.03.2020).

[2] Ligdi González. *Historia de Machine Learning*. URL: https://ligdigonzalez.com/historia-de-machine-learning/. (accessed: 06.03.2020).

[3] *Historia de la Inteligencia Artificial*. URL: https://alexzeroblog.wordpress.com/. (accessed: 06.03.2020).

[4] Eren Gölge. *Brief History of Machine Learning*. URL: https://erogol.com/brief-history-machine-learning/. (accessed: 06.03.2020).

[5] Bernard Marr. *A Short History Of Machine Learning – Every Manager Should Read*. URL: https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/#29a7352d15e7. (accessed: 06.03.2020).

[6] Harkiran Kaur. *Top Machine Learning Trends in 2019*. URL: https://www.geeksforgeeks.org/top-machine-learning-trends-in-2019/. (accessed: 06.03.2020).

[7] Sophia Martin. *10 Most Popular Machine Learning Software Tools in 2020*. URL: https://towardsdatascience.com/10-most-popular-machine-learning-software-tools-in-2019-678b80643ceb. (accessed: 06.03.2020).

[8] Mehedi Hasan. *Top 20 Best AI and Machine Learning Software and Frameworks in 2020*. URL: https://www.ubuntupit.com/best-ai-and-machine-learning-software-and-frameworks/. (accessed: 06.03.2020).

[9] NH Learning Solutions. *7 Industries Leveraging Machine Learning*. URL: https://nhlearningsolutions.com/blog/7-industries-leveraging-machine-learning. (accessed: 07.03.2020).

[10] Daniel Faggella. *Machine Learning Healthcare Applications – 2018 and Beyond*. URL: https://emerj.com/ai-sector-overviews/machine-learning-healthcare-applications/. (accessed: 07.03.2020).

[11] Roman Chuprina. *Machine Learning in Finance: Benefits, Use Cases and Opportunities*. URL: https://www.datasciencecentral.com/profiles/blogs/machine-learning-in-finance-benefits-use-cases-and-opportunities. (accessed: 07.03.2020).

[12] Naveen Joshi. *How AI Can Transform The Transportation Industry*. URL: https://www.forbes.com/sites/cognitiveworld/2019/07/26/how-ai-can-transform-the-transportation-industry/#7f62084c4964. (accessed: 07.03.2020).

[13] *Machine learning in agriculture: How AI helps solve the industrys most pressing challenges*. URL: https://objectcomputing.com/expertise/machine-learning/machine-learning-in-agriculture. (accessed: 07.03.2020).

[14] John Stephenson. *6 Ways Machine Learning will be used in Game Development*. URL: https://www.logikk.com/articles/machine-learning-in-game-development/. (accessed: 07.03.2020).

[15] Alind Gupta. *6 Ways Machine Learning has revolutionized Video Game Industry*. URL: https://www.geeksforgeeks.org/6-ways-machine-learning-has-revolutionized-video-game-industry/. (accessed: 07.03.2020).

[16] Harkiran Kaur. *Top 5 Trends in Artificial Intelligence That May Dominate 2020s*. URL: https://www.geeksforgeeks.org/top-5-trends-in-artificial-intelligence-that-may-dominate-2020s/. (accessed: 07.03.2020).

[17] Redacción APD. *¿Qué es Machine Learning y cómo funciona?* URL: https://www.apd.es/que-es-machine-learning/. (accessed: 15.03.2020).

[18] Inc. Wikimedia Foundation. *Machine learning*. URL: https://en.wikipedia.org/wiki/Machine_learning. (accessed: 15.03.2020).

[19] S.A Iberdrola. *Descubre los principales beneficios del 'Machine Learning'*. URL: https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico. (accessed: 15.03.2020).

[20] Anders Norén. *Clasificación de Aprendizaje automático supervisado*. URL: https://sitiobigdata.com/2019/12/24/clasificacion-de-aprendizaje-automatico-supervisado/#. (accessed: 16.03.2020).

[21] Daniel Rodriguez. *La regresión logística*. URL: https://www.analyticslane.com/2018/07/23/la-regresion-logistica/. (accessed: 16.03.2020).

[22] Regresión Logística para Clasificación. *Jose Martinez Heras*. URL: https://iartificial.net/regresion-logistica-para-clasificacion/. (accessed: 16.03.2020).

[23] Inc. Wikimedia Foundation. *Accuracy and precision*. URL: https://en.wikipedia.org/wiki/Accuracy_and_precision. (accessed: 16.03.2020).

[24] Dave Anderson y George McNeill. "ARTIFICIAL NEURAL NETWORKS TECHNOLOGY". En: *Annalen der Physik* (1992). URL: http://andrei.clubcisco.ro/cursuri/f/f-sym/5master/aac-nnga/AI_neural_nets.pdf.

[25] Oscar García-Olalla Olivera. *Redes Neuronales artificiales: Qué son y cómo se entrenan*. URL: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i. (accessed: 16.03.2020).

[26] Fjodor Van Veen. *The Neural Network Zoo*. URL: https://www.asimovinstitute.org/neural-network-zoo/. (accessed: 17.03.2020).

[27] Steven L. Brunton. *Neural Network Architectures*. URL: https://www.youtube.com/watch?v=oJNHXPs0XDk. (accessed: 17.03.2020).

[28] Inc Wikimedia Foundation. *Ensemble learning*. URL: https://en.wikipedia.org/wiki/Ensemble_learning. (accessed: 17.03.2020).

[29] Wikipedia. *Support vector machine*. URL: https://en.wikipedia.org/wiki/Support_vector_machine. (accessed: 07.04.2020).

[30] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. (accessed: 07.04.2020).

[31] Savan Patel. *Chapter 2 : SVM (Support Vector Machine) — Theory*. URL: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. (accessed: 07.04.2020).

[32] Inc Wikimedia Foundation. *Decision tree learning*. URL: https://en.wikipedia.org/wiki/Decision_tree_learning. (accessed: 03.04.2020).

[33] Anders Noren. *Árbol de decisión en Machine Learning (Parte 1)*. URL: https://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/. (accessed: 03.04.2020).

[34] Anders Noren. *Árboles de decisión en Machine Learning (Parte 2)*. URL: https://sitiobigdata.com/2019/12/14/arboles-de-decision-en-machine-learning-parte-2/. (accessed: 03.04.2020).

[35] guru99. *Unsupervised Machine Learning: What is, Algorithms, Example*. URL: https://www.guru99.com/unsupervised-machine-learning.html. (accessed: 04.04.2020).

[36] Google. *Machine Learning*. URL: https://developers.google.com/machine-learning/glossary#centroid. (accessed: 05.04.2020).

[37] Google. *Clustering algorithms*. URL: https://developers.google.com/machine-learning/clustering/clustering-algorithms. (accessed: 05.04.2020).

[38] Wikipedia. *Association rule learning*. URL: https://en.wikipedia.org/wiki/Association_rule_learning. (accessed: 05.04.2020).

[39] Imad Dabbura. *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*. URL: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. (accessed: 07.04.2020).

[40] E.M. Mirkes. *K-means and K-medoids applet. University of Leicester, 2011*. URL: http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html. (accessed: 06.04.2020).

[41] Matt Brems. *A One-Stop Shop for Principal Component Analysis*. URL: https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c. (accessed: 07.04.2020).

[42] Guru99 2020. *Reinforcement Learning: What is, Algorithms, Applications, Example*. URL: https://www.guru99.com/reinforcement-learning-tutorial.html. (accessed: 03.04.2020).

[43] Ayush Singh. *Reinforcement Learning : Markov-Decision Process (Part 1)*. URL: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da. (accessed: 03.04.2020).

[44] geeksforgeeks. *Q-Learning in Python*. URL: https://www.geeksforgeeks.org/q-learning-in-python/. (accessed: 04.04.2020).

[45] Vida Extra Gallego. *La revolución procedural va a llegar*. URL: https://www.vidaextra.com/industria/la-revolucion-procedural-va-a-llegar. (accessed: 06.05.2020).

[46] Martín González Hermida y Enrique Costa Montenegro. "ESTUDIO DE TÉCNICAS DE GENERACIÓN PROCEDURAL EN EL MUNDO DE LOS VIDEOJUEGOS". En: (2018). URL: http://castor.det.uvigo.es:8080/xmlui/bitstream/handle/123456789/230/TFG%20Mart%C3%ADn%20Gonz%C3%A1lez%20Hermida.pdf?sequence=1&isAllowed=y.

[47] Josh Bycer. *Procedural vs. Randomly Generated Content in Game Design*. URL: https://www.gamasutra.com/blogs/JoshBycer/20150807/250760/Procedural_vs_Randomly_Generated_Content_in_Game_Design.php. (accessed: 06.05.2020).

[48] Boyan Bontchev. "MODERN TRENDS IN THE AUTOMATIC GENERATION OF CONTENT FOR VIDEO GAMES". En: (2017). URL: https://core.ac.uk/download/pdf/156902177.pdf.

[49] Nicolas A. Barriga. "A Short Introduction to Procedural Content Generation Algorithms for Videogames". En: (2018). URL: https://www.researchgate.net/publication/331717755_A_Short_Introduction_to_Procedural_Content_Generation_Algorithms_for_Videogames.

[50] Alexander Hofmann y Dr. Santiago Ontañón Bernhard Rieder. "Using Procedural Content Generation via Machine Learning as a Game Mechanic". En: (2018). URL: https://static1.squarespace.com/static/559921a3e4b02c1d7480f8f4/t/5bc5afdf15fcc0ad8522a29c/1539682276053/Rieder+Bernhard_840.PDF.

[51] Wikipedia. *Procedural Generation*. URL: https://es.wikipedia.org/wiki/Generacion_por_procedimientos. (accessed: 06.05.2020).

[52] Wikipedia. *Generative Music*. URL: https://en.wikipedia.org/wiki/Generative_music. (accessed: 06.05.2020).

[53] Jim Rossignol. *Tag: Procedural Generation*. URL: http://www.bldgblog.com/tag/procedural-generation/. (accessed: 06.05.2020).

[54] spriters-resource. *Spriters-resource*. URL: https://www.spriters-resource.com/nes/supermariobros/. (accessed: 04.02.2020).

[55] Kennedy Odhiambo Ogada. *N-grams for Text Classification Using Supervised Machine Learning Algorithms*. URL: https://pdfs.semanticscholar.org/fa18/59e0bffa436150b785e3d2ff4c6f1bbd4ccd.pdf. (accessed: 01.03.2020).

[56] Daniel Jurafsky James H. Martin. *N-gram Language Models*. URL: https://web.stanford.edu/~jurafsky/slp3/3.pdf. (accessed: 20.02.2020).

[57] Mechtild Stock Wolfgang G. Stock. *Handbook of Information Science*. URL: https://books.google.es/books?id=d1PnBQAAQBAJ&pg=PA169&lpg=PA169&dq=Information+Theory+in+1948+Shannon+n-grams&source=bl&ots=h6CuTsEB-U&sig=ACfU3U13EqY5PyPRYFx6CL9Rx8w9XqUfvg&hl=es&sa=X&ved=2ahUKEwiRp5DFqN7pAhXdDWMBHbaPCQ4Q6AEwAXoECAkQAQ#v=onepage&q=Information%20Theory%20in%201948%20Shannon%20n-grams&f=false. (accessed: 10.03.2020).

[58] Google. *Quantitative Analysis of Culture Using Millions of Digitized Books*. URL: https://books.google.com/ngrams/. (accessed: 08.05.2020).

[59] Wikipedia. *Machine translation*. URL: https://en.wikipedia.org/wiki/Machine_translation. (accessed: 10.05.2020).

[60] Wikipedia. *Bag of words model*. URL: https://en.wikipedia.org/wiki/Bag-of-words_model. (accessed: 09.05.2020).

[61] Wikipedia. *Recurrent neural network*. URL: https://en.wikipedia.org/wiki/Recurrent_neural_network. (accessed: 17.05.2020).

[62] Volodymyr Bilyk. *Recurrent neural network*. URL: https://theappsolutions.com/blog/development/recurrent-neural-networks/. (accessed: 17.05.2020).

[63] Tensorflow. *Word embeddings*. URL: https://www.tensorflow.org/tutorials/text/word_embeddings. (accessed: 19.05.2020).

[64] Diego Calvo. *Función de activación – Redes neuronales*. URL: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/. (accessed: 19.05.2020).