UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

ExoplanetIA
Machine Learning para la detección de exoplanetas

Presentado por Jesús María Herruzo Luque
en Universidad de Burgos — 28 de abril de 2020
Tutor: Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Jesús María Herruzo Luque, con DNI 44372813V, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de abril de 2020

Vº. Bº. del Tutor:

D. Carlos López Nozal

Vº. Bº. del co-tutor:

D. Manuel Hermán Capitán

Vº. Bº. del co-tutor:

D. Alejandro Viloria Lanero

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

Conceptos teóricos

En esta sección se presentan los conceptos teóricos relacionados con este Trabajo Fin de Grado que sirven para facilitar la comprensión. TODO

3.1. Proceso experimental para la generación de modelos

TODO interesante comentar el marco global del proceso de creación de modelos para facilitar la lectura. Puedes ser interesantes algo similar al proceso CRISP-DM.

3.2. Repositorios de datos Kaggle-Kepler

TODO Kaggle [1].

Descripción del conjunto de datos y de los conjuntos de datos disponibles

3.3. Perceptrón Multicapa

El perceptrón multicapa, o red multicapa con propagación hacia delante, es el modelo de aprendizaje profundo por excelencia. Es una generalización del perceptrón simple que surgió debido a la incapacidad de estos para dar solución a problemas no lineales [3].

El objetivo de estas redes es aproximar alguna función f. Por ejemplo, para un clasificador como es nuestro caso, `y = f(x)` mapea un input, x a

[1] https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

una categoría y. La red define un mapeado `y = f(x, \theta)` y aprende los valores de los parametros `\theta` que resultan en la mejor aproximación a la función [1].

Su arquitectura es simple, consistiendo en una capa de entrada, encargada de recibir las señales del exterior y propagarlas a las neuronas de la siguiente capa, una o más capas ocultas, que procesan la información, aplicando una función de activación a los datos recibidos de la capa previa, y una capa de salida, que comunica al exterior la respuesta de la red.

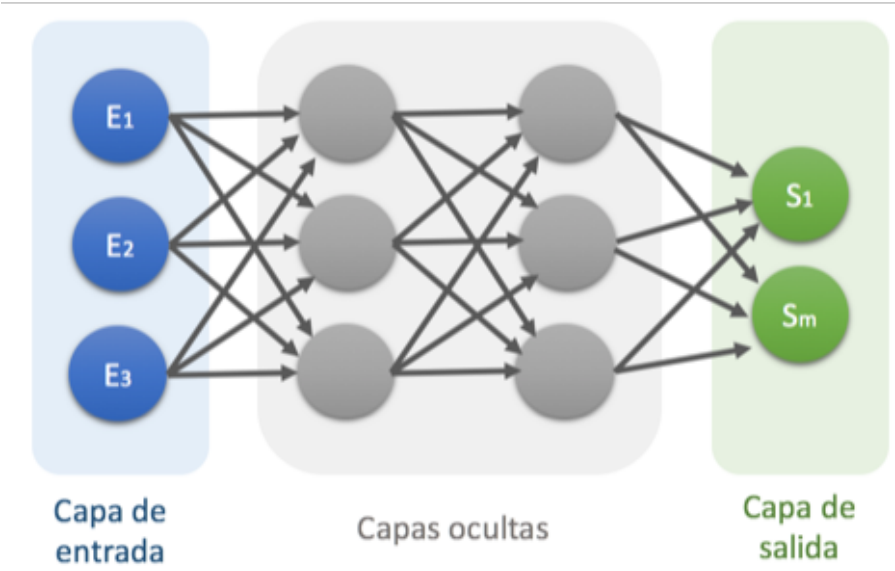

Figura 3.1: Perceptrón multicapa

Consideraciones de diseño

A la hora de diseñar la arquitectura de un perceptrón multicapa hay varios elementos que podemos alterar para tratar de lograr una mejor solución.

Número de neuronas

En algunos casos, especialmente en la capa de entrada y de salida, el número de neuronas viene definido por el problema a resolver. En las capas ocultas, este número puede variar ampliamente. Hay que considerar que un número elevado de neuronas puede provocar que estas memorizen los datos de entrada, proceso conocido como *overfitting*. En este caso, nuestra

red proporcionaría buenos resulados durante con los datos entrenamiento, pero sería incapaz de generalizar y los resultados serian pobres cuando se enfrentase a datos nuevos.

Por otro lado, un número escasos de neuronas puede provocar el efecto contrario, que nuestra red no disponga de la capacidad necesaria para generalizar correctamente. En este caso, conocido como *underfitting*, la red presenta pobres resultados, tanto en el entrenamiento como en los tests.

En nuestro caso, el número de neuronas en la capa de entrada viene determinado, a priori, por el número de características de nuestro dataset, esto es, 3197. Respecto a la capa de salida, dependerá de la función de activación que se vaya usar; en el problema que tratamos de resolver, clasificando los datos en dos categorías, hay o no hay exoplaneta, usaremos dos neuronas.

Número de capas y conexiones

De forma similar al número de neuronas, el número de capas ocultas puede variar ampliamente, contribuyendo especialmente al problema de *overfitting* comentado anteriormente. Además, de acuerdo al teorema de aproximación universal, cuando se usan funciones de activación no lineales, una sola capa oculta es suficiente para representar cualquier función continua en un rango dado, aunque esta capa puede ser demasiado grande y fallar en aprender y generalizar correctamente [1], por lo que es conveniente probar varias aproximaciones. Dado que nuestro problema es, además, no continuo, no debemos ceñirnos a usar una sola capa oculta.

Es también importante como se encuentran conectadas las capas. El modelo es más frecuente es el de capa totalmente conectada, donde cada neurona esta conectada a cada una de las neuronas de la siguiente capa. Hay, sin embargo, otras opciones donde, la más frecuente de ellas, consiste en que la salida de algunas o todas las neuronas de una capa se conectan con la entrada de neuronas de una capa no inmediatamente posterior, haciendo que su valor tenga más peso en el resultado final de la red.

Otro opción respecto a las conexiones entre las capas es usar la técnica conocida como *dropout*. Esta consiste en asignar, durante el proceso de entrenamiento, el peso de determinadas neuronas, seleccionadas de forma aleatoria, a cero, excluyendo de facto su aportación al resultado final de la red. El objetivo de la técnica es reducir el *overfitting*, ya que hace que la red sea menos dependiente del peso especifico de determinadas neuronas [4].

Funciones de activación

Vamos a examinar brevemente las funciones de activación más frecuentes que podríamos usar en nuestro modelo.

La función sigmoide fué de las primeras en usarse de forma masiva. La función está acotada entre [0, 1] y suele usarse en la última capa para representar probabilidades en clasificadores binarios. También es habitual usarla en las capas ocultas de los perceptrones multicapa. Sin embargo, adolece de algunos problemas, quizá el mayor de ellos sea que satura y mata el gradiente, provocando una lenta convergencia.

La tangente hiperbólica es muy similar a la sigmoide, estando igualmente acotada, aunque en un rango mayor, [-1, 1], lo que la hace adecuada para problemas en los que hay que decidir entre dos opciones. A diferencia de la sigmoide, esta centrada en el 0.

Es importante resaltar que la función sigmoide y la tangente hiperbólica se encuentran relacionadas, tal que $tanh(x) = 2 * sigmoid(2x) - 1$ por loq que existe poca diferencia a la hora de usar una u otra.

Tenemos también la función relu, función lineal rectificada por sus siglas en inglés. Esta función no está acotada y deja los valores positivos sin alterar pero transforma los negativos en cero. Tiene un buen desempeño en redes convolucionales a la hora de tratar con imagenes, pero también es la opción por defecto en los perceptrones multicapa. Esto se debe principalmente a dos factores: es poco probable que mate el gradiente y genera redes escasas, esto es, redes con neuronas muertas que no se activan, lo que hace la red más eficiente. Además, su facilidad de cálculo respecto a otras funciones, acorta el tiempo de entrenamiento de la re. Presenta, sin embargo, un importante problema, y es que puede matar a demasiadas neuronas. Para solucionarlo, existe una variante, denominada leaky relu, que, en lugar de anular los valores negativos, los multiplica por un coeficiente para devolver un valor negativo.

Finalmente, hablamos de la funcion softmax, que transforma un vector de entrada en un vector de probabilidades cuyo sumatorio es uno. Es usada frecuentemente en la capa de salida de la red cuando se trata de resolver un problema de clasificación.

De cara al diseño de nuestro modelo, la elección evidente para la capa de salida es usar una función softmax, aunque también se realizarán pruebas usando la función sigmoide para ver si presenta un mejor resultado.

Respecto a las capas ocultas, usaremos principalmente la función relu y, de forma similar a con la capa de salida, haremos pruebas con la sigmoide.

Algoritmo de optimización

El algoritmo de optimización es el encargado de actualizar los pesos de nuestra red para minimizar la perdida. Pytorch ya tiene implementados varios de estos algoritmos, por lo que solo queda decidir cual usar.

El más conocido y uno de los primeros en desarrollarse es el descenso del gradiente. Pytorch implementa el descenso del gradiente estocástico (SGD), que actualiza los pesos tras procesar cada ejemplo, en lugar de hacerlo tras procesar todo el dataset. Este algoritmo presenta algunas dificultades, como elegir la tasa de aprendizaje adecuada y que ese valor se aplique a todos los pesos por igual, así como oscilaciones que dificultan la convergencia en el punto mínimo. Para intentar corregir este último problema, el algoritmo puede configurarse para usar momento, que ayuda a suavizar las oscilaciones añadiendo una fracción de los pasos previos al paso actual.

Usaremos este optimizador como linea base de trabajo con diferentes valores para la tasa de aprendizaje tanto con como sin momento.

Otro de los algoritmos más usados es Adam[2], acrónimo en inglés de estimación adaptativa del momento, que será el otro algoritmo que usaremos.

A diferencia de SGD, Adam calcula tasas de aprendizaje distintas para los parámetros e incorpora momento. Adam es computacionalmente eficiente, tiene pocos requisitos de memoria y facilita la convergencia. Además, al actualizar los parámetros con diferentes tasas de aprendizaje, es menos sensible a una elección no óptima de la tasa de aprendizaje inicial.

3.4. Medidas de desempeño del modelo

El objetivo de nuestro modelo es obtener un clasificador binario que nos indique la probabilidad de que una entrada de datos pertenezca a una de nuestras dos clases: exoplaneta y no exoplaneta.

Con este objetivo en mente, la literatura existente nos indica que la solución óptima suele ser aplicar una función de activación softmax para la capa de salida de la red. Esta función, también llamada función exponencial normalizada, es una forma de regresión logística que normaliza un valor de entrada en un vector de salida que sigue una distribución de probabilidad

cuya suma total es 1. Así pues, el valor de salida de la neurona k-ésima vendrá dado por la función:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

En cualquier caso, estudiaremos otras opciones, como puede ser el caso de la función sigmoide, que nos devuelve un valor en el rango [0, 1], el cual puede ser interpretrado como probabilidad, en nuestro caso, de que sea una estrella con exoplaneta.

Desbalanceo del dataset

Analizando nuestro conjunto de datos, observamos que este se encuentra muy desbalanceado: los casos negativos (no es un exoplaneta) superan ampliamente en número a los casos positivos (si es un exoplaneta).

Esto supone un problema para el aprendizaje de la red ya que, ante cualquier entrada, esta puede *"aprender"* a responder siempre que no es un exoplaneta, acertando en la amplia mayoría de los casos.

Para solventar este problema, siguiendo a Viloria[5], vamos a definir nuestra función de evaluación, con la que juzgaremos el aprendizaje real de nuestra red y su capacidad de predecir el resultado correcto frente a nuevas entradas de datos.

$$f = Acierto * (\alpha * Sen + \beta * Esp)$$

donde `Acierto` representa el ratio de respuestas correctas, `Sen` es la sensibilidad (casos catalagocados como positivos que son realmente positivos), `Esp` es la especificidad (casos negativos correctamente calificados como no exoplanetas) y α y β son dos pesos usados para alterar la importancia de la sensibilidad y la especificidad. Comenzaremos con un valor neutro de 0.5 para cada uno, pero estudiaremos si su ajuste permite obtener un mejor modelo.

Definimos a continuación los ratios de `Acierto`, `Sen` y `Esp`, donde `VP` es el número de verdaderos positivos, `VN` los verdaderos negativos, `FP` los falsos positivos y `FN` los falsos negativos.

$$Acierto = \frac{VP+VN}{VP+VN+FP+FN}$$

$$Sen = \frac{VP}{VP+FN}$$

$$Esp = \frac{VN}{VN+FP}$$

3.5. Bibliotecas de machine learning

De cara a implementar nuestro modelo de red neuronal debemos decidir que lenguajes y bibliotecas vamos a usar. A día de hoy, la mayor parte de los frameworks y bibliotecas que facilitan el desarrollo de redes neuronales funcionan en entornos Python, que puede ser considerado el lenguaje de facto de la industria, aunque existen otras alternativas en lenguajes como R, Mathlab, y en frameworks como Neuroph para Java o Mathematica.

En nuestro caso, vamos a considerar una comparativa de tres bibliotecas de Python:

- Tensorflow es una biblioteca de código abierta desarrollada por Google para uso interno, tanto en investigación como en producción, que posteriormente fue lanzada al público.

- Keras es una API de alto nivel capaz de ejecutarse sobre otros lenguajes o bibliotecas, como Tensorflow, R o Theano, diseñada con el foco en la facilidad de uso.

- PyTorch es una biblioteca de código abierta desarrollada principalmente por Facebook que también presenta una interfaz para C++.

Vamos a examinar diferentes parámetros para ver que nos aporta cada una de ellas.

Velocidad

Los estudios muestran que no hay una diferencia significativa de velocidad entre Tensorflow y PyTorch. Este no es el caso con Keras, que presenta un rendimiento claramente inferior.

Nivel del API

Como se ha comentado, Keras es una API de alto nivel, capaz de correr sobre otras bibliotecas, como Tensorflow o Theano, proporcionando una interfaz común que facilita el desarrollo rápido de proyectos.

Tensorflow proporciona APIs tanto de alto como de bajo nivel, lo que le dota una gran flexibilidad.

Finalmente, PyTorch proporciona sólamente una API de nivel, enfocada en el trabajo directo con matrices.

Arquitectura

Keras presenta una arquitectura simple y fácil de comprender, mientras que tanto Tensorflow como PyTorch presentan arquitecturas más complejas y un código con mayor verbosidad.

La API de PyTorch se encuentra mejor diseñada mientras que la de Tensorflow es un tanto confusa y ha recibido numerosos cambios importantes en cada versión, lo que dificulta mantener un código estable y estar actualizado.

Debuggin

Depurar código en Tensorflow es relativamente complejo y no muy intuitivo. En Keras no es un proceso habitual, dado el alto nivel de sus componentes, lo que tampoco facilita la depuración en caso de algún problema, ya que la mayor parte del código se encuentra en la biblioteca. Sin embargo, PyTorch si ofrece buenas opciones para la depuración, muy similares a las encontradas en IDEs para lenguajes conocidos, como Eclipse o Visual Studio.

Dataset

Los problemas de velocidad de Keras no lo hacen adecuado para trabajar con grandes datasets. No es el caso de Tensorflow o PyTorch, que no tienen este problema de rendimiento.

Documentación y comunidad

Tanto en PyTorch como en Tensorflow se nota el efecto de tener detras a dos grandes empresas tecnológicas. En ambos casos, existen números recursos gratuitos con los que aprender así como una importante comunidad de usuarios que las respaldan y ofrecen su ayuda. Tensorflow tiene más tiempo de desarrollo y su base de usuarios es mayor pero desde el 2018 la popularidad de PyTorch está en constante aumento, especialmente en el ambito académico.

Keras contrasta respecto a las otros dos con una más reducida comunidad y menor documentación.

Puesta en producción

A la hora de poner en producción un modelo previamente entrenado, Keras no dispone de ninguna utilidad en si misma, haciendo uso de las

caracteristicas de Tensorflow. Este permite servir los modelos en un servidor web mediante una API REST o en dispositivos móviles.

PyTorch se apoya en otras bibliotecas para poder exponer sus modelos via web, permitiendo también otras opciones interesantes, como la interfaz con C++, lo que permite convertir los modelos en ejecutables fácilmente.

Resumen de la comparación

Tras analizar las características de las tres bibliotecas, vemos como se adaptan a nuestras necesidades.

Keras es una buena opción para probar y generar modelos de forma rápida, pero no nos permite profundizar en el aprendizaje y compresión de las redes neuronales, ya que la mayor parte del trabajo de nivel es gestionado de forma interna por la biblioteca. Unido a la dificultad de depurar el código y a su peor rendimiento, hace que optemos por no utilizarla.

La decisión entre Tensorflow y PyTorch es más dificil de realizar, ya que ambos aportan características similares: la posibilidad de trabajar con las redes a bajo nivel para poder estudiarlas en detalle, buen rendimiento y variados recursos para aprender, ya sea en forma de tutoriales o de comunidad de usuarios para resolver dudas. Sin embargo, hay dos detalles marcan la diferencia y nos hacen decantarnos por PyTorch: por un lado, la facilidad de depuración del código y, por otro, la posibilidad de generar ejecutables.

Así pues, la biblioteca que finalmente usaremos será **PyTorch**.

Técnicas y herramientas

Se exponen a continuación las herramientas usadas durante el desarrollo del proyecto.

4.1. Python

Python es un lenguaje de programación interpretado de alto nivel. Posee licencia de código abierto y, en los últimos años, se ha convertido en el estandar de facto para los proyectos de machine learning.

4.2. Jupyter Notebook

IDE interactivo de código abierto basado en web. Permite crear y compartir documentos que contienen tanto código como texto markdown.

4.3. Bibliotecas de Pyhton

Torch

Biblioteca de código abierto para aprendizaje automático. Se puede encontrar más información sobre sus características y su comparación con otras bibliotecas similares en la Sección 3.5 Bibliotecas de machine learning

NumPy

Biblioteca que agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar

con esos vectores o matrices.

Pandas

Extensión de NumPy para manipulación y análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series de tiempo.

SciPy

Basado en NumPy, expande esta biblioteca con herramientas y algoritmos para matemáticas, ciencias e ingeniería.

imbalanced-learn

Ofrece técnicas y algoritmos de *under-sampling* y *over-sampling* comúnmente utilizados en conjuntos de datos que muestran un fuerte desequilibrio entre clases.

Matplotlib

Matplotlib es una biblioteca para crear visualizaciones estáticas, animadas e interactivas en Python.

Os

Proporciona una interfaz para utilizar los comandos del sistema operativo.

Time

Aporta funcionalidades para trabajar con objetos de fechas y horas.

Repackage

Biblioteca para invocar paquetes no registrados y acceder a ellos con rutas relativas.

HTML

Funcionalidades para trabajar con documentos HTML.

Flask

Framework minimalista para crear aplicaciones web de forma rápida y sencilla.

Werkzeug

Biblioteca que proporciona diferentes utilidades relativas al desarrollo web.

Wtforms

Añade representación y validación de formularios flexible para el desarrollo web con Python.

Flask-wtf

Integra la biblioteca Wtforms con Flask.

Base64

Proporciona funcionalidades para codificar y decodificar datos binarios.

4.4. Git

Sistema de control de versiones distribuido de código abierto.

4.5. Gitlab

Servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas, ofreciendo una plataforma en la cual se puede realizar todas las etapas del ciclo de desarrollo del software.

4.6. LaTeX

Sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Es usado de forma

especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas.

MiKTeK

Distribución de LaTex para sistemas Windows.

TexMaker

Editor de código abierto de LaTex. Integra variedad de herramientas para desarrollar documentos.

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.

[3] Marvin Minsky and Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[5] A. Viloria-Lanero and V. Cardeñoso-Payo. Integration of skin segmentation methods using anns. 2006.