Redes Neuronales Artificiales: Prácticas y Trabajo

Marcos Esteve Casademunt

Febrero 2020

Contents

1 Ejercicio 1: MNIST

El código desarrollado está en `https://colab.research.google.com/drive/1XohmX2YGhq9ejTX2kW2pgMkNI2_8zzNv`

En este primer ejercicio se propone realizar una solución al problema de reconocimiento de dígitos manuscritos MNIST; este problema se trata de una clasificación de los números del 0 al 9.

El objetivo final es conseguir un modelo basado en redes neuronales que obtenga una tasa de error inferior al 0.8 %. Para ello se han realizado los siguientes cambios sobre el script original:

- Data augmentation: Se ha realizado una aumento de los datos de entrenamiento con el fin de que la red consiga mejores generalizaciones ante datos no observados en la fase de entrenamiento. Para ello se han utilizado los siguientes parámetros:
  - desplazamiento horizontal
  - desplazamiento vertical
  - zoom de la imagen
  - rotación
- Modificación de la red: Se ha reducido la arquitectura de la red neuronal a 2 capas ocultas permitiendo que, al reducir el número de parámetros a estimar sea más complicado que se caiga en sobre entrenamiento y por tanto se permitan mejores generalizaciones reduciendo también el tiempo de entrenamiento. La arquitectura del modelo la podemos apreciar en la figura 1

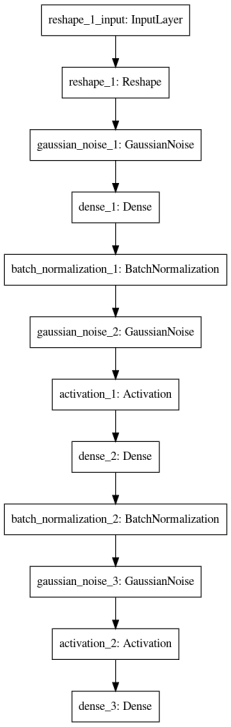

Figure 1: Arquitectura de MNIST

- Modificación de los parámetros tamaño de batch y ruido gausiano

Con todas las modificaciones expuestas anteriormente se consigue un accuracy del **99.28%** con lo que se consigue alcanzar el objetivo propuesto. A continuación, en las figuras 2 y se muestran la evolución de la precisión a lo largo de 70 epochs.

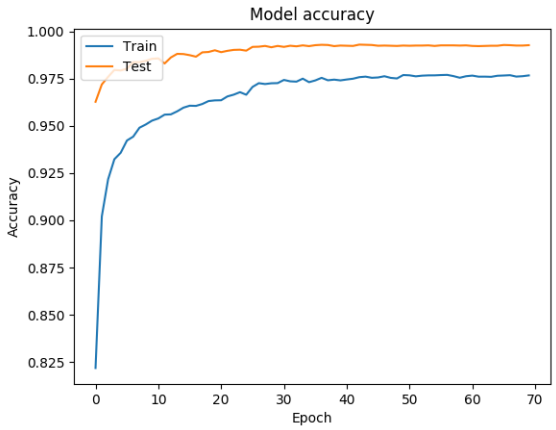

Figure 2: Evolución de la precisión en MNIST

2 Ejercicio 2: CIFAR

El código desarrollado está en `https://colab.research.google.com/drive/1L1klXF1RNHd7LOdb1_YdNUQaQx7digT5`

En este ejercicio se propone realizar una solución al problema CIFAR. Este problema trata de realizar una clasificación de distintos objetos en 10 clases distintas.

El objetivo final consiste en obtener un modelo basado en redes neuronales cuya tasa de error sea inferior al 1 %. Para conseguir este objetivo se han realizado los siguientes cambios sobre el script proporcionado en Github:

- Data augmentation: Se ha realizado una aumento de los datos de entrenamiento con el fin de que la red consiga mejores generalizaciones ante datos no observados en la fase de entrenamiento. Para ello se han utilizado los siguientes parámetros:

  - desplazamiento horizontal
  - desplazamiento vertical
  - zoom de la imagen
  - rotación

  Además se ha probado a utilizar MixUp con alfa 0.2, aunque los mejores resultados no se han utilizado con este. El uso de MixUp permite mezclar imagenes de distintas clases realizando de esta forma un mejor data augmentation con el fin de obtener mejores generalizaciones al mezclar datos de distintas clases, permitiendo de esta manera obtener una red mucho más robusta. Destacar que la implementación de MixUp se ha extraido del Github `https://github.com/yu4u/mixup-generator`

- Modificación del *learning rate*: Se ha implementado un método de de *learning rate annealing* basado en *step decay* y comentado en[1]. Se ha utilizado un *learning rate* inicial de 0.2, un drop de 0.9 y el número de *epochs* necesario para realizar cada drop a 25. A continuación en la figura 3 podemos observar el comportamiento del *learning rate* con el transcurso de las *epochs*

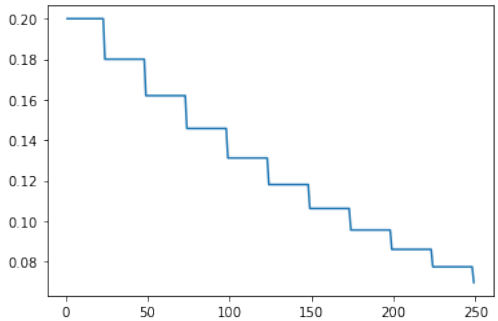

Figure 3: Step Decay

[1] `http://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-`

- Cada vez que se llama a la función CBGN se ha añadido una capa convolución extra y se ha modificado el ruido gausiano

Con todos estos experimentos el mejor resultado obtenido ha sido de un **0.9035** consiguiendo de esta forma el objetivo propuesto.

Además para obtener esta precisión no se ha utilizado finalmente Mix-UP, ya que en los experimentos mostraba una convergencia muy lenta debido a la gran cantidad de variabilidad que introduce este método. Tampoco se ha utilizado step decay como learning rate annealing. Por último en la gráfica 4 podemos apreciar la evolución de la precisión con el transcurso de las epochs

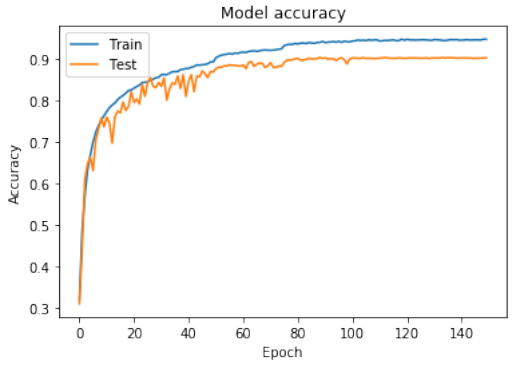

Figure 4: Evolución de la precisión en CIFAR

3 Proyecto Personal

El código desarrollado está en `https://colab.research.google.com/drive/1wlthRaN8SvH7BxhD6F4QCVyBvdSErwOY`

Para el desarrollo del proyecto personal se ha escogido una tarea de clasificación binaria con el objetivo de detectar si un *tweet* tiene un sentimiento positivo o negativo. Para ello se ha utilizado el dataset **sentiment 140** y extraído del link `http://help.sentiment140.com/for-students`.

El objetivo final consiste en obtener un modelo basado en redes neuronales que sea capaz de clasificar los tweets. Para ello se exploraran distintas técnicas desde realizar una extracción de características por *bag of words* y emplear redes neuronales clásicas a utilizar técnicas basadas en *word embeddings* y redes LSTM.

3.1 Dataset

El dataset utilizado consiste en un total de 1.600.000 tweets divididos en dos clases, positivos y negativos. Además cabe destacar que el corpus está balanceado en cuanto al número de muestras de cada clase.

Para el entrenamiento y el testeo de los modelos se ha optado por realizar una partición de los datos en *Hold-out* donde el 80% de los datos se utilizan para entrenamiento y el 20% para test.

3.2 Bag Of Words + Neural Networks

En primer lugar se ha realizado una representación de los tweets utilizando una Bolsa de palabras ponderada por TF-IDF. Este tipo de representación permite representar cada *tweet* como un vector donde para cada posición del vector se indica la frecuencia de aparición de un determinado *token* en ese *tweet*. Además, se ha utilizado un pesado TF-IDF que permite atenuar aquellos *tokens* que tengan presencia en muchos *tweets*. Se han utilizado las siguientes características:

- Se ha realizado un preprocesado de los tweets eliminando aquella información que no aporta información acerca del sentimiento como pueden ser los hashtags, las menciones y los links. Además se ha pasado todo el texto a minusculas.
- Se ha limitado el número máximo de características a 60000
- Se han eliminado las StopWords para el inglés
- Se han considerado distintos tamaños de n-gramas, concretamente se han considerado n $\in$ 1,2,3

Una vez definida la bosa de palabras se ha definido un perceptrón de 2 capas ocultas y se ha entrenado durante 10 epochs. Una vez finalizado el entrenamiento se ha obtenido una precisión de **0.7863**. En la figura 5 podemos apreciar la evolución del accuracy durante el transcurso de las epochs

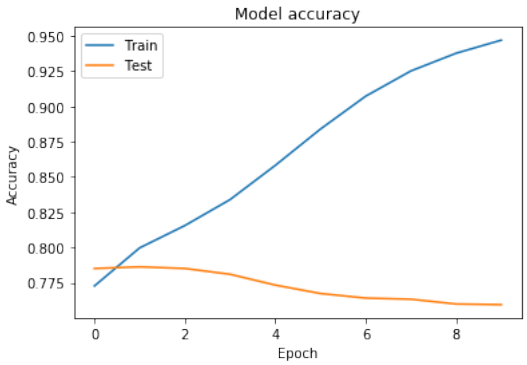

Figure 5: Evolución de la precisión utilizando un MLP

3.3 Embeddings + LSTM

En segundo lugar se ha realizado una representación utilizando word embeddings. Una de las características principales de los embeddings es la capacidad de capturar las relaciones semánticas y sintácticas entre distintos términos. Además el uso de este tipo de representaciones permite tratar con vectores densos de una dimensionalidad significativamente inferior a si utilizáramos representaciones del tipo Bag Of Words.

Se ha definido una arquitectura basada en word-embeddings + redes LSTM para realizar la clasificación de los tweets dependiendo del sentimiento positivo o negativo. Se ha entrenado este modelo consiguiendo una precisión máxima de

0.8258 que como podemos observar es significativamente superior a la obtenida con la representación basada en bolsa de palabras. Además destacar que el tiempo de entrenamiento se reduce a la mitad en comparación con el primer modelo. En la figura 6 podemos apreciar la evolución del accuracy durante el transcurso de las epochs

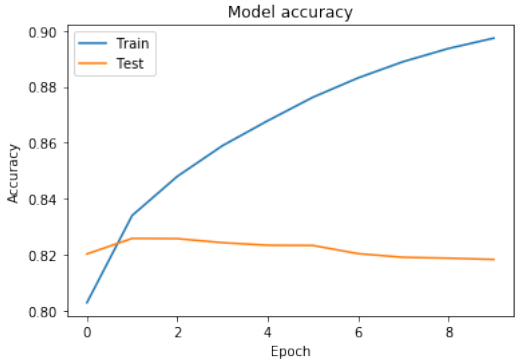

Figure 6: Evolución de la precisión utilizando redes LSTM y word embeddings

3.4 Posibles mejoras

Una vez explorado distintas técnicas para la clasificación del sentimiento en tweets podemos plantear posibles mejoras.

En primer lugar en lugar de entrenar los embeddings junto a la red se podría realizar una gran recopilación de tweets y entrenar unos embeddings de mayor calidad; de esta forma dichos embeddings tendrían una mayor información del contexto semántico y sintáctico de twitter.

En segundo lugar se podría probar a utilizar otras técnicas de representación como BERT para codificar los tweets. Este tipo de técnicas están demostrando marcar el estado del arte en numerosas tareas en procesamiento del lenguaje natural.

Por último, otra técnica que podría funcionar en este tipo de tareas es aprovechar tecnologías de Transfer learning, más concretamente ULMFIT. Esta técnica permite especializar un modelo de lenguaje para realizar tareas de clasificación.