Diseño de una herramienta para la transcripción de caracteres matemáticos mediante una red neuronal entrenada en paralelo

Guzman Castro, Yeily P.[1]

Faculta de Ciencias y Artes, Universidad de Puerto Rico
Calle Post, Mayagüez Puerto Rico

22 de mayo de 2022

Resumen:
Con el objetivo de crear una red neuronal capas de reconocer caracteres matemáticos y transcribirlos en formato `.tex`, se implementa una red neuronal entrenada en paralelo capaz de identificar hasta el momento los números del 0 al 10 escritos desde un canvas. Con esta herramienta se busca facilitar las transcripciones de texto que utilizan ecuaciones y símbolos.

Palabras clave: Red neuronal convolucional, programación en paralelo, reconocimiento de imágenes.

Introducción

En la programación regular se implementan algoritmos que establecen la lógica necesaria y los pasos a seguir para convertir los datos iniciales en resultados; en el aprendizaje automático no se conoce explícitamente cual es el proceso que se usa para generar los resultados. Lo que se quiere es crear un modelo de redes neuronales que tome las entradas y ejemplos de las soluciones que debería arrojar y que pueda aprender por si solo el algoritmo necesario para hacer la conversión.

Una de las cosas más didácticas de la inteligencia artificial es el reconocimiento de imágenes por el interesante hecho de que un código pueda ver, catalogar o entender una foto o video. Antes de los años noventa las redes neuronales, aunque bien eran capaces de reconocer imágenes, al basar su interpretación en la posición y forma de la misma tenía un alto margen de error pues cuando la imagen que se le presentaba no era parecida a las que estaban en su conjunto de entrenamiento, o incluso si sí la clasificaba bien al cambiarla de tamaño o posición, ya no era capaz de reconocerla correctamente, esto cambió con la llegada de las redes neuronales convolucionales.

En el año 1989, Yan LeCun, un científico de la computación y tres de sus compañeros presentaron el artículo *Gradient-Based Learning Applied to Document Recognition* [4] donde muestran la primera red neuronal diseñada específicamente para el reconocimiento de imágenes o más específicamente, de texto. La arquitectura constaba de varias capas que implementaban la extracción de características y luego las clasificaban. Para lograr esto dividen la imagen en campos receptivos que alimentan una capa convolucional que es la que se encarga de extraer las características de la imagen de entrada (Por ejemplo, detectar lineas, círculos, vértices, etc). El siguiente paso es *pooling* que reduce la dimensionalidad de las características extraídas manteniendo la información más importante. Luego se hace una nueva convolución y otro *pooling* que alimenta una red *feedforward* multicapa (un tipo de red donde la información solo se mueve hacia adelante, sin bucles o ciclos). La salida final de la red es un grupo de nodos que clasifican el resultado.

Un problema de optimización común en este ámbito ocurre al momento de entrenar estas redes, ya que requieren procesar una gran cantidad de datos si se desea una buena predicción por parte del modelo. Se ha demostrado que paralelizar el proceso de entrenamiento de una red neuronal que reconoce caracteres disminuye significativamente el tiempo de esta fase [5].

Utilizaremos la plataforma de TensorFlow (biblioteca de código abierto para desarrollar y entrenar modelos de AA) para desarrollar nuestra red, a su vez para el proceso de paralelización se utiliza la API

de `tf.distribute.Strategy` la cual proporciona una abstracción para distribuir su capacitación en varias unidades de procesamiento. Permite realizar un entrenamiento distribuido utilizando modelos existentes y código de entrenamiento con cambios mínimos.

Se trabajo más específicamente con el modelo de entrenamiento `tf.distribute.MirroredStrategy` para realizar la replicación en el gráfico con entrenamiento síncrono en muchas GPU en una máquina . La estrategia esencialmente copia todas las variables del modelo a cada procesador. Luego, usa all-reduce para combinar los gradientes de todos los procesadores y aplica el valor combinado a todas las copias del modelo.

Objetivo General

Diseñar una red neuronal capaz de reconocer una amplia variedad de caracteres y símbolos matemáticos que permita facilitar la transcripción de documentos.

Objetivos específicos

- Implementar la fase de entrenamiento en paralelo con un alto rendimiento.
- Crear una herramienta interactiva que nos permita facilitar el uso de la red.

Revisión bibliográfica

Este trabajo se basa en las indicaciones que se pueden encontrar en la Guía de TensorFlow llamada **Distributed training with TensorFlow** [2], en el cual nos habla de la API `tf.distribute.Strategy` que se utiliza para distribuir la capacitación en múltiples GPU, múltiples máquinas o TPU.

Con esta API se cubren varios tipos de paralelización que pueden resultar útiles según el modelo que se esté implementando y las necesidades del usuraio. La mayoría de las estrategias de paralelización pueden dividirse en dos grandes tipos:

- **Entrenamiento sincrónico versus asincrónico:** Estas son dos formas comunes de distribuir el entrenamiento con paralelismo de datos. En el entrenamiento sincronizado, todos los *workers* entrenan sobre diferentes porciones de datos de entrada sincronizados y agregando gradientes en cada paso. En el entrenamiento asíncrono, todos los *workers* se entrenan de forma independiente sobre los datos de entrada y actualizan las variables de forma asíncrona. Por lo general, el entrenamiento de sincronización se admite a través de `all-reduce` y asíncrono a través de la arquitectura del servidor de parámetros.
- **Plataforma de hardware:** Es posible que desee escalar su capacitación a varias GPU en una máquina, o varias máquinas en una red (con 0 o más GPU cada una), o en Cloud TPU.

En la sección **Distributed training with Keras** [1] se muestra como utilizar la estrategia `tf.distribute.MirroredStrategy` la cual admite entrenamiento distribuido síncrono en múltiples GPU en una máquina. Crea una réplica por dispositivo GPU. Cada variable del modelo se refleja en todas las réplicas. Juntas, estas variables forman una sola variable conceptual llamada `MirroredVariable`. Estas variables se mantienen sincronizadas entre sí mediante la aplicación de actualizaciones idénticas.

Se utilizan algoritmos de reducción total eficientes para comunicar las actualizaciones de variables a través de los dispositivos. All-reduce los tensores agregados en todos los dispositivos sumándolos y poniéndolos a disposición en cada dispositivo. Es un algoritmo fusionado que es muy eficiente y puede reducir significativamente la sobrecarga de sincronización. Hay muchos algoritmos e implementaciones de reducción total disponibles, según el tipo de comunicación disponible entre dispositivos. De forma predeterminada, utiliza la biblioteca de comunicación colectiva de NVIDIA (NCCL) como implementación de reducción total. Puede elegir entre algunas otras opciones o escribir las suyas propias.

Se utilizó entonces las indicaciones presentes en las guías y se convirtió el modelo entrenado en un archivo `h5` con el fin de poder implementarlo en el canvas en un navegador.

Metodología de investigación

Su utilizan como herramientas principales el producto Google Colab de Google Research, el cual permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador; TensorFlow, una plataforma de código abierto de extremo a extremo para el aprendizaje automático y Keras, una biblioteca de código abierto para crear redes neuronales. Podemos separar este proyecto en dos secciones:

Desarrollo de la red neuronal

El código escrito en lenguaje Python en Google Colab se añade como anexo en un archivo `Python File`. En él se muestra la importación de las plataformas y los sets de entrenamiento y prueba. Procedemos luego a definir la entrategia de entrenamiento con el comando

```
strategy=tf.distribute.MirroredStrategy()
```

El cual como se menciona en la introducción admite entrenamiento distribuido síncrono en múltiples GPU en una máquina. Crea una réplica por dispositivo GPU. Cada variable del modelo se refleja en todas las réplicas. Se utilizó este tipo de estrategia ya que al entrenar sobre diferentes porciones de datos de entrada sincronizados y agregando gradientes en cada paso se logra mantener en sincronía todos los GPU que están involucrados en el proceso.

Esto manejará la distribución y proporcionará un administrador de contexto (MirroredStrategy.scope) para construir su modelo en el interior. Se definen luego el número de dispositivos a utilizar que será en este caso de 1. Realizamos la configuración de entrada, Al entrenar un modelo con varias GPU, puede usar la potencia informática adicional de manera eficaz aumentando el tamaño del lote. En general, utilice el tamaño de lote más grande que se ajuste a la memoria de la GPU y ajuste la tasa de aprendizaje en consecuencia. Normalizamos los pixeles de la imagen de un [0, 255]rango a otro [0, 1].

Podemos ahora crear el modelo en el contexto de Strategy.scope, que crea y compila el modelo mediante la API de Keras. Se utilizó una capa convolucional con actuvacion relu, una capa MaxPooling2D, una capa Flatten, una capa Dense de 64 neurona con activación relu y una capa de salida también de tipo Dense con los 10 posibles resultados (números del 0 al 9). Finalmente ejecutamos la red con 12 épocas y evaluamos su desempeño con `Model.evaluate`. Guardamos el modelo como `numeros_conv_ad_do.h5` y empezamos con la siguiente sección.

Creación de la página de prueba

Para que nuestra red sea más interactiva se diseña una página web donde la red puede reconocer nuestra propia escritura y de esta manera hacer una predicción según su entrenamiento. El diseño de dicha página igualmente se anexa a este documento.

En ella podremos encontrar un cuadro central donde con el cursos dibujaremos el número del 0 al 9, en la parte inferior se muestran los botones de predecir, lo que produce que se envíe una verción de 32×32 pixeles a nuestra red para que posteriormente ella nos envíe su predicción, la cual se mostrará en la parte inferior. El botón de limpiar nos limpiará el canvas para ingresar nuevamente otro número.

Figura 1: Vista general de la página interactiva

Para poder utilizar la red neuronal, debemos ingresar al `cmd` del computador, navegar hasta la carpeta donde guardamos el modelo, ingresar el comando `python -m http.server 8000` para finalmente ingresar al link `http://localhost:8000`

Análisis de los datos

En este caso y como un primer paso se utiliza el set de datos de MNIST, el cuenta con un conjunto de imágenes escritas a mano de las cuales 60 000 ejemplos de entrenamiento y un conjunto de prueba de 10 000. Es un subconjunto de un conjunto más grande disponible en NIST. Los dígitos se normalizaron en tamaño y se centraron en una imagen de tamaño fijo.

Sin embargo, se planea ingresar al menos tres datasets más, uno para las 27 letras del abecedario y otro con los principales operadores tales como sumatorias, productorias, los símbolos de suma, resta, multiplicación entre muchos otros. Además de todo esto, se quiere añadir transformaciones de rotación y tamaño a las imágenes de muestra lo que engrosará grandemente los datos haciendo mucho más útil para paralelización del proceso de entrenamiento.

Resultados

Mediante el comando

```
%tensorboard --logdir=logs
```

Podemos visualizar la figura 2 sobre el rendimiento de nuestra red:

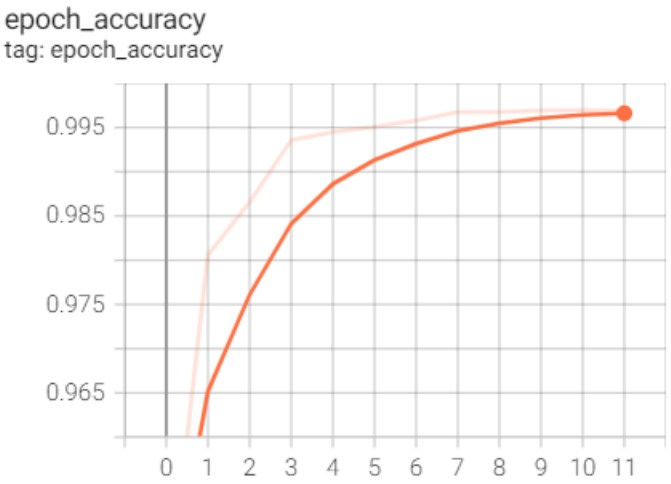

Figura 2: Gráfica del rendimiento de la red (naranja claro) versus cada época. Gráfica suavizada del rendimiento de la red versus cada época (Naranja oscuro)

Vemos además que según el siguiente resultado al aplicar el comando `model.evaluate` es

```
Eval loss: 0.036085404455661774
Eval accuracy: 0.987500011920929
```

Por lo que nuestra red presenta una precisión del 98 %. Por precisión de la red entiéndase la cantidad de éxitos que tiene cuando se le presentan datos del conjunto de prueba. De acuerdo a esto vemos que para la época 12 no hay una tendencia a aumentar el rendimiento de la red, por lo que más épocas no serían necesarias. Algunas pruebas en la página de aplicación nos permite verificar su funcionamiento

Figura 3: Prueba de la red, número 8

Figura 4: Prueba de la red, número 3

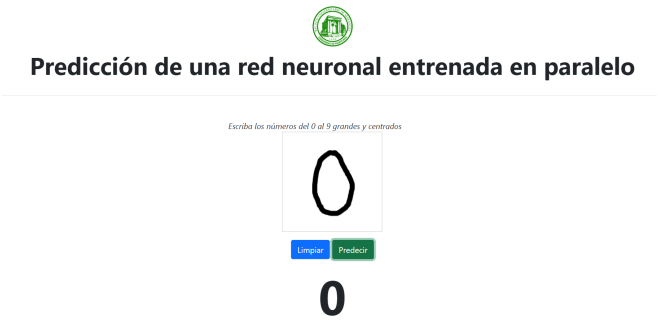

Figura 5: Prueba de la red, número 0

Conclusión

Revisado todo lo anterior, presentamos entonces esta herramienta que hasta el momento es capaz de reconocer los caracteres del 0 al 9 escritos a mano y la cual presenta un porcentaje del 98 % en tan solo 12 épocas. Considerando la cantidad de datos que esta red debe procesar en la fase de entrenamiento y que se planea que haya aún más datos, la paralelización resulta de utilidad.

Mejoras futuras

Se espera ampliar la cantidad y variedad de datos y tokenizar los símbolos operadores. El principal desafío se presenta al momento del segundo objetivo, ya que para esto se hacen necesarios sets de datos con muchos caracteres en una sola imágenes que además son de tamaños variados y sin mencionar que deben estar debidamente etiquetados.

Referencias

[1] *Distributed training with keras*. (s/f). TensorFlow. Recuperado el 22 de mayo de 2022, de https://www.tensorflow.org/tutorials/distribute/keras

[2] *Distributed training with TensorFlow*. (s/f). Recuperado el 22 de mayo de 2022, de https://www.tensorflow.org/guide/distributed_training

[3] García Fuentes, R. (2020). *Paralelización automática del proceso de aprendizaje de una red neuronal Deep Learning* (Bachelor's thesis, Universitat Politécnica de Catalunya).

[4] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11), 2278-2324.

[5] Sierra, E. E., Gómez, G. N., Suárez, A. R., & Pérez, E. C. L. (2006). *Sistema para el entrenamiento paralelo de redes neuronales de propagación hacia atrás*. Ingeniería Industrial, 27(1), 5-pág.