Trabajo Práctico 2
Deep Learning

Felipe Andrés Tenaglia Giunta
Legajo: T-2658/1

14 de octubre de 2016

Ejercicio *1*

Apartado *a*

Para resolver el ejercicio propuesto en el tutorial de Theano, se declaró un nuevo tensor `b = theano.tensor.vector()` y se reemplazó en `out` por la expresión a calcular, en este caso `a ** 2 + b ** 2 + 2 * a * b`, resultando así el código presente en el archivo `theano/Ej1a.py`

Apartado *b*

Los ejercicios fueron resueltos de manera independiente, en tres archivos distintos (`Ej1b_i.py`, `Ej1b_ii.py`, `Ej1b_iii.py`, dentro del directorio `theano`).

Para el primer ítem se indica el tamaño del minibatch en la línea 10, y luego el script calculará cuántos generar, dividiendo luego el conjunto de entrenamiento en los distintos minibatches.

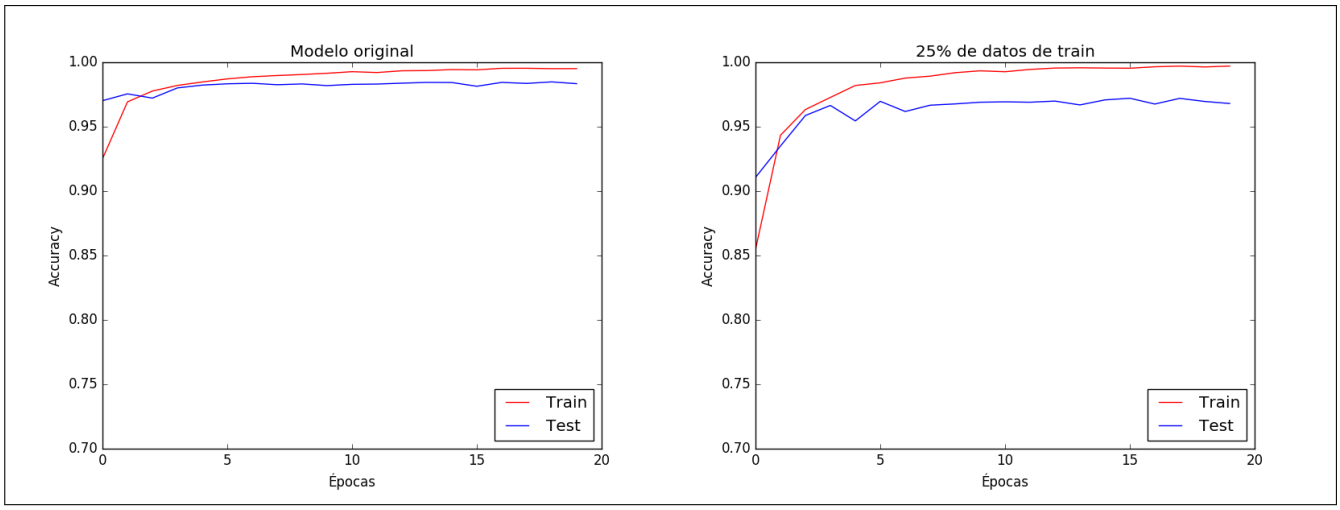
Para el segundo ítem se utilizaron las funciones del TP1 para el preprocesamiento de los datos, presentes en el archivo `TP1.py`. A pesar de la simplicidad de la red (si puede llamarse red, siendo una simple regresión logística) y el tamaño de las muestras de entrada (28x28), el comportamiento es bastante satisfactorio, obteniendo una precisión de 95.75 %, habiendo antes separado 400 imágenes para el conjunto de test.

Para el tercer ítem se declararon los parámetros necesarios para la nueva capa con 100 ReLU y se modificó la función para implementar este cambio. Los nuevos pesos de las ReLU se inicializaron siguiendo el criterio indicado por Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun en *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, es decir, dividiendo los valores generados aleatoriamente por $\sqrt{2/n}$ siendo n la cantidad de parámetros.

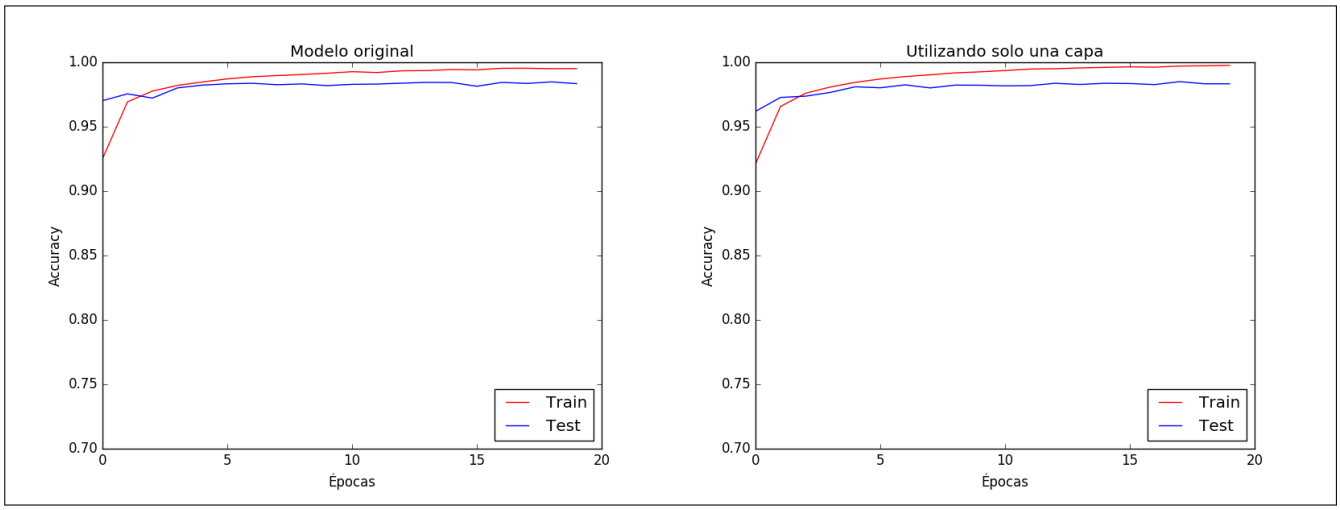
1. Ejercicio *2*

Apartado *a*

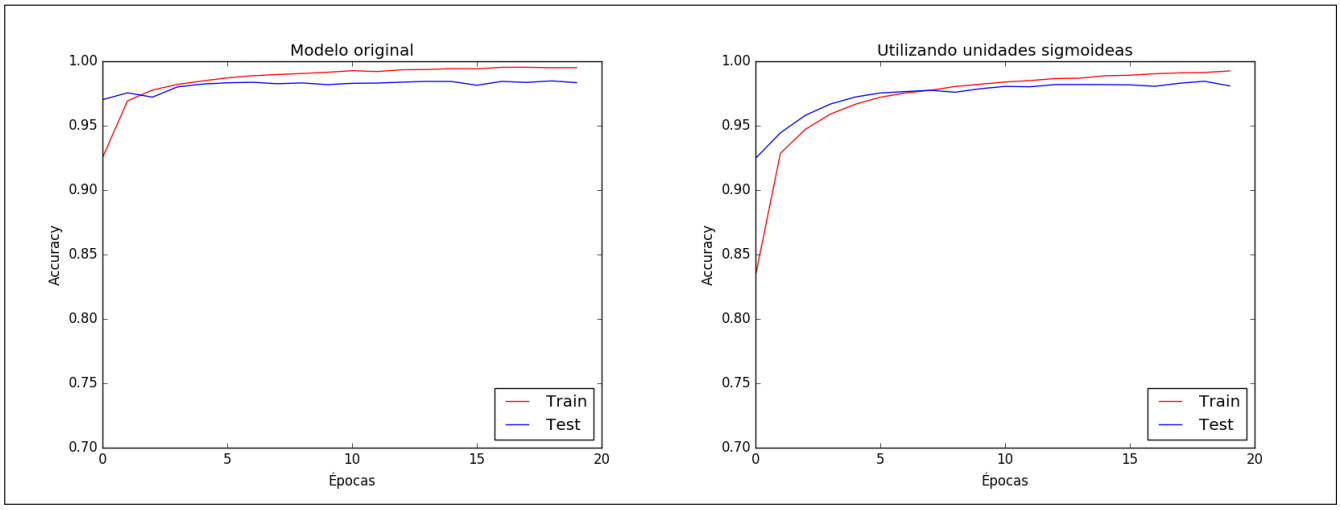
Para comparar resultados, se muestra el desempeño del modelo original junto con la modificación propuesta.



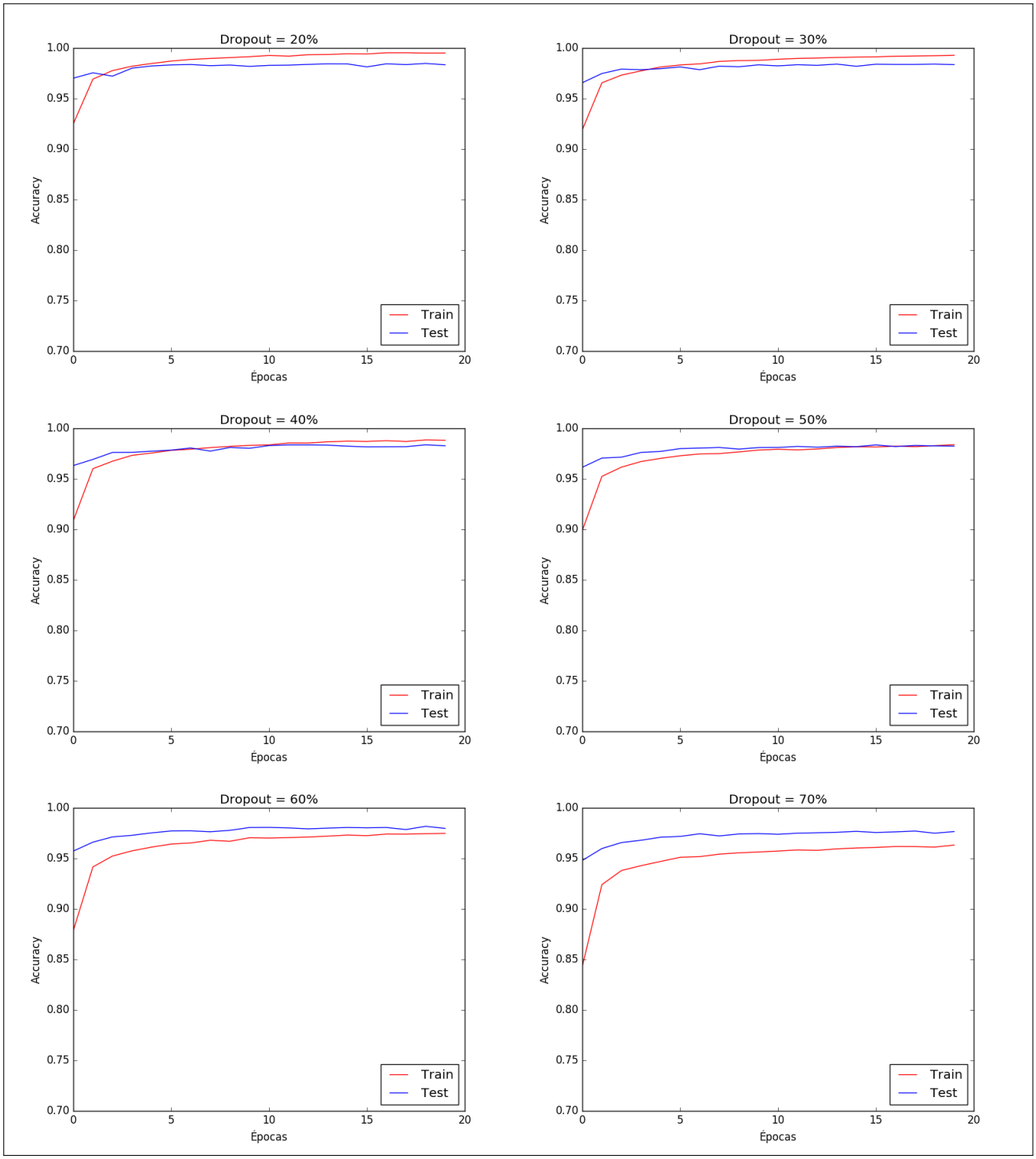
Aquí podemos ver como empeora levemente la velocidad de convergencia, y la precisión en el conjunto de test. Si bien el cambio es muy bajo (de 98.34 % a 96.8 %), es relevante. El motivo es obvio: al reducir el conjunto de entrenamiento, la red no puede aprender lo suficiente para identificar correctamente los dígitos, ajustando así un modelo no tan preciso.

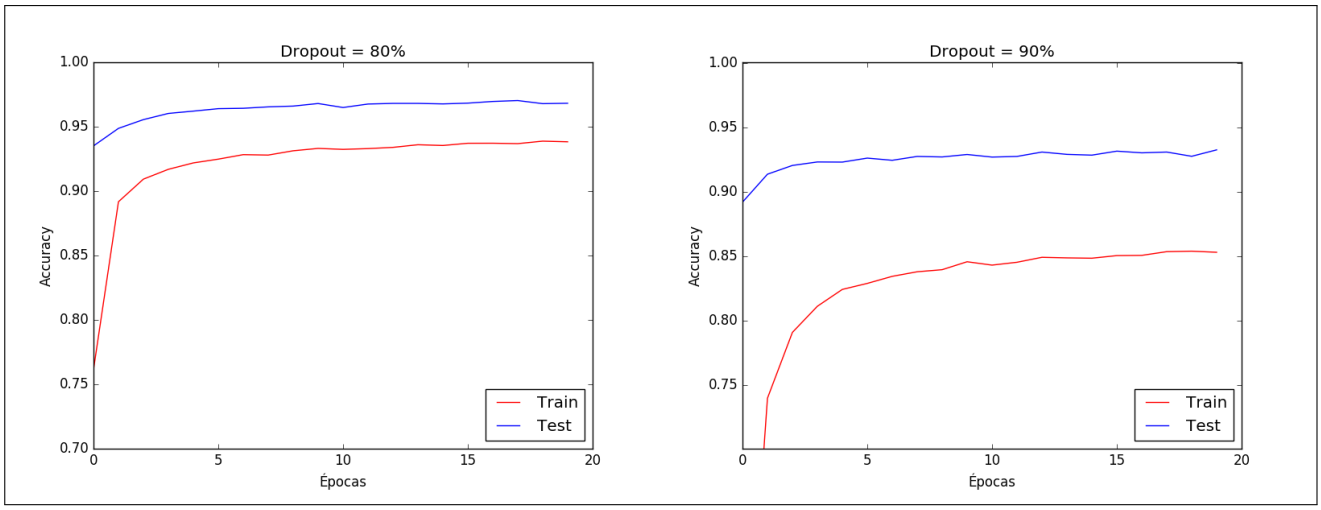


Se esperaba un cambio mucho más grande en el la precisión en test y train, pero esto no fue así. La red se comporta, aún, de una manera muy satisfactoria. Esto puede atribuirse a que la cantidad de muestras es bastante alta (60000 para train).

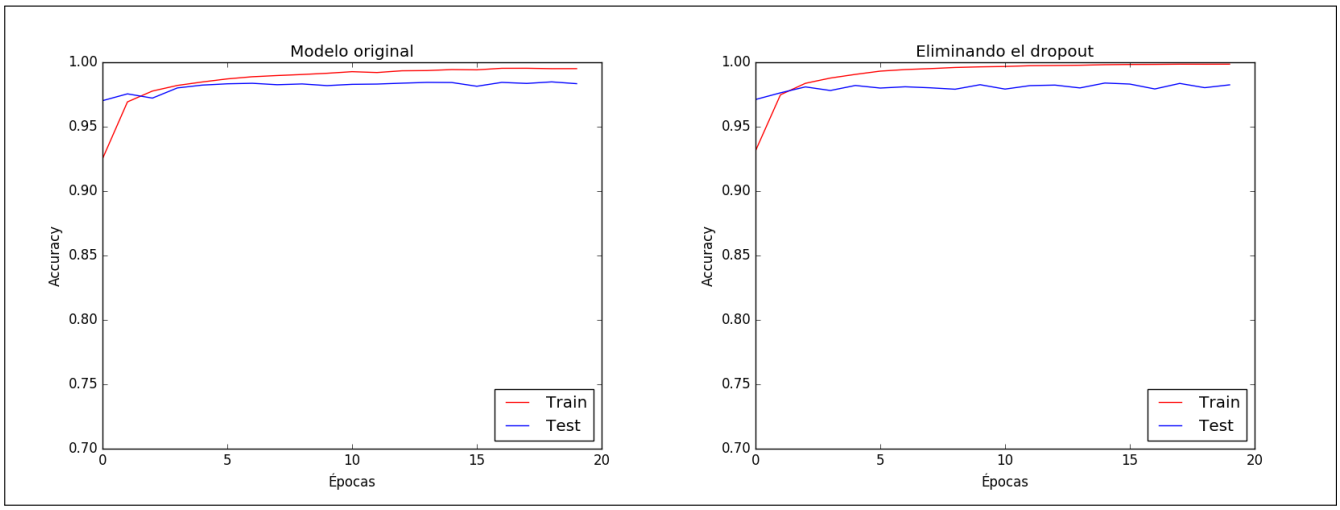


Al utilizar unidades sigmoideas en todas las capas (salvo en la softmax), no se puede notar una pérdida significativa del poder de clasificación de la red, pero si se observa un gran deterioro en la velocidad de convergencia, llegando a la época 10 aún sin converger completamente.





Cuando se incrementa el dropout, podemos ver que la precisión en test no cambia significativamente entre 20 % y 70 %, encontrandosé en torno al 98 %. Cuando el dropout es del 90 %, aumenta considerablemente el error en test, con una diferencia de más del 5 % con el modelo original. Con respecto a la precisión sobre el conjunto de entrenamiento, esta se ve afectada cada vez que aumenta el dropout. Esto es coherente con la función que realiza: al "apagar" neuronas, la red no es capaz de aprender lo suficiente para ajustar de manera adecuada el conjunto de entrenamiento. La precisión en el conjunto de test no se ve tan afectada ya que en esa instancia se utiliza la red en su totalidad.



En este último caso, al eliminar completamente el dropout en toda la red, se puede observar como la red ajusta casi perfectamente al modelo, aunque no sería del todo justo considerar esto como sobreajuste ya que la precisión sobre el conjunto de test se mantiene muy buena (Por encima del 98 %)

Apartado *c*

El formato H5 (o HDF5) presenta varias ventajas cuando deben manejarse grandes volúmenes de datos (como datasets o configuración de redes y sus pesos). Tiene una estructura jerárquica y acceso aleatorio, por lo que es útil para acceder rápidamente a distintas partes del archivo. En el caso particular de Python, la biblioteca `h5py` provee una forma de acceso "lazy", transparente al usuario. Además, al estar contenido en un único fichero, es más probable que la información en disco se almacene de manera contigua, aumentando la velocidad de lectura y mejorando el uso de la caché. Otro punto a resaltar es, al ser un formato que se esta imponiendo, pronto podría llegar a ser un estándar, aumentando la posibilidad de portar datos y configuraciones entre distintas plataformas.

En `keras/Ej2b.py`, además de mostrar el uso de `ImageDataGenerator` se ve como se puede utilizar el módulo `h5py` para exportar datos en este formato.

Keras utiliza, en particular, este formato para guardar en disco la configuración y los pesos entrenados de un modelo, utilizando el método `save()`, en la clase de los modelos. En el archivo `keras/Ej2c.py` se muestra su uso.