1. Theano

El ejercicio 1 se desarrolla en el script *theano_tp2.py*.

a. Revise este tutorial de theano y resuelva el ejercicio propuesto.

El ejercicio propuesto solo requiere crear otra variable T.vector y la operación simbólica es idéntica al enunciado.

b. Modifique este ejemplo de regresión logística para:
   i. Procesar los datos por lotes (minibatches)
   ii. Utilizar como dataset Caltech101 (airplanes vs motorbikes) rescaleado a 28x28 pxl.
   iii. Agregar al modelo una capa de 100 neuronas ocultas con activación ReLU.

Los datos random solo se utilizan para verificar los métodos. Se obtiene aproximadamente el 50% de error (lo esperable).

Con los datos “airplanes vs motorbikes” se obtiene la Tabla 1. El entrenamiento por minibatch parece converger en menor tiempo para una tasa similar de error en entrenamiento. La adición de la capa oculta debería mejorar la capacidad de discriminación no lineal, y la generalización con la función no lineal de tipo relu. Aún así se observa que la tasa de reconocimiento sin la capa oculta es alta, por lo que parece un problema linealmente separable y eso explicaría el margen poco significativo con implementaciones más complejas.

Tabla 1: Resultados sobre el dataset “Airplanes vs. Motorbikes” utilizando el ejemplo de regresión logística modificado.

Experimento	**Exactitud en train**	**Exactitud en test**	**Tiempo [s]**
Batch	95.76 (SD: 0.34)	**95.56 (SD: 0.94)**	6.29 (SD: 0.17)
Minibatch	94.00 (SD: 6.63)	94.81 (SD: 1.04)	**0.52 (SD: 0.02)**
Minibatch + capa oculta sigmoidea	96.00 (SD: 6.63)	94.34 (SD: 2.55)	3.07 (SD: 0.90)
Minibatch + capa oculta relu	97.00 (SD: 4.58)	95.28 (SD: 1.34)	3.21 (SD: 0.69)

2. Keras
    a. Experimente y documente las siguientes modificaciones al ejemplo de redes
        i. Utilizando el 25% de los datos para entrenamiento.
        ii. Utilizando sólo una capa.
        iii. Utilizando unidades sigmoideas.
        iv. incrementando el efecto del dropout.
        v. Eliminando el dropout.:

Experimentos con el script ‘keras_mnist.py’. Se puede observar de la tabla que la reducción del set de entrenamiento es lo que más influye. En este caso los cambios en la implementación no parecen influir de forma significativa.

Tabla 2: Resultados de las diferentes modificaciones del código de ejemplo para MNIST.

Experimento	**Validation Acc**	**Test Acc**	**Mejora relativa al original**
Original	0.9820	0.982	1
i	0.9670	0.967	0.545
ii	0.9818	0.9818	0.989
iii	0.9809	0.9809	0.942
iv	0.9779	0.9779	0.814
v	0.9821	**0.9821**	1.005

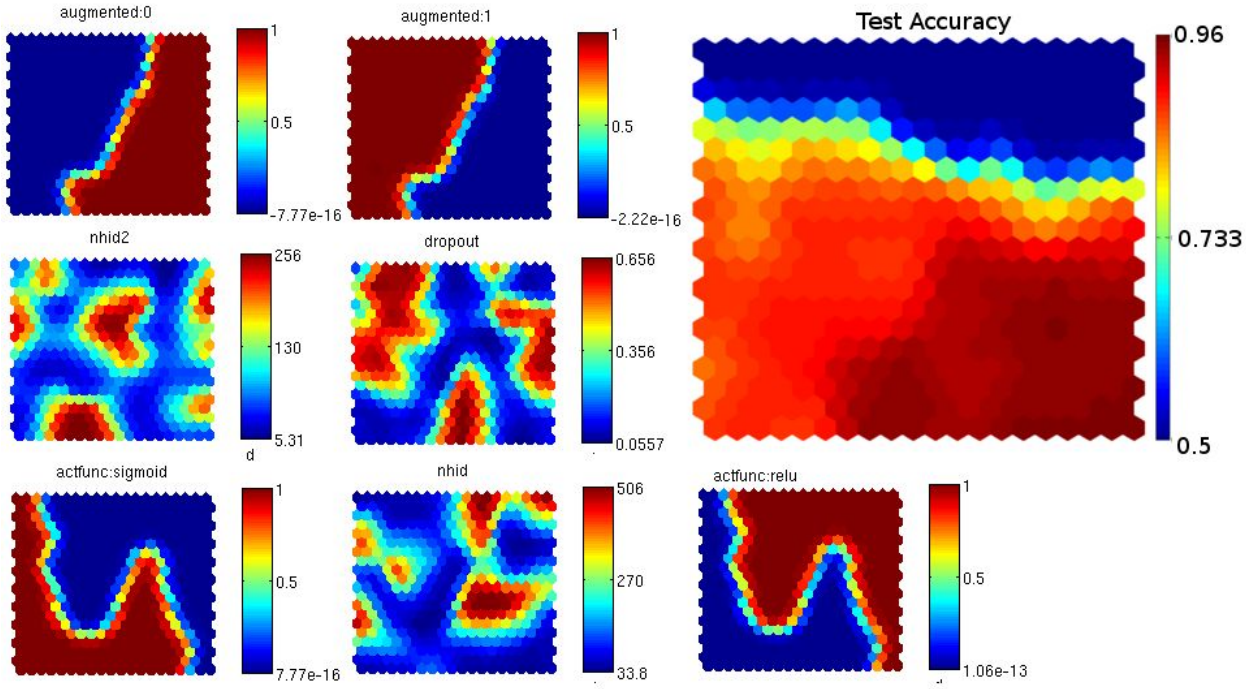
    b. Reimplemente las funciones del TP1, pero usando las primitivas de Keras de lectura y preprocesamiento de datos.

En el script ‘keras_imageTools.py’ se implementa un clasificador para la tarea ‘airplanes vs motorbikes’. Se utilizaron las funciones de preprocesamiento de imágenes de keras. La clase imageDataGenerator provee funciones para, a partir de un directorio, construir un generador de imágenes. Estas ima ǵ enes se extraen por batch y alimentan el modelo. Posee diversas funciones tanto para muestrear las imágenes como para generar datasets aumentados, realizando transformaciones sobre las imágenes como rotaciones, inversiones, entre otras.

Construidos los generadores para entrenamiento y test, se construye un modelo similar al del punto anterior. Es necesario agregar una capa Flatten para reformar el tamaño del tensor de entrada de [Canales x Alto x Ancho] a [Canales*Alto*Ancho x 1]. Al parecer keras no provee funciones para extraer parches en forma directa, aunque como se puede ver los resultados son similares utilizando la imágen completa. Las imágenes se redujeron a 64x64 y se escalaron los valores, todo utilizando el generador de keras.

Se hizo una búsqueda en grilla de diversos hiperparámetros: usar o no la técnica de aumento del dataset (augmentation), el número de neuronas en 1 y 2 capas, el nivel de dropout y la función no-lineal (sigmoidea o relu). Dado el tiempo de cómputo y el espacio a explorar, se corrió una realización por cada set de hiperparámetros.

Para analizar los resultados, se utilizó un mapa auto-organizativo (SOM) entrenado con los hiperparámetros y la exactitud para el conjunto de test[1]. Esto permite observar la relación entre los hiperparámetros que parecen más importantes. La figura 1 muestra el SOM entrenado. Cada imagen representa uno de los hiperparámetros (si es un valor numérico, se representa como una variable, si es categórico, se descompone en una representación binaria). Se puede observar que el aumento del dataset no genera mejores resultados (posiblemente la base de datos es reducida y por lo que se puede ver las imagenes son relativamente homogeneas). La funciones sigmoideas y relu aparecen en diferentes máximos locales . En cuanto al número de neuronas y si usar una segunda capa oculta, no parece tener un efecto determinante en el rango estudiado. Es necesario aclarar que no se fijó una semilla random, y posiblemente esto influya en cambios en el dataset de entrenamiento a partir del generador de keras.



[1] Al hacer este trabajo, me pareció interesante usar esta técnica para evaluar rápidamente las relaciones entre los hiperparámetros y el resultado. No se si tiene mucha validez estadística. No encontré publicaciones o herramientas estándar que sirvan para este análisis en altas dimensiones.

Figura 1: SOM representando hiperparámetros del modelo y la exactitud en la partición de test.

Tabla 3: Mejores 5 resultados de airplanes vs motorbikes.

Aumentado	**N 1er capa**	**N 2da capa**	**Dropout**	**Fun. activación**	**Exactitud test**
No	128	128	0.00	sigmoid	0.980
No	64	128	0.20	relu	0.980
No	64	64	0.10	sigmoid	0.980
No	32	32	0.20	sigmoid	0.980
No	16	128	0.20	relu	0.980
No	16	128	0.10	sigmoid	0.980

c. Investigue la interfaz que provee Keras para salvar datos en el formato h5. Descargue a disco la matriz resultante del ejercicio anterior utilizando el formato h5. Responda: ¿Qué ventajas provee este formato?

El formato HDF5 permite guardar el modelo completo en una estructura optimizada, para analizar o utilizar luego. En keras por ejemplo se puede graficar el modelo (Fig. 2). Para este ejemplo, el archivo HDF5 tiene 150mb, lo que explica la necesidad de contar con este tipo de herramientas para optimizar el almacenamiento del modelo. Otra ventaja de la implementación en keras (más allá del formato de archivo) es que almacena estados internos del entrenamiento, como el del optimizador, y permite retomar un proceso de entrenamiento previo. Esto es necesario dado el gran costo computacional que se espera necesitar.

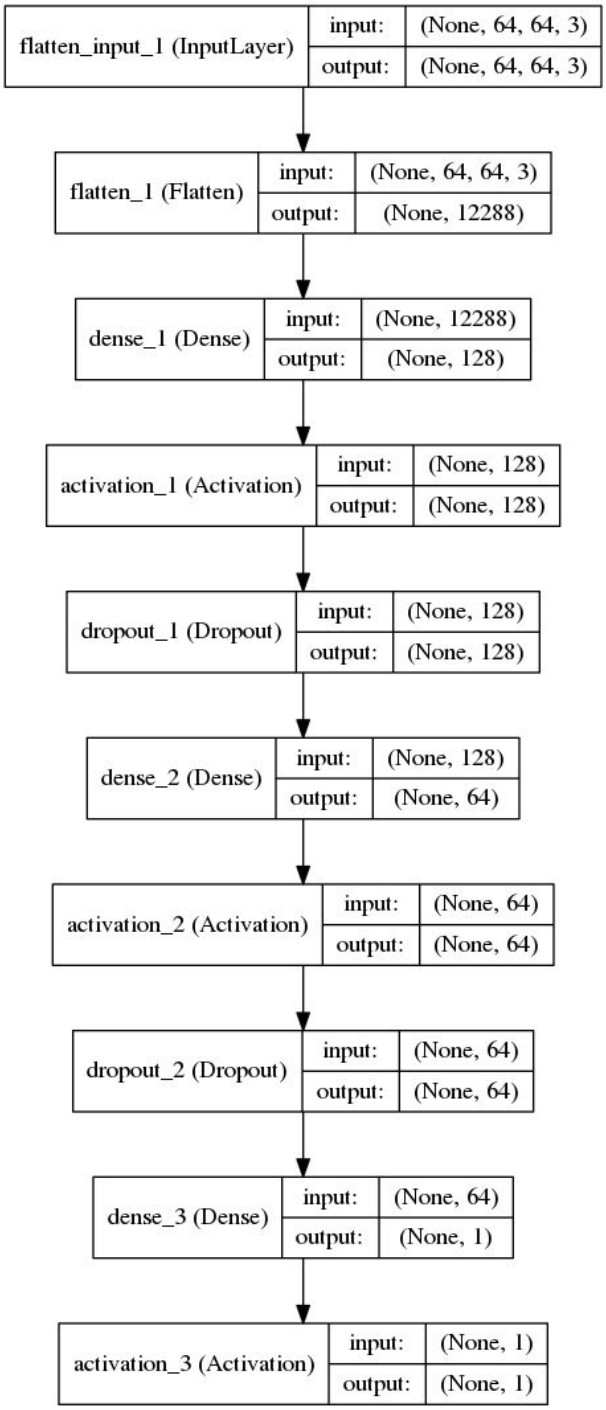


Figura 2: Modelo representado con herramientas de keras.