CC5114 Redes Neuronales y Programación Genética 2018, Primavera

Tarea 1: Red neuronal y Aplicación

Alvaro Leon Sanchez
09 de Septiembre del 2018
https://github.com/aleonss/t1_neural_network

Descripción del Dataset

El dataset utilizado se llama “**Occupancy Detection / Detección de ocupación**”, y podemos encontrarlo en: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+ .

Este corresponde a 3 archivos .csv de 7 columnas, que poseen datos experimentales utilizados para la clasificación binaria de ocupación de la sala (Occupancy), a partir de registros de temperatura, humedad, luz y CO2 que se realizaron por minutos.

Estos archivos son:

- data/datatraining.txt # 8144 filas
- data/datatest.txt # 2666 filas
- data/datatest2.txt # 9753 filas

Detalle de los atributos:

- **date,** fecha en [año-mes-día hora:minuto:segundo] (string).
- **Temperature**, en Celsius (float).
- **Relative Humidity,** humedad relativa % (float).
- **Light**, en Lux (float).
- **CO2**, en ppm (float).
- **HumidityRatio**, Derivado de la temperatura y humedad relativa, en [kg agua-vapor / kg-aire] (float).
- ***Occupancy***, 0 o 1, 0 para no ocupado, 1 para estado ocupado (boolean).

Inicialmente considere parsear “date” para así alimentar la red neuronal con horas y minutos, pero se comentó el código ya que esta información no parece ser relevante para la clasificación (la performance no mejora).

Instrucciones de cómo ejecutar la tarea

La tarea fue implementada en **Python 3.6**, se requiere tener **pip** para instalar las librerías utilizadas, las cuales son:

- **numpy**: Usado para cálculos matemáticos.
- **pandas:** Usado para trabajar con datos en formato .csv .
- **matplotlib:** Usado para graficar clasificaciones de redes neuronales entrenadas.

Matplotlib requiere el módulo “tkinter”, en Ubuntu/Debian puede ser instalado como:

```
sudo apt-get install python3-tk
```

Y en Windows debería venir junto a la instalación de Python 3.

Se pueden instalar las librerías con el comando:

```
pip install -r requirements.txt
```

La tarea consta de 3 archivos en python:

- **neural_network.py:** Clases que implementan la red neuronal.
- **tarea1.py:** Clasificación del dataset usando una red neuronal
- **tests.py:** Test de las clases y métodos implementados.

La clasificación y tests deben ejecutarse como:

```
python tarea1.py
python tests.py
```

Se recomienda usar un “virtual enviroment”, así la ejecución completa en Ubuntu/Debian se resume en:

```
sudo apt-get install python3-venv
python3 -m venv venv
source venv/bin/activate

pip install -r requirements.txt
sudo apt-get install python3-tk

python tarea1.py
python tests.py
```

Análisis de los resultados:

Cómo el número de hidden layers afecta el entrenamiento?

El número de hidden layers, como también el número de de neuronas que posee cada una, afectan directamente a la performance de la red dado un dataset.

En el caso de mi clasificación que toma 5 inputs y retorna uno, puede obtener buenas predicciones (~80% de aciertos) con solo una hidden layer, pero no siempre llegaba a estos resultados. Mejores predicciones y más confiables las pude encontrar con al menos 3 hidden layers, y seguir aumentando layers no mejoraba considerablemente la performance.

En las hidden layers mencionadas, se usó un patrón para asignar el número de neuronas, en donde: la primera debe tener el número mayor de neuronas (>5), y el número de neuronas para la siguientes hidden layers debe ir disminuyendo.
Desconozco que tan efectivo es este patrón, pero se observó cómo aumentar el número de neuronas de una hidden layer a otra no mejora la performance y aumenta el tiempo el tiempo de ejecución.

Cual es la velocidad de la red en procesar la información?

Usando un red neuronal con las capas:

Input	Hidden Layer 0	Hidden Layer 1	Hidden Layer 2	Output
(5)	**16** Neuronas	**8** Neuronas	**4** Neuronas	**1** Neurona

Se obtuvo como performance promedio:

Accuracy:	0.92
Precision:	0.75
Recall:	0.99

Y le tomó a la red neuronal durante entrenamiento y testing:

	Training Data	**Test Data 0**	**Test Data 1**
Nº filas	2665	8143	9752
Tiempo promedio	2228 ms	1518 ms	1829 ms

Cómo el learning rate afecta el entrenamiento?

El learning rate parece tener un efecto distinto en función de datos de entrenamiento, ya que cuando los datos de entrenamientos son pocos (<100), el learning rate tiene un efecto mucho más notorio, y el learning rate “óptimo” depende de los datos. Por otra parte, cuando entrenamos miles de inputs, el learning rate no afecta considerablemente la performance de la predicción.

Importa el orden de los datos de entrenamiento?

Sí, debido a que los datos vienen ordenados temporalmente, fue necesario ordenarlos antes de empezar hacer el entrenamiento, ya que en caso de no hacerlo la red neuronal no aprendía adecuadamente y en general los resultados no superan el 50% (peor que aleatorio).
Pude observar este efecto entrenando la función XOR con la misma secuencia de valores, la cual hacía que la red neuronal siempre retornara un mismo valor (1 o 0) independiente del input.