Máster en Big Data Analytics

Curso 2017-2018

Introducción al Aprendizaje Automático

Práctica 9

Clasificación mediante Redes Neuronales

Jon Ander Gómez Adrián
jon@dsic.upv.es
Departament de Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Índice

1. Motivación

Las redes neuronales son una de las técnicas de *Machine Learning* y en particular de *Pattern Classification* más ampliamente utilizadas. A pesar de su aparente complejidad, su topología permite adaptarlas a casi cualquier problema de clasificación o regresión.

Independientemente de que una persona conozca o no los entresijos de las redes neuronales, o incluso de que sepa como programarlas, lo que sí debe conocer toda persona que se dedique al análisis de datos es cual es su potencial y cómo poder utilizarlas, escogiendo la topología y las funciones de activación según el problema a abordar.

2. Objetivos

2.1. Conceptuales

- Que los alumnos conozcan el efecto de la topología de la red sobre un mismo problema a resolver.

- Que los alumnos entiendan la importancia de la inicialización de los pesos.

  Para ello debe ejecutarse varias veces la misma topología con los mismos parámetros de ajuste sobre un mismo *dataset*, ello permitirá descubrir que unas veces los resultados son mejores y otros peores. Dependiendo de la tarea, las diferencias entre los resultados obtenidos tras varios entrenamientos serán significativas o no.

- Que los alumnos sean conscientes de lo importante que es evitar el *overfitting*, pues según se verá en los ejemplos, las redes siempre se entrenan con un conjunto de muestras de entrenamiento (*training set*) y a cada *epoch* se valida el error con otro conjunto de muestras (*validation set*).

  Es habitual que el *training set* original se particione en dos partes, un subconjunto que será el que se utilice para aprendizaje y otro subconjunto para ver la evolución del error sobre muestras que la red no ha visto durante el entrenamiento. El reparto en porcentaje de ambos subconjuntos que suele utilizarse va desde $\{50\,\%, 50\,\%\}$ a $\{90\,\%, 10\,\%\}$. No obstante, puede probarse que la desproporción sea a la inversa.

  Si se observa la evolución del error sobre el subconjunto utilizado para el aprendizaje y sobre el utilizado para validación, se verá que el de

aprendizaje siempre irá decreciendo, salvo en casos anómalos o reajustes de la red durante las primeras *epochs*. Pero el error sobre el conjunto de validación llegará un momento que comenzará a crecer. Ello es por el fenómeno del *overfitting*, la red conoce muy bien a las muestras utilizadas para entrenarla y pierde capacidad de generalización. Es buen momento para parar cuando el error sobre el conjunto de validación se estanca o comienza a crecer.

2.2. En el manejo de utilidades

- Básicamente lo mismo que ya se ha visto en prácticas anteriores. Saber utilizar ciertas técnicas de transformación de las muestras del espacio de características (o de entrada) a un espacio alternativo, que puede ser de mayor o menor dimensionalidad.

- Saber utilizar funciones que proporciona el *Scikit Learn* para escalar o normalizar los datos de entrada. Ya sea en combinación con técnicas de transformación de un espacio $\mathbb{R}^d$ a otro $\mathbb{R}^M$ o directamente.

3. Tareas

3.1. Tarea 1 - Problema XOR

Repasad el código del siguiente programa y ejecutadlo repetidas veces para ver la capacidad de las redes neuronales como clasificadores.

`test-ann.py`

Este programa Python hace uso de la implementación de las redes facilidada en al carpeta `ann`. El estudio del código de las redes neuronales es opcional y se recomienda que cada persona lo realice por su cuenta si dispone de tiempo. Tener que implementar las redes excede los objetivos del presente curso.

Sí que se pide en esta tarea que los alumnos prueben diferentes topologías de la red y diferentes tipos de función de activación en cada una de las capas.

Los tipos de activación disponibles son: `linear, linear_rectified, binary, sigmoid, tanh` y `softmax`.

Se recomienda utilizar `softmax` y `linear` únicamente en la capa de salida. El primero para clasificación y el segundo para regresión.

Para el ejemplo de esta tarea, el problema XOR, se recomienda poner a verdadero la variable

```
xor_example=True
plot_examples=True
```

para que se representen las muestras del *training set* por pantalla, al menos para una ejecución. De esta manera uno se puede hacer una idea del problema.

Deben probarse distintos tamaños para la capa interna, de 2 a 512 neuronas, se pueden probar las potencias de 2, por ejemplo, y ver qué efecto tiene el número de neuronas ocultas. Las capas de entrada y salida tienen fijado el tamaño, la de entrada según la dimensionalidad de las muestras. La de salida según el número de clases a distinguir.

Deben probarse distintas funciones de activación para la capa oculta, todas salvo `linear` y `softmax`, y dado que es un problema de clasificación, para la salida pueden probarse las siguientes: `linear, softmax, tanh y sigmoid`.

Se recomienda repasar el código de la función `to_one_hot()` para ver cómo preparar un vector de salida para la red según la clase a la que pertenezca la muestra utilizada para entrenamiento en ese instante.

Como última prueba sobre el problema del XOR, las muestras se han separado mediante con un margen entre cada cuadrante.

```
margin=1.0
```

Cambiad el valor del margen a 0 para que las muestras estén juntas, apenas separadas por los ejes de coordenadas, y observad cómo se comportan las redes neuronales y cómo se comporta un clasificador del tipo KDE que se ha estado utilizado como referencia.

¿A qué tipo de clasificador afecta más el hecho de que las muestras no estén separadas por cierto margen?

¿Qué topología de la red y qué funciones de activación se adaptan mejor a cada caso?

Variad el número de muestras, por ejemplo `n_samples` igual a {50,100, 200,500,1000,2000}. ¿Qué efecto tiene el número de muestras con respecto a las redes neuronales y al clasificador KDE cuando las muestras no están separadas vía un margen?

3.2. Tarea 2 - Sobre muestras sintéticas que siguen distribuciones normales o de Gauss

Seguid trabajando con el mismo programa, pero en este caso poned a falso la variable `xor_example` para que automáticamente se ejecute el código correspondiente a la generación de muestras siguiendo distribuciones normales o de Gauss.

Comprobad el efecto de los cambios propuestos en la tarea anterior para este caso.

Dado que las muestras se generan siempre de manera aleatoria, que sirva la precisión obtenida con el clasificador de tipo KDE como referente para ver qué configuraciones de las redes van bien y cuando no es necesario aumentar en número de neuronas de la capa oculta.

3.3. Tarea 3 - Sobre el MNIST

A partir del código anterior más el código utilizado en prácticas anteriores, aplicad las redes neuronales sobre el *dataset* MNIST.

Por el tamaño del *dataset* y por la alta dimensionalidad de las muestras el entrenamiento será lento. Aunque debe probarse sin transformaciones lanzando un proceso que tardará bastante (esto debe de hacerse fuera del laboratorio), se propone, para poder probar en el laboratorio, reducir la dimensionalidad mediante PCA y utilizar un conjunto de muestras reducido para la fase de aprendizaje, por ejemplo 1000 y también probad con 2000.

Probad distintas configuraciones de la red para ver cual es la más adecuada a este problema.

3.4. Tarea 4 - Reconocimiento de caras

Se trata del *dataset* utilizado en la tarea 3 de la práctica sobre SVM.

Esta tarea consiste en aplicar las redes neuronales sobre el mencionado *dataset* realizando las mismas pruebas de configuraciones de las redes como se ha hecho en las tareas anteriores.

Partid del código disponible en la Web de *Scikit Learn* y que se utilizó en la anterior práctica.

4. Conclusiones

El principal objetivo de esta práctica era que el alumno se familiarice con las redes neuronales como clasificadores, que se habitúe a realizar cambios en la topología y a combinar distintas funciones de activación, todo ello con objeto de que conozca el comportamiento de las redes en distintos problemas y según que transformaciones se realicen sobre los datos de entrada.

Sin que el alumno sea un experto en redes a nivel de programarlas, sí es importante que las conozca lo suficiente para aplicarlas a problemas reales a los que se enfrentará. Lo primero será decidir qué toolkit ya programado utilizará, después que sepa cómo realizar pruebas para determinar la topología

y las funciones de activación. En otras palabras, que el alumno acabe siendo un usuario avanzado de las redes. Para ello debe conocer su funcionamiento y los efectos de los parámetros ajustables.

5. Bibliografía

[1] S. authors. Deep Learning. `http://www.deeplearning.net`, 2015. [Online: accessed June 2015].

[2] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Addison-Wesley, second edition, 2000.

[6] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):2–42, 2006.

[7] G. Hinton. A practical guide to training restricted boltzmann machines, 2010.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, JMLR*, 12:2825–2830, 2011.

[9] Wikipedia. Machine Learning. `http://en.wikipedia.org/wiki/Machine_learning`, 2015. [Online: accessed April 2015].