Análisis visual de técnicas no supervisadas de *deep learning* con el paquete dlvisR

David Charte, Francisco Charte, and Francisco Herrera

Universidad de Granada, Granada, España
fdavidcl@correo.ugr.es, francisco@fcharte.com, herrera@ugr.es

Resumen Las técnicas de *deep learning* aplicadas al aprendizaje no supervisado han demostrado su utilidad y potencial, pero carecen del nivel de interpretabilidad que pueden proporcionar otros algoritmos. Además, el ajuste de los parámetros de funcionamiento de este tipo de métodos suele realizarse de forma automática, y no se obtienen explicaciones de cómo influyen en el comportamiento de los modelos y los resultados que estos ofrecen. En este trabajo se presenta una herramienta desarrollada para la plataforma R, el paquete `dlvisR`. Este proporciona un conjunto de utilidades para la visualización de las variables obtenidas internamente por este tipo de modelos respecto de parámetros ajustables por el usuario. Además, un estudio sobre algunos conjuntos de datos reafirma la hipótesis de que la modificación de dichos parámetros tiene consecuencias observables visualmente, cuyo análisis podría aportar conocimiento de interés.

Keywords: deep learning · aprendizaje no supervisado · R · software

1. Introducción

El aprendizaje no supervisado abarca multitud de problemas ampliamente estudiados que tienen diversas aplicaciones presentes en distintos campos, como el tratamiento de imágenes y reconocimiento de objetos [1], análisis semántico [2] y sintáctico del lenguaje [3] o el preprocesamiento de datos y pre-entrenamiento para una posterior fase de aprendizaje [4].

Una familia de técnicas que se pueden utilizar para realizar estas tareas son las de *deep learning* (DL), capaces de construir modelos complejos para los datos a partir de numerosas pequeñas representaciones sencillas. Se pueden englobar por tanto dentro del conjunto de herramientas de *representation learning*, pero se diferencian del resto de estas en que son capaces de aprender varias capas de representaciones, cada una en base a la anterior [5,6]. En los últimos años, gracias al aumento de la capacidad de procesamiento disponible, de la calidad del software abierto dedicado a DL y del volumen de datos publicados para su tratamiento, se ha producido un resurgimiento de este tipo de técnicas, especialmente desde la *Deep Belief Network* [7].

El uso de herramientas de DL para aprendizaje no supervisado puede ser conveniente para fines de extracción de características y reducción de dimensionalidad, considerando que están diseñadas para aprender distintas representaciones parciales o más abstractas de los datos. Sin embargo, existe cierta escasez de interpretabilidad de los resultados que generan, a causa de la estructura de caja negra que exhiben. Esto puede impedir el estudio de las causas de las variaciones de comportamiento que pueden presentar, ya que no se cuenta con representaciones intuitivas del proceso de aprendizaje. Además, en la actualidad estas herramientas se ajustan mediante parámetros que se configuran de forma automática a través de distintos algoritmos existentes [8], pero que no ofrecen una idea comprensible acerca de los motivos por los que unos valores puedan conllevar un mejor rendimiento que otros.

R es un lenguaje de programación dirigido al tratamiento de datos, y como tal proporciona estructuras de datos y funcionalidades básicas para representar y tratar problemas de minería de datos. Además, existe toda una plataforma de paquetes para R denominada CRAN[1], que cuenta con multitud de librerías que facilitan tareas muy diversas, desde lectura y visualización de datos hasta el propio procesamiento mediante distintos algoritmos. En particular, las técnicas más relevantes de DL no supervisado están ya implementadas en paquetes como `h2o`[2] [9] o `darch`[3] [10]. Sin embargo, ninguna de estas implementaciones facilita mecanismos de visualización que permitan obtener una visión del comportamiento de los algoritmos en cuanto a la reducción de dimensionalidad.

El paquete para R que se ha desarrollado, `dlvisR` ("Deep Learning Visualization for R"), aprovecha las implementaciones mencionadas y construye sobre ellas un conjunto de funcionalidades que realizan visualizaciones a partir del aprendizaje conseguido por los algoritmos. Estas permiten extraer información acerca de la influencia en los resultados de las alteraciones en los parámetros, mediante un estudio a lo largo de distintos valores para los mismos. Además, se proporcionan dichas funcionalidades tanto como funciones de R como a través de una interfaz web que facilita las comparativas entre distintos conjuntos de parámetros.

La siguiente sección detalla las principales técnicas de DL aplicables al aprendizaje no supervisado. A continuación, la Sección 3 presenta la nueva herramienta para visualización `dlvisR`. La Sección 4 analiza los resultados obtenidos acerca del comportamiento de estas técnicas y, por último, la Sección 5 expone las conclusiones.

2. Técnicas de DL para aprendizaje no supervisado

Existen varias arquitecturas de DL que permiten realizar un aprendizaje no supervisado sobre los datos, de las cuales una de las más relevantes son los autoencoders. Estos tratan de aprender codificaciones de alto nivel de los datos de

[1] CRAN está disponible en `https://cran.r-project.org/`.
[2] El paquete `h2o` está disponible en `https://cran.r-project.org/package=h2o`.
[3] El paquete `darch` está disponible en `https://cran.r-project.org/package=darch`.

entrada. Otro tipo de herramientas que realizan esta tarea son las máquinas de Boltzmann restringidas (RBM, *Restricted Boltzmann Machines*), que cuentan con un enfoque distinto donde las capas de neuronas han de formar un grafo bipartito. Ambas técnicas son aplicables al aprendizaje de características orientado a mejorar el rendimiento de otras tareas de aprendizaje como la clasificación [11].

Un autoencoder [12] es una red neuronal entrenada para intentar reconstruir los valores de la entrada en la salida, es decir, intenta aprender la función identidad. Internamente, el autoencoder cuenta con una capa que describe una codificación con la que representa a los datos de entrada. Esta codificación, y no la capa de salida, es el resultado de interés cuando se entrena un autoencoder.

Puesto que lo que conviene es obtener una representación de los datos de calidad, no siempre se intenta asociar idénticamente la entrada a la salida, sino que se busca una versión aproximada que resulte en una codificación más compacta o más representativa. De esta forma surgen variantes como los *sparse* autoencoders [13], *denoising* autoencoders [14] o *contractive* autoencoders [15], y combinaciones de estos.

Así, un autoencoder puede tratar de minimizar una función de pérdida de los datos de salida respecto de la entrada, y añadir un ajuste o regularización. En cualquier caso, el modelo básico de autoencoder es el representado en la Figura 1. Este cuenta con una primera y última capa de la misma dimensionalidad, y puede tener una o más capas intermedias a lo largo de las cuales se realiza la codificación y la descodificación.

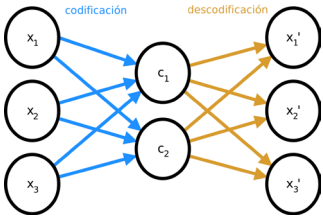

Figura 1. Representación gráfica de un autoencoder simple con una sola capa intermedia

Una vez que el autoencoder aprende una aproximación a la función identidad mediante la codificación interna, esta se puede utilizar para calcular nuevas características para las instancias con el objetivo de tener un modelo basado en los datos que posea una menor dimensionalidad. Para estudiar el comportamiento de un autoencoder visualmente, es de interés que la codificación sea en dos o tres variables, para poder representar gráficamente en dos o tres dimensiones, respectivamente.

3. La herramienta de visualización `dlvisR`

El paquete `dlvisR` para R es un software dedicado a la visualización del comportamiento de técnicas de DL no supervisadas, según los distintos valores que puedan tomar los parámetros de cada una. También incluye visualizaciones realizadas a partir del análisis de componentes principales (PCA) de los datos. Además, facilita la tarea de estudio y comparación mediante una interfaz de usuario web implementada a través del paquete `shiny`[4] [16]. Por otro lado, el desarrollo y la experimentación se pueden realizar mediante guiones de R que hagan uso de las distintas funciones implementadas y documentadas.

La herramienta `dlvisR` es de código abierto y su desarrollo se realiza de forma abierta en un repositorio de control de versiones. Se distribuye bajo licencia MIT, que permite su redistribución y modificación sin limitaciones. Se puede encontrar el código en `https://github.com/fdavidcl/dlvisR`.

3.1. Instalación

Para instalar `dlvisR` se puede hacer uso del paquete de herramientas de desarrollo para R `devtools`[5] [17]. En ese caso, será necesario simplemente ejecutar el siguiente comando en una consola interactiva de R:

```
> devtools::install_github("fdavidcl/dlvisr")
```

Dicho comando instalará únicamente las dependencias fundamentales del paquete. Estas no incluyen las implementaciones de técnicas de DL ya que son paquetes de tamaño considerable y es preferible que el usuario instale únicamente las que necesite para su uso, en lugar de requerirlas todas. Para tener una instalación completa, por tanto, es conveniente instalar el resto de paquetes mediante la siguiente orden:

```
> install.packages(c("h2o", "darch"))
```

Una vez completado este paso, se puede cargar el paquete para usar la funcionalidad que provee. Para ello, basta con ejecutar el comando `library(dlvisR)`. Próximamente se enviará el paquete al repositorio CRAN, de forma que no sean necesarias las herramientas de desarrollo para instalarlo y se pueda usar la función `install.packages()` para tal propósito.

3.2. Funcionalidades disponibles

El paquete `dlvisR` proporciona un conjunto de funciones sencillo y consistente para la creación de modelos y visualizaciones. Por un lado, la función

[4] El paquete `shiny` está disponible en `https://cran.r-project.org/package=shiny`.
[5] El paquete `devtools` está disponible en `https://cran.r-project.org/package=devtools`.

`new_model()` se encarga de realizar la tarea de construir un modelo de menor dimensionalidad a partir del dataset que se obtenga como argumento. Es necesario indicar en el parámetro `type` la técnica a usar, o bien llamar directamente a una de las funciones especializadas, como `new_model.autoencoder()`. Los valores que actualmente acepta el parámetro mencionado son `"pca"` y `"autoencoder"`, pero está planeada la inclusión de otros métodos basados en DL como las RBMs, por lo que podrá tomar más valores en el futuro.

Al utilizar un autoencoder, la función permite ajustar varios parámetros que afectan a la estructura de la red neuronal. En concreto, permite especificar exactamente la cantidad de capas y el número de neuronas por capa mediante el parámetro `layer`. El parámetro acepta un vector donde cada elemento es un entero o bien un número en coma flotante entre 0 y 2 que indica una proporción respecto del número de variables de entrada. Asimismo, la función de activación de las neuronas se puede alterar en el parámetro `activation`, de entre los posibles valores para el paquete `h2o`: `Rectifier`, `Tanh`, `TanhWithDropout`, `RectifierWithDropout`, `Maxout` y `MaxoutWithDropout`. Por último, también se puede aumentar o disminuir el número de veces que se debe iterar el dataset sobre la red a través del parámetro `epochs`. El siguiente es un ejemplo de uso:

```
> iris_model <- new_model(type = "autoencoder",
                          dataset = iris,
                          class_col = 5,
                          layer = c(5, 2, 5),
                          activation = "TanhWithDropout",
                          epochs = 100)
```

El valor de retorno de esta función es un objeto de clase `"dlmodel"`, que incluye el modelo de menor dimensionalidad generado, la columna de clases y un nombre que se puede utilizar si se escoge guardar el modelo en un archivo. Una vez obtenido este objeto, se puede utilizar la versión específica de la función `plot` de R que ofrece el paquete para la clase `"dlvisR"`, para obtener una representación de gráfico de dispersión en 2 o 3 dimensiones del modelo. Así, basta con ejecutar `plot(iris_model)` para obtener tal gráfico.

3.3. Interfaz de usuario web

Para facilitar al usuario la generación de gráficos, el paso de parámetros y la comparación de técnicas, se ha desarrollado una interfaz gráfica para `dlvisR`. Dicha interfaz se puede lanzar llamando a la función `start_gui()`, lo que abrirá una pestaña de navegador para mostrarla.

Como se observa en la Figura 2, la interfaz incluye dos paneles de visualización, cada uno muestra un gráfico y permite configurar las distintas opciones y parámetros en la región inferior.

Para cargar un dataset, basta con pulsar el botón etiquetado *Upload new dataset* y seleccionar un archivo de datos, por ejemplo un dataset en formato ARFF. Tras esto, se podrá escoger el atributo que representa la clase mediante la

Figura 2. Interfaz gráfica de usuario de `dlvisR`

lista de selección que se muestra al lado del nombre del dataset cargado. Así, se evitará entregar a los algoritmos los datos de clase y se colorearán las instancias resultantes de acuerdo con la clase a la que pertenezcan.

4. Experimentación realizada

Se ha llevado a cabo un estudio sobre cómo los cambios en la estructura de los autoencoders influyen en la distribución y organización de las características aprendidas. En concreto, se han observado estas variaciones sobre el dataset WDBC (*Diagnostic Wisconsin Breast Cancer Database*) del repositorio UCI [18], y sobre un subconjunto del conocido dataset MNIST [19]. El primero es un dataset binario formado por 569 instancias con 30 atributos de entrada, orientado al diagnóstico de tumores malignos; mientras que el segundo es un dataset multiclase originalmente con 60000 instancias de entrenamiento y 10000 de test, de 10 clases distintas y 784 variables de entrada, dirigido al reconocimiento de dígitos escritos a mano. En este caso se ha utilizado un subconjunto de 1000 instancias seleccionadas aleatoriamente de la partición de test.

Puesto que la tarea que se persigue en este caso mediante los autoencoders es una extracción de 2 o 3 características sobre las originales, se han calculado los valores que toman en cada una de las instancias y se han situado sobre gráficos de 2 y 3 dimensiones, utilizando un color para indicar la clase a la que pertenecen. De esta forma, se trata de observar si el aprendizaje realizado facilita la separación entre clases. Además, se ha utilizado PCA como algoritmo básico de comparación, tomando las 2 o 3 componentes principales del dataset según el tipo de gráfico que se requiera.

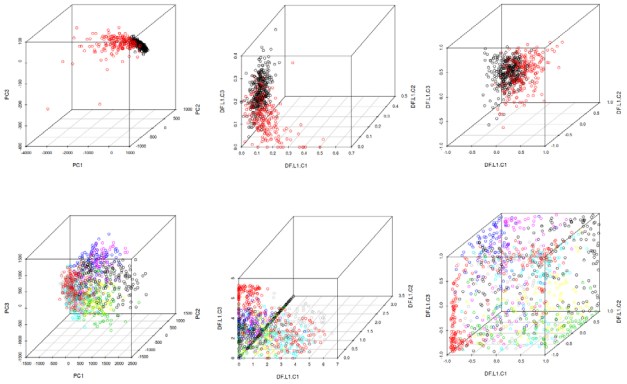

Figura 3. Arriba: dataset WDBC, abajo: dataset MNIST. De izquierda a derecha: PCA, autoencoder con función de activación *Rectifier with dropout* y autoencoder con función de activación *Tanh*

En la Figura 3 se observa el resultado de ejecuciones de PCA frente a autoencoders simples (con una sola capa intermedia de 3 neuronas) utilizando dos de las funciones de activación más usuales: *Rectifier with dropout* y *Tanh*. Se muestra para ambos datasets, donde en WDBC el color negro indica la presencia de tumor maligno, y en MNIST cada color representa un dígito distinto. Lo que se puede apreciar es que, frente a PCA, los autoencoders con función de activación *Tanh* consiguen distribuciones más homogéneas de los datos donde las instancias no se alejan demasiado de su región pero permanecen relativamente separadas del resto de regiones. Este hecho parece acentuarse en el dataset con más clases, MNIST, donde al compactar las 784 variables a únicamente 3, PCA no consigue una separación clara entre muchas clases, mientras que el autoencoder construye algunas regiones donde se concentran la mayoría de ejemplos de una clase.

En la Figura 4 se muestra el comportamiento del autoencoder con función de activación *Rectifier with dropout* al aumentar el número de épocas, o iteraciones del dataset como datos de entrada a la red neuronal. La distribución en este caso tiende a compactarse en una recta conforme se aumenta dicha cantidad. Esto indica que este autoencoder, aunque realiza cierta separación entre la mayoría de instancias de ambas clases, está desaprovechando las dos dimensiones con las que podría representar los datos ya que, al degenerar la distribución en una recta, se podría obtener una representación similar en una variable. La función de activación *Tanh*, sin embargo, no se ve afectada negativamente por el aumento de este parámetro, e incluso tiende a distribuir las instancias de forma que haya

menos solapamiento entre clases, como se plasma en la Figura 5. Lo que se puede deducir es que el número de épocas influye de formas distintas en los modelos según la función de activación escogida, en el caso del *Rectifier with dropout* aparentemente convendría usar un valor pequeño y en el caso de *Tanh* es razonable escoger un valor relativamente alto.

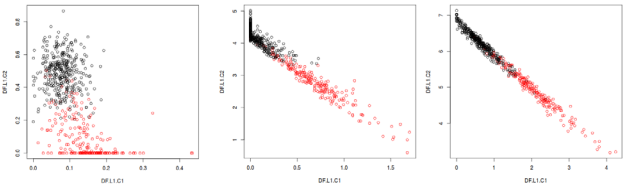

Figura 4. Función de activación *Rectifier with dropout* con número de épocas a 1, 10 y 100

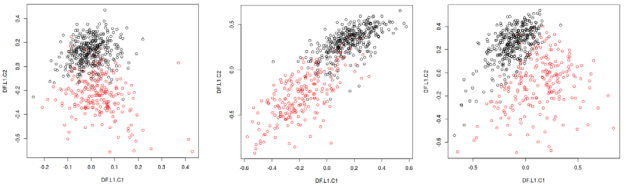

Figura 5. Función de activación *Tanh* con número de épocas a 1, 10 y 1000

En cuanto a la estructura de capas, se ha comprobado cómo puede afectar drásticamente a la distribución de las instancias con las características generadas. Por un lado, aumentar el número de capas puede facilitar que el autoencoder aproveche mejor las dimensiones del espacio de codificación y distribuya así los datos ocupando el mayor espacio posible, separando de esa forma las instancias y posiblemente reduciendo el solapamiento entre clases. Sin embargo, la adición de capas de mayor dimensionalidad puede afectar negativamente, si el resto de parámetros de la red no se adapta convenientemente. Por ejemplo, puede necesitar de un mayor número de épocas, lo que podría indicar que tiene una convergencia más lenta y por tanto requiere un mayor tiempo de cálculo. Ejemplos de estos hechos se pueden ver en la Figura 6, donde se ha utilizado una sola capa de

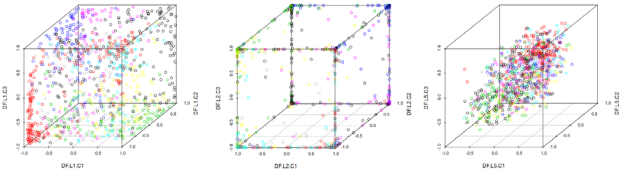

Figura 6. De izquierda a derecha: una, tres y nueve capas intermedias en la red neuronal con la función de activación *Tanh*

tres neuronas en un caso; tres capas con 589, 3 y 589 neuronas respectivamente, y nueve capas con una cantidad de neuronas de 1178 (un 150 % de la dimensionalidad del dataset) a las 3 de la capa intermedia. En concreto, la primera visualización muestra una organización en regiones amplias del espacio, mientras que en la segunda se concentra parte de los datos en la frontera del mismo, y en la tercera aparentemente se pierden las regiones que se podían observar en los otros casos. Otras visualizaciones adicionales se pueden encontrar en la página asociada al proyecto, `http://fdavidcl.me/dlvisr/`.

5. Comentarios finales

Las técnicas de DL dirigidas a aprendizaje no supervisado poseen un gran potencial pero carecen de fácil interpretabilidad. La herramienta presentada, el paquete `dlvisR` para R, es un primer avance en el intento de adquirir nociones acerca de la manera en que las alteraciones en la estructura de las redes neuronales subyacentes pueden afectar a los resultados.

Los experimentos realizados sobre datasets binarios y multiclase muestran que esta puede ser una dirección prometedora en la que profundizar. Además de observarse situaciones en las que intuitivamente los autoencoders obtienen buenas representaciones de los datos, se han deducido visualmente algunos patrones que siguen cuando se aumentan o disminuyen los valores para ciertos parámetros.

Se ha planeado como trabajo próximo la introducción de las RBMs en el paquete, aparte de los autoencoders ya presentes, y en el futuro se podría estudiar la generación de otros tipos de gráficos y representaciones en las que se pueda plasmar mejor el comportamiento de las técnicas.

Agradecimientos. Este trabajo es parcialmente financiado por el Ministerio de Educación bajo el proyecto TIN2014-57251-P.

Referencias

1. M. A. Ranzato, F. J. Huang, Y.-L. Boureau, Y. LeCun, Unsupervised learning of invariant feature hierarchies with applications to object recognition, in: Computer

Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, IEEE, 2007, pp. 1–8.
2. T. Hofmann, Unsupervised learning by probabilistic latent semantic analysis, Machine learning 42 (1-2) (2001) 177–196.
3. M. R. Brent, From grammar to lexicon: unsupervised learning of lexical syntax, Computational Linguistics 19 (2) (1993) 243–262.
4. D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, P. Vincent, The difficulty of training deep architectures and the effect of unsupervised pre-training, in: International Conference on artificial intelligence and statistics, 2009, pp. 153–160.
5. Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) (2013) 1798–1828. doi:10.1109/TPAMI.2013.50.
6. I. G. Y. Bengio, A. Courville, Deep learning, book in preparation for MIT Press (2016).
URL `http://www.deeplearningbook.org`
7. G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554.
8. J. S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.
9. S. Aiello, T. Kraljevic, P. Maj, with contributions from the H2O.ai team, h2o: R Interface for H2O, r package version 3.8.1.3 (2016).
URL `https://CRAN.R-project.org/package=h2o`
10. M. Drees, Implementierung und analyse von tiefen architekturen in r, Master's thesis, Fachhochschule Dortmund (2013).
URL `https://cran.r-project.org/package=darch`
11. A. Coates, A. Y. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: International conference on artificial intelligence and statistics, 2011, pp. 215–223.
12. D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, Tech. rep., DTIC Document (1985).
13. C. Poultney, S. Chopra, Y. L. Cun, et al., Efficient learning of sparse representations with an energy-based model, in: Advances in neural information processing systems, 2006, pp. 1137–1144.
14. P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine Learning, ICML '08, ACM, 2008, pp. 1096–1103.
15. S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, Contractive auto-encoders: Explicit invariance during feature extraction, in: Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 833–840.
16. W. Chang, J. Cheng, J. Allaire, Y. Xie, J. McPherson, shiny: Web Application Framework for R, r package version 0.13.2 (2016).
URL `https://cran.r-project.org/package=shiny`
17. H. Wickham, W. Chang, devtools: Tools to Make Developing R Packages Easier, r package version 1.11.0.9000.
URL `https://github.com/hadley/devtools`
18. M. Lichman, UCI machine learning repository (2013).
URL `http://archive.ics.uci.edu/ml`
19. Y. Lecun, C. Cortes, The MNIST database of handwritten digits.
URL `http://yann.lecun.com/exdb/mnist/`