IX Simposio de Teoría y Aplicaciones de la Minería de Datos (IX TAMIDA)

TAMIDA 7:
DEEP LEARNING

Una primera aproximación a la predicción de variables turísticas con Deep Learning

Daniel Trujillo, Antonio Jesús Rivera, Francisco Charte, María José del Jesus
Instituto Andaluz de Investigación en Ciencia de Datos e Inteligencia Computacional (DaSCI), Departamento de Informática
Universidad de Jaén
Jaén, España
{dtviedma,arivera,fcharte,mjjesus}@ujaen.es

***Resumen*—El turismo es una de las actividades económicas más importantes a nivel mundial, por lo que una correcta planificación de los recursos existentes en función de la demanda es fundamental. En este sentido, el trabajo desarrollado permite comparar la bondad de un nuevo modelo de *deep learning*, LSTM, frente a un modelo clásico ampliamente reconocido, ARIMA. Se ha llevado a cabo un proceso de entrenamiento para obtener los modelos LSTM y ARIMA que, posteriormente se han validado utilizando datos no disponibles durante el aprendizaje.**

Nuestros resultados muestran que los nuevos modelos LSTM obtienen una precisión mayor que el clásico ARIMA, tanto en la validación a priori como en la predicción posterior.

***Palabras clave*—LSTM; ARIMA; time series forecasting**

I. Introducción

La industria del turismo a nivel mundial se considera una de las actividades económicas más importantes, sólo superado por la industria del petróleo y productos derivados [2], según algunos autores. En el caso español, al sector turístico se vinculan un 12,1% del total de empleos en 2013, creciendo desde 9,8% en 2001 [3]. En una actividad tan importante es fundamental una correcta planificación, de manera que, por un lado, se pueda atender adecuadamente a la demanda, y por otro, se pueda realizar el mantenimiento del patrimonio que da lugar a la actividad turística.

Por su naturaleza, de este tipo de actividades emanan conjuntos de datos con una fuerte dependencia temporal, lo que también se conoce como **series temporales** [**?**]. En estos conjuntos de datos, las instancias aparecen ordenadas por un criterio cronológico, al tiempo que la variable de estudio presenta una fuerte dependencia cronológica. Ejemplos de este tipo de conjuntos de datos son la evolución del precio de un bien, del número de pasajeros de un aeropuerto, o del grado de ocupación hotelera en una provincia concreta.

Tradicionalmente, el método que se ha utilizado para analizar este tipo de conjuntos de datos ha sido ARIMA [4]. Este modelo integra análisis de medias móviles y análisis autoregresivo, conocido como modelo ARMA, y lo generaliza para series diferenciadas. Este modelo, junto con las estrategias heurísticas desarrolladas para encontrar sus mejores parámetros, tiene una muy importante fundamentación estadística y se le considera como el estado del arte en el ámbito de predicción de series temporales.

Por su parte, las técnicas de *deep learning* han supuesto una revolución importante en el campo del aprendizaje automático, gracias a evoluciones en los algoritmos de entrenamiento disponibles, así como el aumento de la potencia de cómputo de los ordenadores actuales. Esta combinación de factores posibilitan el uso de estos modelos capaces de reproducir procesos cognitivos complejos reservados a los humanos. Modelos de *deep learning* han conseguido resultados sorprendentes en tareas como el reconocimiento de escritura [5], lo cual los hace prometedores sucesores de los algoritmos tradicionales de aprendizaje automático.

Uno de estos modelos, LSTM (*Long Short Term Memory*), está especialmente diseñado para tareas de análisis de series temporal. Su estructura le permite almacenar conocimiento útil para razonar sobre una serie temporal con un contexto mayor que las técnicas tradicionales, estableciendo una nueva línea de estudio que quizás termine por suceder al actual estado del arte, el modelo ARIMA.

El objeto de este estudio es establecer una comparativa entre el algoritmo más comúnmente usado en predicción de series temporales, ARIMA, y las redes neuronales de tipo LSTM, en un problema real de predicción de datos turísticos de interés para la provincia de Jaén.

Este artículo se estructura de la siguiente forma: En la segunda sección se aportará una breve introducción de los métodos considerados en el estudio. A continuación, se detallará la experimentación llevada a cabo para establecer la comparativa, tras lo cual se mostrarán los resultados obtenidos para finalizar con una discusión de los mismos.

II. Métodos

En esta sección se aporta una descripción a nivel introductorio de los métodos que utilizaremos para llevar a cabo la experimentación, LSTM y ARIMA. Al no ser objetivo de este estudio proporcionar una descripción suficientemente detallada, se aportan las referencias más relevantes en cada caso.

Universidad de Jaén.

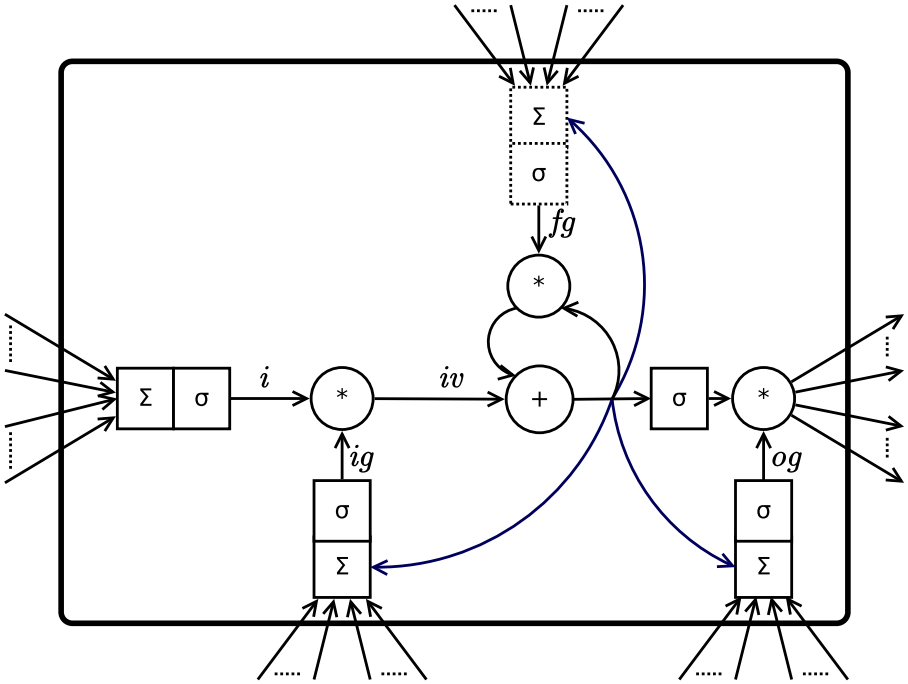

Fig. 1. Representación esquemática de una neurona tipo LSTM.

A. LSTM

Las redes neuronales de tipo LSTM surgen como respuesta a los diversos problemas en torno al flujo del gradiente que sufren los algoritmos de aprendizaje basados en gradiente de redes neuronales artificiales, los cuales se discuten ampliamente en [6], siendo el más recurrente en este tipo de redes el llamado ”vanishing gradient”. Este problema hace referencia al decreciente efecto de las variaciones de un parámetro en la salida final de la red, en la medida en la que el parámetro se aleja de la salida de la red. Esto afecta a las capas de entrada de redes prealimentadas con un número elevado de capas, pero también a redes neuronales recurrentes con amplias dependencias temporales en los datos de entrada.

Las redes neuronales de tipo LSTM han supuesto un importante impulso para las redes neuronales recurrentes, ya que, por los motivos expuestos, los algoritmos de entrenamiento de redes neuronales recurrentes más utilizados no ofrecían ninguna ventaja práctica frente a los mismos algoritmos en redes neuronales prealimentadas [6].

Long Short Term Memory es una arquitectura de neurona artificial cuya estructura le permite mantener un estado interno formado a partir de las evaluaciones que se han realizado, y que influye en las posteriores. Esta estructura le permite memorizar conocimiento que utilizará y actualizará en evaluaciones posteriores.

En su formulación original [7], esta célula define un flujo de información a través del cual, los valores de entrada se modifican mediante operaciones multiplicativas y funciones de activación para obtener su salida.

Sin embargo, posteriores modificaciones a la estructura original se han incorporado de facto en esta formulación, dando lugar a la estructura LSTM más utilizada en la literatura actual, que consta de los siguientes elementos:

- *Input*, *input gate*, *forget gate* y *output gate*: unidades simples, que primeramente aplican pesos a los valores de entrada, a los que se suman el valor de salida de *CEC* resultado de la última evaluación del *CEC* (*peephole connections*), y posteriormente aplica una función de activación.
- CEC (*Constant Error Carrousel*): situado en la parte central de la neurona, está formado por una autoconexión recurrente. Esta autoconexión constituye la memoria de la neurona, cuyo valor se suma a la entrada del *CEC* y se actualiza en cada evaluación de la neurona a su salida.

En la fig. 1 se detalla el interior de una neurona LSTM. En ella se puede apreciar muy facilmente el flujo de informacion dentro de la neurona, de izquierda a derecha, mientras es transformada por las 3 válvulas de información. Asimismo, se puede ver el CEC, justo en el centro de la neurona.

A continuación se proporciona una descripción textual de la neurona LSTM:

1) Los valores de entrada se suministran a las unidades *input*, *input gate* y *forget gate*, obteniendo los valores *i*, *ig* y *fg*, respectivamente
2) *i* e *ig* se combinan multiplicando sus valores elemento a elemento, obtenido *iv*.
3) *fg* se multiplica, elemento a elemento, con el valor de salida del *CEC* en el instante de tiempo anterior, obteniendo el nuevo valor de entrada de *CEC*.
4) Este valor de entrada de *CEC* se suma con *iv*, elemento a elemento, obteniendo la salida de *CEC*. Ésta, además de continuar por el flujo de información, se copia a

las *input*, *forget* y *output gate* a través de las *peephole connections*.

5) Se calcula la salida del *output gate*, *og*, dados los valores originales de entrada y el nuevo valor de salida del *CEC* a través de las peephole connection.
6) Por último, la salida de *CEC* se suministra a una función de activación, cuyo resultado se multiplica elemento a elemento con *og* para obtener la salida final de toda la neurona LSTM.

Esta estructura interna está diseñada para trabajar con series de datos en intervalos mucho más amplios, ya que se asegura, junto con un entrenamiento adecuado, generalmente mediante backpropagation, un flujo de error constante en el *CEC*, mientras que la *input gate* protege el valor recordado en el *CEC* de entradas irrelevantes, y la *output gate* protege a otras células de valores irrelevantes en el *CEC*.

Esta descripción de la estructura estándar es producto, como se ha mencionado antes, de modificaciones sobre la estructura original que se define en [7]. Dichas modificaciones incluyen:

- *Forget gate*: Previamente a su introducción en [8], el valor de salida del *CEC* simplemente se copia a la entrada del *CEC*, mientras que este valor, agregado a los datos de entrada, se suministran a una función de activación. Tras su introducción, se elimina la función de activación, y la salida del *CEC* pasa a ser simplemente la agregación del valor anterior del *CEC* a los datos de entrada.
- *Peephole connections*: Son conexiones con peso desde la salida del *CEC* hasta las *input gate* y *forget gate*, donde se usarán en la siguiente evaluación; y hasta la *output gate*, donde se usará inmediatamente para calcular el valor *og* y la salida final de la LSTM.

B. ARIMA

ARIMA [4] es un modelo ampliamente utilizado en la literatura, por su especial bondad en tareas de predicción de series temporales. El funcionamiento general de este modelo consiste en aproximar la serie temporal mediante una función matemática que pueda ser evaluada en las condiciones que se desean predecir. Esto significa que ARIMA no sólo es útil en tareas predictivas, sino que el ajuste que realiza de la serie temporal puede ser analizado desde un punto de vista descriptivo.

ARIMA se construye a partir de otro modelo, llamado ARMA, generalizándolo para series temporales que han sido diferenciadas. Esto es, una serie temporal obtenida como las diferencias entre cada elemento y el anterior de la serie temporal inicial. El nombre de ARMA proviene de las iniciales *AutoRegressive and Moving Averages*, y consiste en encontrar, primeramente un modelo autoregresivo sobre las diferencias de los valores a predecir, y posteriormente modelar el error de la anterior regresión a través de un modelo de medias móviles.

III. Experimentación

Para conocer el desempeño de los algoritmos se han realizado ejecuciones de ambos sobre un conjunto de datos real,

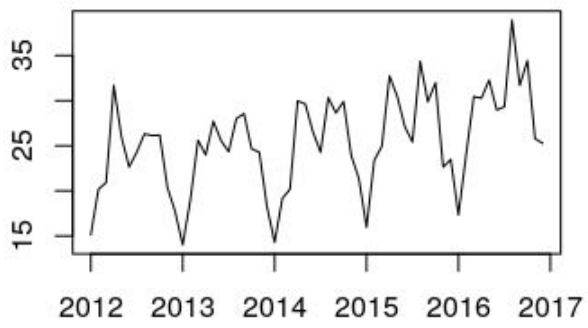

Fig. 2. Visualización de la serie temporal.

obteniendo para cada una de las ejecuciones unas métricas de error que serán objeto de comparación más adelante.

Con el objeto de hacer la comparativa lo más justa posible, el criterio que se ha utilizado a la hora de seleccionar los mejores parámetros del modelo LSTM es la precisión de predicción en el conjunto de entrenamiento. Tras seleccionar los mejores conjuntos de parámetros según este criterio, se ha procedido a reevaluar los modelos en el conjunto de test. De este modo el conjunto de test no interviene en el entrenamiento, como es natural, pero tampoco interviene en la selección del modelo, simulando con mayor precisión un escenario real cuyo objetivo sea el de predecir valores futuros.

A. Métricas de error

En la elaboración de esta experimentación se han considerado 2 métricas de error distintas: La raíz del error cuadrático medio (*rmse*); y el error porcentual absoluto medio (*mape*).

Las métricas de error cuantifican el desvío de una muestra (en este caso, representa la predicción realizada por nuestros modelos) con respecto de otra (los valores reales para la serie temporal).

Siendo E el vector de valores esperados y P el vector de valores predichos, teniendo E y P el mismo número de valores, ambas métricas se definen como sigue:

1) RMSE:

$$RMSE(E,P) = \sqrt{MSE(E,P)} \tag{1}$$

$$MSE(E,P) = \frac{\sum_{i=1}^{n}(E_i - P_i)^2}{n} \tag{2}$$

2) MAPE:

$$MAPE(E,P) = \frac{100}{n}\sum_{i=1}^{n}\frac{E_i - P_i}{E_i} \tag{3}$$

B. Conjunto de datos

Los datos empleados se han obtenido del repositorio público de Instituto Nacional de Estadística. En concreto, se han utilizado los relativos a la Encuesta de Ocupación Hotelera, serie correspondiente al grado de ocupación por plazas de la provincia de Jaén. Estos datos han sido agrupados por la fuente en intervalos de tiempo de un mes, siendo este intervalo la resolución de la que disponemos. En la fig. 2 se puede ver una representación de los datos obtenidos.

Como cabe esperar de una serie temporal relacionada con el sector turístico, se puede apreciar a simple vista una periodicidad de 12 meses en los valores, además de picos de ocupación en los meses de marzo-abril y julio-agosto, correspondiente con los períodos vacacionales de semana santa y verano.

De todos los datos disponibles, se han utilizado los comprendidos entre enero de 2012 y diciembre de 2016. Posteriormente, se han construido 2 conjuntos de datos: El primero de ellos es el conjunto de entrenamiento, y está formado por los datos comprendidos entre enero de 2013 y diciembre de 2015. El resto de datos, los comprendidos entre enero y diciembre de 2016 compondrán el conjunto de datos de test.

A las series temporales extraídas se les han añadido regresores externos adicionales para aumentar la eficiencia de los métodos. Estas columnas adicionales se han decidido con cuidado de no introducir incertidumbre en el conjunto de datos, considerando sólo aquellos regresores que se puedan obtener de forma precisa de antemano. En concreto, se ha añadido información sobre:

- Año
- Mes del año
- Semana santa: Indica si el mes en cuestión contiene parte de la semana santa de ese año.
- Puentes largos: Número de festivos en días martes o jueves en el mes dado.
- Puentes cortos: Número de festivos en días lunes o viernes en el mes dado.

Si bien esta es la estructura general de los conjuntos de datos de partida, por exigencias de cada implementación la estructura se ha adaptado en cada caso:

LSTM: Cada instancia suministrada a la red durante el entrenamiento tiene la forma $(V_n, V_{n-1}, V_{n-11}, V_{n-12}, SS, PC, PL)$, mientras que para la predicción se elimina el valor V_n por motivos obvios.

ARIMA: En el caso de ARIMA, cada instancia tiene la forma $(V_n, V_{n-12}, SS, PC, PL)$

Siendo:

n El mes correspondiente a la instancia en cuestión.
V_i El valor de la serie temporal correspondiente al mes i-ésimo.
SS Indica si en ese mes tiene lugar la Semana Santa.
PC El número de puentes cortos del mes correspondiente.
PL El número de puentes largos del mes correspondiente.

En el caso de la LSTM, se ha optado por añadir el valor de la serie temporal de mes anterior, del año anterior, y del undécimo mes anterior. Esta selección de retardos obedece a los resultados de un análisis preliminar de autocorrelación de la serie temporal, fig. 3, que muestra que estos valores son los más significativos.

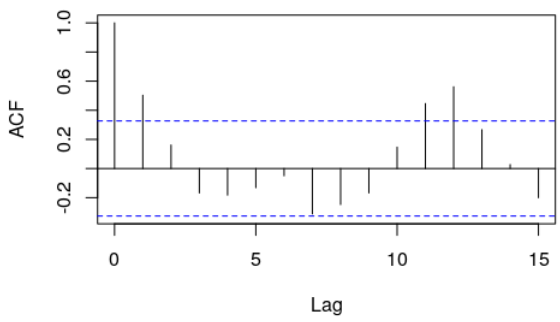

Fig. 3. Función de autocorrelación.

C. Métodos

La experimentación con las redes neuronales recurrentes de tipo LSTM se ha implementado en *Python*, utilizando la librería *Keras* con el backend *TensorFlow*, configurado por defecto. Asimismo, se ha utilizado la librería *Pandas* para la gestión de los datos, *Numpy* para el cálculo numérico, *Matplotlib* para la visualización de los resultados y *Scikit-learn* para el cálculo de métricas de error, así como la normalización de los datos de entrada.

A pesar de que las redes neuronales recurrentes LSTM son un modelo de tipo *deep learning*, el tiempo de entrenamiento por modelo en esta experimentación no ha sido excesivo.

Una vez realizadas ejecuciones preliminares explorando distintas zonas del espacio de configuraciones de parámetros, a la vista de las métricas de error obtenidas, se ha procedido a seleccionar la región del espacio donde encontraremos los modelos que calculen las mejores predicciones:

- Número de LSTM: 25, 50 y 100
- Iteraciones: 100 y 200
- *batch_size*: 1 y 2

Dado que el algoritmo utilizado para el entrenamiento de las redes LSTM, BackPropagation Through Time, parte de un estado inicial aleatorio, para reducir el efecto del azar de los elementos del estado inicial de la red, se han realizado 20 ejecuciones con cada una de las combinaciones de parámetros del espacio de búsqueda reducido definido anteriormente. De esta forma el error medio obtenido está muy escasamente influenciado por algunas ejecuciones cuyos resultados pueden verse desviados por motivos del azar.

Para el entrenamiento del modelo ARIMA se ha utilizado el entorno R de computación estadística, en lugar del lenguaje Python, como se ha hecho anteriormente. Esto es debido a que la implementación de referencia del modelo ARIMA está desarrollada en este lenguaje. En concreto, se ha utilizado la función *auto.arima* del paquete *forecast* [4].

El programa de entrenamiento del modelo ARIMA es visiblemente más simple que en el caso de la red LSTM. Esto es debido a que todo el análisis de parámetros y entrenamiento del modelo se realiza de forma automática por la implementación elegida, utilizando técnicas como las descritas en los trabajos [10], [11]. La existencia de distintas heurísticas

TABLA I
Resultados ejecuciones LSTM - RMSE Entrenamiento

Num LSTM	Iterac.	batch size	RMSE	
			Promedio	Desv. Est.
25	200	1	2,17954	0,01534
50	200	1	2,18941	0,01802
25	200	2	2,19675	0,01309
50	200	2	2,20690	0,00849
100	200	1	2,21396	0,03308
100	200	2	2,22174	0,01000
25	100	1	2,22621	0,01257
50	100	1	2,23581	0,01208
25	100	2	2,23769	0,01105
100	100	1	2,24942	0,02027
50	100	2	2,24990	0,01107
100	100	2	2,27130	0,01539

TABLA II
Resultado ejecución ARIMA

Entrenamiento	Test	
RMSE	RMSE	MAPE
2,40561	1,89010	5,01111

hace posible que este proceso se pueda automatizar teniendo en cuenta métricas comparativas como *Akaike information criterion* [12]

Además, dado que ni ARIMA ni las heurísticas que buscan los mejores parámetros son probabilísticos, no es necesario realizar varias ejecuciones del algoritmo, ya que todas arrojarían exactamente los mismos resultados.

IV. Resultados

En la tabla I se pueden ver los resultados de las mejores ejecuciones conseguidas con redes neuronales de tipo LSTM. En ella se muestra, para la métrica de error RMSE en la partición de entrenamiento del conjunto de datos, el promedio y la desviación típica de 20 repeticiones completas del experimento. Los distintos modelos se muetran en la tabla ordenados según el error cometido en la experimentación. A la vista de estos resultados, seleccionamos el modelo de menor error para la comparativa con ARIMA.

En la tabla II se muestra el resultado de la ejecución realizada con el modelo ARIMA. Como se ha introducido antes, dado que este modelo no es probabilístico, carece de sentido realizar varias repeticiones, ya que todas predicen exactamente igual, por lo que las métricas obtenidas serán idénticas.

La tabla comparativa III se muestran los resultados obtenidos en la partición de test, tanto para el modelo ARIMA como para el mejor modelo LSTM, el cual se ha seleccionado siguiendo el criterio del menor error en la partición de entrenamiento (tabla I). Es fácil observar de esta forma que LSTM obtiene mejores resultados que ARIMA en ambas métricas de error consideradas.

V. Conclusión

En este artículo se ha realizado una comparativa entre un nuevo modelo de red neuronal recurrente, LSTM, y el algoritmo considerado estado del arte en predicción de series temporales, ARIMA, sobre un problema de predicción de series temporales en turismo, justificado por la importancia del sector turístico en la economía mundial.

TABLA III
Comparativa ARIMA - LSTM

	RMSE	MAPE
LSTM	**1,70039**	**4,61407**
ARIMA	1,89010	5,01111

Los resultados de la experimentación llevada a cabo, resumidos en la tabla III, indican que un modelo de red neuronal de tipo LSTM, seleccionado basándonos exclusivamente en datos de entrenamiento, obtiene una mayor precisión en la predicción de los datos de test que el modelo ARIMA clásico, ampliamente utilizado en el campo de series temporales.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de España de Ciencia y Tecnología bajo el proyecto TIN2015-68454-R.

Referencias

[1] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.
[2] C. Altés, "Marketing y turismo," *Madrid: Editorial Síntesis*, 1993.
[3] J. R. Cuadrado Roura, J. M. López Morales *et al.*, "El turismo, motor del crecimiento y de la recuperación de la economía española," 2015.
[4] R. J. Hyndman, Y. Khandakar *et al.*, *Automatic time series for forecasting: the forecast package for R*. Monash University, Department of Econometrics and Business Statistics, 2007, no. 6/07.
[5] V. V. Romanjuk, "Training data expansion and boosting of convolutional neural networks for reducing the mnist dataset error rate," *Naukovi Visti NTUU KPI*, no. 6, pp. 29–34, 2016.
[6] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.
[7] J. Schmidhuber and S. Hochreiter, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.
[8] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," 1999.
[9] J. Brownlee, "Multivariate time series forecasting with lstms in keras-machine learning mastery," *Machine Learning Mastery*, 2017.
[10] E. J. Hannan and J. Rissanen, "Recursive estimation of mixed autoregressive-moving average order," *Biometrika*, vol. 69, no. 1, pp. 81–94, 1982.
[11] V. Gómez, *Automatic model identification in the presence of missing observations and outliers*. Ministerio de Economía y Hacienda, Dirección General de Análisis y Programación Presupuestaria, 1998.
[12] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 199–213.
[13] J. E. Ball, D. T. Anderson, and C. S. Chan, "Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community," *Journal of Applied Remote Sensing*, vol. 11, no. 4, p. 042609, 2017.

Detección de cáncer de piel usando técnicas de aprendizaje profundo

1st Alejandro Polvillo Hall
Departamento de Datos Geographica
Sevilla, España
alejandro@geographica.gs

2nd Juan A. Álvarez-García
Dpto. de Lenguajes y Sistemas Informáticos Universidad de Sevilla
Sevilla, España
jaalvarez@us.es

3rd Cristina Rubio-Escudero
Dpto. de Lenguajes y Sistemas Informáticos Universidad de Sevilla
Sevilla, España
crubioescudero@us.es

***Resumen*—El aprendizaje profundo ha sido muy utilizado para la clasificación de imágenes a partir de la competición ImageNet en 2012. Esta clasificación de imágenes es de gran utilidad en el campo de la medicina, en el que ha habido un gran crecimiento de uso de técnicas de minería de datos en los últimos años. En este trabajo seleccionamos y entrenamos una red de aprendizaje profundo para el análisis de un conjunto de datos de cáncer de piel, obteniendo resultados muy satisfactorios, ya que el modelo ha superado los resultados de clasificación de dermatólogos entrenados haciendo uso de un dermatoscopio, de otras técnicas de aprendizaje automático, y de otras técnicas de aprendizaje profundo.**

***Index Terms*—aprendizaje profundo, imágenes médicas, análisis de datos clínicos**

I. Introducción

El aprendizaje profundo ha sido utilizado en el campo de la visión por computadora durante décadas [5], [9]. Sin embargo, su verdadero valor no se había descubierto hasta la competición de ImageNet en 2012 [8], un éxito que provocó una revolución a través del uso eficiente del procesamiento de unidades gráficas (GPU). El principal poder del aprendizaje profundo radica en su arquitectura [10], [11], que permite discriminación en múltiples niveles de abstracción para un conjunto de características.

Las técnicas de aprendizaje profundo han sido utilizadas con éxito en campos como el de la medicina, en el que el aprendizaje profundo viene a solucionar problemas que presentan los algoritmos de aprendizaje automático con algunas estructuras de datos muy utilizadas en medicina como son las imágenes. La minería de datos clínicos es la aplicación de técnicas de minería de datos a los datos clínicos, con el objetivo de interpretar los datos disponibles. Permite la creación de modelos de conocimiento y proporciona asistencia para la toma de decisiones clínicas. En los últimos 10 años, ha habido un interés creciente en la aplicación de técnicas de minería de datos a los datos clínicos. MEDLINE ha visto un fuerte aumento de factor 10 en el número de trabajos con el término ”minería de datos.[en] su título [7].

Para hacer un entrenamiento completo, el aprendizaje profundo requiere una gran cantidad de datos de entrenamiento etiquetados, un requisito que puede ser difícil para cumplir en el campo de la medicina, donde la anotación de expertos es costosa y las enfermedades (por ejemplo, lesiones) no cuentan con grandes conjuntos de datos. Además, requiere una gran cantidad de recursos computacionales para que el entrenamiento no sea excesivamente lento.

En este trabajo hemos aplicado técnicas de aprendizaje profundo para analizar un conjunto de datos de imágenes de cáncer de piel, obteniendo resultados muy satisfactorios.

En las siguientes secciones describimos las metodologías y resultados obtenidos.

II. Metodología

En esta sección se describen las metodologías utilizadas para el desarrollo de este trabajo.

II-A. Tensorflow

Tensorflow [1] es uno de los mejores frameworks de aprendizaje profundo existente. Ha sido adoptado por un montón de grandes empresas como Airbus, Twitter, IBM y otras más debido a su flexibilidad y polivalencia. Tensorflow es desarrollado por Google, que la usa en todos sus proyectos de aprendizaje automático y aprendizaje profundo.

Tensorflow no es en sí un framework de aprendizaje profundo, es un framework que te permite trabajar de forma muy rápida con matrices gracias a su paralelización en GPU's. Como casi todos los cálculos realizados para entrenar y predecir con una red neuronal son cálculos matriciales hacen de esta herramienta que sea ideal para usarla en la construcción de redes neuronales.

Tensorflow tiene un paquete interno que viene con la funcionalidad necesaria para hacer funcionar una red neuronal convolucional: Capas convolucionales, optimizadores, funciones de optimización, etc...

Una desventaja de Tensorflow es que es necesario escribir mucho código para conseguir algo funcional. El hecho de escribir todo ese código hace que tengas un control total sobre todos los elementos de la arquitectura de la red neuronal. Aunque también puede hacer que cometas muchos errores.

II-B. Keras

Keras [2] es un framework específico de aprendizaje profundo. No es competidor de Tensorflow, pues Keras se ejecuta "encima" de Tensorflow. Keras provee una sintaxis extremadamente fácil para la creación de redes neuronales, y después convierte esta sintaxis a modelos de Tensorflow, usando la potencia de éste para ejecutar toda la maquinaria de aprendizaje.

Debido a que nuestro caso de uso será un caso de uso de red neuronal, y no necesitaremos en principio la manipulación a muy bajo nivel de los modelos de aprendizaje profundo, nosotros optamos por el uso de Keras para el desarrollo del presente proyecto.

III. Resultados

El conjunto de datos seleccionado para evaluar nuestra propuesta ha sido el conjunto de imáganenes de cáncer de piel de The International Skin Imaging Collaboration (ISIC) (https://isic-archive.com/). Es una plataforma que intenta aunar a los profesionales dermatológicos con el objetivo de luchar contra el cáncer de piel. El conjunto de datos está formado por 23906 imágenes. Además de las imágenes, se proporcionan algunos metadatos entre los que destacamos edad, sexo, lugar anatómico del melanoma, tipo de diagnóstico, clase de melanoma y espesor del melanoma.

En primer lugar analizamos el conjunto de datos con respecto a la clase benigno/maligno. Como se puede observar en la Figura 1 está muy desbalanceado.

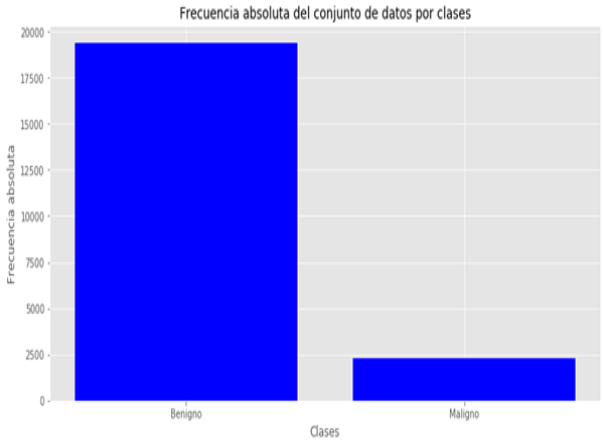

Figura 1. Conjunto de datos de cáncer de piel desbalanceado.

Debido a esto recurrimos a una técnica de remuestreo sobre la clase minoritaria que se encarga de escoger elementos al azar con reemplazo y añadirlos a la clase minoritaria. La ventaja de ésta técnica es que no elimina información, simplemente refuerza la información existente sobre la clase minoritaria. Tras aplicar la técnica el resultado obtenido se puede ver en la Figura 2.

En segundo lugar, partimos el conjunto de datos en entrenamiento (80 %) y test (20 %) asegurándonos de mantener balanceadas las clases en cada uno de los conjuntos.

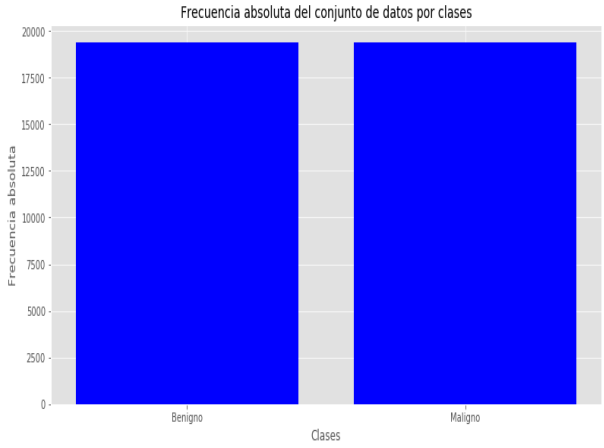

Figura 2. Conjunto de datos de cáncer de piel después del balanceo.

En tercer lugar, aplicamos un mecanismo de aumento de datos consistente en realizar distintas transformaciones aleatorias sobre cada uno de los elementos del conjunto de entrenamiento. Al realizar transformaciones aleatorias sobre el conjunto de datos conseguimos un doble efecto: aumentar el tamaño de nuestros datos y hacer que el sistema generalice mucho mejor. En particular aplicamos las técnicas conocidas como rotación automática, traslación vertical y horizontal de píxeles, cizallamiento, zoom, volteo, filtro de sal y pimienta.

III-A. Selección y aplicación del modelo de aprendizaje profundo

A la hora de enfrentarse a la selección del modelo nos encontramos en un momento de incertidumbre máxima debido a que son muchos los parámetros que tenemos que elegir, entre estos parámetros se encuentran:

- Arquitectura de la red neuronal: Propia, InceptionV3, VGG16, VGG19, Xception, etc...
- Tipo de entrenamiento: Transferencia de conocimiento, desde cero, etc…
- Parámetros de la transferencia de conocimiento: capas a entrenar, entrenamiento completo, etc...
- Algoritmo de aprendizaje: Descenso del gradiente, RMSProp, Adagrad, etc...
- Parámetros del algoritmo de aprendizaje: Tasa de aprendizaje, funciones de activación, épocas, lotes, etc…

Para seleccionar los parámetros adecuados hemos utilizado la búsqueda por rejilla (o grid search en inglés). Esta técnica se basa en la definición de un conjunto de posibles valores que pueden tomar los parámetros, y generar el producto cartesiano de todos los valores de todos los parámetros entre sí, usando cada uno de estos conjuntos de parámetros generados para construir un modelo y evaluar la bondad. En la Tabla I podemos ver todos los parámetros que se han considerado a la hora de buscar el modelo más adecuado.

En nuestra búsqueda rejilla se han fijado 3 parámetros: épocas, lotes y funciones de activación. Las épocas se han fijado puesto que es obvio que a más épocas mejor van a

Cuadro I
Parámetros considerados en la búsqueda del modelo más adecuado

Parámetro	Valores
Modelo	Xception, VGG16, VGG19 y InceptionV3
Tipo de entrenamiento	Transferencia de conocimiento, desde cero
Capas a entrenar	1, 2, 3 y todas
Algoritmo de aprendizaje	Descenso del gradiente, RMSProp, Adam, Adagrad
Tasa de aprendizaje	0.00001, 0.0001, 0.001, 0.01 y 0.1
Épocas	2
Lotes	10
Funciones de activación	ReLu

Cuadro II
Modelo y parámetros seleccionados

Parámetro	Valores
Modelo	InceptionV3
Tipo de entrenamiento	Transferencia de conocimiento
Capas a entrenar	todas
Algoritmo de aprendizaje	Adam
Tasa de aprendizaje	0.0001
Épocas	2
Lotes	10
Funciones de activación	ReLu

funcionar los modelos en su mayoría, así que se fijan a 2 épocas, para que así todas puedan recorrer el conjunto de datos 2 veces. Los lotes van de la mano de la capacidad de la unidad de procesamiento gráfico, aunque pueden influir en el entrenamiento, en este caso se han hecho los cálculos para que el conjunto de lotes sea el máximo que la unidad de procesamiento gráfica puede soportar, así hacemos que el tiempo de búsqueda por rejilla sea menor. La función de activación la he fijado a la función de activación ReLu debido a que todas las arquitecturas usan la ReLu debido a que el aprendizaje se realiza de una manera más rápida con este tipo de función de activación.

El modelo y parámetros que mejor desempeño han tenido, obteniendo un 81 % de exactitud se puede ver en la Tabla II:

El modelo seleccionado como óptimo es el modelo de Inception V3, creado por Google. Tal y como se describe en [12] era de esperar que se seleccionara la transferencia de conocimiento en vez del entrenamiento desde cero, pues posiblemente tenga un desempeño mayor. También se recomienda que, aunque se parta de unos pesos definidos (por la transferencia de conocimiento), se permita el reajuste de algunos pesos en las capas convolucionales, con el objetivo de que se adapte a nuestro problema. Como algoritmo de aprendizaje se usa Adam, con una tasa bastante pequeña. Esta tasa tan pequeña es debido a que gracias a la transferencia de conocimiento estamos muy cerca del óptimo, queremos dar pasos muy pequeños para acercarnos cada vez más al óptimo.

Una vez con nuestro modelo y parámetros definidos, procedemos a aumentar el número de épocas con el objetivo de obtener un mejor modelo y poder evaluar en la siguiente sección. Una vez el entrenamiento del modelo ha finalizado, hemos aplicado el clasificador sobre el conjunto de pruebas, obteniendo los resultados que se pueden ver en la Figura 3.

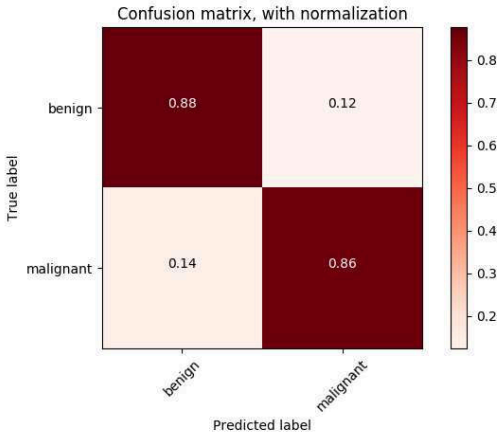

Figura 3. Matriz de confusión normalizada.

III-B. Evaluación del modelo

A continuación hablamos sobre la evaluación del modelo. La exactitud es la métrica por defecto que usa Keras para representar la bondad del modelo a lo largo de las distintas épocas de entrenamiento. En la Figura 4 podemos observar cómo se comporta el clasificador respecto a la exactitud a lo largo de las distintas épocas para los conjuntos de validación y de entrenamiento.

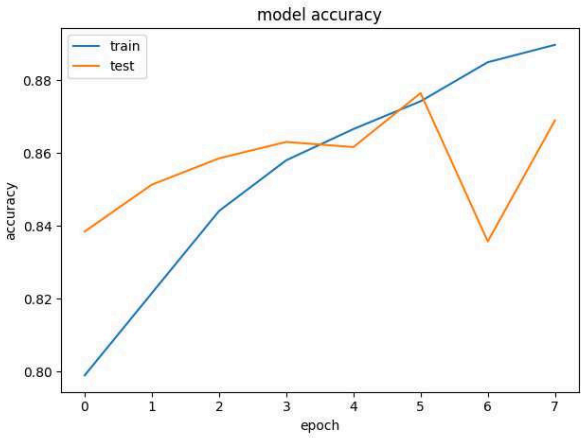

Figura 4. Exactitud del modelo a lo largo de las distintas épocas.

Como se puede observar en la gráfica siempre se encuentran en crecimiento, exceptuando en la época 6 que el conjunto de prueba hace algo difícil de interpretar, pero en la siguiente época se recupera.

El clasificador se ha puesto a entrenar durante 10 épocas, pero hemos activado una opción que permite ahorrar recursos. Esta opción consiste en que, si el clasificador durante la etapa de clasificación comienza a permanecer durante algún tiempo sin variar su exactitud, se pare la etapa de aprendizaje cuando se acabe la época. Por eso se puede observar que, aunque se haya puesto a entrenar durante 10 épocas, en la gráfica sólo se observan 8.

Cuadro III
RESULTADOS OBTENIDOS POR NUESTRO MODELO DE APRENDIZAJE PROFUNDO.

Medida	Valor
Exactitud	86.90 %
Precisión	87.47 %
Sensitividad	86.14 %
Especificidad	87.66 %
Área ROC	0.87

Se proporcionan también las medidas que se pueden ver en la Tabla III con respecto al desempeño de nuestro modelo de aprendizaje profundo.

La Figura 5 representa el área bajo la curva ROC

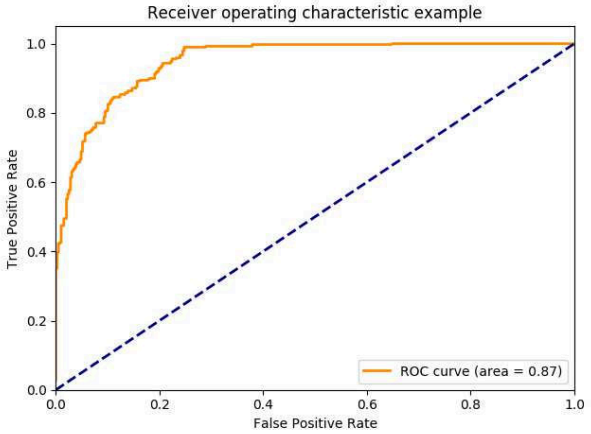

Figura 5. Área bajo la curva ROC.

Las inspecciones visuales sin ayuda de los expertos dermatológicos obtienen una media de un 60 % de exactitud [3], debido a la complejidad de observación de características diferenciadoras a simple vista. Con la ayuda de un experto entrenado junto a un dermatoscopio, la exactitud puede aumentar hasta un 75 % - 84 %.

Partiendo de la base de las afirmaciones anteriormente expuestas, y de las métricas obtenidas por nuestro clasificador, podemos observar a simple vista que nuestro clasificador ha superado en acierto a un experto dermatológico entrenado haciendo uso de un dermatoscopio. Esto parece un buen punto de partida.

Sabiendo que nuestro clasificador es capaz de superar a un experto dermatólogo, procedemos ahora a comparar nuestro clasificador con otras técnicas.

La primera intuición al intentar aplicar técnicas de inteligencia artificial en el análisis de imágenes médicas es usar aprendizaje automático o aprendizaje profundo. Con lo cual usaremos estos dos enfoques con el objetivo de comparar nuestro clasificador y ver qué bueno o malo es respecto a otros clasificadores

En primer lugar, nos centraremos en la comparación de nuestro clasificador con las técnicas de aprendizaje automático.

Cuadro IV
COMPARACIÓN DE RESULTADOS ENTRE APRENDIZAJE PROFUNDO Y APRENDIZAJE AUTOMÁTICO.

Medida	Nuestro clasificador	Aprendizaje automático
Exactitud	86.90 %	77 %
Precisión	87.47 %	71 %
Sensitividad	86.14 %	73 %

Cuadro V
COMPARATIVA ENTRE RESULTADOS DE NUESTRO APRENDIZAJE PROFUNDO Y OTRO APRENDIZAJE PROFUNDO.

Medida	Nuestro clasificador	Otro clasificador
Exactitud	86.90 %	85.5 %
Precisión	87.47 %	63.7 %
Sensitividad	86.14 %	50.7 %
Especificidad	87.66 %	94.1 %
Área ROC	0.87	0.8

El aprendizaje automático sobre el conjunto de datos de ISIC ha obtenido las métricas que se pueden ver en la Tabla IV [4].

Como se puede observar en la tabla, nuestro clasificador supera con creces a los clasificadores de aprendizaje automático enfocados al análisis de imágenes médicas para el conjunto de datos de ISIC. el resultado es lógico, pues los algoritmos de aprendizaje automático no están pensados para extraer características de imágenes, por lo que siempre se presupone un rendimiento menor.

En segundo, compararemos nuestro clasificador con otro clasificador de aprendizaje profundo [6]. El clasificador con el que compararemos nuestro clasificador, es un clasificador usado por un equipo participante de la competición de ISIC. Los resultados se pueden ver en la Tabla V:

Se puede observar, que, aunque el desempeño en la exactitud en ambos clasificadores es parecido, nuestro clasificador se comporta mucho mejor en la mayoría de situaciones. Ellos han conseguido obtener una especificidad muy alta, que es lo que le permite compensar la sensibilidad para obtener una exactitud y AUC ROC bastante aceptables.

IV. Conclusiones

En este trabajo hemos presentado el resultado de aplicar técnicas de aprendizaje profundo a un conjunto de datos de imágenes de cáncer de piel. Se ha descrito la fase de preprocesamiento de los datos, selección de parámetros para el modelo seleccionado y se han calculado distintas métricas para los resultados obtenidos.

En las métricas se pueden observar que el clasificador es bastante homogéneo, y no hay resultados de difícil interpretación. El clasificador tiene un nivel de acierto en ambas clases bastante estable, lo que nos hace indicar que ha generalizado correctamente la detección de cáncer en las imágenes. Nuestro modelo ha superado los resultados de clasificación de dermatólogo entrenado haciendo uso de un dermatoscopio, de otras técnicas de aprendizaje automático, y de otras técnicas de aprendizaje profundo.

En definitiva, el clasificador se ha comportado de manera muy buena frente a este conjunto de imágenes respecto a

otros clasificadores y técnicas. Aún sabiendo que las imágenes médicas son complicadas de tratar, el clasificador ha sabido generalizar los detalles que separan una imagen de un paciente que padece cáncer de piel de otro que no lo padece, consiguiendo así predecir con exactitud en la mayoría de casos.

Agradecimientos

Este trabajo ha sido financiado por los proyecto del Ministerio de Economía y Competitividad TIN2014-55894-C2-1-R y TIN2017-82113-C2-1-R.

Referencias

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] François Chollet et al. Keras, 2015.

[3] Noel CF Codella, David Gutman, M Emre Celebi, Brian Helba, Michael A Marchetti, Stephen W Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, et al. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). In *Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on*, pages 168–172. IEEE, 2018.

[4] Machine Learning for ISIC Skin Cancer Classification Challenge. https://hackernoon.com/machine-learning-for-isic-skin-cancer-classification-challenge-part-1-ccddea4ec44a.

[5] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[6] David Gutman, Noel CF Codella, Emre Celebi, Brian Helba, Michael Marchetti, Nabin Mishra, and Allan Halpern. Skin lesion analysis toward melanoma detection: A challenge at the international symposium on biomedical imaging (isbi) 2016, hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1605.01397*, 2016.

[7] Jimison Iavindrasana, Gilles Cohen, Adrien Depeursinge, H Müller, R Meyer, and Antoine Geissbuhler. Clinical data mining: a review. *Yearbook of medical informatics*, 18(01):121–133, 2009.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[12] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines

David Charte
Dept. CS and AI
University of Granada
Granada, España
fdavidcl@ugr.es

Francisco Charte
Dept. CS
University of Jaén
Jaén, España
fcharte@ujaen.es

Salvador García
Dept. CS and AI
University of Granada
Granada, España
salvagl@decsai.ugr.es

María J. del Jesus
Dept. CS
University of Jaén
Jaén, España
mjjesus@ujaen.es

Francisco Herrera
Dept. CS and AI
University of Granada
Granada, España
herrera@decsai.ugr.es

***Abstract*—This is a summary of our article published in Information Fusion [1] to be part of the CAEPIA-18 Key Works.**

***Index Terms*—Autoencoders, Feature fusion, Feature extraction, Representation learning, Deep Learning, Machine Learning**

I. Summary

The performance of most machine learning algorithms depends on the quality of the input data, and in particular on its features. Many techniques exist which combine their information into a new feature set, with the aim of improving learned models. This new set is usually of lower dimension and contains more abstract variables. Procedures which result in new feature sets can be named after several terms: feature engineering, feature learning, representation learning, feature selection, feature extraction and feature fusion. Manifold learning algorithms, especially those based on artificial neural networks (ANNs), fall into the last category.

Autoencoders (AEs) are feedforward ANNs with a symmetric *encoder-decoder* structure (see Fig. 1), where the middle layer represents an encoding of the input data. They are trained to reconstruct their inputs onto their output layer while some restrictions prevent them from copying the data along. AE components can generally be represented as functions: an encoder f and a decoder g which map inputs x to outputs r: $r = g(f(x))$. The desired encoding would be $y = f(x)$.

Typical activation functions such as linear, binary or ReLU are of limited use in AEs, while logistic, hyperbolic tangent and SELU are more common. According to whether the dimension of their encoding is lower or higher than that of the data, AEs can be either *undercomplete* or *overcomplete*, respectively. If they have just one hidden layer they are called *shallow*, and otherwise they are considered *deep*.

A taxonomy of the main variants of AEs has been proposed according to the properties of the inferred model:

This work is supported by the Spanish National Research Projects TIN2015-68454-R and TIN2014-57251-P, and Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

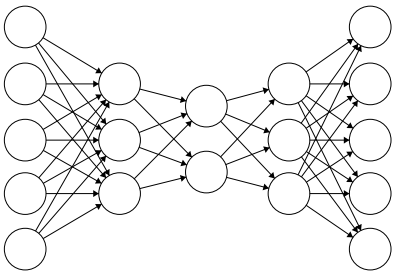

Fig. 1. A possible neural architecture for an autoencoder.

- Lower dimensionality
  - Basic [2]: the basis for most autoencoders, a symmetric feedforward ANN with a distance between the outputs and the inputs as the objective function, trainable with stochastic variants of gradient descent (SGD, AdaGrad, RMSProp, Adam...). Training can be done layer by layer in a stacked fashion.
  - Convolutional [3]: layers in this variant are convolutional, which explicitly consider a 2-dimensional structure for processing images.
  - LSTM-based [4]: models sequential data by placing Long-Short-Term Memory units as encoders and decoders.
- Regularization
  - Sparse [5]: introduces a penalty term for encodings with many activations, thus resulting in a low activation average. The Kullback-Leibler divergence can be used to attract the AE to the desired average activation value.
  - Contractive [6]: achieves local invariance to changes in many directions around the training samples by adding a penalty term based on the Frobenius norm of the Jacobian matrix of the encoder.
- Noise tolerance
  - Denoising [7]: its generated features are less sensitive to noise by training with corrupted versions of

input data.
  - Robust [8]: uses a special loss function, correntropy, in order to build more robust features. Correntropy measures the probability density that two events are equal, and is less affected by outliers than other distance metrics.

- Generative model
  - Variational [9]: applies a variational Bayesian approach to encoding, assuming a latent, unobserved random variable generates the observations. It tries to approximate the distribution of the latent variable given the data.
  - Adversarial [10]: brings adversarial learning to AEs, simultaneously training a discriminator and a generator (the encoder), each competing with the other.

AEs are also the inspiration for other, more complex, neural structures. For example, autoencoder trees which involve decision trees, dual-autoencoders which learn two latent representations, and recursive AEs which introduce more pieces of input as the model deepens.

AEs can be related to the wide range of existing feature fusion methods: in the linear case, they are equivalent to PCA when linear activations and mean squared error loss function are used. As a consequence, they can be seen as generalizations of PCA which allow nonlinearities and other objective functions. They are more easily applicable than Kernel PCA since in that case the choice of kernel highly alters the behavior of the model. Other nonlinear approaches such as Isomap and Locally Linear Embedding can be compared to the contractive AE, as they attempt to preserve the local structure of the data. Restricted Boltzmann Machines are alternatives to AEs for greedy layer-wise initialization, but their models are hard to simulate than those of AEs.

The usual purpose of AEs is performing feature fusion in order to improve classification and regression performance, and to facilitate other unsupervised tasks which can be hard in high-dimensional scenarios, such as clustering. Most applications of AEs can be summarized in the following categories:

- Classification: reducing or transforming the training data in order to achieve better performance in a classifier.
- Data compression: training AEs for specific types of data to learn efficient compressions.
- Detection of abnormal patterns: identification of discordant in- stances by analyzing generated encodings.
- Hashing: summarizing input data onto a binary vector for faster search.
- Visualization: projecting data onto 2 or 3 dimensions with an AE for graphical representation.
- Other purposes: reconstruction of deteriorated images, noise reduction in automatic speech recognition, curation of biological databases, tagging digital resources.

In [1] we propose some guidelines which can help when designing AEs for different tasks. The following is an overview of these:

- Architecture: a starting point is choosing the desired length of the encoding and the type of units for the neural structure.
- Activations and loss function: sigmoid-like functions generally work well in the encoding layer. Place linear or ReLU at the output when using mean squared error, or logistic activation when using cross-entropy error.
- Regularizations: can be used according to the desired properties in the encoding. Weight decay can prevent overfitting, sparsity can be useful in many scenarios and contraction can find a lower-dimensional manifold.

The main software libraries and frameworks which allow for the construction and training of AEs are Tensorflow, Caffe, Torch, MXNet, and Keras. Software pieces which specifically implement AEs are packages Autoencoder and SAENET of the CRAN repository, H2O and yadlt for Python.

A case study is provided in [1] with the well known MNIST handwritten digits dataset, covering several adjustable parameters such as the encoding length, the optimization algorithm, activation functions and some of the predominant AE variants.

References

[1] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, pp. 78 – 96, 2018.

[2] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4, pp. 291–294, 1988.

[3] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks and Machine Learning – ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I*, T. Honkela, W. Duch, M. Girolami, and S. Kaski, Eds. Springer Berlin Heidelberg, 2011, pp. 52–59.

[4] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *International Conference on Machine Learning*, 2015, pp. 843–852.

[5] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[6] S. Rifai, Y. Bengio, P. Vincent, and Y. N. Dauphin, "A generative process for sampling contractive auto-encoders," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Omnipress, 2012, pp. 1811–1818.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. ACM, 2008, pp. 1096–1103.

[8] Y. Qi, Y. Wang, X. Zheng, and Z. Wu, "Robust feature learning by stacked autoencoder with maximum correntropy criterion," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6716–6720.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[10] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.