Inteligencia Computacional
Master en Ingeniería Informática

Práctica 1: Redes Neuronales

Luis Alberto Segura Delgado

Miércoles 9 de Diciembre de 2015

Índice

1 Introducción

En esta práctica el objetivo es familiarizarse con técnicas de aprendizaje basadas en Redes Neuronales para resolver un problema concreto, en este caso, el reconocimiento de patrones numericos.
En clase se han visto las redes neuronales más básicas, las originales. Y a partir de estás se ha ido profundizando hasta conocer el funcionamiento y las mejoras que se han ido añadiendo para que sean capaces de resolver mejor y de forma más eficiente los problemas, resolviendo los problemas iniciales que se descubrieron en los modelos más básicos. De esta forma, ahora conocemos una técnica de aprendizaje automático más que podemos usar para aquellos problemas que se nos planteen en el futuro, siempre que aplicar redes neuronales sea una estrategia correcta para resolver el problema.

2 Implementación

El trabajo de implementación realizado ha sido principalmente desarrollar por mi mismo una red neuronal multicapa y entrenarla ajustando los diferentes parámetros (tasa de aprendizaje, neuronas de cada capa, capas ocultas..) de forma que se obtuviese la menor tasa de error en la clasificación de los ejemplos del problema propuesto.

2.1 Red Neuronal Multicapa Simple

En primer lugar se ha implementado una red neuronal multicapa sencilla. La implementación se ha realizado en un nuevo lenguaje de programación similar a C/C++ llamado *Rust*[1]. La implementación en detalle se puede ver en el código que se entrega junto a esta documentación.

La implementación es sencilla, y nos permite crear una red neuronal con el número de capas ocultas que deseemos. Simplemente tenemos que indicar el número de entradas (neuronas de entrada), el número de capas ocultas, el número de neuronas ocultas (todas las capas tendrán el mismo número de neuronas), el número de salidas de la red y la tasa de aprendizaje de la red a la hora de entrenar con BackPropagation. Automáticamente se creará una red neuronal con la topología indicada. La inicialización de los pesos es aleatoria, con valores entre -0.5 y 0.5, y el término *bias* se inicializa para todas las neuronas a 1.0.

Una vez que tenemos nuestra red neuronal creada, ésta debe poder ser ejecutada, para ellos se implementa una función *ejecutar* que simplemente realiza las operaciones de multiplicar pesos por entradas y acumularlos, y finalmente calcular la salida final de las neuronas (y) en base a la función de activación, que en este caso es la función *Sigmoide*.

$$o = \sum_{i=0}^{n} w_i x_i + b_i \tag{1}$$

$$y = \frac{1}{1 + e^{-o}} \tag{2}$$

Para cada neurona se calcula su salida aplicandole la función *Sigmoide*, haciendo propagación hacia delante, finalmente obtenemos la salida de la red para unas entrada determinadas.

Y ya solamente quedaba implementar un método para entrenar nuestra red neuronal. Este método es BackPropagation. Después de pelear mucho con las fórmulas y su implementación, finalmente parece que está implementado correctamente y funciona. El método de entrenamiento concreto es un entrenamiento *online*, es decir, los pesos de las neuronas se actualizan cada vez que le pasamos una entrada (foto de dígito) en función del error cometido al clasificar. Como sabemos, cada vez que se le pasa una entrada, se ejecuta

[1] `https://www.rust-lang.org`

la red, de forma que obtenemos la salida de la red. Se calcula el error de la clasificación y se propaga hacia atrás a todas las neuronas de la red para actualizar sus pesos en la medida en la que afectan a la salida (al error que se produce).

Una vez que tenemos esta red neuronal sencilla y su método de entrenamiento, solo queda ponerla a entrenar para empezar a obtener los primeros resultados. Tras una serie de experimentos y pruebas con diferentes configuraciones (tasa de aprendizaje, número de neuronas ocultas, etc), vimos que los errores no se reducían suficiente, solían quedar en torno al 20%, por lo que se han implementado algunas mejoras que se nos han ocurrido para tratar de reducir el error (de entrenamiento).

2.2 Mejora en BackPropagation: *BackPropagation con Refuerzo*

En primer lugar se nos ocurrió que, como nuestra red neuronal se quedaba atascada en un porcentaje de error en torno al 20% para el conjunto de entrenamiento, quizás podríamos "forzar" la red a aprender aquellos ejemplos que parecía que le costaban más. Para ello, se implementó una modificación de BackPropagation (*BackPropagation con Refuerzo*) que, en primer lugar, entrenaba una época[2]. Después de entrenar en una época solamente, se calcula la tasa de error y se guardan los ejemplos en los que se ha equivocado. Y a continuación, se entrena una época usando solamente los ejemplos en los que ha fallado. Para este nuevo entrenamiento que solamente usa los ejemplo erróneos, la tasa de error se reduce para evitar que se "olviden" los ejemplos que ya se aprendieron. El entrenamiento con los ejemplos erróneos se realiza tantas veces como se indique. De forma que, una vez que se entrena la primera vez con salidas falladas, se vuelve a calcular el error y se guardan los nuevos errores, para entrenar con los nuevos errores solamente, y en cada iteración (época de refuerzo) se reduce progresivamente la tasa de error (esto de reducir la tasa de error parece funcionar bastante bien). Además de reducir la tasa de error poco a poco, como solo pasamos una parte de los datos (los que se han predicho mal) el tiempo de entrenamiento es menor que si pasamos todos los ejemplos en todas las épocas.

Con este método de entrenamiento, nuestra red neuronal mostró mejoras, reduciendo su tasa de error en el conjunto de entrenamiento del 20% hasta un 11%. Buscando mejorar aún más la tasa de error (pues aún era alta), se implementaron nuevas ideas.

2.3 Mejora en BackPropagation: *BackPropagation Adaptativo*

Como en el BackPropagation con refuerzo parecía dar buenos resultados ir reduciendo progresivamente la tasa de aprendizaje (η) con forme avanzamos de época, surgió la idea de utilizar esta idea pero sobre todo el conjunto de entrenamiento. De esta forma, lo que se hacía era hacer un entrenamiento *online* en el que en cada época, se reducía la tasa de error, de forma que los "saltos" hacía la solución óptima fuesen más pequeños conforme nos acercamos a ésta. En todas las épocas se usaba el conjunto completo de entrenamiento (esta es la diferencia real entre la mejora anterior y esta). La reducción de la tasa de aprendizaje se fijó finalmente como sigue:

$$\eta_{e+1} = \eta_e(e * 0.1) \tag{3}$$

Con la esperanza de que los resultados mejorasen, se probaron diferentes formas de ir actualizando la tasa de aprendizaje en cada época, y la que mejor pareció funcionar fue la fórmula anterior. Aún así, los resultados no mejoraron demasiado y la tasa de error se mantuvo en torno al 11-12%.

[2]pasada de todos los datos de entrenamiento una vez (los dígitos 60000)

2.4 Generando Ruido

Otra de las cosas que se ha probado ha sido introducir ruido en los datos, pues como vimos en clase, nos puede ayudar a que nuestra red aprenda características (patrones) más generales quitando de en medio la información que realmente no es útil, y que, al haber limpiado los datos ya no se distingue de los patrones que realmente podrían interesarnos. Por ejemplo, al limpiar las imágenes, los fondos y las esquinas son todas negras, lo que puede llevar a la red a aprender patrones relacionados con las esquinas, cuando no es lo que queremos pues no es la información importante. Al introducir ruido, las esquinas variarán y obviará esos patrones (ya que no existirán).
Para introducir ruido, se elige al azar los pixels y el nivel de ruido. Si el pixel tiene un valor menor a 20, se cambia su valor con probabilidad del 2%, a un nuevo valor de entre 0 y 120. Además, el entrenamiento se hace ahora con el doble de datos, pues tenemos la imagen original y una nueva imagen (que es una copia) a la que se le ha introducido ruido.
Los resultados mejoraron, pero no demasiado, pues como se verá en los resultados, seguía en torno al 10-11%.

3 Resultados

Después de realizar todas las modificaciones comentadas y de probar diferentes parámetros para ellas, se han obtenido varios resultados, algunos de los mejores son los siguientes.[3]

Tabla 1: **Resultados**

Algoritmo	**Épocas**	**Capas Ocultas**	**Neuronas Ocultas**	η	**E.Entren.**	**E.Test**
BackPropagation Simple	3	1	200	0.1	19.95%	18.81%
BackPropagation Refuerzo	3	1	300	0.1	12.048%	10.95%
BackPropagation Refuerzo	3	1	256	0.1	12.83%	12.07%
BackPropagation Refuerzo	10	1	400	0.1	11.99%	11.24%
BackPropagation Refuerzo Con ruido	3	1	200	0.1	11.94%	11.15%
BackPropagation Refuerzo Con ruido	2	1	400	0.1	10.62%	10.1%
BackPropagation Refuerzo Con ruido	2	1	600	0.07	10.55%	9.74%
BackPropagation Refuerzo Con ruido	2	1	600	0.05	10.16%	9.6%
BackPropagation Refuerzo Con ruido	3	1	600	0.04	10.20%	9.8%
BackPropagation Adaptativo	241	0	0	0.09	8.63%	8.75%

Estos son algunos de los mejores resultados que he obtenido con mi implementación sencilla de red neuronal. El mejor de los resultados con una red con capas ocultas es aquel con el mayor número de neuronas ocultas y usando los datos con ruido generados. Cuantas más neuronas tengamos, la red neuronal será capaz de reconocer patrones más complicados. Sin embargo, el mejor resultado se ha conseguido para una red neuronal

[3]Los experimentos con ruido se realizan con el doble de datos (imagen original + copia con ruido, para cada imagen).

sin capas ocultas (solamente con las capas de entrada y salida).
Estos resultados se han conseguido después de muchos experimentos modificando los parámetros, principalmente la tasa de error, para ajustarla de forma que el error se redujese lo máximo posible.

Como podemos ver en estos resultados, la tasa de aprendizaje es relativamente alta, y podemos ver el efecto de *Early Stopping*. Como vemos, la tasa de error sobre los datos de entrenamiento es mayor que la tasa de error sobre los datos de test. Esto es uno de los efectos del Early Stopping, no sobreaprendemos los datos de entrenamiento y obtenemos una red neuronal más general, lo que permite que frente a los datos de test funcionen mejor las generalizaciones que ha realizado.

En resumen, podemos ver como, conforma vamos refinando la forma de entrenar a nuestra red, ésta, es capaz de aprender mejor los ejemplos del problema. Mezclando las dos ideas más interesantes que se nos han ocurrido conseguimos que nuestra red haya mejorado poco a poco sus resultados. Por un lado, tenemos la idea de reforzar el aprendizaje pasándole una y otra vez los ejemplos que no es capaz de aprender, y por otro lado, generamos más datos introduciendo ruido en los datos originales para que sea capaz de generalizar y aprender mejor.

Algunas ideas que se quedan sin poder realizarse por falta de tiempo son, por ejemplo, desarrollar una red neuronal más sofisticada con capas convolutivas que permitan aprender mejor y sean capaces de reconocer los patrones en diferentes lugares de una imagen. También sería interesante tratar de paralelizar la red neuronal de modo que sea más rápido el aprendizaje y de forma que en el mismo tiempo podamos entrenar más la red.

Por otro lado puede que nos preguntemos por qué no se han realizado otros experimentos, como por ejemplo, probar con más capas. Esto si que se ha hecho, se han realizado diferentes experimentos usando varias capas ocultas, principalmente 2 (pues la propagación de error hacía atrás tiende a 0 y las capas iniciales acaban por no tener sentido, pues no ajustan prácticamente nada sus pesos). Pero los resultados no han sido mejores y el tiempo necesario para entrenar la red era bastante mayor (muchas más conexiones, pesos, para ajustar). Por ello, no se han realizado tantos experimentos con varias capas ocultas, además, los experimentos con una capa oculta daban mejores resultados que los que se realizaron con varias de ellas.

4 Conclusiones

Como conclusiones finales a los experimentos realizados y a la práctica en general, podemos sacar que:

- Las redes neuronales son una técnica muy interesante para aplicar en problemas de clasificación. Como hemos visto, nos permite detectar y "aprender" patrones de forma que no sea útil para detectar esos patrones en nuevos datos que se produzcan.
- Es una técnica que no solo nos permite detectar patrones, sino que es adaptable a diferentes problemas, es decir, en cualquier problema en el que tengamos unas entradas a las cuales les corresponden unas salidas podemos aplicar redes neuronales para tratar de resolver el problema. No es dependiente del problema, no es un algoritmo que dependa de cada problema de forma específica.
- Aunque las redes neuronales con suficientes neuronas en las capas ocultas son capaces de aprender correctamente la función que representan los datos (la función del problema por la cual obtenemos una salida concreta para cada entrada), no ocurre lo mismo con las capas. Es decir, cuantas más capas, no quiere decir que mejor aprenda (al menos usando BackPropagation como algoritmo de entrenamiento). Como sabemos, conforme se propaga el error por las capas, éste se va haciendo cero, por lo que cuantas más capas tengamos menos variarán los pesos en las capas iniciales. Esto al final quiere decir que

poniendo muchas capas al final lo único que conseguiremos es aumentar el número de pesos a ajustar, pero no una mejora en la red, pues las neuronas de las capas iniciales no ajustarán (no aprenderán).

- En el aspecto práctico, esta claro que realizar tu propia implementación de una red neuronal no es lo mejor, pues pasarás horas peleando para comprender lo mejor posible el funcionamiento (principalmente del método de entrenamiento, Backpropagation) e intentando implementarlo correctamente. Lo normal, y lo mejor, es utilizar implementaciones ya hechas, que sepamos que ya están suficientemente probadas y que son correctas, además de que están optimizadas correctamente y paralelizadas. Así usamos unas herramientas que funcionan correctamente y bien, y además, nos centramos en el problema concreto que debamos resolver. Aunque en el caso de está práctica, la idea de implementar la red neuronal era enterarme bien de como funciona una red neuronal, y para eso lo mejor es implementarla tu mismo.

La práctica me ha gustado mucho, me ha parecido muy interesante. Me hubiese gustado poder mejorar más la red, como he comentado, me hubiese gustado paralelizar algunos cálculos e implementar alguna que otra mejora, pero por falta de tiempo (demasiadas prácticas) no ha sido posible. Aún así, me ha gustado la práctica y esto muy contento de haber podido aprender como funciona una red neuronal.

Para concluir, decir que el código, esta documentación y algunos resultados que se han obtenido (entre ellos los resultados expuestos en esta documentación) estarán disponibles en Github por si son de utilidad para alguien. La dirección es `https://github.com/segura2010/Rust-NeuralNet-IC` y estará disponible pocos días después de la entrega de la práctica.