

VNS con reducción del espacio de búsqueda para entrenamientos de redes neuronales*

Ignacio José García del Amo Belén Melián Batista José A. Moreno Pérez

Dept. de Estadística, I.O. y Computación

ETS Ingeniería Informática

Universidad de La Laguna

38271 La Laguna

{igdelamo,mbmelian,jamoreno}@ull.es

Resumen

El VNS (Variable Neighborhood Search) es un tipo de búsqueda metaheurística de reciente aparición basado en un cambio sistemático en la estructura de entornos. Esta metaheurística ha alcanzado éxitos notables sobre todo en problemas donde el espacio de búsqueda tiene una cierta estructura de vecindad que puede ser aprovechada en el diseño del VNS. El entrenamiento de redes neuronales es un problema que, aunque de reconocida relevancia en Inteligencia Artificial, ha sido escasamente estudiado desde el punto de vista de las metaheurísticas. En este trabajo se propone la aplicación de la metaheurística VNS al entrenamiento de redes neuronales, mostrando una sencilla implementación que permite obtener soluciones de calidad comparable a otros métodos más tradicionales, con una mejora considerable en el tiempo de entrenamiento. El método propuesto se basa en una definición de estructuras de entornos teniendo presente la topología de la red dada por las conexiones y las capas. Este procedimiento limita la exploración del espacio de pesos asociados a la estructura, acelerando así el proceso.

1. Introducción

Las redes neuronales artificiales permiten aproximar funciones no lineales de varias variables. Para ello es necesario definir la estructura de la red y ajustar una serie de parámetros asociados a los elementos de la red llamados *pesos*. Si la variable de salida es categórica se trata de un problema de clasificación y si es una variable continua es un problema de predicción o aproximación.

Dentro del enfoque del aprendizaje supervisado, una red neuronal es entrenada para una tarea de clasificación o predicción con un conjunto de datos de entrenamiento ajustando los parámetros para minimizar los errores que se producen en dicho conjunto. El conjunto de entrenamiento E está constituido por una serie de valores de entrada de la función y sus correspondientes salidas. Si la función f tiene por entradas un vector x y la salida es una variable $f(x)$, el error que comete la red neuronal con pesos w que aplicada a los valores x proporciona la predicción $p(x, w)$ es la diferencia entre $f(x)$ y $p(x, w)$. Una de las formas de medir el error que se produce en un conjunto de entrenamiento E es mediante la raíz del error cuadrático medio (RMSE) calculado como:

$$RMSE(E, w) = \sqrt{\frac{1}{|E|} \sum_{x \in E} (f(x) - p(x, w))^2}$$

Por tanto, el entrenamiento de la red se in-

*Este trabajo ha sido parcialmente financiado por los proyectos TIC2002-04242-C03-01 (70 % son fondos FEDER) y PI042004/088

terpreta como un problema de optimización no lineal consistente en minimizar dicho error de entrenamiento a través de un conjunto óptimo de pesos w^* . Es decir encontrar los pesos w^* tales que:

$$RMSE(E, w^*) = \min_w RMSE(E, w).$$

En este problema se pueden aplicar técnicas específicas y técnicas generales de optimización global como las metaheurísticas.

Sin embargo, el objetivo fundamental es la generalización, es decir minimizar el error también fuera del conjunto de entrenamiento. Por tanto, se buscan procedimientos de entrenamiento de la red capaces de extraer del conjunto de entrenamiento la información extensible al universo del que proviene descartando lo específico del mismo. Para medir la generalización del proceso de entrenamiento se utiliza otro conjunto V de valores distintos de los de E sobre los que validar la red. El rendimiento de la red se mide entonces por el error cuadrático medio en este conjunto de validación con los pesos obtenidos por el proceso de entrenamiento, llamado *error de validación* que se calcula por.

$$RMSE(V; E, w^*) = \sqrt{\frac{1}{|V|} \sum_{y \in V} (f(y) - p(w^*, y))^2}$$

El conjunto de validación se suele tomar independiente del conjunto de entrenamiento, para comprobar la capacidad de generalización de la red. Con el objeto además de detener el entrenamiento de la red antes de que empiece a sobreajustar los pesos, se suele emplear un tercer conjunto de test, T . Dicho conjunto se usa en la fase de entrenamiento para determinar cuándo la red deja de aprender las características generales de la red y empieza a memorizar el conjunto de entrenamiento. Para ello, cada cierto número de iteraciones se calcula el error del conjunto T (pero sin modificar los pesos en función de este error). Cuando el error de test alcanza un mínimo y empieza a crecer de nuevo, quiere decir que está sobreajustando.

En general, si el modelo contempla pocos parámetros será difícil ajustar la red a los datos, y si tiene demasiados y la red es suficientemente general se producirá un sobreajuste al conjunto de entrenamiento por el que se minimiza excesivamente el error de entrenamiento en detrimento del error de validación. Sin embargo, no existe un criterio bien definido que determine cuál debe ser el número de parámetros o ni siquiera la arquitectura de la red óptimos para un problema dado.

Una red neuronal para la predicción o aproximación de una función consta de una serie de nodos o neuronas de entrada (una por cada variable de la función), una neurona de salida (o varias, si la función es multidimensional) que están interconectadas a través de un conjunto variable de neuronas ocultas. Un tipo de estructura para una red neuronal muy utilizada es la red multicapa en la que, además de la capa de neuronas de entrada y la capa de neuronas de salida, las neuronas ocultas están también organizadas en capas y todos los enlaces conectan neuronas de capas diferentes.

El modelo más sencillo consiste en una red neuronal con una única capa oculta de h neuronas de forma que el conjunto de neuronas N consta de una capa de entrada N_I , una capa de salida N_O y una capa oculta N_H ; $N = N_I \uplus N_H \uplus N_O$. Además todas las conexiones van desde neuronas de la capa de entrada a neuronas de la capa oculta o desde la capa oculta a la capa de salida. Varios resultados teóricos sustentan la suficiencia de este modelo con una única capa oculta para aproximar arbitrariamente bien cualquier función real continua (*ver [1]*), aunque para ello necesita un número de neuronas ocultas suficientemente elevado (suponiendo que el tiempo de entrenamiento sea también suficientemente grande).

Dentro de una red neuronal multicapa, podemos numerar las neuronas de forma consecutiva desde la capa de entrada a la de salida. Aquí consideramos redes con una capa de entrada de n neuronas (n es el número de variables de entrada), una capa oculta de h neuronas y una única neurona de salida. Por tanto, identificaremos las neuronas de entrada con

el índice $i = 1, \dots, n$, las neurona de la capa oculta con el índice $n + j$, $j = 1, \dots, h$ y la neurona de salida es la neurona $n + h + 1$. Dado un patrón de entrada $x = (x_1, x_2, \dots, x_n)$ para las neuronas de entrada a la red, cada neurona de la capa oculta recibe una entrada de cada una de las neuronas de entrada i , $i = 1, \dots, n$, a las que está conectada y envía su salida a la neurona de salida.

En los modelos usuales, las entradas de las neuronas de las capas oculta y de salida son combinaciones lineales de los pesos asociados a los enlaces y las salidas de las neuronas de la capa anterior. Por tanto, la entrada para una neurona de la capa oculta j sería

$$x_j = w_j + \sum_{i=1}^n x_i w_{ij}; j = n + 1, \dots, n + h.$$

donde w_j es el peso asociado al *sesgo* de la capa anterior (se puede interpretar como una pseudo-neurona cuya salida vale siempre 1). Cada neurona de la capa oculta, transforma su entrada en una salida de la forma $y_j = g(x_j)$ siendo la función *sigmoide* $g(x) = 1/(1 + e^{-x})$ una de las más usadas. En cambio, para problemas de predicción, en la capa de salida se utiliza normalmente como función de activación una función lineal (por ejemplo, la propia función identidad).

2. La Metaheurística VNS

Las *metaheurísticas* son estrategias generales para diseñar procedimientos heurísticos que resuelvan un problema de optimización mediante un proceso de búsqueda en un cierto espacio de soluciones alternativas. Los procesos de búsqueda heurística están generalmente basados en transformaciones de las alternativas que determinan una estructura de entornos en el espacio de soluciones. La Búsqueda de Entorno Variable (*Variable Neighbourhood Search*, VNS) es una metaheurística propuesta sólo hace unos años (ver [6], [3] y [4]) que está basada en un principio simple: cambiar sistemáticamente de estructura de entornos dentro de la búsqueda.

Un problema de optimización consiste en encontrar, dentro de un conjunto X de soluciones

factibles, la que optimiza una función $f(x)$. Si el problema es de minimización se formula como sigue:

$$\text{mín}\{f(x)|x \in X\} \quad (1)$$

donde x representa una *solución* alternativa, f es la *función objetivo* y X es el *espacio de soluciones* factibles del problema. Una *solución óptima* x^* (o mínimo global) del problema es una solución factible donde se alcanza el mínimo de (1).

Si X es un conjunto finito pero de gran tamaño se trata de un problema de *optimización combinatoria*. Si $X = \mathbb{R}^n$, hablamos de *optimización continua*. La mayoría de los problemas de optimización que surgen en aplicaciones prácticas son NP-duros y para abordarlos se necesitan métodos de optimización heurística (al menos para instancias de gran tamaño o como solución inicial para algún procedimiento exacto). Las metaheurísticas se han mostrado como una herramienta apropiada para abordar este tipo de tareas [5].

Una *estructura de entornos* en el espacio de soluciones X es una aplicación $\mathcal{N} : X \rightarrow 2^X$ que asocia a cada solución $x \in X$ un entorno de soluciones $\mathcal{N}(x) \subset X$, que se dicen *vecinas* de x . Las metaheurísticas de búsqueda local aplican una transformación o movimiento a la solución de búsqueda y por tanto utilizan, explícita o implícitamente, una estructura de entornos. Denotemos por \mathcal{N}_k , $k = 1, \dots, k_{max}$, a un conjunto finito de estructuras de entornos en el espacio X . Los entornos \mathcal{N}_k pueden ser inducidos por una o más métricas introducidas en el espacio de soluciones X . La mayoría de las heurísticas de búsqueda local usan sólo una estructura de entornos.

Una solución $x^* \in X$ es un *mínimo global* del problema (1) si no existe una solución $x \in X$ tal que $f(x) < f(x^*)$. Decimos que $x' \in X$ es un *mínimo local* con respecto a \mathcal{N}_k , si no existe una solución $x \in \mathcal{N}_k(x') \subseteq X$ tal que $f(x) < f(x')$. Una búsqueda local descendente cambia la solución actual por otra solución mejor de su entorno, por tanto tienen el peligro de quedarse atascada en un mínimo local. Las metaheurísticas basadas en procedimientos de búsqueda local aplican distintas formas de continuar la búsqueda después de

encontrar el primer óptimo local.

La VNS está basada en tres hechos simples:

1. Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
2. Un mínimo global es mínimo local con todas las posibles estructuras de entornos.
3. Para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

Esta última observación, que es empírica, implica que los óptimos locales proporcionan información acerca del óptimo global. Puede ser, por ejemplo, que ambas soluciones tengan características comunes. Sin embargo, generalmente no se conoce cuales son esas características. Es procedente, por tanto, realizar un estudio organizado en las proximidades de este óptimo local, hasta que se encuentre uno mejor.

Los hechos 1 a 3 sugieren el empleo de varias estructura de entornos en las búsquedas locales para abordar un problema de optimización. El cambio de estructura de entornos se puede realizar de forma determinística, estocástica, o determinística y estocástica a la vez.

La *búsqueda de entorno variable básica* (*Basic Variable Neighbourhood Search*, BVNS) combina cambios determinísticos y aleatorios de estructura de entornos. Los pasos de la VNS básica se dan en la figura 1.

La condición de parada puede ser, por ejemplo, el máximo tiempo de CPU permitido, el máximo número de iteraciones, o el máximo número de iteraciones entre dos mejoras. Frecuentemente los entornos \mathcal{N}_k sucesivos están anidados. Obsérvese que la solución x' se genera al azar en el paso (2a) para evitar el ciclo, que puede ocurrir si se usa cualquier regla determinística.

3. Aplicación del VNS a las redes neuronales

Como ya se ha comentado antes, una de las características más importantes del VNS es su capacidad de explorar el espacio de soluciones de un problema restringiendo la búsqueda a

Inicialización

Seleccionar un conjunto de estructuras de entornos $\mathcal{N}_k, k = 1, \dots, k_{max}$, que se usarán en la búsqueda; encontrar una solución inicial x ; elegir una condición de parada.

Iteraciones

Repetir, hasta que se cumpla la condición de parada, la siguiente secuencia:

- (1) Hacer $k \leftarrow 1$;
- (2) Repetir, hasta que $k = k_{max}$, los pasos:
 - (a) Agitación Generar al azar una solución x' del k -ésimo entorno de x ($x' \in \mathcal{N}_k(x)$);
 - (b) Búsqueda local Aplicar algún método de búsqueda local con x' como solución inicial; denótese con x'' el mínimo local así obtenido.
 - (c) Moverse o no Si la solución obtenida x'' es mejor que x , hacer $x \leftarrow x''$ y $k \leftarrow 1$; en otro caso, hacer $k \leftarrow k + 1$.

Figura 1: VNS Básica; BVNS

un determinado entorno de las soluciones consideradas en cada iteración.

En el caso del entrenamiento de redes neuronales, el problema consiste en encontrar un conjunto de pesos w^* de la red que minimice el error de clasificación de los conjuntos de entrenamiento, test y validación. Por tanto, estamos ante un problema de optimización global que tiene un espacio de búsqueda de soluciones w -dimensional, y donde parece factible utilizar el VNS como método de optimización.

Por las características del VNS, se suele considerar una solución de un problema como un vector donde las componentes se corresponden con los valores de los parámetros a ajustar. Para aplicar esto al caso de las redes neuronales, la traducción entre *pesos* de la red y *vector de parámetros* de la solución consiste simplemente en colocar uno a continuación del otro todos los pesos que llegan a una neurona, y todo esto para todas las neuronas de la red.

Sin embargo, ya el hecho de que exista una

organización de los pesos no sólo por neuronas, sino también por capas, nos da una idea de que existe una estructura interna que podemos aprovechar para realizar la búsqueda por entornos. Por ejemplo, si una red tiene la clásica arquitectura con una única capa oculta, parece intuitivo pensar que no tendrá la misma influencia en el error de la red un peso que esté entre la capa de entrada y la oculta (cuyo valor se ve ponderado por la función de activación de la neurona oculta, y a su vez, por los pesos de la capa oculta), que un peso que conecte una neurona oculta con una de salida, cuyo valor afecta directamente en la salida de la red. Es precisamente esta idea la que nos lleva a pensar que podemos aprovechar la inherente organización de los pesos de una red para generar entornos de exploración para con el VNS.

En el experimento llevado a cabo, se ha usado la variante del BVNS para realizar una búsqueda en el espacio de soluciones del problema que permita encontrar un conjunto de pesos w^* que produzca errores comparables a los de otros métodos más tradicionales como el *Backpropagation*. Las características del BVNS utilizado en este problema son las siguientes:

- Se considera que una solución s_1 pertenece al entorno k de otra solución s_2 si se diferencia de ella sólo en un conjunto específico de parámetros (pesos). En concreto, los parámetros que se permiten variar son los pesos que conectan la capa anterior con las neuronas k y $k + 1$, dejando el resto de pesos fijos. La idea que hay detrás de este criterio es que, dejando fijos el resto de pesos, el algoritmo tratará de mejorar la contribución individual de cada neurona a la solución final. Nótese que el criterio permite variar también los pesos de la neurona $k + 1$ junto con los de la neurona k . El motivo de ello es que así se evita particionar los pesos en conjuntos disjuntos, y por tanto permite captar también la relación entre conjuntos de neuronas a la hora de contribuir a la solución final. Además, se ha añadido una condición más que

dice que según aumenta el número de iteraciones del algoritmo, se va variando también el número de neuronas que se consideran dentro del entorno k de una solución, pudiendo ser de forma cíclica $\{k+1\}$, $\{k+1, k+2\}$ o $\{k+1, k+2, k+3\}$. En principio, cualquier neurona de la capa oculta o de salida puede ser elegida, por lo que es necesario mapear k a los índices de dichas neuronas.

- Para la parte de *agitación* del BVNS, se sustituyen los pesos del entorno k de una solución por valores aleatorios dentro del rango $[-d, d]$, donde d viene determinado por:

$$d = \frac{1}{\sqrt{\text{no.entradas}}}$$

donde *no.entradas* es el número de entradas que recibe la neurona k a la que llega el peso que estamos considerando [8].

- Como método de búsqueda local, se ha utilizado el *Simplex* [9], que no utiliza ninguna información del gradiente del error, pero que sirve a los propósitos de minimización de funciones. Este método, que no debe ser confundido con el clásico algoritmo de programación lineal, se basa en la construcción de un simplex n -dimensional, es decir, un conjunto de $n+1$ puntos (soluciones) y a partir de sus evaluaciones (el valor del error de la red para cada solución) y de ciertas operaciones geométricas de expansión y contracción, busca un mínimo local. En dicho método de búsqueda local se limita la búsqueda a las soluciones que pertenezcan al entorno de una solución de partida dada. Esto reduce de forma efectiva el espacio de búsqueda, pasando de ser w -dimensional a d -dimensional, donde d es el número de conexiones con la capa anterior de las neuronas seleccionadas en el entorno de la solución inicial.

4. Experiencia computacional

Para llevar a cabo las pruebas experimentales, hemos utilizado una serie de funciones medi-

ante las cuales hemos generado 3 ficheros con 200 muestras para cada una de ellas, tomando valores para x_1 en el rango $[-100, 100]$, y para x_2 en el rango $[-10, 10]$. Estos ficheros se han usado respectivamente en las fase de entrenamiento, test y validación. Las funciones usadas para generar los datos han sido recogidas de la literatura, y son las siguientes:

1. Suma:

$$f(x_1, x_2) = x_1 + x_2.$$

2. Producto:

$$f(x_1, x_2) = x_1 * x_2.$$

3. División

$$f(x_1, x_2) = \frac{x_1}{|x_2| + 1}.$$

4. *Easom*:

$$f(x_1, x_2) = -\cos x_1 \cos x_2 \exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)).$$

5. *Shubert*:

$$f(x_1, x_2) = \left(\sum_{j=1}^5 j \cos(j+1)x_1 + j \right) \left(\sum_{j=1}^5 j \cos(j+1)x_2 + j \right).$$

6. *Schwefel*:

$$f(x_1, x_2) = 418,9829n - \left(x_1 \sin \sqrt{|x_1|} + x_2 \sin \sqrt{|x_2|} \right)$$

La implementación de la red neuronal se ha hecho con las siguientes características:

- Una única capa oculta de neuronas, cada una de las cuales usaba como función de activación una variante de la familia de las sigmoidales ajustada para ser aproximadamente lineal en el rango de entrada $[-1, 1]$ y con salidas entre $[-1, 1]$ en ese mismo rango de entrada [8].

- Los datos de entrada han sido normalizados para que tanto las entradas como las salidas tengan media 0 y desviación típica 1 [8], [7].

- Aunque los datos han sido normalizados, los errores de entrenamiento, test y clasificación se calculan en base a las diferencias entre la salida de la red y la salida de la función a modelar *sin normalizar*.

- El criterio de parada utilizado ha sido que el algoritmo realice 20 iteraciones seguidas sin disminuir el error de test.

Los resultados para el conjunto de validación obtenidos con el BVNS se han comparado con los obtenidos para el *Backpropagation* (BP) y se presentan en la figura 2. En la tabla se presentan los errores medios de validación en 10 ejecuciones, junto con su desviación media. El tiempo de ejecución del Backpropagation se limitó a 10 minutos, y en ningún caso el tiempo de ejecución del BVNS superó los 30 segundos.

Función	BP	BVNS
1	1,77 ± 1,6	1,65 ± 0,68
2	8,59 ± 3,94	46,4 ± 11,06
3	1,68 ± 0,22	6,93 ± 2,28
4	0,19 ± 0,06	0,05 ± 0,01
5	13,67 ± 0,02	22,36 ± 0,33
6	2,93 ± 0,53	20,21 ± 10,59

Figura 2: Errores de validación

5. Conclusiones

Aunque los resultados obtenidos no son mejores que los del Backpropagation, son comparables en magnitud. Hay que tener en cuenta que el punto clave del VNS a la hora de encontrar mínimos es precisamente la función de búsqueda local. En este caso se ha usado el *Simplex*, que no utiliza información del gradiente, en contraposición con el Backpropagation, que sí la utiliza. Por otra parte, los tiempos de ejecución del Backpropagation (en

algunos casos, de hasta 10 minutos de duración), contrastan con los escasos 30 segundos del VNS. El dato a resaltar de este experimento no es tanto la bondad específica de las soluciones producidas por uno u otro método (cabe esperar que el VNS producirá soluciones de mayor calidad simplemente sustituyendo el *Simplex* por otro método de minimización más sofisticado), sino la capacidad del VNS de reducir el espacio de búsqueda de las soluciones, considerando en cada iteración un subconjunto de pesos del total de la red.

Referencias

- [1] E.K. Blum, L.K. Li (1991) Approximation theory and feedforward networks. *Neural Networks* 4 (4), 511–515.
- [2] A. El-Fallalhi, R. Martí, L. Lasdon. (2005) Path Relinking and GRG for Artificial Neural Networks. *European Journal of Operational Research*, en prensa.
- [3] P. Hansen, N. Mladenovic. (2003) Variable neighbourhood search. In Fred Glover and Gary A. Kochenberger (eds.) *Handbook of Metaheuristics*, chapter 6, Kluwer, 2003.
- [4] P. Hansen, N. Mladenovic., J.A. Moreno Pérez (2003) Búsqueda de Entorno Variable. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*. Numero 19, Volumen 2, páginas 77-92.
- [5] B. Melián, J.A. Moreno Perez, J.M. Moreno-Vega, J. páginas (2003) Metaheurísticas: Una visión global. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*. Numero 19, Volumen 2, páginas 7-28.
- [6] N. Mladenović, P. Hansen (1997) Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100.
- [7] C.M. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
- [8] R.O. Duda, P.E. Hart, D.G. Stork. *Pattern Classification*, 2nd edition, Ed. Wiley-Interscience, 2001.
- [9] J. A. Nelder, R. Mead (1965) *Computer Journal*, vol. 7:308–313.