Evolución de Controladores Basados en Redes Neuronales

Javier Apolloni, Carlos Kavka y Patricia Roggero

LIDIC
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950
D5700HHW - San Luis - Argentina
Tel: 02652-420823
Fax: 02652-430224
e-mail: {javierma,ckavka,proggero}@unsl.edu.ar

Resumen

En los últimos años se han utilizado los algoritmos evolucionarios con mucho éxito para la obtención de controladores, en particular controladores representados como redes neuronales. La principal desventaja de estos métodos es que la solución obtenida no se puede adaptar a cambios en el sistema. Con el objeto de resolver este problema, se propone en este trabajo el almacenamiento de poblaciones intermedias obtenidas durante el proceso evolucionario. De esta manera, frente a cambios en el sistema bajo control, se puede iniciar una nueva evolución a partir de una población intermedia, con lo cual el tiempo necesario para obtener un nuevo controlador se reduce drásticamente.

1 Introducción

El diseño de controladores es una tarea difícil. En algunos casos los controladores pueden ser definidos a mano ajustando un conjunto de parámetros. Sin embargo, si el sistema a ser controlado, o una aproximación de él, puede ser simulado en una computadora, es conveniente usar un algoritmo para determinar automáticamente la mayor cantidad de parámetros posible.

Un ejemplo de un sistema típico que debe ser controlado es el de un tanque en el que se desea mantener un nivel constante de combustible. El sistema consta de una válvula a través de la cual se regula el flujo del combustible que ingresa al tanque y un flotante (sensor) que permite medir el nivel actual de combustible. El controlador debe determinar el grado de apertura de la válvula para mantener el nivel constante, en base a la información suministrada por el flotante. En términos más formales, debe aprender los parámetros de las ecuaciones diferenciales que controlan la relación entre la apertura de la válvula y el nivel indicado por el flotante.

Otro ejemplo es el control de temperatura de un líquido a través de un calentador eléctrico. El controlador debe aprender a mantener constante la temperatura en base a la información recibida por un sensor (termómetro) determinando el tiempo que el calentador debe permanecer encendido o apagado. Esta relación puede ser particularmente compleja si el calentador requiere tiempo para comenzar a calentar efectivamente el líquido, o si sigue aún caliente luego de que es apagado.

La principal desventaja de estos métodos es que la solución obtenida no se puede adaptar a cambios en el sistema. Por ejemplo, si la válvula de control de entrada de líquido sufre un desgaste por uso, la relación entre su apertura y el ingreso de líquido cambia, por lo que un controlador estático no funcionará adecuadamente.

En los últimos años se han propuesto muchos algoritmos para generar controladores. Algunos de los más exitosos están basados en algoritmos genéticos, principalmente basados en una técnica conocida como computación evolucionaria [5, 4, 1]. A través de esta técnica se pueden evolucionar estructuras y obtener la más adecuada para resolver una tarea dada. En particular, en muchas situaciones la estructura evolucionada es una red neuronal.

Las principales ventajas del uso de las redes neuronales como controladores son [6]: (a) las redes neuronales son resistentes al ruido, un problema que está siempre presente en el ámbito de control; (b) sus primitivas son del más bajo nivel posible y (c) las redes neuronales pueden aprender a través de entrenamiento.

Sin embargo, el uso de técnicas evolucionarias no resuelve el problema de la adaptabilidad. Con el objeto de resolver este problema, se propone en este trabajo el almacenamiento de poblaciones intermedias obtenidas durante el proceso evolucionario. De esta manera, frente a cambios en el sistema bajo control, se puede iniciar una nueva evolución a partir de una población intermedia, con lo cual el tiempo necesario para obtener un nuevo controlador se reduce drásticamente. Esta estrategia se justifica dado que los controladores de poblaciones intermedias han sido seleccionados porque logran resolver relativamente bien el problema, aunque no completamente, y es en las poblaciones intermedias en las cuales existe diversidad genética, la cual se pierde en las poblaciones finales en las que ya se ha producido la convergencia.

2 Modelo de Red RBF

El modelo de red neuronal seleccionado es RBF. Esta red consta de tres capas [3, 2]. La primera capa está compuesta por unidades de entrada en un número igual a la cantidad de entradas del controlador. La segunda capa es una capa oculta cuyas unidades no lineales están conectadas directamente a todas las unidades de entrada. La capa de salida consiste de una unidad de salida por cada salida del controlador, conectada a todas las unidades de la capa oculta.

La función de activación de las unidades de la capa oculta es la Función de Base Radial (RBF) definida por:

$$y = \sum_{i=1}^{n} e^{-\frac{(x_i - c_i)^2}{\sigma^2}} \tag{1}$$

donde n es el número de entradas, x es el vector de entrada, c y σ son parámetros de cada unidad. Esta función tiene la forma de una campana de Gauss cuyo centro está representado por el vector c y su amplitud por σ.

La función de activación de las unidades de la capa de salida es una función lineal definida por:

$$z = \sum_{i=1}^{m} w_i * y_i \tag{2}$$

donde m es el número de unidades de la capa oculta, y_i es la salida de la i-ésima unidad de la capa oculta y w_i es el peso asociado a la conexión que une la i-ésima unidad de la capa oculta con esta unidad de salida.

Este modelo de red tiene tres propiedades importantes [3]: (a) es un aproximador universal dado que puede aproximar arbitrariamente bien cualquier función continua; (b) tiene la propiedad de mejor aproximación debido a que dada una función no lineal desconocida, siempre existe una elección de coeficientes que aproxima esta función mejor que todas las otras elecciones posibles y (c) provee una solución óptima dado que minimiza una función que mide cuanto se desvía la solución de su valor real.

3 Algoritmo Evolucionario

Los algoritmos evolucionarios son una técnica de búsqueda global en la que las soluciones posibles son codificadas como strings (usualmente llamados cromosomas). Cada solución posible se evalúa en el problema específico. A través del operador de crossover se pueden combinar strings y obtener strings con mejor fitness. El operador de mutación puede ser utilizado para aplicar pequeñas perturbaciones a los strings.

Si las redes neuronales se representan como strings, el algoritmo evoluciona poblaciones de redes neuronales, evaluándolas y asignándoles un valor de fitness. Las mejores redes neuronales se combinan y nuevas redes son creadas.

Este esquema es particularmente interesante, dado que la mayoría de los algoritmos para redes neuronales sólo buscan valores adecuados para un conjunto de parámetros (usualmente llamados pesos), pero sobre una estructura de red definida a priori.

La codificación que se propone es la siguiente: un individuo Ind de tamaño m, con $m \geq 1$ se especifica formalmente como una secuencia (lista) de unidades RBF:

$$Ind = (R_1, R_2, \ldots, R_m) \tag{3}$$

Cada unidad R_i se especifica formalmente como una triupla de la siguiente forma:

$$R_i = (c_i, \sigma_i, w_i)$$

donde c_i representa el centro de la unidad RBF, σ_i es la amplitud y w_i el peso asociado.

El algoritmo evolucionario puede obtener redes RBF arbitrarias dado que esta representación codifica tanto la estructura de la red como los pesos. Se usan operadores de crossover y mutación estándar para individuos de longitud variable.

4 Un ejemplo

Esta sección presenta un ejemplo en el que se utiliza el algoritmo evolucionario descripto anteriormente, para obtener un controlador para una función f y para una función g (que es f levemente modificada). Las funciones f y g se definen de la siguiente manera:

$$f(x,y) = sin(2\pi x) + sin(2\pi y) \qquad 0 \leq x \leq 1, 0 \leq y \leq 1 \tag{4}$$

$$g(x,y) = \begin{cases} sin(2\pi x) + sin(2\pi y) & 0 \leq x \leq 0.8, 0 \leq y \leq 0.8 \\ (sin(2\pi x) + sin(2\pi y)) * 1.1 & 0.8 < x \leq 1, 0.8 < y \leq 1 \end{cases} \tag{5}$$

La figura 1 (izquierda) muestra el error con respecto al número de generaciones en una ejecución típica del algoritmo para la obtención de un controlador para la función f. Las poblaciones intermedias obtenidas en las generaciones 20 y 50 son almacenadas para su posterior uso como poblaciones iniciales para la evolución del controlador para la función g.

La figura 1 (derecha) muestra el error con respecto al número de generaciones para obtener un controlador para la función g partiendo de: (a) una población aleatoria, (b) la población intermedia obtenida en la generación 20 y (c) la población intermedia obtenida en la generación 50.

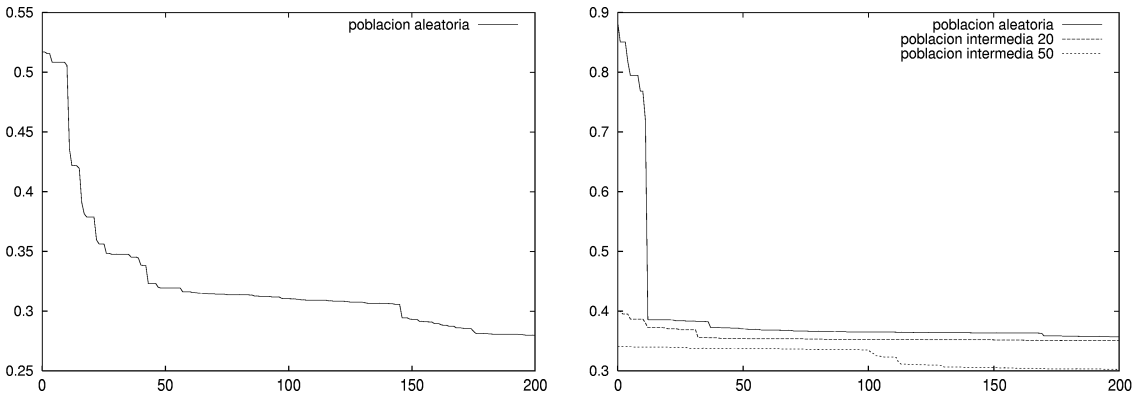

Figura 1: Error con respecto al número de generaciones para la función f (izquierda) y para la función g (derecha) partiendo de: (a) una población aleatoria, (b) una población intermedia obtenida en la generación 20 y (c) una población intermedia obtenida en la generación 50.

5 Conclusiones

El algoritmo evolucionario propuesto permite obtener controladores basados en redes neuronales. Para resolver el problema de cambios en los sistemas a ser controlados se propone el almacenamiento de poblaciones intermedias.

El gráfico de la derecha en la figura 1 muestra que el algoritmo evolucionario puede obtener más rápidamente un controlador, para una función levemente modificada, partiendo de poblaciones intermedias (obtenidas durante la evolución del controlador para la función original).

Estas poblaciones intermedias pueden ser almacenadas durante el proceso evolucionario sin incrementar la complejidad del algoritmo.

Actualmente se está trabajando en la aplicación de este concepto a problemas de control como los mencionados en la sección 1.

Referencias

[1] Wu Geng feng Carlos Kavka, María Liz Crespo and Fu Zhong quian. Fuzzy systems generation through symbiotic evolution. In *Symposium on Engineering of Intelligent Systems*, February 1998.

[2] Emile Fiesler and Russel Beale, editors. *Handbook of Neural Computation*. Institute of Physics Publishing, 1997.

[3] Simon Haykin. *Neural Networks, A Comprehensive Foundation*. Macmillan College Publishing Company, 1994.

[4] Risto Miikulainen and David Moriarty. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, (22):11–33, 1996.

[5] Risto Miikulainen and David Moriarty. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, (5), 1998.

[6] Stefano Nolfi. Evolving non-trivial behaviors on real robots: a garbage collecting robot. Technical Report 96-04, Institute of Psychology C.N.R. - Rome, April 1996.