Informe Proyecto Algoritmo Genético

Víctor Vaquero Martínez

14 de diciembre de 2018

Resumen

Análisis, propuesta e implementación de un algoritmo genético, para crear un controlador de un doble péndulo invertido a través de una red neuronal. Se demostrara el funcionamiento básico de este tipo de algoritmos, sus ventajas e inconvenientes junto con una visualización de la solución final.

1. Introducción

El problema elegido es un clásico de estudio en la teoría de control con múltiples papeles desarrollando tanto su modelado como algoritmos para su control optimo [1]. El sistema consiste en tres objetos diferenciados, dos barras y una base móvil. Ambas barras están conectadas entre si y una de ellas a su vez a la base en un punto de pivotaje.

Este es un sistema caótico por lo que su comportamiento es muy sensible a pequeñas variaciones en las condiciones iniciales. Es además un sistema no lineal lo que dificulta la creación de un controlador. Este ha de, moviendo la base en una de las dos direcciones permitidas, mantener ambas barras en posición vertical con el mínimo movimiento posible.

1.1. Estructura

A partir de esta sección se dará una breve descripción de la simulación del problema llevada a cabo. Luego se explicara el algoritmo elegido en detalle mediante ejemplos ilustrativos y se presentara la implementación de este. Por ultimo se analizara el tiempo de ejecución de dicha implementación mediante un análisis del orden de cada iteración además del coste total del algoritmo a través de una aproximación numérica.

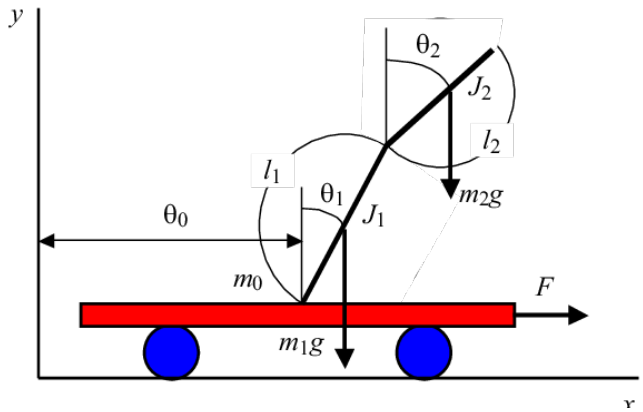


Figura 1: Esquema del sistema

2. Simulación

Llevamos a cabo una simplificación del problema al tratarlo como 2D y además evitando tener en cuenta el costoso calculo de colisiones.

Usaremos el desarrollo analítico de el papel [2] como base. Gracias a eso obtenemos las ecuaciones diferenciales que gobiernan nuestro sistema, dado un vector $y := \begin{bmatrix} q & \theta_1 & \theta_2 \end{bmatrix}^T$ de entrada, con q posición horizontal de la base y θ_1 y θ_2 como ángulos de la primera y segunda barra respectivamente; masas m_1, m_2 y m_3; longitudes de ambas barras l_1 y l_2; amortiguamiento d y fuerza sobre la base u obtenemos:

$$\underbrace{\begin{bmatrix} m+m_1+m_2 & l_1(m_1+m_2)cos\theta_1 & m_2 l_2 cos\theta_2 \\ l_1(m_1+m_2)cos\theta_1 & l_1^2(m_1+m_2) & l_1 l_2 m_2 cos(\theta_1-\theta_2) \\ l_2 m_2 cos\theta_2 & l_1 l_2 m_2 cos(\theta_1-\theta_2) & l_2^2 m_2 \end{bmatrix}}_{=:M(y)} \underbrace{\begin{bmatrix} \ddot{q} \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}}_{=\ddot{y}}$$
$$= \underbrace{\begin{bmatrix} l_1(m_1+m_2)(\dot{\theta}_1)^2 sin\theta_1 + m_2 l_2 (\dot{\theta}_2)^2 sin\theta_2 \\ -l_1 l_2 m_2 (\dot{\theta}_2)^2 sin(\theta_1-\theta_2) + g(m_1+m_2) l_1 sin\theta_1 \\ l_1 l_2 m_2 (\dot{\theta}_1)^2 sin(\theta_1-\theta_2) + g l_2 m_2 sin\theta_2 \end{bmatrix} - \begin{bmatrix} d\dot{q} \\ d\dot{\theta}_1 \\ d\dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} u \\ 0 \\ 0 \end{bmatrix}}_{=:f(y,\dot{y},u)} \quad (1)$$

Ahora tomando $x = \begin{bmatrix} y \\ \dot{y} \end{bmatrix}$ obtenemos

$$\dot{x} = \frac{d}{dt}\begin{bmatrix} y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ M^{-1}(y)f(y,\dot{y},u) \end{bmatrix} \tag{2}$$

ya preparado para pasarlo a nuestro algoritmo de resolución numérica, en este caso elegimos el método clásico Runge-Kutta[3] por ser sencillo y rápido de aplicar.

Este se basa en una aproximación de la derivada en 4 puntos diferentes, al comienzo del intervalo, en el punto medio, en el punto medio pero usando la aproximación previa y en el final del intervalo usando la previa (penúltima) aproximación:

$$\begin{aligned} \vec{a}_n &= \vec{f}(\vec{x}_n) \\ \vec{b}_n &= \vec{f}(\vec{x}_n + \frac{h}{2}\vec{a}_n) \\ \vec{c}_n &= \vec{f}(\vec{x}_n + \frac{h}{2}\vec{b}_n) \\ \vec{d}_n &= \vec{f}(\vec{x}_n + h\vec{c}_n) \\ \vec{x}_{n+1} &= \vec{x}_n + \frac{h}{6}(\vec{a}_n + 2\vec{b}_n + 2\vec{c}_n + \vec{d}_n) \end{aligned} \tag{3}$$

3. Algoritmos Genéticos[4]

Estos se basan en realizar una búsqueda en el espacio de soluciones posibles (en nuestro caso, los posibles pesos de una red) a través de una aproximación de la evolución biológica. Es decir, evolucionan un conjunto base de individuos, usualmente aleatorio, mediante sucesivas iteraciones (o generaciones) de mutaciones y recombinaciones genéticas entre individuos.

Ademas de la creación de nuevas soluciones al problema, existe una función de adaptación al medio por el que se seleccionan a las muestras mas prominentes que se suponen con una mayor probabilidad de tener una descendencia mas apta. Si esto se repite durante cientos o a veces miles de generaciones se obtiene (no siempre) la solución optima global al problema.

La clave de este sistema esta en la creación de una población con la mayor variabilidad posible para evitar caer en un optimo local.

3.1. Redes neuronales

En este caso como hemos comentado previamente implementaremos este algoritmo sobre una red neuronal[5]. Estas son funciones matemáticas que relacionan una o múltiples entradas con una o múltiples salidas a través de patrones

de interconexión entre ellas. Cada neurona es a su vez una función con múltiples entradas de las previas neuronas que se multiplican por unos pesos, se suman por un sesgo y por ultimo se pasan por una función no lineal como un función sigmoidea $f(x) := \frac{1}{1+e^{-x}}$ o una RELU $f(x) = max(0, x)$.

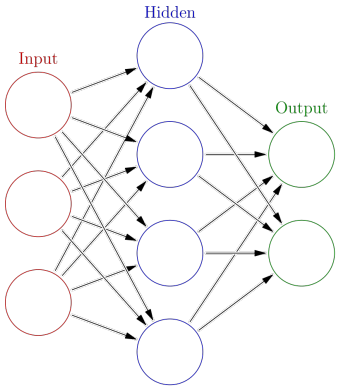


Figura 2: Esquema de una red neuronal

Para este problema en particular hemos elegido la función $tan^{-1}(x)$ ya que las mas clásicas funciones usadas tienen rangos entre 0 y 1 y nosotros necesitamos que al menos la función de salida tenga valores tanto positivos, movimiento a la derecha, como negativos, movimiento a la izquierda.

Además seleccionamos por ensayo un tamaño de red de dos capas ocultas para una estructura final de red de 6 entradas, 20 neuronas primera capa, 20 neuronas segunda capa y colapsamos a una sola salida de nuevo a través de la función $tan^{-1}(x)$. La entrada corresponde al vector de estado y derivadas actuales del sistema y la salida se corresponde con la fuerza elegida u a la que se someterá a la base en la próximo instante t.

3.2. Recombinación y Mutación

El primer paso para obtener un nuevo conjunto de soluciones es la recombinación entre pares de redes. En nuestro caso consistirá en coger para la red «hija» el peso del «padre» con probabilidad α_1 y el peso de la «madre» con probabilidad $(1 - \alpha_1)$ para cada valor o gen de cada neurona.

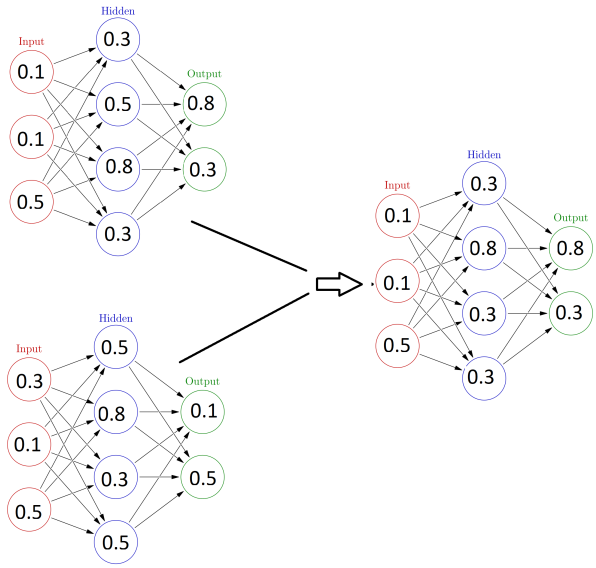
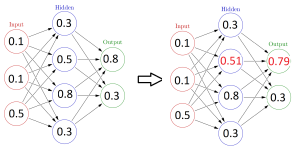
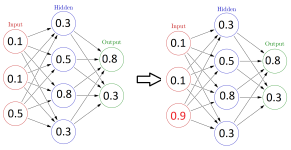


Figura 3: Combinación de dos neuronas diferentes

Una vez hecho esto, se mutara la nueva red de dos formas diferentes, con probabilidad α_2 se modificara ligeramente cada peso de manera lineal y con probabilidad α_3 se tomara un nuevo valor de una distribución normal.



(a) Mutación lineal



(b) Mutación con peso aleatorio

3.3. Implementación y pseudocódigo

Para comenzar, creamos un conjunto aleatorio de redes neuronales con pesos y sesgos inicializados a través de una normal 0 1. Con esto pasamos al principal

bucle de nuestro programa, el entrenamiento. Este toma toda la población actual y la evaluá contra nuestra medida de adaptación, en este caso se ha elegido el tiempo total de duración de la red en la simulación con la condición de cese de mantener ambos extremos de las barras por encima de la linea horizontal de la base.

Algorithm 1 Algoritmo Genético

Entrada: P - Tamaño Población, S - N^0 Supervivientes, C - Numero de Ciclos
Salida: Red

1: **procedure** ENTRENAMIENTO(P, C) ▷ $\mathcal{O}(C * P \log P)$
2: $redes \leftarrow nuevaPoblacion(P)$ ▷ $\mathcal{O}(P)$
3: $maxPunt \leftarrow -\infty$
4: **for** $i \leftarrow 1, C$ **do** ▷ $\mathcal{O}(C * P \log P)$
5: $puntos \leftarrow evaluar(redes)$ ▷ $\mathcal{O}(P)$
6: $supervivientes \leftarrow getSurvivientes(redes, S, puntos)$ ▷ $\mathcal{O}(P \log P)$
7: $redes \leftarrow reproducir(supervivientes, P)$ ▷ $\mathcal{O}(P)$
8: **if** $max(puntos) > maxPunt$ **then**
9: $maxPunt \leftarrow max(puntos)$ ▷ $\mathcal{O}(1)$
10: $mejorRed \leftarrow mejor(redes)$ ▷ $\mathcal{O}(1)$
11: **end if**
12: **if** $maxPunt$ no mejora en X pasos **then**
13: $redes \leftarrow nuevaPoblacion(P)$ ▷ $\mathcal{O}(P)$
14: **end if**
15: **return** $maxPunt, mejorRed$
16: **end for**
17: **end procedure**

Una vez se han evaluado todas con las mismas condiciones iniciales, se eliminan a las menos aptas y se seleccionan un numero S de supervivientes inferior a P. Estos se reproducen entre si y su descendencia se muta. Esta descendencia y los supervivientes de la previa generación constituyen la nueva población de nuestro algoritmo. Esto ultimo se realiza para evitar que en una generación dada el máximo rendimiento baje.

Algorithm 2 Algoritmo Genético

Entrada: S - Supervivientes, P - Tamaño población, $S < P$
Salida: Nuevas redes

```
1: procedure REPRODUCIR(S)                                   ▷ O(P)
2:     redes ← []
3:     while redes.size() < P do                             ▷ O(P)
4:         for each r1 ∈ S do                                ▷ O(min{S^2, P})
5:             for each r2 ∈ S do                            ▷ O(S)
6:                 if r1 ≠ r2 then
7:                     red ← combina(r1, r2)                 ▷ O(1)
8:                     red ← mutar(red)                      ▷ O(1)
9:                     redes.push(red)                       ▷ O(1)
10:                end if
11:            end for
12:        end for
13:    end while
14:    return redes
15: end procedure
```

Debido a que es posible que el algoritmo se estanque, si no ha habido mejoras durante X generaciones se eliminan todas excepto un numero limitado de redes tomadas de entre las mejores de la población y se crean un nuevo conjunto totalmente aleatorio de redes.

Como ultima adicción, para no perder el total de la variabilidad de las redes se permite que pasen un numero fijo de redes de entre el grupo restante.

Como se observa tanto en el código 1 como en el 2, el tiempo de ejecución de nuestro algoritmo es de $\mathcal{O}(C * P \log P)$ teniendo en cuenta que el tiempo de ejecución de la simulación para cada red es constante, pues el tamaño de cada red esta fijado, y que lo mismo pasa tanto para la combinación como para la mutación de una red.

3.4. Coste numérico

Hemos elegido, por ensayo y error, diferentes parámetros para cada constante necesaria para el funcionamiento tanto de la simulación como del algoritmo. Si bien los de la simulación son un tanto arbitrarios, pues diferentes valores dan diferentes simulaciones sobre las que se podría aplicar el algoritmo, los parámetros del algoritmo como la probabilidad de mutación, de recombinación y el numero de supervivientes son muy sensibles a cambios y pequeñas modificaciones dan resultados muy dispares.

Las constantes finalmente elegidas son: población de tamaño 100, numero de supervivientes por generación 10, probabilidad de mutación lineal 0,8 muy alta para fomentar la búsqueda de mejores soluciones de manera local; probabilidad

de heredar los parámetros de un padre u otro 0,4, aunque se ha probado con valores mas extremos como 0,3 o 0,2. Esto facilita la creación de nuevas redes con soluciones muy similares a las del optimo de la generación previa, pues es muy probable que hereden gran parte o la totalidad de sus parámetros de un solo padre. Por ultimo mantenemos a 5 «perdedores» y si el máximo se estanca durante 30 generaciones se reinician todos excepto los supervivientes.

Para calcular el coste hemos llevado a cabo repetidas iteraciones del algoritmo con diferentes tamaños de población con numero de supervivientes $0{,}1Poblacion$ para obtener una media de generaciones necesarias para llegar al optimo. Por desgracia hemos encontrado que cualquier valor, dadas las anteriores constantes, alejado del valor escogido no permite llegar a ningún optimo. En el caso de Poblaciones de tamaño 10, 25 y 50 no se alcanza ninguna solución, durando de media entre 200 y 300 iteraciones de la simulación antes de caer. Lo contrario pasa con tamaños de población 500 y 1000, que se mejora pero con demasiada lentitud, pasando mas de 400 generaciones sin mejoras notables.

Creemos que esto se debe a que todos los parámetros del algoritmo han sido optimizados con un tamaño de población en mente y por ende al modificarlo seria necesario volver a ajustarlos.

Referencias

[1] *Alexander Bogdanov* Optimal Control of a Double Inverted Pendulum on a Cart December 2004

[2] Equations of motion for an inverted double pendulum on a cart (in generalized coordinates)

[3] Método Runge-Kutta para resolución numerica de ecuaciones diferenciales `https://es.wikipedia.org/wiki/M\%C3\%A9todo_de_Runge-Kutta`

[4] Algoritmos genéticos, metodología y variantes `https://es.wikipedia.org/wiki/Algoritmo_gen\%C3\%A9tico`

[5] Redes neuronales, enlaces y funciones de activación `https://es.wikipedia.org/wiki/Red_neuronal_artificial`