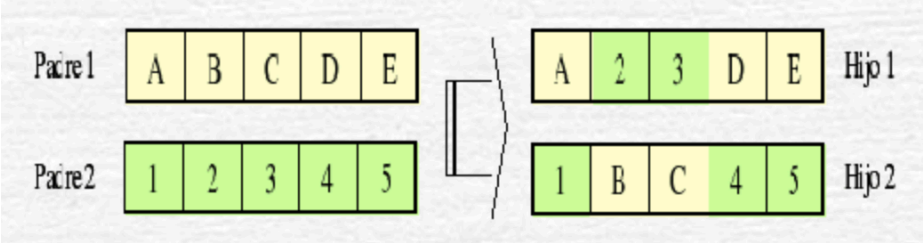
Inteligencia Artificial
Grado en Ingeniería Informática en Sistemas de Información - Curso 2017/18
MODIFICACIÓN TRABAJO DE LA ASIGNATURA- 12/01/2018

IMPORTANTE: Descarga del Aula Virtual el fichero MaterialExamen.zip necesario para el desarrollo del examen y el código del trabajo entregado (puede usar su código propio enviado a través del aula virtual o el código puesto a su disposición en el fichero MaterialExamen.zip). Debe entregar a través de la actividad habilitada para el examen un fichero llamado TusApellidos.zip que contenga los ficheros fuentes .java que hayáis creado o modificado.

Modificación A (1 punto)

Modifique el algoritmo genético para que:

1) La población inicial este formada por 16 individuos en lugar de 12.
2) La población se reproduzca por defecto realizando un cruce en dos puntos aleatorios, tal como se muestra en la siguiente figura.



3) La nueva población contenga al mejor individuo, una mutación de este, 6 individuos seleccionados con el método de la ruleta para ser padres y sus 6 hijos frutos de la reproducción, y dos mutaciones seleccionadas al azar entre los 6 padres y los 6 hijos generados, teniendo de nuevo una población de 16 individuos.

Modificación B (2,5 puntos)

En el juego dino implementado en el trabajo el dinosaurio solo tiene como posibles acciones: agachado (valor 0) y salto (valor 1). Sin embargo, en el juego real de Google el dinosaurio puede caminar erguido (valor 0), puede saltar (valor 1) y puede caminar agachado (valor 2), dando lugar a un problema de clasificación multiclase con 3 clases. Se pide hacer las modificaciones necesarias para que se consideren las tres posibles acciones del dinosaurio.

El fichero `dinoMultiClase.csv` disponible en el material del examen contiene un conjunto de entrenamiento con lecturas realizadas en el juego. La primera columna se corresponde con la distancia a la que se encuentra el obstáculo, la segunda columna indica el tamaño del obstáculo, la tercera la velocidad a la que corre el dinosaurio y la última indica la acción (0 erguido, 1 salto, 2 agachado).

Tenga en cuenta que:

1) La red neuronal tendrá ahora 3 neuronas en la capa de salida, lo que modifica el número de pesos que relacionan la capa oculta con la capa de salida.
2) Para calcular el fitness debe tener en cuenta que una vez aplicado el proceso forward se debe obtener una matriz con 3 columnas, conteniendo cada columna la probabilidad de que el dinosaurio camine erguido, salte o camine agachado, respectivamente. Para calcular la predicción se debe elegir la clase que da lugar a una probabilidad máxima como se muestra en el ejemplo a continuación:

Resultado proceso forward				**Predicción**
0.7	0.2	**0.8**	→	2
0.4	**0.6**	0.2	→	1
…				
…				
0.6	0.2	0.4	→	0

```
public class GeneticNNDemo {
…
for (int i = 0; i < 16; i++) {
   population.add(fitnessFunction.generateRandomIndividual(_pesosSize));
}

public class GeneticNN {
…
protected Individual<A> reproduce2(Individual<A> x, Individual<A> y) {

   int c1 = randomOffset(individualLength);
   int c2 = randomOffset(individualLength);

   if (c2 < c1) {
      int aux = c2;
      c2 = c1;
      c1 = c2;
   }

   List<A> childRepresentation = new ArrayList<A>();
   childRepresentation.addAll(x.getRepresentation().subList(0, c1));
   childRepresentation.addAll(y.getRepresentation().subList(c1,c2));
   childRepresentation.addAll(x.getRepresentation().subList(c2,
     individualLength));

   Individual<A> child = new Individual<A>(childRepresentation);
   return child;
}

    protected Individual<A> nextGeneration(Set<Individual<A>>
population, FitnessFunction<A> fitnessFn) {

        Set<Individual<A>> newPopulation = new HashSet<Individual<A>>();
        List<Individual<A>> populationAsList = new
ArrayList<Individual<A>>(population);

        //6 padres método ruleta
        Individual<A> x = randomSelection(populationAsList, fitnessFn);
        Individual<A> y = randomSelection(populationAsList, fitnessFn);
        Individual<A> z = randomSelection(populationAsList, fitnessFn);
        Individual<A> t = randomSelection(populationAsList, fitnessFn);
        Individual<A> h = randomSelection(populationAsList, fitnessFn);
        Individual<A> j = randomSelection(populationAsList, fitnessFn);

        //6 hijos de esos padres
        Individual<A> child = reproduce(x, y);
        Individual<A> child2 = reproduce(z, t);
        Individual<A> child3 = reproduce(h, j);
        Individual<A> child4 = reproduce(x, z);
        Individual<A> child5 = reproduce(y, h);
        Individual<A> child6 = reproduce(z, j);

        List<Individual<A>> aux = new ArrayList<Individual<A>>();
```

```
        aux.add(x);
        aux.add(y);
        aux.add(z);
        aux.add(t);
        aux.add(h);
        aux.add(j);
        aux.add(child);
        aux.add(child2);
        aux.add(child3);
        aux.add(child4);
        aux.add(child5);
        aux.add(child6);

        //Añadimos 6 Padres
        newPopulation.add(x);
        newPopulation.add(y);
        newPopulation.add(z);
        newPopulation.add(t);
        newPopulation.add(h);
        newPopulation.add(j);

        //Añadimos 6 hijos
        newPopulation.add(child);
        newPopulation.add(child2);
        newPopulation.add(child3);
        newPopulation.add(child4);
        newPopulation.add(child5);
        newPopulation.add(child6);

        //Añadimos mejor individuo
        newPopulation.add(retrieveBestIndividual(population,
fitnessFn));

        //Añadimos mutación del mejor individuo
        newPopulation.add(mutate(retrieveBestIndividual(population,
fitnessFn)));

        //Añadimos 2 mutaciones entre los 6 padres y los 6 hijos
        newPopulation.add(mutate(aux.get((int) Math.floor(Math.random()
* aux.size()))));
        newPopulation.add(mutate(aux.get((int) Math.floor(Math.random()
* aux.size()))));

        aux.removeAll(aux);
        population.clear();
        population.addAll(newPopulation);
        return retrieveBestIndividual(population, fitnessFn);

        /* CODE END */
    }
```

```
public class GeneticNNDemo {
…
private static final int _pesosSize =
(((_inputLS+1)*_hiddenLS)+(_outputLS*(_hiddenLS+1)));
…
}

public double exactitud(double[][] theta1, double[][] theta2) {
        int m = datosX.size();
        double[][] matrizH1 = new double[m][_hiddenLS];
        double[][] vectorH2 = new double[m][_numLabels];
        double[] vectorPrediccion = new double[m];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < _hiddenLS; j++) {
                matrizH1[i][j] = theta1[j][0];
                for (int k = 1; k < _inputLS + 1; k++) {
                    matrizH1[i][j] += datosX.get(i).get(k - 1) *
theta1[j][k];
                }
            }
        }
        matrizH1 = sigmoide(matrizH1);

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < _numLabels; j++) {
                vectorH2[i][j] = theta2[j][0];
                for (int k = 1; k < _hiddenLS + 1; k++) {
                    vectorH2[i][j] += matrizH1[i][k - 1] * theta2[j][k];
                }
            }
        }
        vectorH2 = sigmoide(vectorH2);
        for (int i = 0; i < m; i++) {
            int posmax = 0;
            for(int j=1;j<_numLabels;j++){
                if(vectorH2[i][j]>=vectorH2[i][posmax]){
                    posmax=j;
                }
            }
            vectorPrediccion[i] = posmax;
        }
        int contador = 0;
        for (int i = 0; i < m; i++) {
            if (vectorPrediccion[i] == datosY.get(i).doubleValue()) {

                contador++;
            }
        }
        System.out.println(((1.0 * contador / m) * 100));
        return ((1.0 * contador / m) * 100);
    }
```