greerviau / SnakeAI

Code Issues 6 Pull requests Actions Projects Wiki

master

SnakeAI / SnakeAI / NeuralNet.pde

greerviau Redesigned neuralnet architecture to be scalable, now can designate n... … History

1 contributor

Raw Blame

179 lines (151 sloc) 5.35 KB

```
class NeuralNet {

  int iNodes, hNodes, oNodes, hLayers;
  Matrix[] weights;

  NeuralNet(int input, int hidden, int output, int hiddenLayers) {
    iNodes = input;
    hNodes = hidden;
    oNodes = output;
    hLayers = hiddenLayers;

    weights = new Matrix[hLayers+1];
    weights[0] = new Matrix(hNodes, iNodes+1);
    for(int i=1; i<hLayers; i++) {
       weights[i] = new Matrix(hNodes,hNodes+1);
    }
    weights[weights.length-1] = new Matrix(oNodes,hNodes+1);

    for(Matrix w : weights) {
       w.randomize();
    }
  }

  void mutate(float mr) {
     for(Matrix w : weights) {
        w.mutate(mr);
     }
  }

  float[] output(float[] inputsArr) {
     Matrix inputs = weights[0].singleColumnMatrixFromArray(inputsArr);

     Matrix curr_bias = inputs.addBias();

     for(int i=0; i<hLayers; i++) {
        Matrix hidden_ip = weights[i].dot(curr_bias);
        Matrix hidden_op = hidden_ip.activate();
        curr_bias = hidden_op.addBias();
     }

     Matrix output_ip = weights[weights.length-1].dot(curr_bias);
     Matrix output = output_ip.activate();

     return output.toArray();
  }

  NeuralNet crossover(NeuralNet partner) {
     NeuralNet child = new NeuralNet(iNodes,hNodes,oNodes,hLayers);
     for(int i=0; i<weights.length; i++) {
        child.weights[i] = weights[i].crossover(partner.weights[i]);
     }
     return child;
  }

  NeuralNet clone() {
     NeuralNet clone = new NeuralNet(iNodes,hNodes,oNodes,hLayers);

     for(int i=0; i<weights.length; i++) {
        clone.weights[i] = weights[i].clone();
     }

```

```
      return clone;
   }

   void load(Matrix[] weight) {
       for(int i=0; i<weights.length; i++) {
          weights[i] = weight[i];
       }
   }

   Matrix[] pull() {
      Matrix[] model = weights.clone();
      return model;
   }

   void show(float x, float y, float w, float h, float[] vision, float[] decision) {
      float space = 5;
      float nSize = (h - (space*(iNodes-2))) / iNodes;
      float nSpace = (w - (weights.length*nSize)) / weights.length;
      float hBuff = (h - (space*(hNodes-1)) - (nSize*hNodes))/2;
      float oBuff = (h - (space*(oNodes-1)) - (nSize*oNodes))/2;

      int maxIndex = 0;
      for(int i = 1; i < decision.length; i++) {
         if(decision[i] > decision[maxIndex]) {
            maxIndex = i;
         }
      }

      int lc = 0;  //Layer Count

      //DRAW NODES
      for(int i = 0; i < iNodes; i++) {  //DRAW INPUTS
          if(vision[i] != 0) {
            fill(0,255,0);
          } else {
            fill(255);
          }
          stroke(0);
          ellipseMode(CORNER);
          ellipse(x,y+(i*(nSize+space)),nSize,nSize);
          textSize(nSize/2);
          textAlign(CENTER,CENTER);
          fill(0);
          text(i,x+(nSize/2),y+(nSize/2)+(i*(nSize+space)));

      }

      lc++;

      for(int a = 0; a < hLayers; a++) {
        for(int i = 0; i < hNodes; i++) {  //DRAW HIDDEN
            fill(255);
            stroke(0);
            ellipseMode(CORNER);
            ellipse(x+(lc*nSize)+(lc*nSpace),y+hBuff+(i*(nSize+space)),nSize,nSize);
        }
        lc++;
      }

      for(int i = 0; i < oNodes; i++) {  //DRAW OUTPUTS
          if(i == maxIndex) {
            fill(0,255,0);
          } else {
            fill(255);
          }
          stroke(0);
          ellipseMode(CORNER);
          ellipse(x+(lc*nSpace)+(lc*nSize),y+oBuff+(i*(nSize+space)),nSize,nSize);
      }

      lc = 1;

      //DRAW WEIGHTS
      for(int i = 0; i < weights[0].rows; i++) {  //INPUT TO HIDDEN
         for(int j = 0; j < weights[0].cols-1; j++) {
             if(weights[0].matrix[i][j] < 0) {
                stroke(255,0,0);
             } else {
                stroke(0,0,255);
             }
             line(x+nSize,y+(nSize/2)+(j*(space+nSize)),x+nSize+nSpace,y+hBuff+(nSize/2)+(i*(space+
         }
      }

      lc++;

      for(int a = 1; a < hLayers; a++) {
        for(int i = 0; i < weights[a].rows; i++) {  //HIDDEN TO HIDDEN
           for(int j = 0; j < weights[a].cols-1; j++) {
               if(weights[a].matrix[i][j] < 0) {
                  stroke(255,0,0);
               } else {
                  stroke(0,0,255);

               }
               line(x+(lc*nSize)+((lc-1)*nSpace),y+hBuff+(nSize/2)+(j*(space+nSize)),x+(lc*nSize)+(
           }
        }
```

```
          lc++;
       }

       for(int i = 0; i < weights[weights.length-1].rows; i++) {  //HIDDEN TO OUTPUT
          for(int j = 0; j < weights[weights.length-1].cols-1; j++) {
              if(weights[weights.length-1].matrix[i][j] < 0) {
                 stroke(255,0,0);
              } else {
                 stroke(0,0,255);
              }
              line(x+(lc*nSize)+((lc-1)*nSpace),y+hBuff+(nSize/2)+(j*(space+nSize)),x+(lc*nSize)+(lc
          }
       }

       fill(0);
       textSize(15);
       textAlign(CENTER,CENTER);
       text("U",x+(lc*nSize)+(lc*nSpace)+nSize/2,y+oBuff+(nSize/2));
       text("D",x+(lc*nSize)+(lc*nSpace)+nSize/2,y+oBuff+space+nSize+(nSize/2));
       text("L",x+(lc*nSize)+(lc*nSpace)+nSize/2,y+oBuff+(2*space)+(2*nSize)+(nSize/2));
       text("R",x+(lc*nSize)+(lc*nSpace)+nSize/2,y+oBuff+(3*space)+(3*nSize)+(nSize/2));
    }
 }
```