Υπολογιστική Φυσική Στερεάς Κατάστασης
Neural Networks

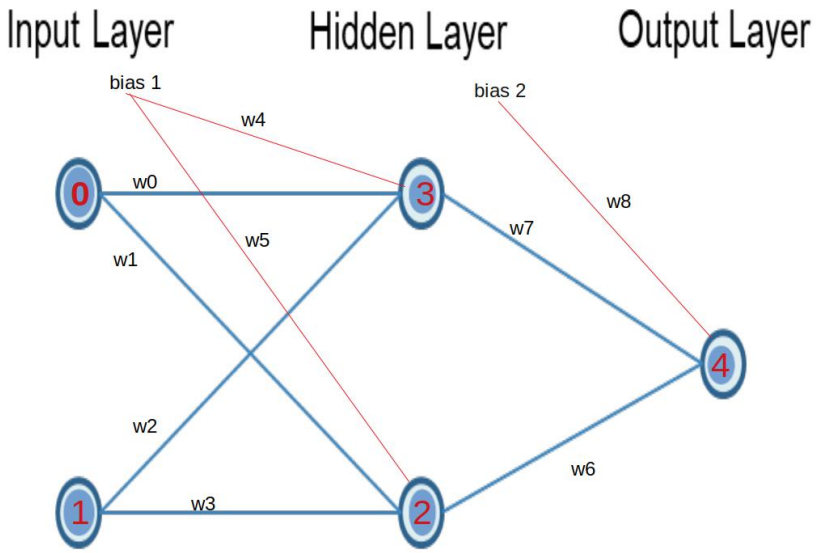
Σεϊτανίδου Δήμητρα

21 Ιουλίου 2020

Νευρωνικά δίκτυα

Σε αυτή την άσκηση θα δημιουργήσουμε ένα τεχνητό νευρωνικό δίκτυο (artificial neural network). Στόχος μας είναι να εκπαιδεύσουμε το δίκτυο έτσι ώστε όταν δίνουμε στην είσοδο συγκεκριμένες τιμές να παίρνουμε το επιθυμητό αποτέλεσμα στην έξοδο. Το δίκτυο θα αποτελείται από τρία επίπεδα, το επίπεδο εισόδου, το επίπεδο εξόδου και ένα κρυφό ενδιάμεσο επίπεδο. Το επίπεδο εισόδου και το κρυφό έχουν από δύο κόμβους το καθένα ενώ το επίπεδο εξόδου έχει έναν. Ο κάθε κόμβος του επιπέδου εισόδου είναι συνδεδεμένος με καθένα από τους κόμβους του κρυφού επιπέδου και αυτοί αντίστοιχα είναι συνδεδεμένοι με τον κόμβο στο επίπεδο εξόδου. Επίσης οι κόμβοι του κρυφού επιπέδου και του εξόδου συνδέονται με κόμβο εκτός επιπέδων που ονομάζεται bias. Οι συνδέσεις δεν έχουν κατεύθυνση έχουν όμως βάρη και αυτά τα βάρη είναι αυτά που πρέπει να υπολογίσουμε για να μπορέσουμε να πούμε ότι έχουμε εκπαιδεύσει το δίκτυο. Στο παρακάτω σχήμα φαίνονται τα ονόματα των κόμβων και των συνδέσεων.

Σχήμα 1: Artificial neural network



Έχουμε τέσσερις πιθανούς συνδυασμούς εισόδου: 0 και 0, 0 και 1, 1 και 0, 1 και 1. Τα όμοια ζευγάρια πρέπει να δίνουν έξοδο 0 και τα ανόμοια 1. Ξεκινάμε με τυχαίες τιμές για τα

βάρη, ανάμεσα στο -1 και 1. Οι κόμβοι 0 και 1 πάντα έχουν τιμή y_0, y_1 ίση με έναν από τους τέσσερις πιθανούς συνδυασμούς που αναφέραμε παραπάνω. Οι τιμές των υπόλοιπων κόμβων καθορίζονται σύμφωνα με τις παρακάτω σχέσεις:

$$y_3 = \phi(u_3)$$
$$y_2 = \phi(u_2)$$
$$y_4 = \phi(u_4)$$

όπου η εξίσωση $\phi(u)$ δίνεται από τη σχέση:

$$\phi(u) = \frac{1}{1 + e^{-x}}$$

Οι ποσότητες u δίνονται από τις

$$u_3 = w_0 * y_0 + w_2 * y_1 + w_4$$
$$u_2 = w_1 * y_0 + w_3 * y_1 + w_5$$
$$u_4 = w_6 * y_2 + w_7 * y_3 + w_8$$

Αφού βρούμε τις τιμές όλων των κόμβων υπολογίζουμε το σφάλμα που έχει η έξοδος με την επιθυμητή τιμή t (target).

$$error = 1/2(y_4 - t)^2$$

Αν το σφάλμα είναι μεγαλύτερο από 0.01 τότε διορθώνουμε τα βάρη πηγαίνοντας προς τα πίσω, από την έξοδο στην είσοδο, με μια διαδικασία που λέγετε back propagation. Τα βάρη αλλάζουν σύμφωνα με τη σχέσης $w_i = w_i + \Delta w_i$ όπου Δw_i:

$$\Delta w_0 = \eta\, \delta_3\, y_0 \qquad \Delta w_1 = \eta\, \delta_2\, y_0 \qquad \Delta w_2 = \eta\, \delta_3\, y_1$$
$$\Delta w_3 = \eta\, \delta_2\, y_1 \qquad \Delta w_4 = \eta\, \delta_3 \qquad \Delta w_5 = \eta\, \delta_2$$
$$\Delta w_6 = \eta\, \delta_4\, y_2 \qquad \Delta w_7 = \eta\, \delta_4\, y_3 \qquad \Delta w_8 = \eta\, \delta_4$$

και $\eta = 0.2$ το ρυθμός εκμάθησης (learning rate) και για το επίπεδο εξόδου $\delta_4 = y_4(1 - y_4)(t - y_4)$ ενώ για τα υπόλοιπα $\delta_i = y_i(1 - y_i) \sum \delta w$.

Αυτή η διαδικασία επαναλαμβάνεται είτε μέχρι τα σφάλματα να γίνουν επαρκώς μικρό, στην οποία περίπτωση θεωρούμε το δίκτυο εκπαιδευμένο, είτε αν περάσουν 10000 εποχές όπου μία εποχή περνάει μόλις δώσουμε και τα τέσσερα ζευγάρια τιμών εισόδου.

Αποτέλεσμα

Αφού τρέξουμε τον κώδικα σύμφωνα με τα όσα περιγράψαμε παραπάνω βρίσκουμε τις εξής τιμές για τα βάρη:

$$w_0 = -0.275832 \qquad w_1 = -5.72209 \qquad w_2 = -1.98348$$
$$w_3 = -0.664434 \qquad w_4 = -0.174571 \qquad w_5 = -2.40081$$
$$w_6 = -0.995157 \qquad w_7 = -1.06965 \qquad w_8 = 0.298674$$

Στην επόμενη παράγραφο δίνεται αναλυτικά και ο κώδικας.

Ο κώδικας

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
using namespace std;

double fi(double u){
  return 1/(1 + exp(-u));
}

int main(int argc, char const *argv[]) {
  int i,j,t,epoch;
  int nw = 9;
  int ny = 5;
  double x,u,err;
  double w[nw];
  double dw[nw];
  double y[ny];
  double d[ny];
  double er_max = 0.01; // maximum acceptable error
  double h = 0.2; //learing rate
  bool trained;

  srand(4372);
  ofstream f("weights.txt");

  //initial weights
  for(i=0;i<nw;i++){
    x = -1.0 + ((double) rand() / (RAND_MAX))*(2.0);
    w[i] = x;
  }

  epoch = 1;
  do {
    trained = true;
    for(j=1;j<5;j++){
      // input & target
      switch (j) {
        case 1:
          y[0] = 1;
          y[1] = 1;
          t = 0;
          break;
        case 2:
          y[0] = 1;
          y[1] = 0;
          t = 1;
```

```
          break;
        case 3:
          y[0] = 0;
          y[1] = 1;
          t = 1;
          break;
        case 4:
          y[0] = 0;
          y[1] = 0;
          t = 0;
          break;
      }

      for(i=2;i<ny-1;i++){
        u = w[i-2]*y[0] + w[i]*y[1] + w[i+2];
        y[i] = fi(u);
      }

      u = w[6]*y[2] + w[7]*y[3] + w[8];
      y[4] = fi(u);

      err = pow(y[4]-t,2)/2;
      if(err<er_max){
        continue;
      }

      trained = false;

      // back propagation
      d[4] = y[4]*(1-y[4])*(t-y[4]);
      d[3] = y[3]*(1-y[3])*(d[4]*w[7]);
      d[2] = y[2]*(1-y[2])*(d[4]*w[6]);
      d[1] = y[1]*(1-y[1])*(d[3]*w[2] + d[2]*w[3]);
      d[0] = y[0]*(1-y[0])*(d[2]*w[1] + d[3]*w[0]);

      dw[0] = h*d[3]*y[0];
      dw[1] = h*d[2]*y[0];
      dw[2] = h*d[3]*y[1];
      dw[3] = h*d[2]*y[1];
      dw[4] = h*d[3];
      dw[5] = h*d[2];
      dw[6] = h*d[4]*y[2];
      dw[7] = h*d[4]*y[3];
      dw[8] = h*d[4];

      for(i=0;i<nw;i++){
        w[i] = w[i] + dw[i];
      }
    }
```

```
97      if(trained == true){
98        break;
99      }
100     epoch = epoch + 1;
101   } while(epoch<10000);
102
103   for(i=0;i<nw;i++){
104     f << w[i] << endl;
105   }
106
107   return 0;
108 }
```