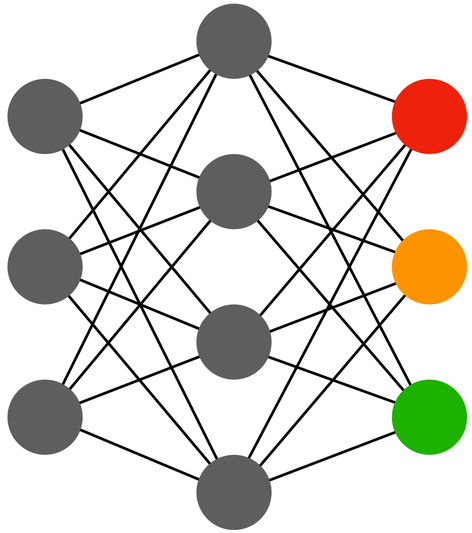
AI_TMS

מערכת הבינה המלאכותית



סמואל ארביב 19.05.2020

רשת הנוירונים

רשת הנוירונים נבנתה מאפס בשפת c++. הרשת הינה גנרית לחלוטין, וניתנת לשימוש בכל מערכת, מורכבת ככל שתהיה.

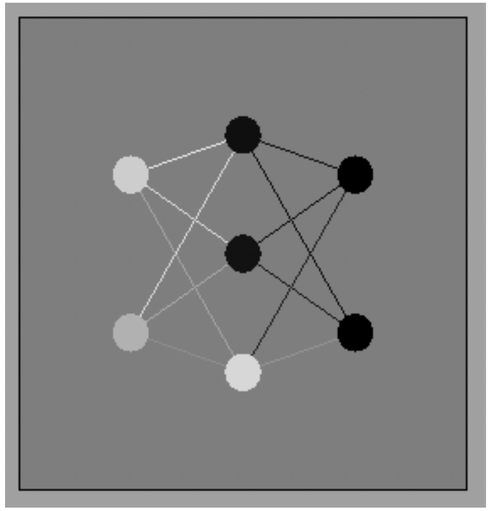
השימוש ברשת הנוירונים פשוט מאוד, ופעילותה יעילה ומהירה.

על מנת לבנות רשת, יש ליצור 'טופולגיה' רצויה עבורה, ולשלוח אותה לפונקצית הבניה.

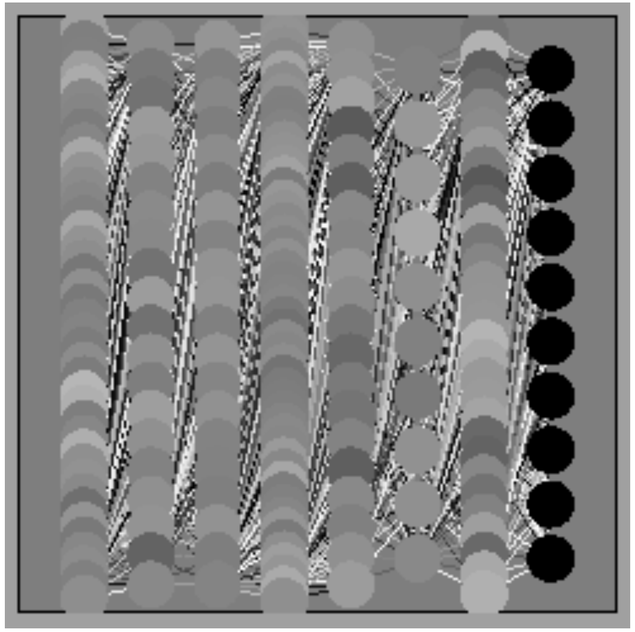
כאמור, הרשת יכולה להיות פשוטה מאוד, אך גם מורכבת מאוד.

```
vector<unsigned> topology;

// input neurons : max lane density, max queue length
topology.push_back(2);
// hidden neurons
topology.push_back(3);
// output neurons : priority points, phase time
topology.push_back(2);
```



```
// input neurons
topology.push_back(40);
// hidden neurons
topology.push_back(20);
topology.push_back(20);
topology.push_back(50);
topology.push_back(20);
topology.push_back(10);
topology.push_back(30);
// output neurons
topology.push_back(10);
```



אלגוריתם גנטי

האלגוריתם הגנטי מבוסס כל עקרון ’החזק שורד’, ומטרתו לשמר לטפח תכונות ’גנטיות’ חזקות, ולהשמיט תכונות חלשות.

כשירותה של רשת נוירונים נקבעת על ידי ביצועה בסימולציה עליה היא שולטת.

ככל שזרימת התחבורה בסימולציה גבוהה יותר, כך הרשת שקבעה את המדיניות באותה העת תקבל ציון גבוה יותר.

מוגדר דור שהוא מערך של רשתות נוירונים. הדור הראשון מאותחל באופן רנדומלי לחלוטין.

```
vector<Net> Net::Generation = vector<Net>();
```

```
for(unsigned i = 0; i < Net::PopulationSize; i++)
{
    Net::Generation.emplace_back(topology);
}
```

```
Net::CurrentNet = &(Net::Generation[Net::CurrentNetIndex]);
```

הגדרת רשת נוירנים באופן רנדומלי נעשית על ידי הצבת ערכים רנדומלים במשקלי רשת הנוריונים.

```
// create random output weights
for (int c = 0; c < numOutputs; c++)
{
    // push a random into the output weights
    output_weights_.push_back(Connection());
    output_weights_.back().weight = randomize_weight();
}
```

```
static double randomize_weight() { return random() / double(RAND_MAX); }
```

לאחר שהורץ דור שלם של רשתות רנדומליות, ניתן לכל רשת ציון.

נבצע נורמליזציה של ציוני הרשתות: כל רשת במערך תקבל מאפיין הנקרא ‘כושר’,
והוא יבטא את חלק ציונו בסך הציונים של הדור.

הכושר הוא אקספוננציאלי ביחס לציון, וזאת על מנת להגדיל את משמעותם של הבדלים קטנים בקצה העליון של תחום הציונים.

סך הכושר של כל הרשתות בדור הוא 1, נשתמש בעובדה זו בהמשך.

```
void Net::NormalizeFitness(vector<Net> &oldGen) {
    double sum = 0;
    for(unsigned i = 0; i < oldGen.size(); i++)
    {
        double score = pow(oldGen[i].score_, 2);
        oldGen[i].score_ = score;
        sum += score;
    }

    for(unsigned i = 0; i < oldGen.size(); i++)
    {
        oldGen[i].fitness_ = oldGen[i].score_ / sum;
    }
}
```

לאחר שביצענו נורמליזציה על דור הרשתות וקבענו לכל אחד מהם מאפיין ’כושר’, ניצור את הדור הבא ונבצע ’אבולוציה’.

יצירת דור חדש תלויה בדור הקודם: בעת יצירת דור חדש, נבחרת בכל פעם באופן רנדומלי (למחצה) רשת מהדור הקודם, והיא משוכפלת אל תוך הדור החדש.

```
vector<Net> Net::Generate(const vector<Net> &oldGen) {
    vector<Net> newGen;
    for(unsigned i = 0; i < oldGen.size(); i++)
    {
        newGen.push_back(Net::PoolSelection(oldGen));
        newGen.back().mutate(0.2);
    }
    return newGen;
}
```

אלגוריתם הבחירה, או PoolSelection הוא אלגורתים לבחירה רנדומלית המושפעת מכשירותה של רשת.

המשמעות היא, שעבור ערך רנדומלי כלשהו, הסיכוי שרשת תיבחר שווה לכושר שלה.

אופן הפעולה הוא:

ניצור ערך רנדומלי **r** בין 0 ל-1.

כל עוד $r > 0$, נוריד ממנו כל פעם את הכושר של רשת אחרת, בסדר עוקב

כאשר r מתאפס לאחר הורדת כושר של רשת כלשהי, רשת זו תיבחר.

ניתן לדמות זאת לכך שכל רשת מקבלת 'נתח' בתחום בין 0 ל-1 בהתאם לכושר שלה. ככל שנתח זה גדול יותר, הסיכוי שיכלול ערך נרדומלי כשלהו גדול יותר.

```
Net Net::PoolSelection(const vector<Net> &oldGen) {
    unsigned index = 0;
    double r = rand() / double(RAND_MAX);

    while(r > 0)
    {
        r -= oldGen[index].fitness_;
        index++;
    }
    index--;
    return oldGen[index];
}
```

לאחר שבחרנו רשתות לשכפל לדור החדש באמצעות האלגוריתם, נבצע עליהם מוטציה.

המוטציה, עבור ערך נתון **X**, תבצע שינויים רנדומליים בכל רשת נוירונים:

עבור כל משקל ברשת, ישנו סיכוי של X שערכו יוגדר מחדש רנדומלית.

לאחר יצירת הדור החדש, נריץ כל רשת שבו ונחזור חלילה.

אימפלמנטציה במערכת

תפקידה של רשת בכל סימולציה היא לקבוע את מדיניות הפעלת הרמזורים בכל צומת.

כל מחזור פאזות מכיל מערך של פאזות. פאזה מכילה מספר נתיבים שלא בהכרח נמצאים באותו הכביש, שיפתחו וייסגרו בהתאם למצב בפאזה.

את המערך הזה נמיין ונשמור ממויין בכל עת, לפי **עדיפות הפאזות**. הפאזה שתיפתח תהיה הפאזה האחרונה המערך, כלומר בעלת העדיפות הגבוהה ביותר. כאשר היא תיסגר, תיתפח הפאזה העוקבת במערך, שהיא כעת הפאזה בעלת העדיפות המקסימלית.

כאמור, עדיפות הפאזות מחושבת **בכל רגע**, ולכן שינויים בתחבורה באים לידי ביטוי בזמן אמת במדיניות הפאזות.

```
void Cycle::cycle_phases() {
    // if cycle has a minimum of 2 phases
    if (number_of_phases_ >= 2)
    {
        // when current phase is closed, advance to next phase and open it
        if (!phases_.back()->GetIsOpen())
        {
            Phase *backPhase = phases_[number_of_phases_ - 1];
            phases_[number_of_phases_ - 1] = phases_[number_of_phases_ - 2];
            phases_[number_of_phases_ - 2] = backPhase;

            phases_[number_of_phases_ - 1]->Open();

        }
            // constantly sort the list by their priority score
        else
        {
            // calculate the priority of each phase
            calculate_priority();
            // sort(arr[0:-2])
            partial_sort(phases_.begin(),
                         phases_.end() - 1,
                         phases_.end() - 1,
                         compare_priority);
        }
    }
}
```

על מנת לחשב את עדיפותה של פאזה, נשתמש **ברשת הנוירונים.**

עבור פאזה במחזור רמזורים, רשת הנוירונים מקבלת כקלט:

1. **ערך הצפיפות המקסימלי** בין כל הנתיבים השייכים לאותה פאזה.
2. **אורך התור המקסימלי** בין כל הנתיבים השייכים לאותה פאזה.

עבור פאזה במחזור רמזורים, רשת הנוירונים מזחירה כפלט:

1. **ציון עדיפות הפאזה** בהתאם לקלט
2. **משך זמן הפתיחה** האופטימלי

```
void Cycle::calculate_priority() {

    for (int p = 0; p < number_of_phases_ - 1; p++)
    {
        // get input values
        phases_[p]->GetInputValues(input_values_);
        // feed data through nerual net
        Net::CurrentNet->FeedForward(input_values_);
        // get data from output layer
        Net::CurrentNet->GetResults(output_values_);

        // set the output data
        phases_[p]->SetPhasePriority(output_values_[0]);
        phases_[p]->SetCycleTime(clamp(
            float(output_values_[1]) * Settings::MaxCycleTime,
            Settings::MinCycleTime,
            Settings::MaxCycleTime));
    }
}
```

בכל עת, נשמר במקביל העתק של הרשת שציונה היה הגבוה ביותר עד כה. בממשק במשתמש, ישנה האופציה להריץ את הרשת החזקה ביותר.

כמו כן, ניתן להמשיך להפעיל את האלגוריתם הגנטי ולהמשיך את האבולוציה כל הזמן ללא הפסקה. האבולוציה אינה מקבעת את הרשתות לצורה כשלהי, אלה מנסה לשפר כל הזמן את הציון.

המשמעות היא שהאלגוריתם מסוגל לבצע שינויים דרסטיים לאבולוציה של הרשתות בהתאם למצב משתנה בתחבורה.

כך לדוגמה, במצב של גשם בו מכוניות יסעו לאט יותר, ידע האלגוריתם לבצע את האבולוציה בהתאם, ויוכל לשנות את צורת הרשת והמדיניות בפרק זמן קצת יחסית.

קוד מקור - https://github.com/samuelarbibe/AI_TMS

סרטון הפעלה - https://www.youtube.com/watch?v=BLz_PdU2oyo

סרטון תקציר - https://www.youtube.com/watch?v=xJOcDKXWJXo