Métaheuristique pour l'optimisation TP5 120 crédit

Deniz Sungurtekin

November 2020

1 Introduction

Le but de ce travail pratique est de nous familiariser avec l'algorithme "Particle Swarm Optimization" (PSO) et aux réseaux de neurones artificiels. PSO est une méthode qui s'inspire du monde des vivants, elle se base sur la collaboration des individus entre eux pour résoudre un problème donné. Nous allons utiliser cette méthode pour entraîner notre réseau, afin qu'il soit capable de classifier des chiffres écris à la main. Nous avons à notre disposition un ensemble de donné X contenant 200 échantillons décrivant une image par 400 de ses pixels et un ensemble de donné Y possédants les labels correspondants, 1 si l'image représente un "2" et 0 si elle représente un "3". A travers ce rapport, nous allons expliquer la méthodologie utilisée, les résultats obtenues, l'impact du nombre de particule et celui du "velocity cut-off" sur ces résultats

2 Méthodologie

Dans un premier temps, notre objectif est de trouver les meilleurs matrices et vecteurs Θ_1, Θ_2, puisque c'est eux qui contiennent nos poids qui vont permettre à notre réseaux d'obtenir la meilleur classification de nos images. Pour cela, il est nécessaire d'implémenter PFO, comme expliqué dans l'énoncé chaque particule explore les valeurs possibles de positions dans le domaine [-0.5,0.5] avec des valeurs de vélocité initialisé à 0 (Vecteur de la même taille que le vecteur position, s*(m+1)+s+1 avec s = 25 et m = 400). Chaque particule initialise aléatoirement son vecteur position, puis calcule l'énergie associée pour stocker la meilleur obtenue. Ensuite, il suffit de prendre la meilleur énergie parmi toute les particules et mettre à jour leur vecteur vélocité et position de la manière suivante:

$$\begin{aligned} \boldsymbol{v}_{t+1}^i &= \omega \boldsymbol{v}_t^i + c_1 r_1 (\boldsymbol{b}_t^i - \boldsymbol{s}_t^i) + c_2 r_2 (\boldsymbol{b}_t^G - \boldsymbol{s}_t^i) \\ \boldsymbol{s}_{t+1}^i &= \boldsymbol{s}_t^i + \boldsymbol{v}_{t+1}^i \end{aligned}$$

avec w = 0.9, $c_1 = c_2 = 2$, $0 < r_1 < 1, 0 < r_2 < 1$.

Puis, il suffit simplement d'appliquer le pseudo-code fourni:

Algorithm 1 PSO algorithm

Initialize
t = 0 ;
Initialize randomly the positions $\boldsymbol{s}_0^i$ in the domain to be explored and the velocities $\boldsymbol{v}_0^i = 0$ for all particles $i = 1..N$.
Do
For each particle :
Calculate its fitness J_t^i ;
If $J_t^i \leq J_{best}^i$ then $J_{best}^i = J_t^i, \boldsymbol{b}_t^i = \boldsymbol{s}_t^i$
End for
Calculate $J_{best}^G = min_i J_{best}^i$, update $\boldsymbol{b}_t^G = \boldsymbol{b}_t^{arg(min_i J_{best}^i)}$
For each particle :
Randomly generate r_1, r_2
Update particle velocity by formula (1)
Update particle position by formula (2)
End for
t = t + 1 ;
Until (end criteria are met)

A la différence près que dans mon code j'ai préféré inclure un compteur "frozen" et "diff" qui permettent respectivement de détecter si le système ne trouve pas de meilleur solution durant tmax/5 itération et de repérer si la solution s'améliore uniquement de manière négligeable 5 fois de suite.(différence de 0.001 avec l'anciennne meilleur solution) Cela permet d'arrêter le processus si le système ne fait plus de progrès notable.

De plus, afin d'éviter une croissance trop grande du vecteur vélocité, on introduit le "velocity cut-off" qui donne une valeur absolue maximal des composantes de notre vecteur. Cette condition est vérifié après la mis à jour du vecteur vélocité à chaque itération. Si elle n'est pas respectée, la valeur de la composante en question est fixé à la valeur maximal/minimal définie. Nous discuterons de son impact dans la prochaine section.

Une fois nos poids Θ_1 et $\Theta 2$ obtenus par PSO, il suffit de calculer $y_{predict}$ en arrondissant la valeur obtenue à l'entier le plus proche après avoir appliqué notre réseau de neurone avec Θ_1 et $\Theta 2$ sur chaque ligne de X.

3 Résultat

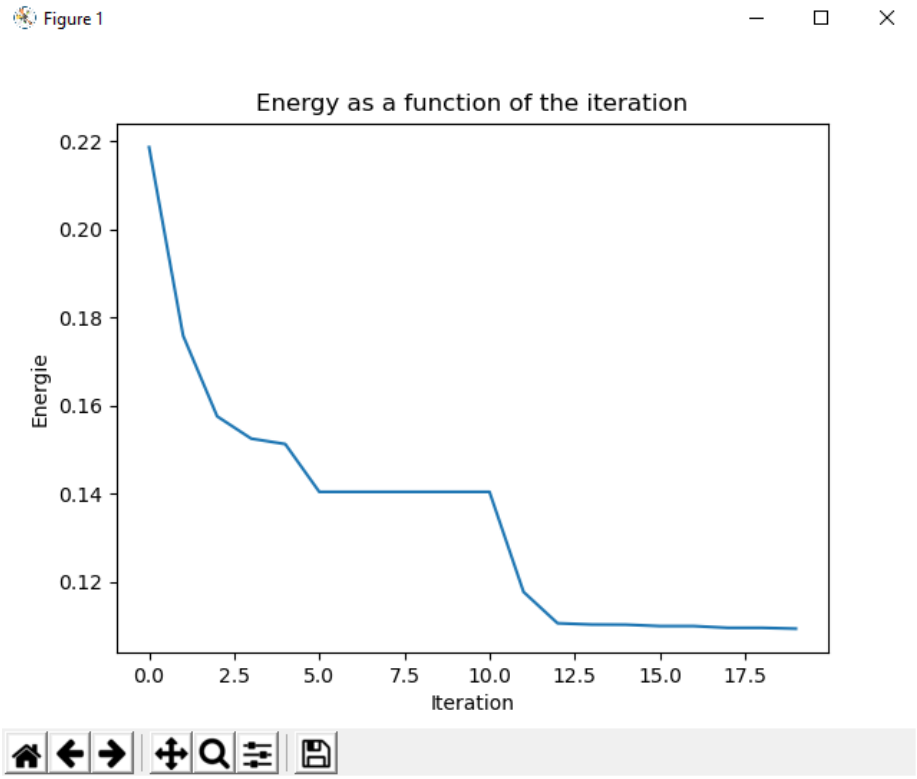
Voici les résultats obtenues après avoir lancé 10 fois notre algorithme PSO:

```
[0.10135039736431782, 0.11084569225728944, 0.09239251887020117, 0.10104896811506718, 0.07792808338655474, 0.08983906423912698,
0.08588657069586088, 0.07069178127422371, 0.09094176913081414, 0.10938461427651187]
```

On remarque que les 10 valeurs optimales obtenues semblent assez stable puisqu'elles ne varient que très peu, entre 0.11 et 0.7. De plus, ce résultat semble satisfaisant puisque celui-ci calcule la moyenne des différences entre y_k et $h_{\Theta 1,\Theta 2}(x_k)$ au carré

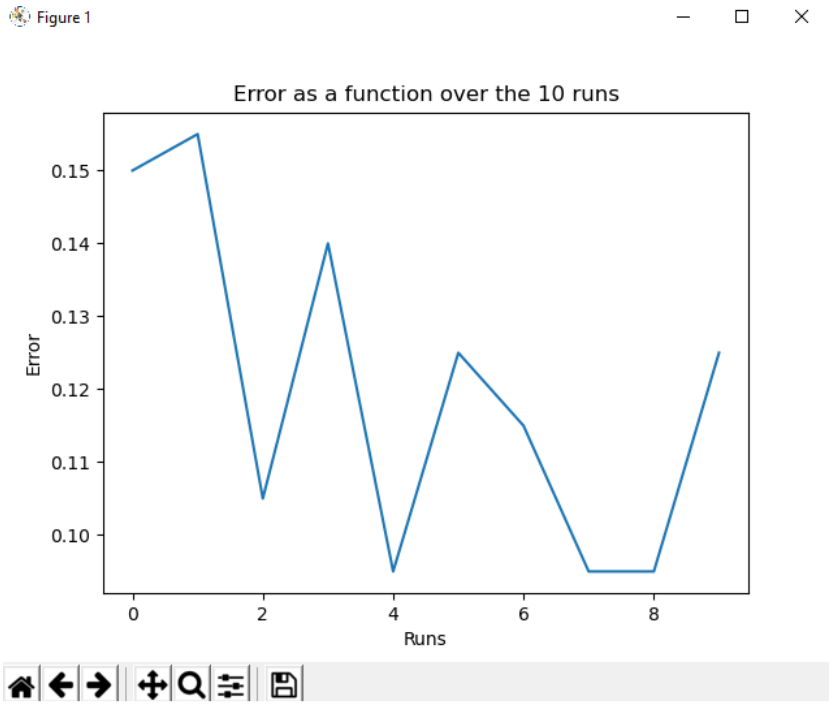
avec uniquement 100 itérations. J'ai choisi ce nombre d'itération (Avec 20 particules), car les résultats me semblent correct pour un temps d'exécution rapide. En outre, avec 100 itérations le système a fait les améliorations importante et à déjà commencé à effectuer des améliorations négligeable de l'ordre de 0.001. Il serait donc facilement possible d'augmenter la précision de notre réseau de neurone mais cela coûtera beaucoup de temps pour un gain de précision infime.

Voici maintenant l'évolution de la meilleur énergie parmi toute les particules durant le 10ème run de PSO:



Ici, le processus s'est arrêté à la 20ème itération, car le système ne présentait aucune amélioration notable (>0.001) durant les cinq dernière fois où un gain de précision a été trouvé. On remarque clairement une décroissance très rapide durant les 5 premières itérations qui continue de baisser jusqu'à atteindre une valeur de 0.1, où il va commencer à diminuer beaucoup plus lentement et se stabiliser.

Observons maintenant l'erreur de prédiction pour chacune de ces runs:



On observe une précision qui varie de 84% à 95%, ce qui semble très correct pour un nombre d'itération très bas et donc un temps d'exécution rapide.

Discutons désormais de l'impact du nombre de particule et du "velocity cut-off" sur le résultat. Forcément, plus le nombre de particule est grand plus le résultat s'améliore rapidement, car cela favorise l'exploration des positions possibles dans le domaine. Par conséquent, pour obtenir un résultat stable, un grand nombre de particule demande moins d'itérations mais augmente considérablement leur temps d'exécution. De plus, dû à l'initialisation aléatoire du vecteur position, un grand nombre de particule favorise la diversification des positions et donc permet de rapidement trouver des minimums locaux.

En ce qui concerne la vélocité, elle représente la distance parcourue par une particule depuis la position courante. Par conséquent, on peut déduire qu'une grande vélocité favorise l'exploration de l'espace de recherche et qu'une petite vélocité favorise son exploitation. En d'autre mots, plus on limite cette vélocité par un "velocity cut-off" plus le système aura des difficultés à diversifier ses solutions. Cependant, il peut être bénéfique de limiter cette vélocité dans le temps pour faciliter l'exploration et ensuite de la réduire pour permettre une bonne exploitation des meilleurs solutions.