En fördjupad genomgång av Partikelsvärmsoptimering (PSO)

Sebastian Öhman

1 januari 2025

1 Bakgrund och grundidé

Partikelsvärmsoptimering (*Particle Swarm Optimization*, PSO) utvecklades av James Kennedy och Russell Eberhart 1995 och är en svärmbaserad metod för att lösa optimeringsproblem, ofta i ett kontinuerligt sökutrymme. Grundtanken är inspirerad av hur fågelflockar eller fiskstim rör sig i koordinering mot en plats som erbjuder bäst ”värde” (exempelvis mest föda).

I PSO är varje ”partikel” en möjlig lösning på det aktuella problemet. Partikeln har följande egenskaper:

1. **Position**: En vektor $\mathbf{x}_i$ som beskriver partikelns nuvarande läge i sökutrymmet (t.ex. i $\mathbb{R}^n$).
2. **Hastighet**: En vektor $\mathbf{v}_i$ som beskriver hur partikeln rör sig i sökutrymmet.
3. **Personbästa** ($\mathbf{pbest}_i$): Den bästa position (lägst kostnad eller högst ”fitness”) som partikeln själv har uppnått hittills.
4. **Globala bästa** (**gbest**): Den övergripande bästa positionen (eller värdet) som hela svärmen har hittat.

Varje iteration uppdateras partikelns hastighet och position utifrån:

- Inertiavikten (w), som styr hur mycket den tidigare hastigheten behålls.
- En ”kognitiv” komponent (c_1), som drar partikeln mot dess personbästa.
- En ”social” komponent (c_2), som drar partikeln mot det globala bästa.

Dessutom multiplicerar man ofta dessa komponenter med slumpvärden för att tillföra extra mångfald och undvika att fastna i lokala optimum.

2 Typiska formler

Anta att vi har N partiklar i en svärm, och var och en är en vektor i $\mathbb{R}^n$. För partikel i:

2.1 Hastighetsuppdatering

$$\mathbf{v}_i^{(t+1)} = w \cdot \mathbf{v}_i^{(t)} + c_1\, r_1 \big(\mathbf{pbest}_i^{(t)} - \mathbf{x}_i^{(t)}\big) + c_2\, r_2 \big(\mathbf{gbest}^{(t)} - \mathbf{x}_i^{(t)}\big),$$

där

- w: inertiavikt (t.ex. mellan 0.4 och 0.9).
- c_1: kognitiv koefficient (exempelvis runt 1–2).
- c_2: social koefficient (exempelvis runt 1–2).
- r_1, r_2: slumptal i intervallet $[0, 1]$.

2.2 Positionsuppdatering

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}.$$

2.3 Uppdatering av personbästa

Om partikel i:s nya position $\mathbf{x}_i^{(t+1)}$ ger ett bättre värde (lägre kostnadsfunktion eller högre "fitness") än dess nuvarande $\mathbf{pbest}_i$, så sätts

$$\mathbf{pbest}_i^{(t+1)} = \mathbf{x}_i^{(t+1)},$$

annars behålls det gamla värdet.

2.4 Uppdatering av globalt bästa

Om partikel i har bättre värde än det befintliga globala bästa $\mathbf{gbest}^{(t)}$, så sätts

$$\mathbf{gbest}^{(t+1)} = \mathbf{x}_i^{(t+1)},$$

annars behålls den gamla **gbest**.

3 En enkel översikt av arbetsflödet

1. **Initiering**: Slumpa startpositioner och hastigheter för alla partiklar. Sätt $\mathbf{pbest}_i$ för varje partikel till dess startposition och välj **gbest** som den position (över alla partiklar) som har bästa värdet i denna initiala uppsättning.
2. **Iterera**:
   - Beräkna för varje partikel hur bra dess nuvarande position är (utvärdera kostnadsfunktionen eller fitnessfunktionen).
   - Uppdatera partikelns personbästa om den nya positionen är bättre.
   - Uppdatera globalt bästa om någon partikel hittat en bättre lösning.
   - Uppdatera partikelns hastighet ($\mathbf{v}_i$) och position ($\mathbf{x}_i$) enligt formlerna ovan.
3. **Avslut**: När ett fördefinierat stoppkriterium är uppfyllt (exempelvis maximalt antal iterationer, tillfredsställande lösningskvalitet eller tidsbegränsning) avslutas algoritmen, och **gbest** rapporteras som det bästa resultatet.

4 Exempel på pseudokod

Nedan visas en förenklad pseudokod för PSO. Språk och format kan förstås variera beroende på implementering, men tankesättet är detsamma:

```
# Parametrar:
#   N: antal partiklar
#   max_iter: maximala antal iterationer
#   w: inertiavikt
#   c1, c2: kognitiv och social koefficient
#   problem_function(x): den kostnads- eller fitnessfunktion som ska minimeras/maximeras
#   D: dimensionen på sökutrymmet (t.ex. 2, 3 eller högre)

# 1. Initiering
```

```
particles = en lista med N partiklar
for i in range(N):
    particles[i].x = slumpa en D-dimensionell position
    particles[i].v = slumpa en D-dimensionell hastighetsvektor
    particles[i].pbest = particles[i].x
    particles[i].best_value = problem_function(particles[i].x)

# Sätt gbest till bästa pbest bland alla partiklar:
gbest = position hos den partikel som har lägsta (eller högsta) best_value
gbest_value = problem_function(gbest)

iteration = 0
while iteration < max_iter:
    for i in range(N):
        # 2.1 Utvärdera nuvarande position
        current_value = problem_function(particles[i].x)

        # 2.2 Uppdatera personbästa
        if current_value är bättre än particles[i].best_value:
            particles[i].pbest = particles[i].x
            particles[i].best_value = current_value

        # 2.3 Uppdatera globalt bästa
        if current_value är bättre än gbest_value:
            gbest = particles[i].x
            gbest_value = current_value

    # 2.4 Uppdatera hastigheter och positioner
    for i in range(N):
        # Slumptal för variation
        r1 = slumpvärde i [0,1]
        r2 = slumpvärde i [0,1]

        # Hastighetsuppdatering
        particles[i].v = w * particles[i].v
                         + c1 * r1 * (particles[i].pbest - particles[i].x)
                         + c2 * r2 * (gbest - particles[i].x)

        # Positionsuppdatering
        particles[i].x = particles[i].x + particles[i].v

    iteration = iteration + 1

# 3. Avslut
return gbest, gbest_value
```

5 Parametrar och praktiska överväganden

- **Inertiavikt** (w): Påverkar balans mellan global och lokal utforskning. Ett lägre w får svärmen att snabbt konvergera mot ett område, medan ett större w ger större utforskning av sökutrymmet.

- **Kognitiv och social komponent (c_1, c_2)**: Höga värden kan göra att partiklarna rör sig mer okontrollerat. Låga värden kan istället göra att algoritmen konvergerar långsamt eller fastnar i lokala optimum. En vanlig tumregel är $c_1 + c_2 = 4$, exempelvis $c_1 = 2$ och $c_2 = 2$.
- **Slumptal (r_1, r_2)**: De slumpmässiga komponenterna bidrar till att partiklarna inte rör sig i exakt samma riktning mot *pbest* och *gbest*, vilket hjälper till att undvika lokala optimum.
- **Antalet partiklar (N)**: För få partiklar riskerar att inte täcka sökutrymmet tillräckligt, medan för många partiklar kan öka beräkningstiden utan att nödvändigtvis ge bättre resultat.
- **Stoppkriterium**: Till exempel maximalt antal iterationer, tidsbegränsning eller en fördefinierad tröskel för funktionsvärdet.

6 Exempel på tillämpningar

- Optimering av neurala nätverks arkitektur (t.ex. hitta optimal struktur eller optimala hyperparametrar).
- Kontinuerliga optimeringsproblem inom till exempel industriella processer, molekylmodellering eller ekonomiska modeller.
- Funktioner med flera lokala extrempunkter (där klassiska gradientmetoder riskerar att fastna).
- Tuning av algoritmparametrar inom andra AI- eller maskininlärningsområden.

7 Sammanfattning

Partikelsvärmsoptimering är en enkel men kraftfull metod för att lösa optimeringsproblem, särskilt i högdimensionella eller icke-linjära sammanhang. Genom att kombinera varje partikels eget minne av bästa position (**pbest**) med hela svärmens bästa hittills funna lösning (**gbest**) kan PSO hitta bra – och ibland utmärkta – lösningar på många svårdefinierade problem.

Att justera parametrar som inertiavikt, antalet partiklar, samt de kognitiva och sociala koefficienterna är centralt för att lyckas i praktiken. Trots sin enkelhet är PSO en av de mer populära och välstuderade metoderna inom biologiskt inspirerad optimering och fortsätter att hitta användning inom flera forsknings- och industriområden.