LICENCIATURA EM ENGENHARIA INFORMÁTICA
2023/2024

Inteligência Computacional

Garbage Classification
Gravitational Search Algorithm
Meta 2

Bruno Oliveira - 2019136478
Micael Eid – 2019112744

Contents

Computação Swarm

O paradigma da computação swarm é inspirado no comportamento coletivo de organismos sociais, como insetos, pássaros e peixes, que interagem uns com os outros para realizar tarefas complexas de forma coordenada. Essa abordagem tem como objetivo aplicar princípios da inteligência coletiva observada na natureza para resolver problemas computacionais.
Na computação swarm, um grande número de agentes simples, chamados de "partículas" ou "indivíduos", interagem localmente entre si para atingir um objetivo global. Cada agente segue regras simples, geralmente baseadas em interações locais e comunicação direta com os vizinhos, sem a necessidade de uma entidade central de controlo.

Relativamente ao treino de redes neurais, o paradigma de computação swarm pode ser aplicado de várias maneiras:

- Otimização de Hiperparâmetros: Os parâmetros de uma rede neural, como taxas de aprendizagem e arquitetura da rede, podem ser ajustados usando algoritmos de otimização baseados em swarm. As partículas representam diferentes configurações de hiperparâmetros e a colaboração entre elas poderia levar a descobertas mais eficientes e melhores configurações.

- Treino Distribuído: Em vez de depender de uma única instância de treino centralizado, o treino da rede neural pode ser distribuído entre várias máquinas ou dispositivos, cada um representando uma partícula no swarm. A comunicação local entre essas instâncias pode permitir uma convergência mais rápida e eficiente durante o treinamento.

- Otimização de Topologia da Rede: A estrutura da rede neural pode ser ajustada dinamicamente com base no feedback local. Cada partícula pode representar uma configuração específica da arquitetura da rede, e a troca de informações entre elas pode levar a uma adaptação coletiva da topologia da rede para melhor desempenho.

- Deteção de Anomalias e Adaptação Dinâmica: O swarm pode ser utilizado para monitorizar o desempenho da rede neural em tempo real. Se uma parte da rede começar a comportar-se de maneira atipica, as partículas podem se reorganizar dinamicamente para se adaptar à mudança nas condições, proporcionando uma maior robustez ao sistema.

- Seleção de Conjunto de Dados: As partículas podem explorar diferentes subconjuntos de dados durante o treino, permitindo uma abordagem mais abrangente para a generalização da rede neural.

Gravitational Search Algorithm (GSA)

O Algoritmo de Busca Gravitacional (GSA) é uma técnica de otimização baseada em fenómenos gravitacionais que ocorrem na natureza. Desenvolvido por Rashedi, Nezamabadi-pour e Saryazdi em 2009, o GSA é inspirado no comportamento de corpos celestes que interagem entre si devido à força gravitacional.

Vantagens

- GSA é relativamente simples de entender e implementar, tornando-o acessível para utilizadores menos experientes.
- Comparado a alguns algoritmos de otimização, como o PSO, o GSA possui menos parâmetros para ajustar., o que facilita na configuração do algoritmo.
- GSA tende a ser eficaz na exploração global do espaço de busca, graças à influência gravitacional entre as partículas. Isso pode ser útil para problemas onde encontrar a solução global é crucial.
- Inspiração nas leis físicas da gravidade, resultando em comportamentos interessantes e eficientes.

Desvantagens

- Por vezes o GSA pode convergir para a solução ótima de maneira mais lenta em comparação com outros algoritmos.
- Apesar de ter menos parâmetros que alguns algoritmos, a escolha dos mesmos pode influenciar significativamente o desempenho do algoritmo.
- A eficácia do GSA pode variar dependendo das características específicas do problema.
- O desempenho do GSA pode ser afetado por problemas de escala, especialmente em espaços de busca de alta dimensão. Isso pode tornar o GSA menos eficiente para problemas complexos.

PSO e GSA

O Partical Swarm Optimization (PSO) baseia-se em interações sociais para otimização, o GSA é inspirado nas leis físicas da gravidade descobertas por Isaac Newton. A escolha entre eles dependerá da natureza específica do problema e das características desejadas para a otimização. É muitas vezes necessário exemplos práticos em problemas específicos para determinar qual dos algoritmos funciona melhor.

Semelhanças:

- Ambos os algoritmos são inspirados em comportamentos coletivos de entidades naturais (partículas no PSO e corpos celestes no GSA).
- Ambos procuram um equilíbrio entre a exploração do espaço de busca em busca de novas soluções e a exploração de regiões promissoras.

Diferenças:

- No PSO, as partículas são movidas com base em sua melhor posição pessoal e na melhor posição global encontrada pela população. No GSA, a movimentação é influenciada pela força gravitacional entre partículas, calculada com base em suas massas e distâncias.
- PSO usa uma abordagem de atualização contínua das posições, enquanto o GSA emprega uma abordagem baseada em leis físicas para calcular as posições em cada iteração.
- GSA, devido à sua influência gravitacional, tende a ser mais eficiente na exploração global do espaço de busca, enquanto o PSO pode ser mais sensível a ficar preso em ótimos locais.
- PSO requer a configuração de parâmetros como coeficientes de inércia, fatores de aprendizagem. O GSA tem menos parâmetros para ajustar, o que simplifica a implementação.

Função Benchmark (Ackley)

Para comparação fizémos testes com esta função benchmark pedida no enunciado e depois comparámos com o GSA.

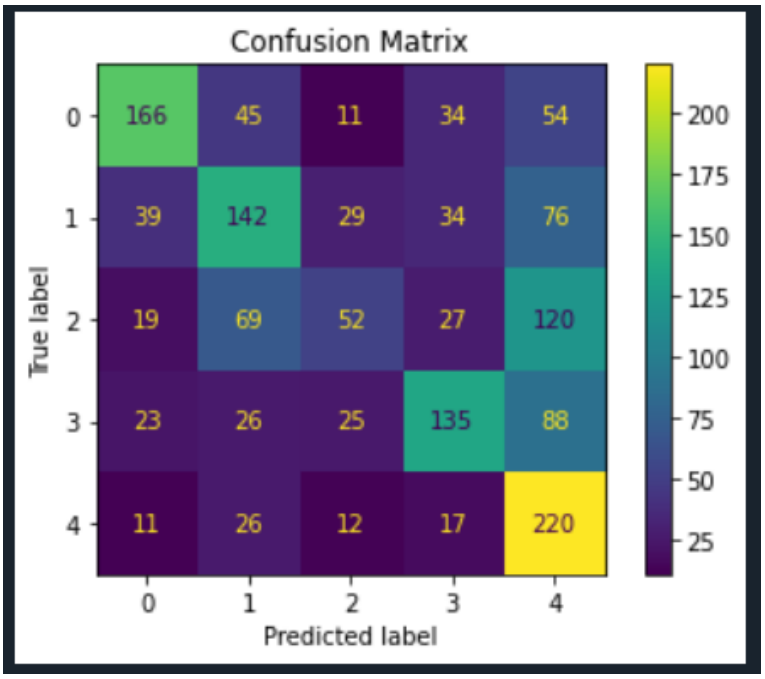

Figure 1Resultados da Função Ackley 48%

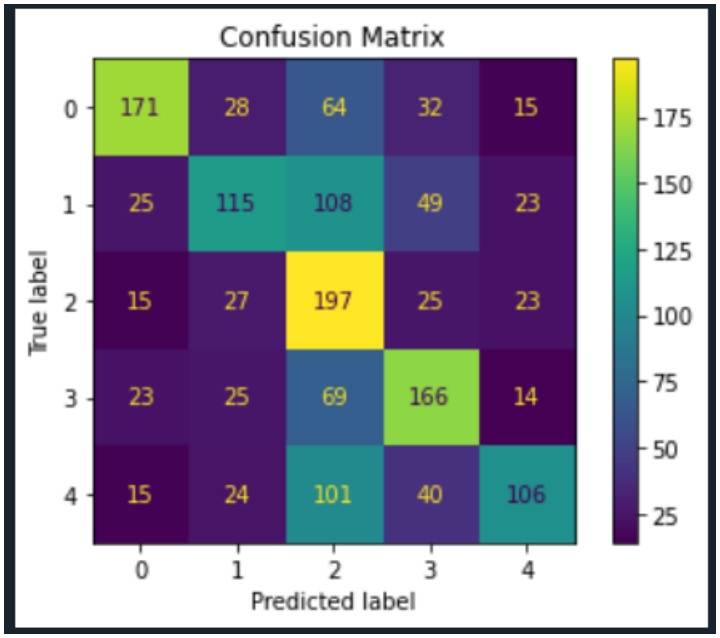

Figure 2Função GSA 50%

Otimização de Hiperparâmetros

Mudanças de Código

Na meta 1 o nosso trabalho foi realizado em Matlab o que provou ser bastante trabalhoso e ineficiente tendo em conta os trabalhos realizados em Python pelos colegas. Contudo, nesta meta o TP foi realizado na linguagem Python, o que permitiu realizar várias mudanças no Código previamente feito, obtendo assim melhor poder de computação.

Foi alterado o seguinte:

- Reshape das imagens – As imagens passaram de uma resolução de 28x28 para 100x100
- Cor das Imagens – As imagens passaram do tom acinzentado para RGB

Com estas alterações conseguimos obter valores bastante melhores do que na meta anterior.

Hiperparâmetros:

Hiperparâmetros utilizados para otimização:

- Nº de neurónios
- Nº de camadas
- Funções de otimização

Foram realizados vários testes com os seguintes valores para os hiperparâmetros:

- Nº de Neurónios: 50, 100, 150, 200, 250 e 500
- Nº de camadas: 1 a 5
- Funções de otimização: softmax, ReLu e sigmoide

Parâmetros do Gravitational Search Algorithm utilizados para otimização:

- Nº de agentes
- lb
- ub
- GO
- Iterações
- Dimensions

Foram realizados vários testes com os seguintes valores para estes parâmetros:

- Nº de agentes: 2, 3, 5, 10 e 50
- lb [1, 50]
- up [5, 500]
- GO: 2 e 3 (valor default é 9.8)
- Iterações: 5, 10, 20 e 100

Estes testes tiveram em conta x número de épocas compreendidas entre 5 e 150 tanto no modelo normal como no modelo GSA.

Para os testes a métrica de avaliação utilizada como referência foi a Accuracy.

Código Utilizado

Tendo em conta os testes realizados, estes foram os parâmetros com resultados mais significativos.

```
# GSA parameters
global dimension
num_agents = 100
lb = [1, 50]
ub = [5, 500]
G0 = 3
iteration = 200
```

Figure 3 Parâmetros globais do GSA

```
#Treinar o modelo normal
new_model.fit(img_train, labels_train, epochs=150, batch_size=32, validation_split=0.2)
```

Figure 4Parâmetros do treino do modelo

```
dimension = 2

def gravitational_force(m1, m2, r, G):
    return G * (m1 * m2) / r**2

def fitness_function(position):
    """
    Evaluate the fitness of a solution for the GSA optimization based on gravitational forces.

    Parameters:
    - position (numpy array): The position of a solution in the search space.
    - dimensionality (int): The dimensionality of the search space.

    Returns:
    - fitness (float): The fitness value of the solution.
    """
    G = 6.674 * 10**-11  # gravitational constant

    # Assuming each element of the position array represents a mass
    masses = position

    # Calculate total gravitational force among masses
    total_force = 0.0
    for i in range(dimension):
        for j in range(dimension):
            if i != j:
                r = np.abs(i - j)  # Simple distance assumption for illustration
                total_force += gravitational_force(masses[i], masses[j], r, G)

    # Fitness is the negative of the total gravitational force (since we typically maximize fitness)
    fitness = -total_force

    return fitness
```

Figure 5 Função Fitness do GSA

Conclusão e discussão de resultados

Contrariamente ao trabalho realizado na meta 1, esta meta foi bastante mais trabalhosa tendo em conta a complexidade e a valorização da mesma.
Devido a uma falta de documentação do nosso algoritmo, a nossa implementação do mesmo demorou muito mais tempo do que era suposto, comprometendo o nosso desenvolvimento do PSO pedido para esta meta.
Contudo e tendo em conta estas adversidades conseguimos implemetar o nosso algoritmo obtendo um aumento relativamente à meta passada.
Na meta passada a nossa accuracy era de 45% (média) e aumentou para 56% (em média) em praticamnete todos os testes sendo que o nosso melhor resultado foi de 58%.
Esperamos que na meta final o nosso valor de accuracy consiga estar em valores acima de 90%.

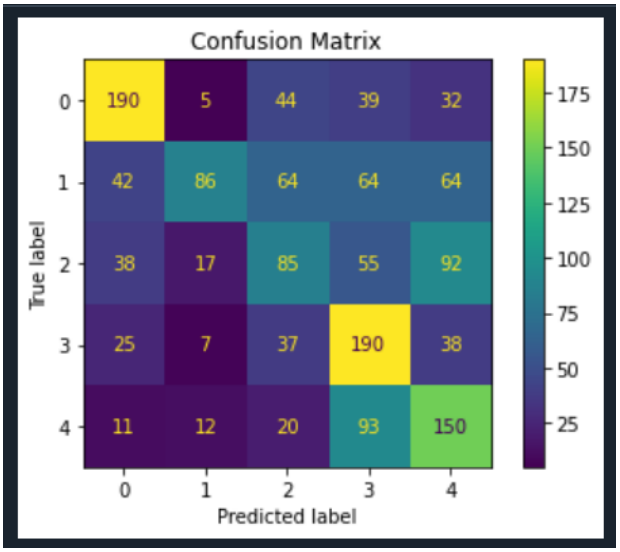

Figure 6 Teste Base 47%

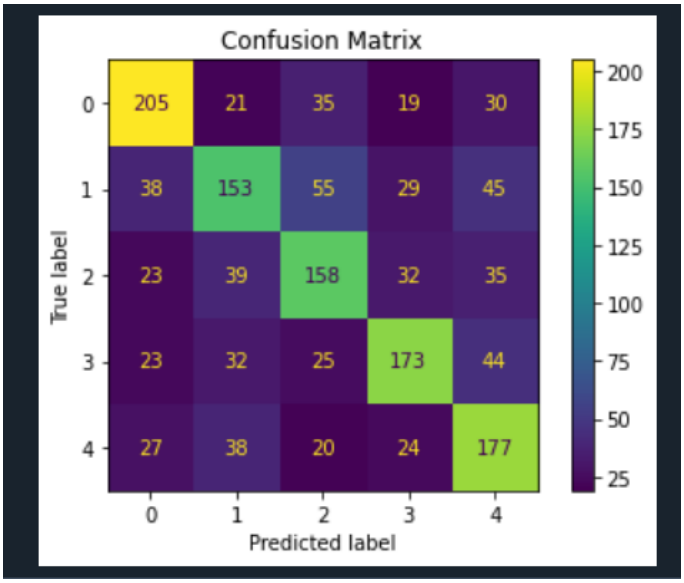

Figure 7Teste com Algoritmo 58%

Referências

Filipe Fernandes, colega da disciplina de IC.
ChatGPT - https://chat.openai.com
GitHub - https://github.com/SISDevelop/SwarmPackagePy