Particle Swarm Optimization

1st Semester of 2023-2024

Petcu Mircea
mircea.petcu@s.unibuc.ro

Tudoroiu Simona
simona.tudoroiu@s.unibuc.ro

Abstract

In this project, we used Particle Swarm Optimization (PSO) algorithm to train a weighted ensemble of models for classification and regression. We extended the application of this algorithm for training neural networks such as Multi-Layer Perceptron (MLP) and Recurrent Neural Network (RNN), in situations where their performance is not the best. Using this approach, we aimed to improve the generalization and efficiency of the MLPs in the context of tabular data and we wanted to overcome the vanished and exploding gradient problems of the Vanilla RNNs.

1 Introduction

1.1 Problem Statement

It is known that training an ensemble of weak models is beneficial for generalization. Therefore, we tried to extend this concept by training models with unequal voting rights, both in the context of classification and regression. We observe that neural networks show unsatisfactory performance on tabular data due to its irregular patterns, and their smooth decision boundary, given by the gradients, does not perform well on irregular patterns. For the difficulties associated with vanishing gradient and exploding gradient in training recurrent neural networks, we will use the Particle Swarm Optimization (PSO) algorithm to try to overcome these problems.

1.2 Contributions per Member

- Simona Tudoroiu:

  - training of the ensemble of models for classification

  - the documentation

- Mircea Petcu:

  - training of the ensemble of models for regression

  - training of the MLP

  - training of the RNN

1.3 Summary of the Approach

We tried to adapt the Particle Swarm Optimization (PSO) algorithm for building an ensemble of models for classification and regression. More precisely, we assigned each particle's position a set of weights corresponding to the classifiers, thus transforming each particle into a distinct solution. The same strategy was applied in the context of regression. As for training the neural networks, we assigned each network's parameter a dimension from the position of one particle, transforming the particles into full networks.

1.4 Why We Chose to Approach this Project

The motivation for choosing this project derived from previous success in implementing and using techniques for ensemble of models. Our goal is to expand and improve these techniques, taking them to a higher level of complexity and effectiveness.

1.5 Related Work

- 'Particle Swarm Optimization' (Kennedy and Eberhart, 1995) - The original paper on this subject. This helped us with understanding the Particle Swarm Optimization Algorithm and how to use it.

- 'Inertia Weight Strategies in Particle Swarm Optimization' (Bansal et al., 2011) - This paper shows adaptation techniques for the inertia, where we got inspiration for the project.

- 'The Parameters Selection of PSO Algorithm influencing On performance of Fault Diagnosis' (He et al., 2016) - This helped us as a starting point for choosing the hyperparameters for Particle Swarm Optimization.

- 'Good Parameters for Particle Swarm Optimization' (Pedersen, 2010) - This helped us as a starting point for choosing the hyperparameters for Particle Swarm Optimization.

1.6 Insights and Future Learning Goals in Project Context

"I studied this optimization algorithm in detail, identifying both its strengths and limitations. I concluded that it is possible to operate without gradients within neural networks, thus opening up the possibility of unlocking the potential of the vanilla recurrent network. This alternative, currently not widely used due to the previously mentioned problems, has potential in terms of its adaptability. Although we have not advanced in depth in this direction, we have identified the potential of neural networks to handle tabular data as well. Next, I prefer to direct my investigations to the field of neural networks applied to tabular data, given the possibility of training specialized networks for this purpose." - M.P.

"I have explored the Particle Swarm Optimization algorithm in detail, revealing not only its practical operation but also the underlying concepts. In the learning process, I understood that this algorithm draws inspiration from the collective behavior of animals, being the very interesting analogy from nature to an optimization algorithm. For me, this algorithm sparked interest in looking for more concepts in the field of Artificial Intelligence that are related to nature and its behavior." - S.T.

2 Approach

The method we adopted involved an initial documentation phase, where we investigated the behaviour of the algorithm to gain a deep understanding and determine how it could be implemented within our specific problem.

2.1 Particle Swarm Optimization Algorithm

The algorithm starts by initializing particles in the n-dimensional space, where $n \in \mathbb{N}$. Each particle receives a position and a velocity, this being initialized randomly, preferably as uniform as possible. The goal of the algorithm is to minimize or maximize an objective function. A swarm is composed of several particles, and the training of the swarm is carried out in a specific number of iterations. At each iteration, each individual evaluates its own objective function.

If an individual is in a personal best with respect to the objective function, it updates its personal best. If an individual has the best value of the objective function within the entire swarm, the group best objective function is updated accordingly. After the updates, each particle adapts its own velocity with the following formula:

$$vel_{n+1} = w \cdot vel_n + c_1 \cdot r_1 \cdot (pb - pos) + \\ c_2 \cdot r_2 \cdot (gb - pos),$$

where c1 represents the cognitive constant (how much personal performance matters), c2 represents the social constant (how much the group performance matters), w represents inertia (how random it should be done the actions of the particles), r1 and r2 are two random variables between 0 and 1, vel represents the velocity, pb represents the personal best, gb represents de group best and pos represents the position. The position is adapted according to the following formula:

$$pos = pos + vel$$

where pos represents the position and vel represents the velocity. This happens for all particles and the whole process is repeated at each iteration, until the number of iterations ends or a desired solution is reached.

2.2 The Ensemble of Models for Classification

To accomplish this task, we used a tabular dataset intended for a binary classification problem, in which the objective was to determine the obesity status of the individual (obese or non-obese). The dataset is quite unbalanced and is characterized by small dimensions. To achieve robust evaluation, we split the dataset into three distinct parts: training, validation, and testing.
In the first step, we used the training dataset to train base classifiers. Afterwards, we used the validation dataset to train an ensemble using the PSO algorithm. The test dataset has been reserved for performance evaluation purposes.
Our implementation is defined by assigning a distinct weight to each classifier. Every particle in our system has a position and a velocity. The position of the particle is defined by the weights associated with each classifier, thus generating a c-dimensional space, where c represents the number

of classifiers. The initial position values are selected from a uniform distribution between 0 and 1, while the velocity is initialized between -2 and 2, uniformly distributed in a c-dimensional space. Choosing a uniform distribution for the initialization of positions and velocities is intended to ensure a wide range of solutions. Our primary objective, as measured by the objective function, was model accuracy.
After recalculating the positions, we normalize them using the L1 function. Our ensemble makes predictions in a collective way, so first, we transform the weights into probabilities using the softmax function, being required to be in the interval (0, 1) and their sum to be 1. Each model predicts the label in the one-hot format, multiplied by its own weight. The final label for that data point is determined to be the maximum value above dimension 1 in the combined result of the ensemble.
As base classifiers, we trained a variety of models: XGBoost, CatBoost, Random Forest Classifier, Logistic Regression, Support Vector Machine Classifier, Gaussian Naive Bayes, K Neighbors Classifier, Gradient Boosting Classifier, Ada Boost Classifier and Extra Trees Classifier. We observed that in the case of an ensemble of models where some of them dominate the others in performance, the performance of the ensemble tends to approach but not exceed the performance of the best performing individual classifier. We also experimented with including models with approximately equal performance, but this approach did not lead to notable performance, as the ensemble did not outperform either the equal voting ensemble or the best performing individual classifier. This situation is due to the fact that when the models have roughly equal performance, the algorithm tends to balance the weights, and if there is a significant discrepancy, it tends to give a predominant vote to the model with the best performance.

2.3 The Ensemble of Models for Regression

The process of training regression models has significant similarities with training the classification models. The major difference lies in the objective function, which is no longer accuracy, but Mean Squared Error (MSE). The procedure for predicting the current value is carried out by the following formula:

$$\sum_{i=0}^{c}(w_i \cdot model_i \cdot predict(x)),$$

where c is the number of classifiers, x is the current datapoint, w represent the weight, model is the regression model, and predict(x) is the prediction function of this model. So, there is no more one-hot predicting, but directly calculated sum of the predictions.
The results show a small increase in performance for the weighted ensemble compared to the unweighted model and the most efficient regression model.

2.4 Training Multilayer Perceptron (MLP) with Particle Swarm Optimization

We adapted the Particle Swarm Optimization algorithm to train a feed-forward neural network with two hidden layers, using the same training data as for the classifiers. The network setup includes two hidden layers, each with 30 neurons. Since we train without gradients and do not require differentiable functions, we chose to use the argmax operation instead of softmax for the final logits, a more computationally efficient operation. Accuracy was used as the objective function instead of cross-entropy or other classification-specific loss functions, directly optimizing accuracy without requiring differentiable functions.
In this approach, the position of a particle represents all the parameters corresponding to that network, and a particle is represented by a neural network. Uniform initialization was similarly applied. To map the weights of the network to the vector represented by the position of a particle, we performed a flatten operation on each weight matrix of each layer and kept them in order from left to right. The PSO algorithm runs normally, and for computing the objective function for each particle, it is mapped back to the matrix form of the layers. Finally, the best particle (the solution) is selected. The results obtained were slightly better compared to training a multiclass perceptron (MLP) with the same architecture from the scikit-learn library, which uses gradient-based training.

2.5 Training Vanilla Recurrent Neural Network with Particle Swarm Optimization

Same as the method we applied in Multilayer Perceptron (MLP) training, we trained a Vanilla Recurrent Neural Network with a hidden size of 30 neurons. In this instance, the data set consisted of stock price prediction for the Apple company.

For this type of networks it is known that it has difficulties in learning because the gradients tends to become very low numbers or very high numbers, so the training phase is becoming ineffective of unstable. Using the Particle Swarm Optimization algorithm in the training process effectively addresses the vanished gradient and exploding gradient problems common to Vanilla Recurrent Neural Networks and can lead to improved results. We set a window size of 30 datapoints in the training process, and the process was similar to that of training the MLP. As the objective function, we chose mean absolute error and compared the performance to training a Vanilla Recurrent Neural Network based on gradients in PyTorch. The results obtained by the PSO approach indicated superior performance compared to gradient-based training.

2.6 Experiments

To approach the task of creating an ensemble of models for both classification and regression, we adopted two distinct strategies. In the first strategy, we trained models with approximately equal performance, while in the second, we opted for models with varying levels of performance. To train MLP (Multilayer Perceptron) networks, we used Particle Swarm Optimization. In this approach, we set up an MLP network with 2 hidden layers, each with 30 neurons, and explored various parameters for PSO. In parallel, we also trained an MLP using the Scikit-Learn library, with the same architecture, but on various numbers of epochs at each run, to evaluate the impact of the number of epochs on the performances. For RNN (Recurrent Neural Network), we adopted a method similar, training an RNN with PSO and experimenting with different parameter configurations. In parallel, we also trained an RNN in PyTorch, using the Stochastic Gradient Descent (SGD) algorithm, and varied the number of epochs to analyze its influence on the obtained results. All the results can be seen in the attached tables.

3 Limitations

Even though we have made numerous attempts to adjust the parameters of the PSO algorithm, it remains a rather sensitive algorithm to hyperparameters and requires a considerable time investment for optimization. Additionally, as previously highlighted in the documentation, it is observed that the algorithm does not achieve notable results in certain tasks.
Another important limitation is the extended time required for the algorithm to convergence in the process of training networks, compared to classical gradient-based training.

4 Conclusions and Future Work

In conclusion, although the ensemble of regression models has made small improvements in performance, there is potential for further development to achieve significant performance. Regarding the ensemble of classifiers, once the regression ensemble is developed, the same approach can be applied to the classification models. We were able to successfully train a feed-forward neural network and a vanilla recurrent neural network, thus solving the vanishing gradient and exploding gradient problem in the recurrent one and allowing the use of undifferentiated functions in the training process of the feed-forward network.
For future improvements, we can consider implementing an parametric-efficient algorithm using Particle Swarm Optimization. Given the absence of gradients in training networks with this type of algorithm and the limitations of neural networks in processing tabular data at the present time, a promising direction would be to train a specialized model for these types of data, capturing correlations between features, such as TabNet and Tab-Transformer. Another possible approach to training networks would be hybrid training, which consists in the first phase of training with gradients until an initial solution is reached, followed by overcoming a possible local optima using Particle Swarm Optimization.
Furthermore, considering the relatively small size of the trained models, the application of this algorithm can be extended to a much larger scale to truly evaluate its benefits.

References

J. C. Bansal, P. K. Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, and Ajith Abraham. 2011. Inertia weight strategies in particle swarm optimization. In *2011 Third World Congress on Nature and Biologically Inspired Computing*, pages 633–640.

Yan He, Wei Jin Ma, and Ji Zhang. 2016. The parameters selection of pso algorithm influencing on performance of fault diagnosis.

J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. 4:1942–1948 vol.4.

M. E. H. Pedersen. 2010. Good parameters for particle swarm optimization.

Metric for PSO ensemble	Metric for regular ensemble	Metric for best model
0.7662	0.7727	0.7662
155260521.1093	156300240.3966	164218254.0702

Table 1: The metric for the classification ensemble, is accuracy and the weights for it are: [0.24347927, 0.52465163, 0.2318691]. The metric for regression ensemble, is mean squared error and the weights are: [0.396 0.162 0.441].

Metric for PSO ensemble	Metric for regular ensemble	Metric for best model
0.7662	0.7662	0.7922
129303405.8766	141915955.4077	110726212.7664

Table 2: The metric for the classification ensemble, is accuracy and the weights for it are: [0.4482963 , 0.28344656, 0.26825715]. The metric for regression ensemble, is mean squared error and the weights are: [0.13 0.13 0.13 0.13 0.13 0.352].

Accuracy for PSO trained MLP	Accuracy for gradient trained MLP
0.685	0.6785

Table 3: Best results for PSO trained MLP and best results for gradient-based trained MLP from Scikit-Learn

Mean Absolute Error for PSO trained RNN	Mean Absolute Error for gradient trained RNN
122.4317	211.0051

Table 4: Best results for PSO trained RNN and best results for gradient-based trained RNN from Pytorch.