4.3 Grey wolf optimization

Optimization is essentially everywhere, from engineering design to economics and from holiday planning to Internet routing. As money, resources and time are always limited, the optimal utilization of these available resources is crucially important.

In general, an optimization problem can be written as

optimize $\boldsymbol{f_1(x), \ldots, f_i(x), \ldots, f_N(x)}$, $\boldsymbol{x = (x_1, \ldots x_d)}$

subject to,

$$\boldsymbol{h_j(x) = 0 \text{ , } (j = 1, 2 \ldots J)}$$

$$\boldsymbol{g_k(x) <= 0 \text{ , } ((k = 1 \ldots K)}$$

where $f_1, \ldots, f_N$ are the objectives, while h_j and g_k are the equality and inequality constraints, respectively. In the case when N=1, it is called single-objective optimization. When N≥2, it becomes a multi-objective optimization problem whose solution strategy is different from those for a single objective. This article mainly concerns single-objective optimization problems.

Different types of optimization algorithms

Deterministic optimization algorithms:

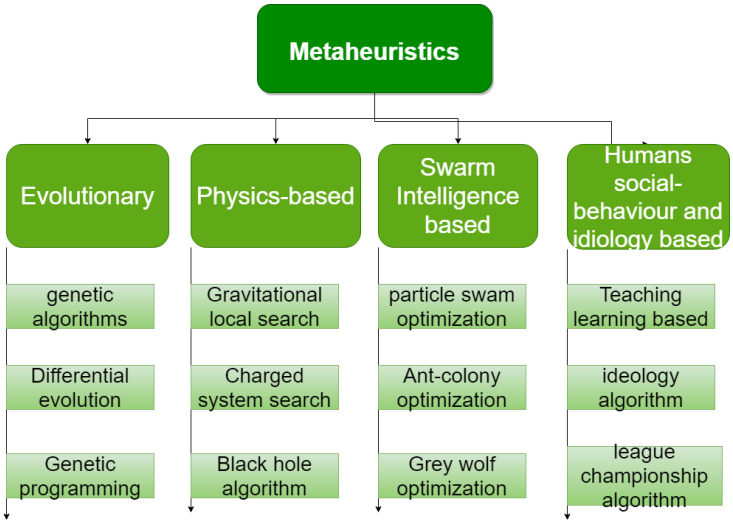
Deterministic approaches take advantage of the analytical properties of the problem to generate a sequence of points that converge to a globally optimal solution. These approaches can provide general tools for solving optimization problems to obtain a global or approximately global optimum.

Examples: linear programming, nonlinear programming, and mixed-integer nonlinear programming, etc.

Heuristics and metaheuristics:

A meta heuristic is a higher-level procedure or heuristic which aims to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem. They are used especially when incomplete or imperfect information is available or when there is limited computation capacity.

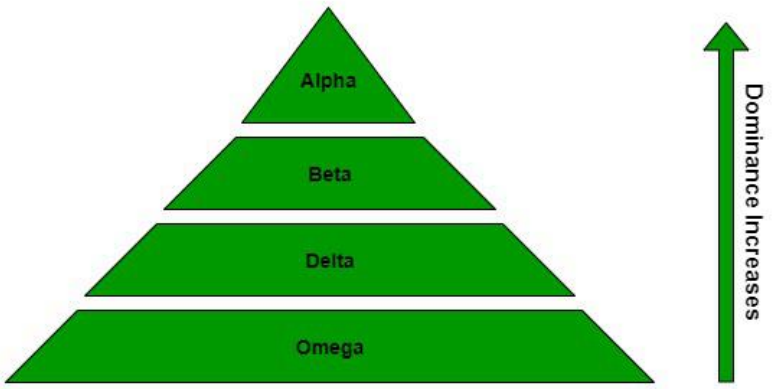
Meta heuristics make relatively few assumptions about the optimization problem being solved and so may be usable for a variety of problems.



Inspiration of the algorithm

Grey wolf optimizer (GWO) is a population-based meta-heuristics algorithm that simulates the leadership hierarchy and hunting mechanism of grey wolves in nature, and it's proposed by Seyedali Mirjalili et al. in 2014**.**

- **Grey wolves are considered apex predators, which are at the top of the food chain**
- **Grey wolves prefer to live in groups (packs), each group contain 5-12 individuals on average.**
- **All the individuals in the group have a very strict social dominance hierarchy as demonstrated in the accompanying**



- Alpha α wolf is considered the dominant wolf in the pack and his/her orders should be followed by the pack members.
- Beta β are subordinate wolves, which help the alpha in decision-making and are considered as the best candidate to be the alpha.
- Delta δ wolves have to submit to the alpha and beta, but they dominate the omega. There are different categories of delta-like Scouts, Sentinels, Elders, Hunters, Caretakers etc.

- Omega ω wolves are considered as the scapegoat in the pack, are the least important individuals in the pack and are only allowed to eat at last.

Whale Optimization Algorithm (WOA)

The step-by-step procedure to obtain an optimum value (maximum or minimum) of an objective function is called an Optimization Algorithm.

Meta-heuristic optimization algorithms are becoming more and more popular in engineering applications because they:

- rely on rather simple concepts and are easy to implement
- do not require gradient information
- can bypass local optima
- can be utilized in a wide range of problems covering different disciplines.

Many scholars and researchers have developed several metaheuristics to address complex/unsolved optimization problems**.**

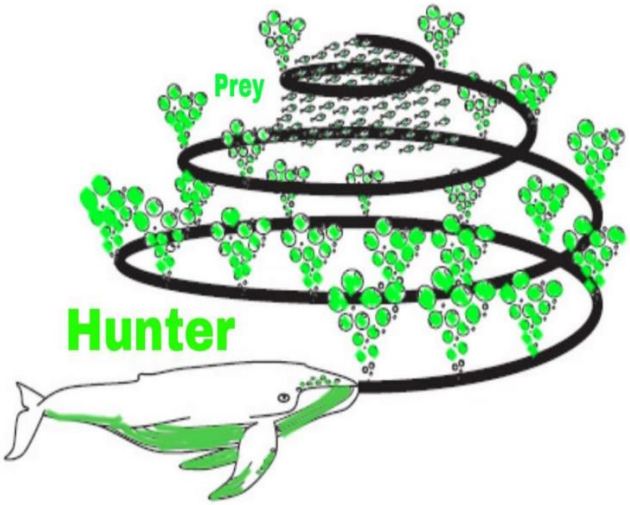
Inspiration of the algorithm:

Whale optimization algorithm (WOA): A nature inspired meta-heuristic optimization algorithm which mimics the hunting behaviour of humpback whales. The algorithm is inspired by the bubble-net hunting strategy

Foraging behavior of Humpback whales is called bubble-net feeding method. Humpback whales prefer to hunt school of krill or small fishes close to the surface. It has been observed that this foraging is done by creating distinctive bubbles along a circle or '9'-shaped path as shown in Fig. 1.

Two maneuvers associated with bubble net feeding are 'upward-spirals' and 'double-loops'.

- **In 'upward-spirals'** maneuver humpback whales dive around 12 m down and then start to create bubble in a spiral shape around the prey and swim up toward the surface.
- **'double-loops'** maneuver includes three different stages: coral loop, lobtail, and capture loop



Mathematical Model

Bubble-net feeding is a unique behaviour that can only be observed in humpback whales. In whale optimization algorithm (WOA) the spiral bubble-net feeding maneuver is mathematically modeled in order to perform optimization

- WOA simulated hunting behaviour with random or the best search agent to chase the prey
- WOA uses a spiral to simulate bubble-net attacking mechanism of humpback whales

Encircling Prey

Current best candidate solution is assumed to be closes to target prey and other solutions update their position towards the best agent

$$\vec{D} = | \vec{C}.\vec{X}_{best}(t) - \vec{X}(t) |$$

(1)

$$\vec{X}(t+1) = \vec{X}_{best}(t) - \vec{A}.\vec{D}$$

(2)

Where t indicates the current iteration, $\vec{A}$ and $\vec{C}$ are coefficient vectors, $\vec{X}_{best}$ is the position vector of the best solution, and $\vec{X}$ indicates the position vector of a grey wolf

$$\vec{A} = 2\vec{a}\overrightarrow{r_1} - \vec{a}$$

(3)

$$\vec{C} = 2\overrightarrow{r_2}$$

(4)

Where $\overrightarrow{r_1}$, $\overrightarrow{r_2}$ are random vectors in [0, 1].

Bubble-net attacking method (exploitation phase)

In order to mathematically model the bubble-net behaviour of humpback whales, two approaches are designed as follows

- **1. Shrinking encircling mechanism**
    - This behaviour is achieved by decreasing the value of $\vec{a}$. a is decreased from 2 to 0 over the course of iterations.
- **2. Spiral updating position**
    - $\overrightarrow{D'} = \mid \vec{X}_{best}(t) - \vec{X}(t) \mid$
    - $\vec{X}(t+1) = \overrightarrow{D'}.e^{bl}.\cos(2\pi l) + \vec{X}_{best}(t)$ (5)
    - l is a random number in [-1, 1]

Search for prey:

Humpback whales search randomly according to the position of each other

$$\vec{D} = \mid \vec{C}.\vec{X}_{rand}(t) - \vec{X}(t) \mid$$

(6)

$$\vec{X}(t+1) = \vec{X}_{rand}(t) - \vec{A}.\vec{D}$$

(7)

Firefly Algorithm

The Firefly Algorithm (FA) is a nature-inspired optimization algorithm based on the flashing behavior of fireflies. Fireflies use bioluminescence to communicate and attract mates. The algorithm simulates this behavior to optimize solutions to various optimization problems.

Using this knowledge about the fireflies we can idealize some of the flashing characteristics of the fireflies and develop firefly-inspired algorithms. The following rules are used to describe the standard FA(Firefly Algorithm)

- All fireflies are unisex, so one firefly will be attracted to other fireflies regardless of their sex.
- Attractiveness is proportional to a firefly's brightness. Thus for any two flashing fireflies, the less brighter one will move toward the brighter one. The attractiveness is proportional to the brightness, both of which decrease as their distance increases. If there is no brighter one than a particular firefly, it will move randomly.
- The brightness of a firefly is affected or determined by the landscape of the objective function.

For a maximization problem, the brightness can simply be proportional to the value of the objective function. Other forms of brightness can be defined in a similar way to the fitness function in genetic algorithms.

Algorithm Explanation

1. Initialize the population of fireflies with random positions in the search space.

2. Define the fitness of each firefly based on the objective function.

3. Evaluate the fitness of each firefly based on the objective function.

4. Update the position of each firefly by considering its attractiveness towards other fireflies and their light intensity.

5. Update the light intensity of each firefly based on its fitness value.

6. Repeat the movement and updating light intensity steps until the termination criteria are met.

7. Extract the best solution found after the specified number of iterations.

Multi-objective Optimization

Multi-objective optimization is the process of optimizing multiple objective functions at once. In deep learning, this means minimizing several loss terms that are representing a different requirement or goal, The outcome of such optimization is not just a single optimal solution but a set of trade-off solutions known as the Pareto front.

When We Need It

Multi-objective optimization becomes necessary in many scenarios, such as:

- **Multi-task learning:** Balancing loss across different tasks.
- **Regularization and generalization:** Combining performance with model robustness.
- **Physics-Informed Neural Networks (PINNs):** Reconciling empirical loss and physical constraints.
- **Generative modeling:** Ensuring sample quality and diversity in GANs or diffusion models.
- **Reinforcement learning:** Managing vector-valued reward functions across multiple goals.

Key Strategies for Multi-Objective Optimization

There are several approaches to tackle MOO in deep learning. These can be broadly categorized into scalarization-based methods, gradient-based methods and front-approximation strategies.

1. Scalarization Methods

Scalarization is the simplest and most commonly used method. It combines multiple objectives into a single loss function using a weighted sum:

$$L_{\text{total}} = \sum_i w_i L_i$$

Where L_i is the individual objective and $w_i$is the weight assigned to it.

Advantages

- Easy to implement with existing optimizers.
- Efficient and differentiable.
- Works well when objectives are not strongly conflicting.

Limitations

- Struggles with non-convex Pareto fronts.
- Requires careful tuning of weights.
- Only gives one solution per run, not the full Pareto front.

Variants of scalarization include Chebyshev scalarization (which minimizes the worst-case objective) and conic scalarization (for better geometry in trade-off space).

2. Gradient-Based Methods

When scalarization falls short especially when objectives are strongly conflicting, Gradient-based methods offer more precise control. Some of the most popular include:

- **Multiple Gradient Descent Algorithm (MGDA):** Finds a common descent direction that improves all objectives.
- **PCGrad:** Projects gradients to avoid destructive interference.
- **CAGrad:** Combines gradients in a convex way to ensure balanced updates.

Advantages of Gradient-Based Methods

- They adaptively balance competing losses during training.
- Eliminate the need for manual weight tuning.
- Suitable for multi-task learning where gradient conflicts are frequent.

3. Pareto Front Approximation

Rather than outputting a single compromise solution, these methods aim to approximate the entire Pareto front. Two major approaches include:

- **Evolutionary Algorithms (MOEAs):** Algorithms like NSGA-II evolve a population of models over time to explore the solution space. They are powerful but often slow and less suited for high-dimensional deep learning models.
- **Preference-Conditioned Models:** These models take in a preference vector and output a solution optimized for that trade-off. This allows users to query solutions from the Pareto front interactively.

4. Interactive and Continuation-Based Methods

Interactive methods allow human feedback or evolving objectives to guide the optimization process. Continuation-based methods use second-order approximations to iteratively trace the Pareto front. These are less common in current deep learning practice but are gaining interest, especially for scenarios requiring dynamic or adaptive trade-off control.

Applications Across Domains

Multi-objective optimization is not just a theoretical concept it's increasingly being integrated into real-world systems across domains:

- **Multi-task Learning:** MGDA and PCGrad are widely used to train neural networks that handle several tasks without one dominating the learning process.
- **Generative Models:** Diffusion models and GANs benefit from balancing realism, diversity and stability using gradient-based MOO.
- **Reinforcement Learning:** In Multi-Objective Reinforcement Learning (MORL), policies are optimized over multiple reward signals. Scalarization and preference-based policy conditioning are popular techniques.
- **PINNs and Scientific ML:** Physical consistency is often modeled as a separate loss, balanced against data loss using MOO techniques.
- **Responsible AI:** When building fair or privacy-aware models, fairness constraints or penalties are added as secondary objectives.

Challenges in Multi-Objective Optimization

Despite its promise, multi-objective deep learning still faces several open challenges:

- **Lack of standardized benchmarks:** Most datasets don't strongly exhibit conflicting objectives, making it hard to evaluate MOO methods fairly.
- **Scalability issues:** Evolutionary and Pareto front approximation methods often don't scale well to large models or datasets.
- **Dynamic trade-offs:** Current methods rarely support human-guided objective adjustment during training.
- **Non-convexity:** Many real-world Pareto fronts are non-convex, limiting the effectiveness of scalarization-based methods.

Hyperparameter Optimization

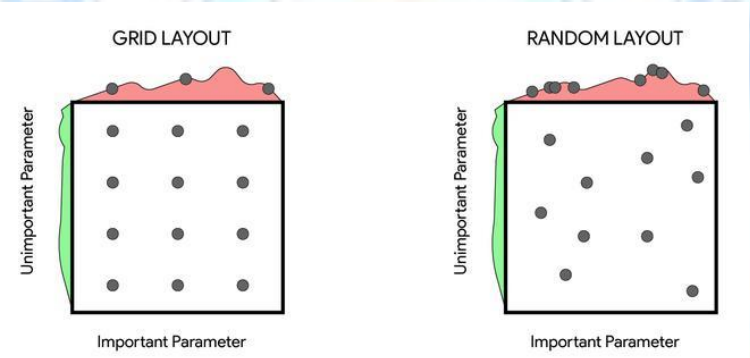
Hyperparameters are those parameters that we set for training. Hyperparameters have major impacts on accuracy and efficiency while training the model. Therefore it needed to be set accurately to get better and efficient results. Let's discuss some Hyperparameters Optimization Methods to optimize the hyperparameter.

Hyperparameters Optimization Technique

Exhaustive Search Methods

Let's first discuss some Exhaustive Search Methods to optimize the hyperparameter.

- **Grid Search:** In Grid Search, the possible values of hyperparameters are defined in the set. Then these sets of possible values of hyperparameters are combined by using Cartesian product and form a multidimensional grid. Then we try all the parameters in the grid and select the hyperparameter setting with the best result.
- **Random Search:** This is another variant of Grid Search in which instead of trying all the points in the grid we try random points. This solves a couple of problems that are in Grid Search such as we don't need to expand our search space exponentially every time add a new hyperparameter



Drawback:

Random Search and Grid Search are easy to implement and can run in parallel but here are few drawbacks of these algorithm:

- If the hyperparameter search space is large, it takes a lot of time and computational power to optimize the hyperparameter.
- There is no guarantee that these algorithms find local maxima if the sample is not meticulously done.

Bayesian Optimization:

Instead of random guess, In bayesian optimization we use our previous knowledge to guess the hyper parameter. They use these results to form a probabilistic model mapping

hyperparameters to a probability function of a score on the objective function. These probability function is defined below.

$$P\left(\frac{score(y)}{hyperparameters(x)}\right)$$

This function is also called "surrogate" of objective function. It is much easier to optimize than Objective function. Below are the steps for applying Bayesian Optimization for hyperparameter optimization:

1. Build a surrogate probability model of the objective function
2. Find the hyperparameters that perform best on the surrogate
3. Apply these hyperparameters to the original objective function
4. Update the surrogate model by using the new results
5. Repeat steps 2–4 until n number of iteration

Sequential Model-Based Optimization:

Sequential Model-Based Optimization (SMBO) is a method of applying Bayesian optimization. Here sequential refers to running trials one after another, each time improving hyperparameters by applying Bayesian probability model (surrogate).

There are 5 important parameters of SMBO:

- Domain of the hyperparameter over which .
- An objective function which outputs a score which we want to optimize.
- A surrogate distribution of objective function
- A selection function to select which hyperparameter to choose next. Generally we take Expected Improvement into the consideration
- A data structure contains history of previous (score, hyperparmeter) pairs which are used in previous iterations.

There are many different version SMBO hyperparameter optimization algorithm. These common difference between them is the surrogate functions. Some surrogate function such as Gaussian Process, Random Forest Regression, Tree Prazen Estimator. In this post we will discuss Tree Prazen Estimator below.

Tree Prazen Estimators:

Tree Prazen Estimators uses tree-structure for optimizing the hyperparameter. Many hyperprameter can be optimized by using this method such as number of layers, optimizer in the model, number of neurons in each layer. In tree prazen estimator instead of calculating

$P\left(\frac{y}{x}\right)$we calculate $P\left(\frac{x}{y}\right)$and P(y) (where y is an intermediate score that decides how good this hyperparameter values such as validation loss and x is hyperparameter).

In first of Tree Prazen Estimator, we sample the validation loss by random search in order to initialize the algorithm. Then we divide the observations into two groups: the best performing one (e.g. the upper quartile) and the rest, taking y^* as the splitting value for the two groups.

Then we calculate the probability of hyperparameter being in each of these groups such as

$$\boldsymbol{P\left(\frac{x}{y}\right)}\textbf{= f(x) if y<y* and }\boldsymbol{P\left(\frac{x}{y}\right)}\textbf{= g(x) if y>y*.}$$

The two densities and g are modelled using Parzen estimators (also known as kernel density estimators) which are a simple average of kernels centred on existing data points.

P(y) is calculated using the fact that p(y<y*)= f(y*) which defines the percentile split in the two categories.

Using Baye's rule (i.e. p(x, y) = p(y) $P\left(\frac{x}{y}\right)$), it can be shown) that the definition of expected improvements equivalent to f(x)/g(x).

In this final step we try to maximize the \frac{f(x)}{g(x)}

Drawback:

The biggest disadvantage of Tree Prazen Estimator that it selects hyperparameter independently of each other, that somehow effects the efficiency and computation required because in most of the neural networks there are relationships between different hperparameters

AutoML

Automated Machine Learning (AutoML) simplifies the machine learning development process by automating complex tasks like model selection and tuning. It helps democratize AI by making machine learning accessible to individuals and organizations with limited expertise. AutoML is increasingly used across industries to accelerate adoption and deployment of ML solutions.

- It automates model selection, training, and hyperparameter tuning.
- Reduces the need for deep data science expertise.
- Speeds up development and deployment of ML models.
- Enables wider adoption of machine learning across sectors

AutoML working

1. Problem Definition & Data Input

- User provides dataset and defines the task (e.g., classification, regression, forecasting).
- AutoML identifies data types (numerical, categorical, text, images).

2. Automated Data Preprocessing

- Handles missing values and outliers.
- Encodes categorical features and scales numerical data.
- Performs automatic feature engineering and selection.

3. Model Selection & Training

- Tries multiple algorithms (e.g., tree-based models, neural networks, boosting methods).
- Selects candidate models based on the task and dataset characteristics.

4. Hyperparameter Optimization

- Automatically tunes parameters using techniques like grid search, random search, or Bayesian optimization.
- Compares performance across configurations.

5. Evaluation & Validation

- Applies cross-validation and appropriate evaluation metrics (e.g., accuracy, F1-score, RMSE).
- Selects the best-performing model.

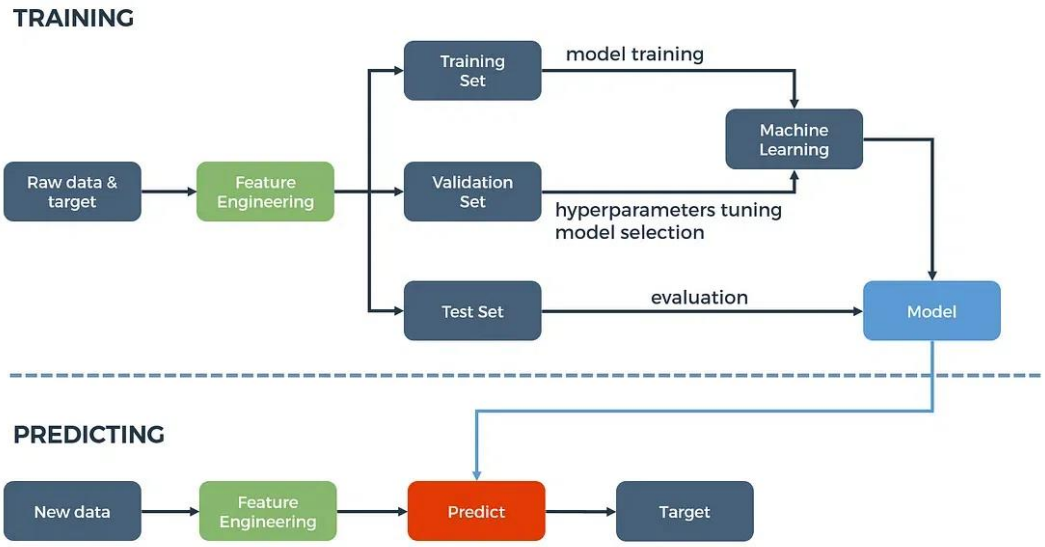
6. Deployment Preparation

- Exports the trained model.
- Generates APIs or deployment-ready artifacts (in advanced AutoML systems).

7. Monitoring & Maintenance

- Tracks model performance in production.
- Detects drift and retrains models when necessary.

AutoML automates the entire machine learning lifecycle, making it easier to apply ML techniques across domains such as tabular data analysis, image recognition, natural language processing, and forecasting.



AutoML Systems

Automation Targets in AutoML systems can automate several steps involved in building a machine-learning model. Some of them are as follows:

- **Data preparation:** Handling missing values, scaling features, encoding categorical variables, and splitting data into training and testing sets.
- **Feature engineering:** Extracting useful features from raw data.
- **Model selection & Training:** Choosing an appropriate machine learning algorithm and hyperparameters.
- **Hyperparameter tuning:** Fine-tuning the settings of a machine learning model to optimize performance.
- **Ensemble modelling:** Combining multiple models to improve performance.
- **Deployment:** Putting a trained model into production
- **Monitoring and Maintenance:** After the deployement model can be maintained and monitered based the new data.