Exploring an Evolutionary Approach for Task Generation in Meta-Learning with Neural Processes

Kerem Yoner[1]
Supervisor(s): Joery Vries[1]**, Matthijs Spaan**[1]**,**
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Kerem Yoner
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan , Joery Vries, Pradeep Murukannaiah

Exploring an Evolutionary Approach for Task Generation in Meta-Learning with Neural Processes

Kerem Yoner (5500761)
TU Delft, Netherlands

Abstract

This paper explores the application of evolutionary algorithms to enhance task generation for Neural Processes (NPs) in meta-learning. Meta-learning aims to develop models capable of rapid adaptation to new tasks with minimal data, a necessity in fields where data collection is costly or difficult. By integrating evolutionary strategies, we aim to enhance the efficiency and robustness of NPs. We evaluate our approach using 1-D function regression problems, where Genetic Algorithm generates diverse and challenging tasks. Our results show that the evolutionary approach improves learning efficiency and model performance, achieving lower Root Mean Squared Error (RMSE) compared to traditional methods.

1 Introduction

Meta-learning is an area of machine learning that focuses on developing models capable of adapting to new tasks with minimal data [Vettoruzzo *et al.*, 2023]. This capability is crucial in many real-world applications where collecting data is costly or difficult. Traditional machine learning models often require large amounts of data and long training phases, which is not feasible in scenarios such as personalized healthcare, robotics, and real-time language translation [Zou, 2023]. The data, or more generally the curriculum, that the model is trained on is crucial to model's learning efficiency and its generalization capabilities [Vanschoren, 2018]. This research addresses this problem by exploring a curriculum strategy that employs evolutionary algorithms to optimize task generation, thereby improving the model's learning efficiency and robustness.

Neural Processes (NPs), introduced by [Garnelo *et al.*, 2018], offer a flexible approach to meta-learning by combining the strengths of neural networks and Gaussian processes. NPs can learn distributions over functions, enabling them to adapt to new tasks quickly. More recently, evolutionary algorithms have been applied to reinforcement learning to create adaptive environments that challenge and improve the learning agents [Green *et al.*, 2019]. However, the application of evolutionary algorithms to meta-learning tasks, specifically using NPs, remains unexplored.

The primary focus of this research is to explore the integration of neural processes for meta-learning with evolutionary algorithms for task generation. The key research question is: How can evolutionary algorithms be utilized to optimize task generation in neural processes for meta-learning?

2 Background

2.1 Model Description

NPs are a class of models that merge the capabilities of neural networks and Gaussian processes to learn distributions over functions, allowing for rapid adaptation to new tasks. The architecture of NPs typically comprises the following components:

1. **Encoder**: This component encodes the context points (X_c, Y_c) into latent representations. Given a set of context points (x_i, y_i) for $i = 1, \ldots, N_c$, the encoder outputs a set of representations $r_i = \text{Encoder}(x_i, y_i)$. These representations are then aggregated to form a global latent representation $r = \text{Aggregate}(r_1, \ldots, r_{N_c})$.
2. **Latent Variable Model**: The latent variable model infers a distribution over the latent variables z conditioned on the aggregated context representation r. This distribution is denoted as $q(z \mid r)$. The latent variable z captures the underlying structure of the function being modeled.
3. **Decoder**: The decoder uses the latent variables z to generate predictions for the target points X_t. For each target point $x_t \in X_t$, the decoder outputs a predictive distribution $p(y_t \mid x_t, z)$.

The training of NPs involves maximizing the Evidence Lower Bound (ELBO), which balances the log-likelihood of the observed data and the Kullback-Leibler (KL) divergence between the approximate posterior $q(z|r)$ and the prior $p(z)$. The ELBO for NPs is given by:

$$\text{ELBO} = E_{q(z|r)} \left[\sum_{t=1}^{N_t} \log p(y_t|x_t, z) \right] - \text{KL}(q(z|r) \| p(z))$$

where N_t is the number of target points.

By maximizing the ELBO, NPs effectively learn to encode context information into latent variables that generalize well

to new target points, providing a powerful mechanism for meta-learning across diverse tasks. [Garnelo *et al.*, 2018] is the main source of the model that is used.

3 Methodology

The solution uses a NP model and utilizes an evolutionary curriculum strategy to enhance meta-learning. The motivation for this approach is to leverage the adaptability and efficiency of evolutionary algorithms to create diverse and challenging tasks, thereby improving the model's learning potential and generalization capabilities.

1-D function regression is chosen as the meta-learning problem, where the context and target points are sampled from a Fourier function. Every data point represents a different Fourier function, and the Fourier functions are parameterized by the number of cosine waves they include, amplitude of each wave, phase of each wave and the period.

$$y(x) = a_0 + \sum_{i=1}^{n} a_i \cos\left(2\pi i \frac{x - \phi_i}{T}\right)$$

where:

- n is the number of cosine waves.
- a_i are the amplitudes of each wave.
- ϕ_i are the phase shifts.
- T is the period.
- x is the input variable.

For comparison of the evolutionary approach, a baseline is trained by randomly sampling Fourier functions from a range of the mentioned parameters. The range of the parameters that the Fourier functions are sampled from constitutes the main task distribution. Model definition, amount of computation used for training and number of samples that the model is trained on are kept constant between the baseline and the evolutionary approach to make a fair comparison.

Main training loop for the evolutionary approach is given by the below pseudo-code.

Algorithm 1 Training Loop with Evolutionary Approach

```
Initialize variables: train_till_eval, training_steps, best, losses
while training_steps ¡ total_training_samples do
    dataloader ← create new dataset
    for each batch in dataloader do
        if training_steps ≥ total_training_samples then
            break
        end
        preprocess batch
        if train_till_eval ≥ eval_intervals and training_steps
        ≠ 0 then
            evaluate model  reset train_till_eval
        end
        train model on batch  update training_steps and
        train_till_eval
        if minimum loss improves then
            update best model
        end
        if any issue detected then
            break
        end
        if training_steps % evolution_interval == 0 and train-
        ing_steps ≠ 0 then
            apply evolutionary algorithm  break
        end
    end
end
```

The set of tasks that the model is trained on is newly created at constant intervals using the tasks generated by evolutionary algorithm. New set of tasks that the model is trained on consists of both randomly sampled Fourier functions and Fourier functions generated by the evolutionary algorithm to make sure the model isn't trained for only a specific part of the task distribution which is generated by the evolutionary algorithm.

Basic Genetic Algorithm (GA) is used as the evolutionary algorithm [Kramer, 2017]. Each candidate is represented with the parameters of a Fourier function. Fitness of each candidate is calculated by the loss of the model on the context and target points sampled from the Fourier function represented by the candidate. Below is the high-level overview of genetic algorithm:

Algorithm 2 Evolutionary Algorithm

Input: pop_size, generations, n_range, period_range, amplitude_range, phase_range, mutation_rate, top_k, retain_rate, rng, model, params
Output: best_individuals, best_fitness_log, best_fitnesses
initialize population initialize best_fitness_log
for *each generation* **do**
  evaluate fitness of population
  retain top performers
  select parents based on fitness create next generation via crossover
  apply mutations
  update population
end
evaluate final population fitness
select top$_k individuals$
return best_individuals, best_fitness_log, best_fitnesses

Population is initialized by randomly sampling Fourier functions with the given parameter ranges. Genetic operators make sure that the new candidates are not out of the bounds of the ranges given for the main task distribution.

For evaluation of the evolutionary approach and the baseline, test points are sampled from the task distribution and the performances are measured with the below metrics.

1. **Root Mean Squared Error (RMSE)**: Measures the average magnitude of the error between predicted and actual values, penalizing larger errors more heavily. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

where $\hat{y}_i$ are the predicted values and y_i are the actual values.

2. **Negative Log-Likelihood (NLL)**: Measures the log likelihood of the target points given the predicted means and standard deviations from the model. It is calculated as follows:

$$\log p(y_{\text{target}}|\hat{y}, \sigma) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_{\text{target}} - \hat{y})^2}{2\sigma^2}\right)\right)$$

The NLL is then the negative mean of these log likelihoods:

$$\text{NLL} = -\frac{1}{n}\sum_{i=1}^{n}\log p(y_{\text{target},i}|\hat{y}_i, \sigma_i)$$

where n is the number of target points, y_{target} are the actual target values, $\hat{y}$ are the predicted means, and σ are the predicted standard deviations.

Using both RMSE and ECE provides a comprehensive evaluation of the model's accuracy and reliability.

4 Experimental Setup

The hyperparameters and configurations used are detailed in this section. The implementation relies on JAX, Flax, Optax, and NetKet libraries. The model architecture is a NP with the following components:

Component	Details
Embedding	MLP([64, 64], Leaky ReLU, LayerNorm)
Projection Posterior	NonLinearMVN
Output Model	ResBlock(MLP([128, 128], Leaky ReLU, LayerNorm)), Dense(2)
Aggregator	MeanAggregator

Table 1: Model Architecture

Parameters for the GA are given in the table below. The parameter choice for the GA is done by doing multiple trials with different parameters which ensures the generation of high fitness candidates through generations.

Parameter	Value
Population Size	75
Generations	10
Mutation Rate	0.20
Top k Selection	75
Retain Rate	0.35

Table 2: Evolutionary Algorithm Hyperparameters

Below table gives the parameter ranges for the Fourier functions in the main task distribution. Fourier functions are sampled uniformly over the ranges for these parameters.

Parameter	Range
n (number of terms)	3 to 6
Period	0.05 to 2.0
Amplitude	0.1 to 10.0
Phase	0.0 to π

Table 3: Ranges for Fourier Function Parameters

Below table gives the hyper-parameters used for the model and configurations for the training loop.

Parameter	Value
Test Resolution	512
Posterior MC Samples	1
Batch Size	64
KL-Divergence Penalty	1×10^{-4}
Target Samples	32
Context Samples	64
Optimizer	AdamW (learning rate 1×10^{-3}, weight decay 1×10^{-6})
Training Steps	64000
Evolution Interval	512
Tasks by Evolution Proportion	0.3

Table 4: Model and Training Hyperparameters

5 Results and Discussion

Plots are created by aggregating the results of five experiments ran with different seeds. Red lines on the plots of the evolutionary approach represents the points at the training where a new set of tasks generated with the GA is introduced.

5.1 Analysis of Training Losses

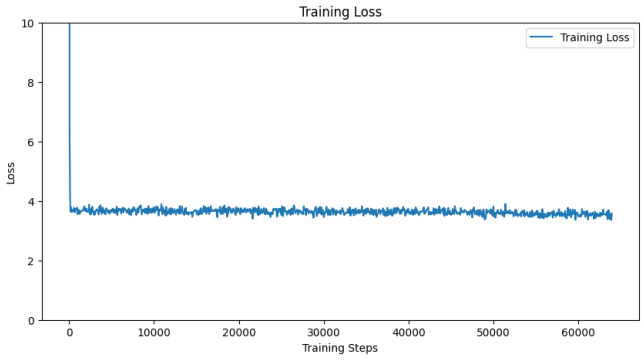

Figure 1: Training Loss for the Baseline Model

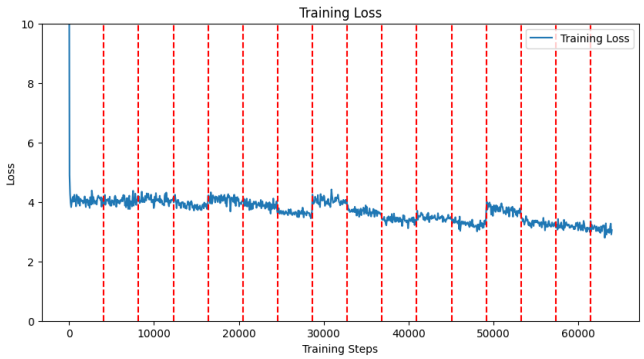

Figure 2: Training Loss for the Evolutionary Model

Training loss curve for the evolutionary approach, compared to baseline, have spikes right after the introduction of the new set of training tasks, and stabilizes after some amount of training steps. This is explained by the fact that the new set of tasks introduced includes tasks that the model had already higher loss at that step of the training. It supports that the GA algorithm works as intended, to generate tasks that the model is currently not performing well. However, the spikes are not consistent due to using also the randomly sampled tasks along with the GA generated tasks. Moreover, the tasks that the GA generates depends on the current parameters of the model, which may result in the average difficulty of the new set of tasks differing at different points.

5.2 Analysis of Test Errors

Looking at the curves for the test performances, we see similar spikes at the points of introduction of new set of tasks. This can be explained by the fact that the model is trained on data points that are focused on a specific part of the task distribution, which are generated by GA.

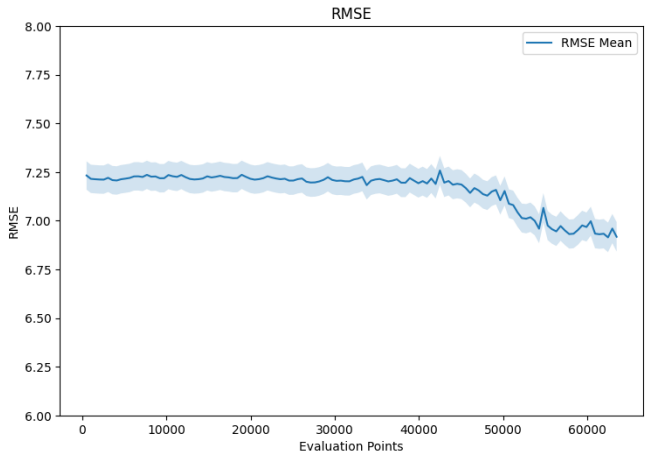

Figure 3: Test RMSE for the Baseline Model

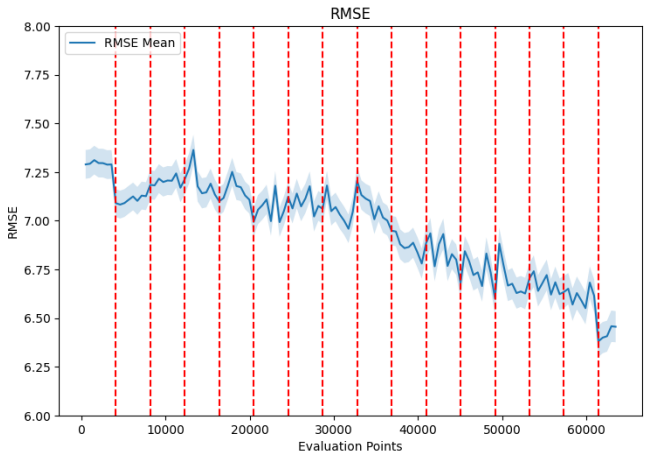

Figure 4: Test RMSE for the Evolutionary Model

Comparing the test performances of the baseline and the evolutionary approach, evolutionary approach is able to outperform the baseline in the RMSE metric. RMSE for the evolutionary approach does not follow a trend that is as stable as the baseline; however, the downward trend starts earlier than the baseline. Also, at the end of the training, RMSE is lower 10% lower compared to the baseline. Given the fact that both models are trained with same amount of samples and computation, it shows that the evolutionary approach increased the sample efficiency in the learning process.

5.3 Analysis of Test Errors

CEE metric for the evolutionary approach does not differ significantly compared to the baseline. We see a similar stability difference, which can be explained by the same reasons previously mentioned for the stability of training and test performances.

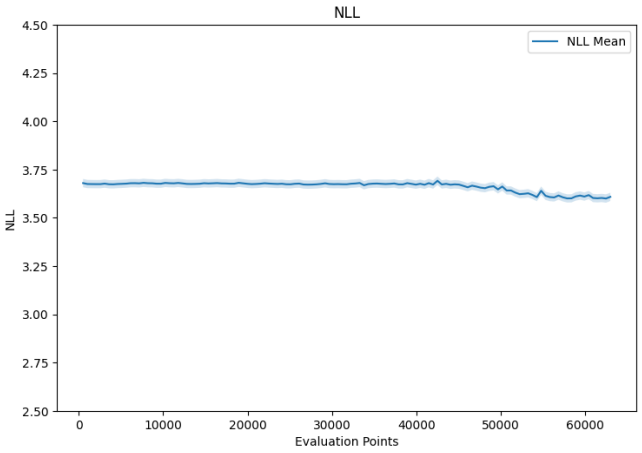

Figure 5: Test NLL for the Baseline Model

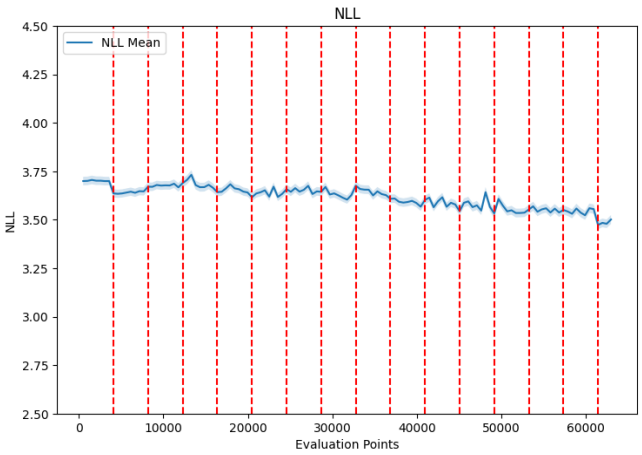

Figure 6: Test NLL for the Evolutionary Model

6 Conclusion and Future Work

In this paper, we explored the integration of Neural Processes (NPs) for meta-learning with an evolutionary algorithm-based curriculum strategy for task generation. Our findings demonstrate that the evolutionary approach enhances the model's learning efficiency compared to a baseline that employs random task sampling. The evolutionary curriculum strategy led to a more effective adaptation to new tasks, as evidenced by the lower RMSE in the evolutionary approach.

Future work can build upon this foundation by exploring several avenues: Future work can extend this foundation by exploring:

- **Diverse Task Distributions**: Apply the approach to more complex, high-dimensional tasks and real-world datasets.
- **Advanced Evolutionary Strategies**: Investigate sophisticated evolutionary strategies like multi-objective optimization and co-evolutionary algorithms.
- **Hybrid Approaches**: Combine evolutionary algorithms with other meta-learning techniques, such as reinforcement learning and generative models.
- **Theoretical Analysis**: Conduct a deeper theoretical analysis of the convergence properties and performance guarantees of the evolutionary curriculum strategy. For example, out-of-task distribution performance can be measured.

7 Responsible Research

No unethical practices were involved in this study. All data used in our experiments were synthetically generated, ensuring no privacy or consent issues. For reproducibility purposes, detailed descriptions of our methods, datasets, and code are provided to allow other researchers to replicate and validate our findings.[1]

References

[Garnelo *et al.*, 2018] Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[Green *et al.*, 2019] Michael Cerny Green, Benjamin Sergent, Pushyami Shandilya, and Vibhor Kumar. Evolutionarily-curated curriculum learning for deep reinforcement learning agents, 2019.

[Kramer, 2017] Oliver Kramer. *Genetic Algorithms*, pages 11–19. Springer International Publishing, Cham, 2017.

[Vanschoren, 2018] Joaquin Vanschoren. Meta-learning: A survey, 2018.

[Vettoruzzo *et al.*, 2023] Anna Vettoruzzo, Mohamed-Rafik Bouguelia, Joaquin Vanschoren, Thorsteinn Rögnvaldsson, and KC Santosh. Advances and challenges in meta-learning: A technical review, 2023.

[Zou, 2023] Lan Zou, editor. *Meta-learning: Theory, Algorithms and Applications*. Elsevier and MICCAI Society book series. Academic Press, an imprint of Elsevier, 2023.

[1] Github Repository: https://github.com/keremoner/meta-learning-evolution