Hyperparameter Tuning for Milvus via HOBO

Xiang Pan

September 25, 2021

Contents

1 Introduction

For solving the optimization of milvus hyperparameters, we use the Bayesian Optimization and Hyperband(BOHB)[2] as our parameter search method.

2 Methods

There are several level hyperparameters in milvus, including index-type, index-params and search-params. To get an end-to-end solution for index, we use BOHB in different level. For Index Type, to eminent the randomness of BO for index-type(which means BO may not fully explore some specific type due to init poor performance), we set two index type optimization mode(Loop and BO).

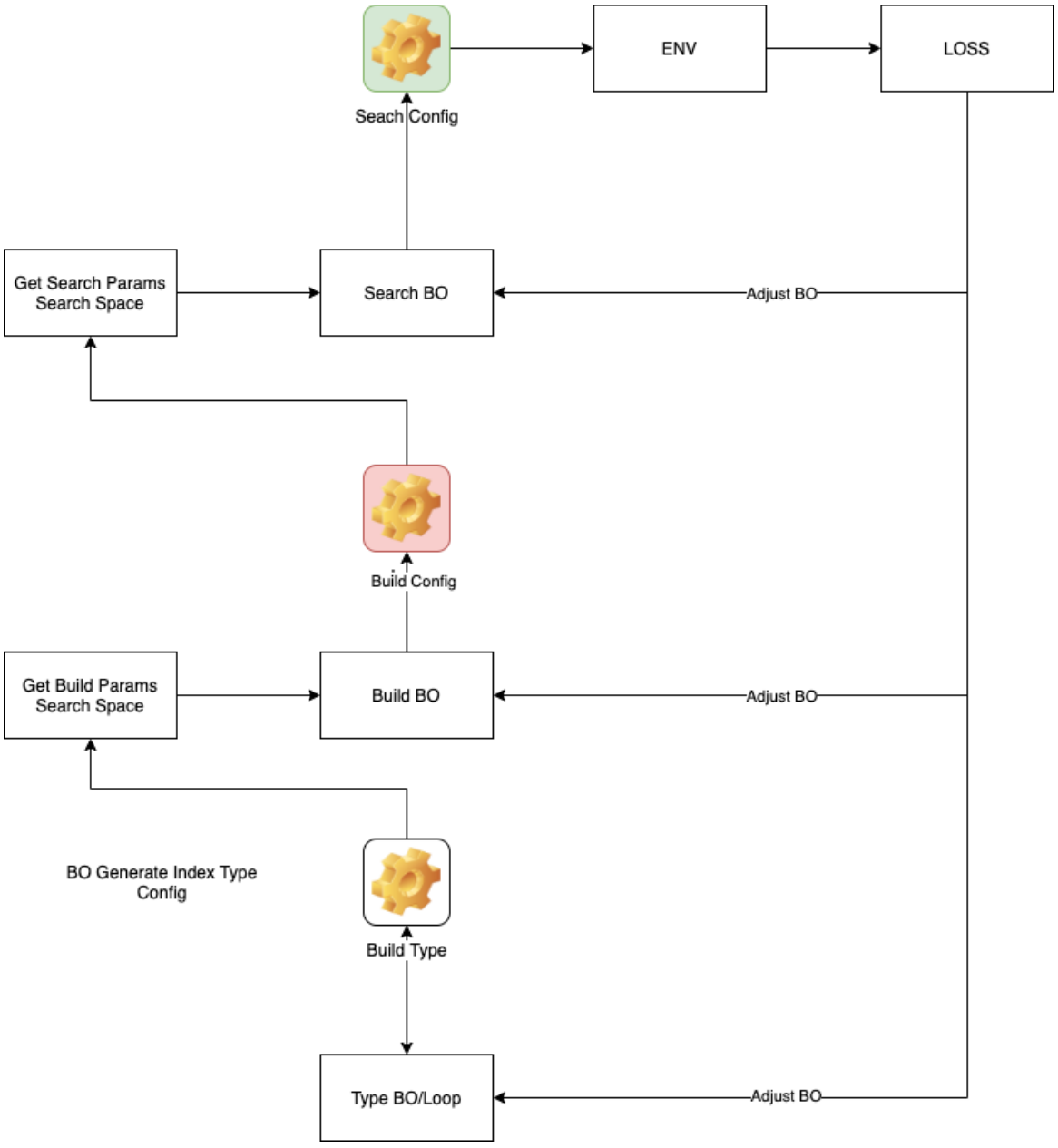

Figure 1: Flow Diagram

2.1 Loss Function

We use laplace method to convert constraint BO to unconstraint version. Our loss function is as below:

$$Loss = sign(recall, threshold) - query\_per\_sec \tag{1}$$

$$Sign(recall, threshold) = \begin{cases} recall - threshold & recall > threshold \\ \lambda * (threshold - x) & recall <= threshold, \end{cases} \tag{2}$$

Here we use λ = 100000 for Lagrange method, and *threshold* = 95.

3 Implemtation

3.1 Hardware Information

Our method is tested on server with: CPU: Intel Core i7-8700 CPU @ 4.6GHz and RAM: 32083MiB.

3.2 Core Code Annotations

3.2.1 ENV

ENV is a helper class configure basic milvs related variables.

Listing 1: ENV class config

```
class ENV():
def __init__(self, args = None):
    print("ENV")
    # docker related information
    host = '127.0.0.1'
    port = '19530'

    # get milvus client and collection_name
    self.client = Milvus(host, port)
    self.collection_name = args.collection_name

    # get query_vectors and set top_k
    self.query_groundtruth = self.get_groundtruth()
    self.query_vectors = self.get_query()
    self.top_k = 100

    # get status by curretn db
    self.index_type = None
    self.index_params = None
    self.refresh_status()

    # set datadim, which is needed by some serch constraint
    global gDataDim
    gDataDim = 128

    # based on the input type, get the default build config
    if args.op == "build_params":
```

```
        self.target_index_type = get_index_type(args.index_type)
        self.target_index_params = \
        get_default_build_config(self.target_index_type)

        is_build = False
        if self.index_type != self.target_index_type:
            is_build = True
        elif self.index_params != self.target_index_params:
            is_build = True

        if is_build:
            self.env_build_input(self.target_index_type,\
                self.target_index_params)
            self.refresh_status()

    # set search space
    self.default_build_config = get_default_build_config(self.index_type)
    self.search_configspace = get_search_configspace(self.index_type,\
        self.index_params)
    self.build_configspace = get_build_configspace(self.index_type)
```

Listing 2: Refresh Status

```
# base on user's input, get the target search space
def refresh_status(self):
    """
    refresh status
    reset index_type and index_params
    reset all config space
    """
    status, stats = self.client.get_index_info(self.collection_name)
    self.index_type = stats._index_type
    self.index_params = stats._params

    # set config space
    self.default_build_config = get_default_build_config(self.index_type)
    self.build_configspace = get_build_configspace(self.index_type)
    self.search_configspace = get_search_configspace(self.index_type,\
        self.index_params)
```

3.2.2 Search Config Class

Listing 3: Search Config Class

```
class HNSW_build_search_shared_config(object):
def __init__(self):
    self.M =  cs.IntegerUniformHyperparameter('M', 4, 64)
    self.efConstruction =  \
        cs.IntegerUniformHyperparameter('efConstruction', 8, 512)
    top_k = 100
    self.ef =  cs.IntegerUniformHyperparameter('ef', top_k, 512)
    self.configspace = \
    cs.ConfigurationSpace([self.M, self.efConstruction, self.ef], seed=123)
```

We use constant class to configure the default search space.

3.2.3 ENV Input

Listing 4: Input

```python
# given full env put
def config_input(self, config):
    # check current index type
    is_build = False

    self.refresh_status()
    if self.index_type != config['index_type']:
        is_build = True
    elif self.index_params != config['index_params']:
        is_build = True

    if is_build:
        self.env_build_input(config['index_type'] ,config['index_params'])
        self.refresh_status()


    recall, query_per_sec = self.env_search_input(config['search_params'])
    self.search_params = config['search_params']

    return recall, query_per_sec
```

For input, we check the current status and mark the change that need to be done. Once we have changed the index of current database, we refresh the ENV status.

3.3 User Input Parser

Listing 5: Input Parser

```python
if args.op == "build_type":
    build_type_search_spcae =\
    [IndexType.IVF_FLAT, IndexType.IVF_PQ, IndexType.IVF_SQ8, IndexType.HNSW]
    if args.build_type_op_method == "BO":  # BO
        index_type = \
        cs.CategoricalHyperparameter('index_type', build_type_search_spcae)
        index_type_configspace = cs.ConfigurationSpace([index_type], seed=123)
        type_opt = \
        BOHB(index_type_configspace,\
         build_type_evaluate, max_budget=10, min_budget=1)
        type_logs = type_opt.optimize()
    else:                                  # Loop
        for index_type in build_type_search_spcae:
            env.target_index_type = index_type
            env.refresh_status()
            opt = \
            BOHB(get_build_configspace(env.target_index_type), \
            build_evaluate, max_budget=10, min_budget=1)
            logs = opt.optimize()
```

The input parser is trival.

3.4 HOBO

Listing 6: HOBO

```
# Based on the HOBO class and our search space configuration, \
# we can build a BOHB object. After that, we can call the optimize() \
# method to start the optimization process.
def build_type_evaluate(params, n_iterations):
env.target_index_type = params['index_type']
env.refresh_status()
if args.build_search_share_space:
    opt = BOHB(get_build_search_shared_configspace(env.target_index_type),
               build_search_share_space_evaluate,
               max_budget=n_iterations,
               min_budget=1,
               eta = 10)
    logs = opt.optimize()
else:
    opt = BOHB(get_build_configspace(env.target_index_type),
               build_evaluate,
               max_budget=n_iterations,
               min_budget=1,
               eta = 10)
    logs = opt.optimize()
return logs.best['loss']
```

4 Results

HOBO Results

Set 25, 2021

Results

Index Type Optimization

Method	index_type	M	efConstruction	ef	recall	query_per_sec	loss
BOHB(Index Type Loop)	'HNSW'	17	445	114	99.68	16782.6	-16777.9
BOHB(Index Type BO)	'HNSW'	22	274	106	99.75	18522	-18517.2

	index_type	nlist	M	nprobe	recall	query_per_sec	loss
Grid Search	'HNSW'	4	158	200	97.11	18331.74	-18329.63

Index Parameters Optimization

IVF_FLAT

	index_type	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_FLAT'	2883	54	99.68	14911	-14906.3
Grid Search	'IVF_FLAT'	14601	101	100.0	14402.03	-14397.03

IVF_SQ8

	index_type	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_SQ8'	8405	46	98.86	13827.5	-13823.7
Grid Search	'IVF_SQ8'	5401	101	99.49	13080.62	-13076.13

IVF_PQ

	index_type	m	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_PQ'	128	3800	205	98.1	1289.00	-1285.90
Grid Search	'IVF_PQ'	64	1	1	95.08	1733.67	-7629.25

HNSW

Method	index_type	M	efConstruction	ef	recall	query_per_sec	loss
BOHB	'HNSW'	18	92	157	99.85	17868.6	-17863.8
Grid Search	'HNSW'	4	158	200	97.11	18331.74	-18329.63

5 Attemption

5.1 VAE

Trying to conpress the embedding dimension and get a more milvus-friendly representation space, we are using VAE to preprocess the given datasets.

5.1.1 Measure of Rank keeping

We define the overlap and exact-match to measure the rank-keeping performance.

$$overlap = \frac{1}{|D|} \sum_{eb \in D} \frac{\text{knn of f(eb)} \cap \text{knn of eb}}{\text{knn of eb}} \tag{3}$$

$$overlap = \frac{1}{|D|} \sum_{eb \in D} \frac{\sum_{neb \in \text{neibor of eb}} \text{rank f(neb) == rank of neb}}{k} \tag{4}$$

5.1.2 Methods

Minimum Reconstruct Error

$$Loss_{MRE}(eb) = decoder(encoder(eb)) - eb \tag{5}$$

We use $compressed\_eb = encoder(eb)$ for milvus search.

Distance Preserving

$$Loss_{RP}(e1, e2) = \|e1 - e2)\|_2 - \|encoder(e1) - encoder(e2)\|_2 \tag{6}$$

Contrastive Learning The key idea of contrastive learning is to use contrastive loss to make the embedding keep the relative distance, which "may" be useful for keeping the rank.

$$Loss_{CL}(a, p, n) = \|a - p\|_2 - \|a - n\|_2 \tag{7}$$

Similarly, we have,

$$Loss_{CL}(a, p, n, f) = \|f(a) - f(p)\|_2 - \|f(a) - f(n)\|_2 \tag{8}$$

f is the encoder function.

For a, p and n, a is the anchor, p is the positive and n is the negative. Positive and negative can be decided by their relative distance.

- If p is closer to a than n, then p is positive and n is negative.
- If p is top-k nearest neighbors of a but n is not, then p is positive and n is negative.

Our empricial result is that the loss of contrastive learning and vae can not preserving the relative distance rank between embedding.

5.2 Graph Based Message Passing

We use message passing to make the more similar embedding closer. First, we build the knn graph g for given dataset D. Over graph g, we define the following message passing function,

$$u' = u + reduction(aggregation(v, e)) \tag{9}$$

For each node u in g, v is their neighbors, e is the edge between u and v, and aggregation is the aggregation function.

We use $aggregation = e\_mul\_v$ and $reduction \in \{mean, max, min, sum\}$. The result shows that max get the best performance, which may suggest that simply average the neighbors' embedding is not enough. However, the best overlap performance is around 0.65, which is not good enough for ranking preserving.

5.3 Minimum Distortion Embedding

We use the method proposed in [1], which is to minimize the distortion of embedding rank while comperssing the embedding dimension.

References

[1] Akshay Agrawal, Alnur Ali, and Stephen Boyd. Minimum-Distortion Embedding. page 179.

[2] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. page 10.