allennlp-optuna

himkt

Nov 23, 2021

CONTENTS:

CHAPTER

ONE

INSTALLATION

You can install allennlp-optuna by `pip`.

```
pip install allennlp-optuna
```

Then you have to create `.allennlp_plugins`.

```
echo 'allennlp_optuna' >> .allennlp_plugins
```

You can check if allennlp-optuna is successfully installed by running `allennlp --help`.

```
usage: allennlp [-h] [--version]  ...

Run AllenNLP

optional arguments:
-h, --help     show this help message and exit
--version      show program's version number and exit

Commands:

    best-params  Export best hyperparameters.
    evaluate     Evaluate the specified model + dataset.
    find-lr      Find a learning rate range.
    predict      Use a trained model to make predictions.
    print-results
                Print results from allennlp serialization directories to the console.
    retrain      Train a model.
    test-install
                Test AllenNLP installation.
    train        Train a model.
    tune         Train a model.
```

Can you see `best-params`, `retrain`, and `tune` in the help? If so, congratulations! You have installed allennlp-optuna.

CHAPTER

TWO

TUTORIAL

2.1 AllenNLP configuration

2.1.1 Original configuration

Here is the example of AllenNLP configuration.

- `imdb.jsonnet`

```
local batch_size = 64;
local cuda_device = -1;
local num_epochs = 15;
local seed = 42;


local embedding_dim = 32;
local dropout = 0.5;
local lr = 1e-3;
local max_filter_size = 5;
local num_filters = 32;
local output_dim = 64;
local ngram_filter_sizes = std.range(2, max_filter_size);

{
  numpy_seed: seed,
  pytorch_seed: seed,
  random_seed: seed,
  dataset_reader: {
    lazy: false,
    type: 'text_classification_json',
    tokenizer: {
      type: 'spacy',
    },
    token_indexers: {
      tokens: {
        type: 'single_id',
        lowercase_tokens: true,
      },
    },
  },
  datasets_for_vocab_creation: ['train'],
```

(continues on next page)

(continued from previous page)

```
  train_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/train.jsonl
↪',
  validation_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/dev.
↪jsonl',
  model: {
    type: 'basic_classifier',
    text_field_embedder: {
      token_embedders: {
        tokens: {
          embedding_dim: embedding_dim,
        },
      },
    },
    seq2vec_encoder: {
      type: 'cnn',
      embedding_dim: embedding_dim,
      ngram_filter_sizes: ngram_filter_sizes,
      num_filters: num_filters,
      output_dim: output_dim,
    },
    dropout: dropout,
  },
  data_loader: {
    shuffle: true,
    batch_size: batch_size,
  },
  trainer: {
    cuda_device: cuda_device,
    num_epochs: num_epochs,
    optimizer: {
      lr: lr,
      type: 'sgd',
    },
    validation_metric: '+accuracy',
  },
}
```

2.1.2 Setup for allennlp-optuna

You have to change values of hyperparameter that you want to optimize. For example, if you want to optimize a dimensionality of word embedding, a change should be following:

```
< local embedding_dim = 32;
---
> local embedding_dim = std.parseInt(std.extVar('embedding_dim'));
```

The sample configuration looks like following:

- `imdb_optuna.jsonnet`

```
local batch_size = 64;
local cuda_device = -1;
```

(continues on next page)

(continued from previous page)

```
local num_epochs = 15;
local seed = 42;

local embedding_dim = std.parseInt(std.extVar('embedding_dim'));
local dropout = std.parseJson(std.extVar('dropout'));
local lr = std.parseJson(std.extVar('lr'));
local max_filter_size = std.parseInt(std.extVar('max_filter_size'));
local num_filters = std.parseInt(std.extVar('num_filters'));
local output_dim = std.parseInt(std.extVar('output_dim'));
local ngram_filter_sizes = std.range(2, max_filter_size);

{
  numpy_seed: seed,
  pytorch_seed: seed,
  random_seed: seed,
  dataset_reader: {
    lazy: false,
    type: 'text_classification_json',
    tokenizer: {
      type: 'spacy',
    },
    token_indexers: {
      tokens: {
        type: 'single_id',
        lowercase_tokens: true,
      },
    },
  },
  datasets_for_vocab_creation: ['train'],
  train_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/train.jsonl
↪',
  validation_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/dev.
↪jsonl',
  model: {
    type: 'basic_classifier',
    text_field_embedder: {
      token_embedders: {
        tokens: {
          embedding_dim: embedding_dim,
        },
      },
    },
    seq2vec_encoder: {
      type: 'cnn',
      embedding_dim: embedding_dim,
      ngram_filter_sizes: ngram_filter_sizes,
      num_filters: num_filters,
      output_dim: output_dim,
    },
    dropout: dropout,
  },
  data_loader: {
```

(continues on next page)

(continued from previous page)

```
    shuffle: true,
    batch_size: batch_size,
  },
  trainer: {
    cuda_device: cuda_device,
    num_epochs: num_epochs,
    optimizer: {
      lr: lr,
      type: 'sgd',
    },
    validation_metric: '+accuracy',
  },
}
```

Well done, you have completed the setup AllenNLP configuration for allennlp-optuna.

2.2 Defining search space

Next, it's time to define a search space for hyperparameter. A search space is represented as JSON element. For example, a search space for embedding dimensionality looks like following:

```
{
  "type": "int",
  "attributes": {
    "name": "embedding_dim",
    "low": 64,
    "high": 128
  }
}
```

`type` should be `int`, `float`, or `categorical`. `attributes` is arguments that Optuna takes. `name` is a name of hyperparameter. `low` and `high` are the range of a parameter. For categorical distribution, `choices` is available. For more information about `attributes`, please see the Optuna API reference (`suggest_float`, `suggest_int`, and `suggest_categorical`).

The entire example of AllenNLP configuration for allennlp-optuna is following:

- `hparams.json`

```
[
  {
    "type": "int",
    "attributes": {
      "name": "embedding_dim",
      "low": 64,
      "high": 128
    }
  },
  {
    "type": "int",
    "attributes": {
      "name": "max_filter_size",
```

(continues on next page)

(continued from previous page)

```
      "low": 2,
      "high": 5
    }
  },
  {
    "type": "int",
    "attributes": {
      "name": "num_filters",
      "low": 64,
      "high": 256
    }
  },
  {
    "type": "int",
    "attributes": {
      "name": "output_dim",
      "low": 64,
      "high": 256
    }
  },
  {
    "type": "float",
    "attributes": {
      "name": "dropout",
      "low": 0.0,
      "high": 0.5
    }
  },
  {
    "type": "float",
    "attributes": {
      "name": "lr",
      "low": 5e-3,
      "high": 5e-1,
      "log": true
    }
  }
]
```

2.3 Optimize hyperparameters by allennlp cli

2.3.1 Optimize

You can optimize hyperparameters by:

```
allennlp tune \
    imdb_optuna.jsonnet \
    hparams.json \
    --serialization-dir result \
    --study-name test
```

2.3.2 Get best hyperparameters

```
allennlp best-params \
    --study-name test
```

2.3.3 Retrain a model with optimized hyperparameters

```
allennlp retrain \
    imdb_optuna.jsonnet \
    --serialization-dir retrain_result \
    --study-name test
```

2.4 Advanced configuration for Optuna

You can choose a pruner/sample implemented in Optuna. To specify a pruner/sampler, create a JSON config file.

- optuna.json

```
{
  "pruner": {
    "type": "HyperbandPruner",
    "attributes": {
      "min_resource": 1,
      "reduction_factor": 5
    }
  },
  "sampler": {
    "type": "TPESampler",
    "attributes": {
      "n_startup_trials": 5
    }
  }
}
```

Next, we have to add *optuna_pruner* to *epoch_callbacks*.

- imdb_optuna_with_pruning.jsonnet

```
local batch_size = 64;
local cuda_device = 0;
local num_epochs = 15;
local seed = 42;

local embedding_dim = std.parseInt(std.extVar('embedding_dim'));
local dropout = std.parseJson(std.extVar('dropout'));
local lr = std.parseJson(std.extVar('lr'));
local max_filter_size = std.parseInt(std.extVar('max_filter_size'));
local num_filters = std.parseInt(std.extVar('num_filters'));
local output_dim = std.parseInt(std.extVar('output_dim'));
local ngram_filter_sizes = std.range(2, max_filter_size);
```

(continues on next page)

(continued from previous page)

```
{
  numpy_seed: seed,
  pytorch_seed: seed,
  random_seed: seed,
  dataset_reader: {
    lazy: false,
    type: 'text_classification_json',
    tokenizer: {
      type: 'spacy',
    },
    token_indexers: {
      tokens: {
        type: 'single_id',
        lowercase_tokens: true,
      },
    },
  },
  train_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/train.jsonl
↪',
  validation_data_path: 'https://s3-us-west-2.amazonaws.com/allennlp/datasets/imdb/dev.
↪jsonl',
  model: {
    type: 'basic_classifier',
    text_field_embedder: {
      token_embedders: {
        tokens: {
          embedding_dim: embedding_dim,
        },
      },
    },
    seq2vec_encoder: {
      type: 'cnn',
      embedding_dim: embedding_dim,
      ngram_filter_sizes: ngram_filter_sizes,
      num_filters: num_filters,
      output_dim: output_dim,
    },
    dropout: dropout,
  },
  data_loader: {
    shuffle: true,
    batch_size: batch_size,
  },
  trainer: {
    cuda_device: cuda_device,
    // NOTE add `optuna_pruner` here!
    epoch_callbacks: [
      {
        type: 'optuna_pruner',
      }
    ],
```

(continues on next page)

(continued from previous page)

```
    num_epochs: num_epochs,
    optimizer: {
      lr: lr,
      type: 'sgd',
    },
    validation_metric: '+accuracy',
  },
}
```

Finally, you can run optimization with pruning:

```
allennlp tune \
    imdb_optuna_with_pruning.jsonnet \
    hparams.json \
    --optuna-param-path optuna.json \
    --serialization-dir result/hpo \
    --study-name test-with-pruning
```

2.5 Hyperparameter optimization at scale!

you can run optimizations in parallel. You can easily run distributed optimization by adding an option *–skip-if-exists* to *allennlp tune* command.

```
allennlp tune \
    imdb_optuna.jsonnet \
    hparams.json \
    --optuna-param-path optuna.json \
    --serialization-dir result \
    --study-name test \
    --skip-if-exists
```

allennlp-optuna uses SQLite as a default storage for storing results. You can easily run distributed optimization **over machines** by using MySQL or PostgreSQL as a storage.

For example, if you want to use MySQL as a storage, the command should be like following:

```
allennlp tune \
    imdb_optuna.jsonnet \
    hparams.json \
    --optuna-param-path optuna.json \
    --serialization-dir result/distributed \
    --study-name test \
    --storage mysql://<user_name>:<passwd>@<db_host>/<db_name> \
    --skip-if-exists
```

You can run the above command on each machine to run multi-node distributed optimization.

If you want to know about a mechanism of Optuna distributed optimization, please see the official documentation: https://optuna.readthedocs.io/en/stable/tutorial/004_distributed.html

CHAPTER

THREE

API REFERENCE

3.1 Commands

3.1.1 Best hyperparameter utilities

allennlp_optuna.commands.best_params.**fetch_best_params**()

allennlp_optuna.commands.best_params.**show_best_params**()

class allennlp_optuna.commands.best_params.**BestParam**

3.1.2 Retraining model interface

allennlp_optuna.commands.retrain.**train_model_from_args_with_optuna**()

class allennlp_optuna.commands.retrain.**Retrain**

Retraining a model.

3.1.3 Optimization interface

allennlp_optuna.commands.tune.**tune**()

class allennlp_optuna.commands.tune.**Tune**

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX