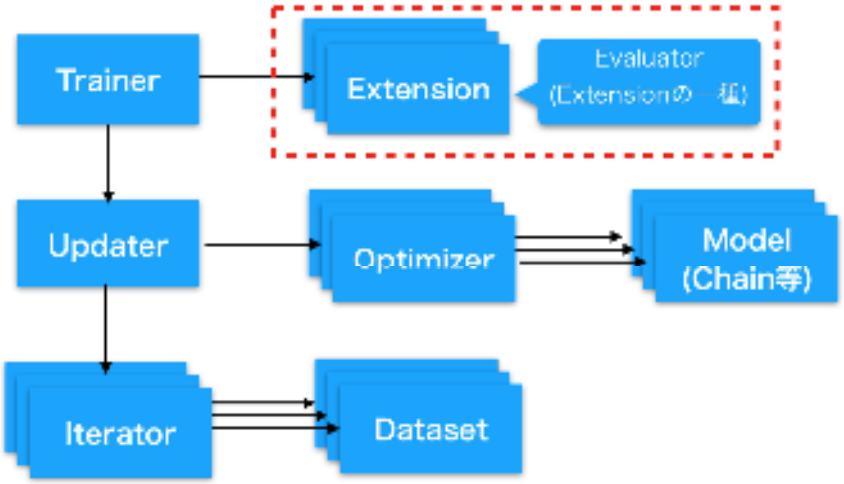
# 午睡二時四十分

2017-10-24

## chainer: Evaluatorを自作してトレーニング中のモデルの評価を柔軟に行う

### Evaluatorとは




DNNの訓練を行う中でモデルの訓練が意図通り進んでいるかを評価したくなることが多いと思います。

Chainerでは定義したモデルの訓練を行う際にそのモデルの評価を行うための仕組みとしてEvaluatorという 仕組みを持っています。

このEvaluatorは 以前解説したExtensionの一種として作られています。

本質的にはExtensionを自分で自作すればモデルの評価ももちろん可能なのですが、 Evaluatorを継承してカスタマイズすることでaccuracyやlossの計上、イテレータの状態管理などをスマートに行うことができます。

### chainer 標準のEvaluator

まず、継承元になるEvaluatorの作りを確認してみましょう。 Evaluatorについては、chainer標準の extensions.Evaluatorがかなり汎用的に作られており、自作せずに済むのならそれに越したことはないので。

Trainer extensions — Chainer 2.0.0 documentation

Evaluatorは、まず下記のようなオブジェクトを受け取ります。

- iterator: 評価用のデータセット、ミニバッチサイズ等が設定されたイテレータオブジェクト
- target: 評価対象となるモデル、もしくはモデルの列挙されたdict。
- converter: イテレータから取り出した(データ, ラベル)のタプルを訓練用のミニバッチに変換する関数
- device: 評価計算を行うために利用するGPU番号
- eval_hook: 評価前に実行される関数(なくてもok)
- eval_func: 評価を行うために呼び出される関数 (指定されない場合、targetに渡したモデルの**call**が代わりに利用される)

chainer.training.extensions.evaluator — Chainer 2.0.0 documentation

で、それぞれ主要なメソッドの動作をざっくり確認すると

プロフィール

id:mizti

読者になる 23

- __init__ :
  - 渡された引数をインスタンス変数として格納する。
  - 特に、targetにモデルのdictではなく単一のLinkが渡された場合、そのlinkを"main"という名前で辞書登録しなおす
- __call__:
  - Reporter objectを作成して、targetとして渡された各リンクのを監視対象に指定する。
  - evaluateメソッドを呼び出し、その結果をreporterを使ってreportする。 (__call__の戻り値は参照されておらず、reporter_module.reportに渡したdictが印字対象な点に注意)
- evaluate:
  - 渡されたiterator("main")からbatchを取り出す
  - 渡されたモデル("main")、もしくはeval_funcにbatchから取り出したデータ/ラベルを入力する(ここでobserverにaccuracy/lossが記録される)
  - observerに書かれたaccuracy/lossをsummaryに蓄積。
    - summaryはDictSummaryのインスタンス。DictSummaryはキー毎に投入された値の回数や平均値、二乗和を蓄積でき、平均や分散などの統計値を取り出せます
  - 最後にsummaryに蓄積された値を平均して呼び出し元(__call__)に返却

ということをしています。

つまり、標準のEvaluatorではこういうことが可能です。

- 単一のモデルと単一イテレータの評価。
- モデルの**call**を評価対象にしても良いし、eval_funcで関数を渡して評価に使っても良い

標準のEvaluatorは複数のモデルをtargetに辞書として受け取ることはできますが、 evaluateメソッドが'main'のみを用いて評価するようになっているため、複数のモデルを評価に用いることはできません。

## Evaluatorを自作する

逆に標準のEvaluatorではできないこと、例えば

- 複数のモデルやイテレータを使った評価(たとえばGANのGeneratorとDiscriminatorなど)
- accuracyやloss以外の指標値の出力(一応、eval_funcを使えば可能ですが)

などがしたい場合にはEvaluatorを自作すると良いかと思います。 Updaterの自作をした場合には対応するEvaluatorを作りたいことが多いかと思います。

## 拡張例

様々な拡張方法があると思いますので、やりたいことベースでchainer標準のEvaluatorを継承して独自のEvaluatorを定義する幾つか例を挙げていきます。

### ①とにかく指定した値をログやレポートに表示させたい

印字させたい項目を項目名をkey、スカラー値をvalueに持つ辞書をreporter_module.reportに渡せば、 とりあえず指定した値をログやレポートに表示させられます。

```
from chainer import reporter as reporter_module
from chainer.training import extensions

class MyEvaluator(extensions.Evaluator):
    def __call__(self, trainer=None):
        result = {"hoge": 4, "piyo": 88}
        reporter_module.report(result)
        return None
```

出力:

```
    {
        (略)
        "hoge": 4.0,
        "piyo": 88.0
    }
```

**アクセスの多い記事**

できるだけ丁寧にGANとDCGANを理解する

指定したファイルの更新があったらコマンドを自動実行するシェルスクリプト

chainerのupdaterを自作して複雑なネットワークを訓練する

生成モデルpix2pixを動かしてみる

seabornによる統計データ可視化（ポケモン種族値を例に）(1)

chainerのReportで受け取る辞書(dict)の各値はスカラー値であることが必須です（文字列やリストは渡せません）。

### ②複数のモデルを用いて評価を行う

影響しあう複数のモデルを並列に訓練しているなどで評価を行いたい場合、 evaluateでtargetに指定するモデルをself._targetsから取り出す際に指定するorループで順に呼び出すなどすると良いと思います。

呼び出し元:

```
trainer.extend(MyEvaluator(test_iter, {"model1": model, "model2":model2}, device=args.gpu))
```

Evaluator側:

```
class MyEvaluator(extensions.Evaluator):
    default_name="myval"
    def evaluate(self):
        #target = self._targets['main']

        summary = reporter_module.DictSummary()
        for name, target in six.iteritems(self._targets):
            iterator = self._iterators['main']
            #target = self._targets['main']
            eval_func = self.eval_func or target

            if self.eval_hook:
                self.eval_hook(self)

            if hasattr(iterator, 'reset'):
                iterator.reset()
                it = iterator
            else:
                it = copy.copy(iterator)

            #summary = reporter_module.DictSummary()
            for batch in it:
                observation = {}
                with reporter_module.report_scope(observation):
                    in_arrays = self.converter(batch, self.device)
                    with function.no_backprop_mode():
                        if isinstance(in_arrays, tuple):
                            eval_func(*in_arrays)
                        elif isinstance(in_arrays, dict):
                            eval_func(**in_arrays)
                        else:
                            eval_func(in_arrays)

                summary.add(observation)
        return summary.compute_mean()
```

色々書いてあるように見えますが、元のevaluate()からの変更点は

1. クラス変数default_nameを指定している(下記のログ出力のようにReporterがログ項目の接頭辞にしてくれます)
2. target = self.*targets['main']*ではなく*self*.targetsからループで取り出すようにしている
3. summaryの宣言をそのループの外側に書いた

だけです。

self._targetにはEvaluator定義時に指定したモデルが入っているのですが、

- 単一のモデル渡す（そのモデルが"main"という名前で__init__内で辞書登録される）
- モデルを辞書で渡す

のどちらでも良いようになっています。

このようにすることで

```
{
    （略）
    "myval/model1/loss": 0.07286249771073926,
    "myval/model1/accuracy": 0.9748000055551529,
    "myval/model2/accuracy": 0.0888000001013279,
    "myval/model2/loss": 2.3258586740493774
}
```

のように各モデルに対する評価を出力できます。

### ③モデルの評価中に独自指標値を出力

Inceptionのように一つのモデルから複数の出力がある場合など、accuracyとloss以外の指標を計測してログに出力したいことも多いと思います。そのような場合には

```
def evaluate(self):
    iterator = self._iterators['main']
    target = self._targets['main']
    eval_func = self.eval_func or target

    if self.eval_hook:
        self.eval_hook(self)

    if hasattr(iterator, 'reset'):
        iterator.reset()
        it = iterator
    else:
        it = copy.copy(iterator)

    summary = reporter_module.DictSummary()

    for batch in it:
        observation = {}
        with reporter_module.report_scope(observation):
            in_arrays = self.converter(batch, self.device)
            with function.no_backprop_mode():
                if isinstance(in_arrays, tuple):
                    eval_func(*in_arrays)
                elif isinstance(in_arrays, dict):
                    eval_func(**in_arrays)
                else:
                    eval_func(in_arrays)

        summary.add({MyEvaluator.default_name + '/currenttime': int(time.time())})
        print(observation)
        summary.add(observation)

    return summary.compute_mean()
```

- `summary.add({MyEvaluator.default_name + '/currenttime': int(time.time())})` を足しています

これは `reporter_module.report_scope(observation)` のスコープ内で `chainer.reporter.report(dict)` が呼び出されると、 observationにdictが追加されるという仕組みを用いています

例ではUnixtimeをログに出していますが、モデルや出力に関する適切な数値を渡すことで 評価中のモデルについて都合の良い指標を出力できます。

（私の場合だと例えば、文字列を認識するモデルに対して正解文字列までの編集距離を出力するのに使っていました)

### ④GANのUpdater / Evaluator

下記の記事がGAN用のUpdater / Evaluatorの対の実装例になっているので、GANを実装したい方は参考に できると思います。

chainerで少し複雑なモデルを初めて扱うことになると評価をどうしようか迷うと思いますが(実際迷いました)、このようにEvaluatorを拡張することで柔軟に対処できるようになるかと思います。

mizti 154日前



## 関連記事

2017-10-25
chainer: トレーニングモジュールの拡張方法まとめ
chainerのチュートリアルをこなしたあと、 mnistなどライブラリ...

2017-09-23
chainer: Extensionを自作してディープラーニングの訓練に独自処理を挟み込む
なぜExtensionを自作するのか Chainerのモデルのトレーニング中...

2017-09-17
chainerのupdaterを自作して複雑なネットワークを訓練する
なぜupdaterの自作が必要か chainerで様々なニューラルネットを...

2017-05-20
Tensorflow + Jupyterのsave & restore時のトラブルとその回避方法
Tensorflowでモデルを保存しようとする場合にsaveしたモデルをr...

2017-01-13
chainer: 独自datasetを定義する方法
chainerで独自データセットクラスを作るための方法を明示的に示...

« chainer: トレーニングモジュールの拡張方... chainer: Extensionを自作してディープラ... »