SKORCH

a scikit-learn wrapper for PyTorch

Marian Tietz
marian.tietz@ottogroup.com

Benjamin Bossan
b.bossan@newyorker.com

Less code

typical train loop

```
for epoch in range(args.start_epoch, args.epochs):
    start_time = time.time()
    train(train_loader, model, ema_model, optimizer, epoch, training_log)
    LOG.info("--- training epoch in %s seconds ---" % (time.time() - start_time))

    if args.evaluation_epochs and (epoch + 1) % args.evaluation_epochs == 0:
        start_time = time.time()
        LOG.info("Evaluating the EMA model:")
        ema_prec1 = validate(eval_loader, ema_model, ema_validation_log, global_step, epoch + 1)
        LOG.info("--- validation in %s seconds ---" % (time.time() - start_time))
        is_best = ema_prec1 > best_prec1
        best_prec1 = max(ema_prec1, best_prec1)
    else:
        is_best = False

    if args.checkpoint_epochs and (epoch + 1) % args.checkpoint_epochs == 0:
        save_checkpoint({
            'epoch': epoch + 1,
            'global_step': global_step,
            'arch': args.arch,
            'state_dict': model.state_dict(),
            'ema_state_dict': ema_model.state_dict(),
            'best_prec1': best_prec1,
            'optimizer' : optimizer.state_dict(),
         }, is_best, checkpoint_path, epoch + 1)
```

epoch iteration
time measurement
validation
checkpointing

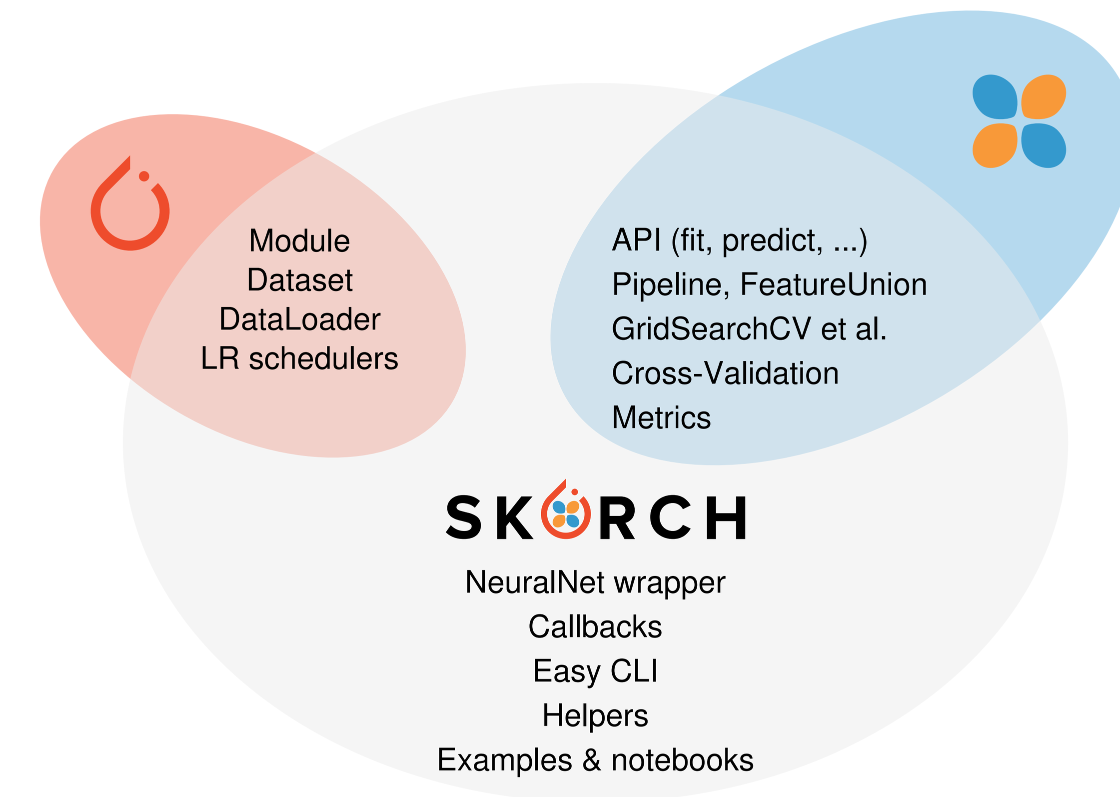
using skorch

```
net = NeuralNetClassifier(
    model,
    max_epochs=10,
    lr=0.1,
    # Shuffle training data on each epoch
    iterator_train__shuffle=True,
    callbacks=[
        Checkpoint(),
    ],
)

net.fit(X, y)
```

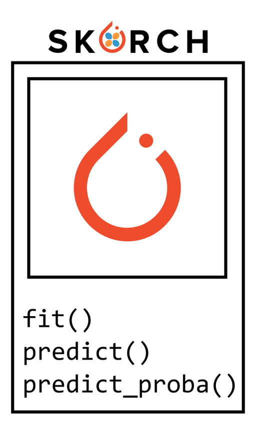
skorch handles the common boiler plate of training neural networks: iterating over your data in epochs, measuring execution time, book keeping, model validation and is easily extensible using callbacks. It proides a unified interface for setting training parameters including your model parameters, for example: optimizer__lr=0.03 or module__ema_weight=0.999.

Scope



Full scikit-learn integration

By providing the sklearn API to PyTorch modules, skorch acts as a bridge between both worlds, enabling the use of sklearn metrics, pipelines, cross-validation and grid/random search. Additionally, users gain compatibility with packages that build upon sklearn, such as imblearn and cleanlab.



You have a metric learning model and want to evaluate the model on a classification data set? No problem, just combine it with sklearn's **KNeighborsClassifier**:

```
model = Pipeline([
    ('embedder', net),
    ('clf', KNeighborsClassifier()),
])
model.fit(X, y)
```

Along with pipelines and metrics, scikit-learn integration also comes with great support for parameter searches and validation tools. Running your model through a grid search is as easy as with every other sklearn model:

```
from sklearn.model_selection import GridSearchCV

params = {
    'optimizer__lr': [0.02, 0.002],
    'module__dropout_p': [0, 0.5],
}
model = NeuralNetClassifier(MyModule)
gs = GridSearchCV(model, params, cv=5)
gs.fit(X_train, y_train)
```

skorch exposes all important parameters, including model initalization parameters, to include them in parameter searches.

PyTorch compatibility

skorch aims for full compatibility of PyTorch's features to combine the strengths of PyTorch with the versatility of scikit-learn. For example, you can train your model using your trusty torchvision `Dataset` or simply use your favorite CSV file loaded as pandas `DataFrame` - skorch handles both natively while leveraging PyTorch's `DataLoader`.

Fitting pandas DataFrames

```
df = pd.read_csv('my_data.csv')
y = df.pop('y')
net.fit(df, y)
```

Fitting PyTorch datasets

```
ds = torchvision.datasets.MNIST(...)
net.fit(ds, y=None)
```

Software statistics

3.5+

1.0+

coverage 98%

Deployment

Saving and loading trained models is easy as skorch implements storing weights and history using **pickle** or torch's **state_dict** format. This makes it easy to move models between machines, especially since skorch also handles missing CUDA devices.

Saving using `torch.save`

```
net = NeuralNet(MyModule)
net.fit(X, y)
net.save_params(
  f_params='mynet.pt',
  f_optimizer='optimizer.pt',
  f_history='history.json',
)
```

Saving using `pickle`

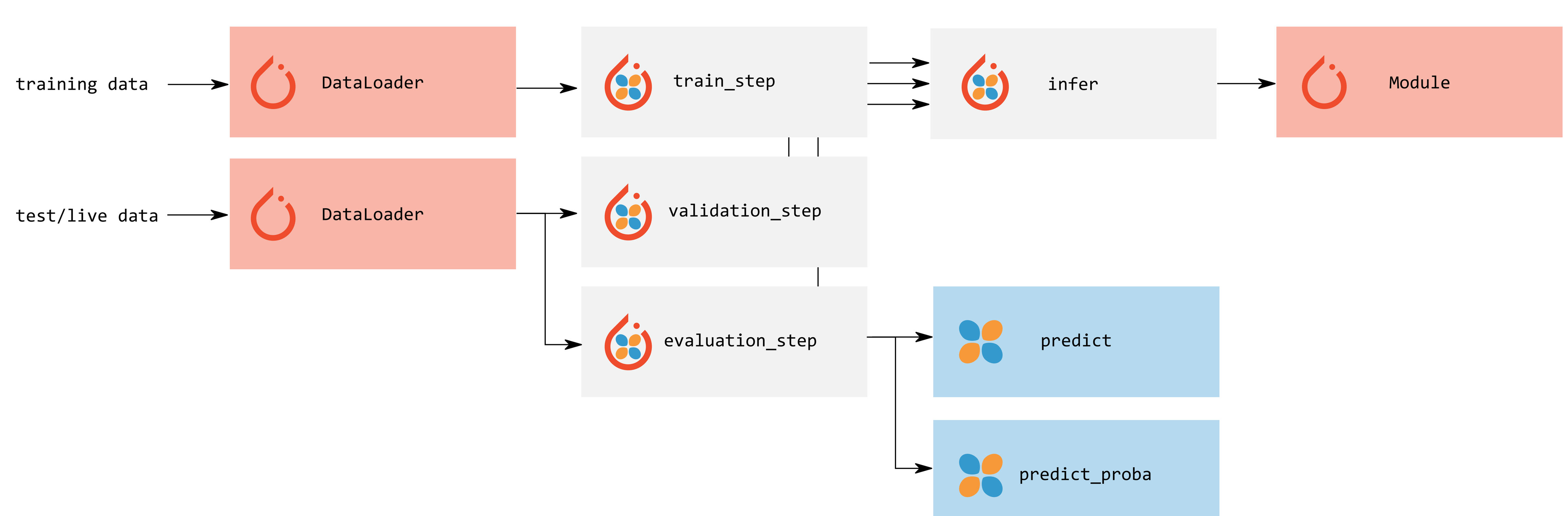
```
net = NeuralNet(MyModule)
net.fit(X, y)
with open('mynet.pkl', 'wb') as f:
    pickle.dump(net, f)
```

Loading using `pickle`

```
with open('mynet.pkl', 'wb') as f:
    net = pickle.dump(net, f)

y_pred = net.predict(X)
```

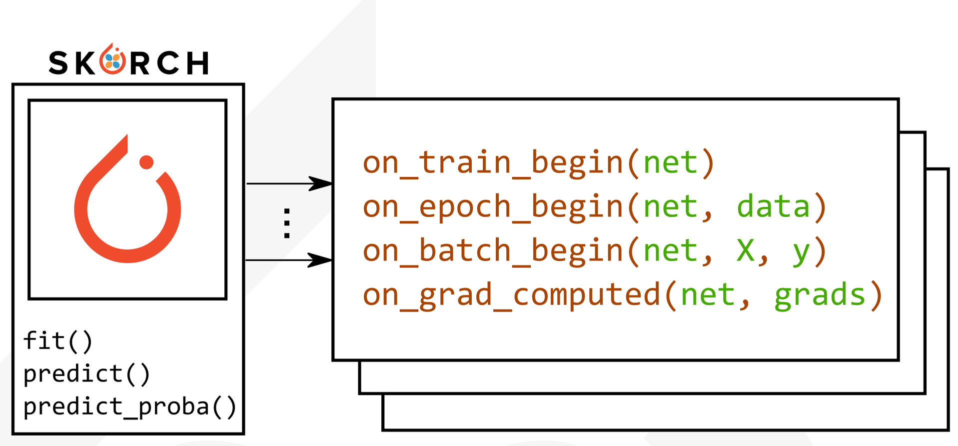
Extensible



An important design decision of skorch was to be as extensible as possible while providing sensible default implementations. In practice, this means that the NeuralNetwork class, as well as its decendants, have a method for each important step of the training and inference loop which can be overridden easily but provides a usable default implementation which fits the typical cases. As a consequence it is easy to do simple classification tasks but it is also possible to implement involved training loops by overriding the corresponding methods, `train_step()` for example, of the NeuralNet class.

Callbacks

While skorch is extensible through inheritance, some functionality is modularized into callbacks which can be plugged into the training process of your module. skorch offers a host of callbacks for epoch/batch-wise scoring, learn rate scheduling, layer initialization, tensorboard logging, gradient clipping and many more.



Combining it all

```
model = torchvision.models.resnet18(pretrained=True)
model.conv1 = torch.nn.Conv2d(1, 64, ...)
model.fc = torch.nn.Linear(512, 10, bias=True)

net = NeuralNetClassifier(
    model,
    criterion=torch.nn.CrossEntropyLoss,
    callbacks=[
        skorch.callbacks.Freezer('conv[2-9].*'),
        skorch.callbacks.ProgressBar(),
    ],
)
net.fit(mnist_train, y_train)
```

It is very simple to just fire up, e.g., jupyter notebook, download your favourite pre-trained network, modify it and run it on a classification task of your choice. There is practically no lock-in with skorch as it can handle PyTorch models as they come and it integrates nicely into the world of classical machine learning as well as deep learning.

```
In [*]: net.fit(mnist_train, y_train)

  epoch    train_loss    valid_acc    valid_loss       dur
-------  ------------  -----------  ------------  --------
      1        0.4331       0.9226        0.2597  288.3052

8% 36/469 [00:26<05:11, 1.39it/s, train_loss=0.171, valid_loss=0.382]
```