Second tutorial session

Vincent Dumoulin

January 22, 2015

Outline

1. Pylearn2 introduction
2. Porting numpy + MNIST + MLP to Pylearn2
3. Useful scripts in Pylearn2
4. Writing a new model in Pylearn2

Pylearn2 introduction

What is Pylearn2?

- Machine learning *prototyping* library
- Built on top of Theano
- Flexible and modular
- Developed at LISA

Pylearn2 is not:

- a "black box" machine learning library
- intended for novice users

Pylearn2 introduction

Pylearn2 features

- SGD loop implementation
- Various learning rules available (*e.g.* momentum, AdaDelta, RMSProp)
- Automatic monitoring of various quantities
- Automatic saving of model and parameters based on various criteria
- MLP framework allows mix-and-match between different types of layers
- YAML framework allows to train a model based on a human-readable experiment description

Porting numpy + MNIST + MLP to Pylearn2

Examples

Listing 1: MNIST + MLP in Pylearn2

```
from pylearn2.costs.cost import MethodCost
from pylearn2.datasets.mnist import MNIST
from pylearn2.models.mlp import MLP, Sigmoid, Softmax
from pylearn2.train import Train
from pylearn2.training_algorithms.sgd import SGD
from pylearn2.training_algorithms.learning_rule import Momentum, MomentumAdjustor
from pylearn2.termination_criteria import EpochCounter

train_set = MNIST(which_set='train', start=0, stop=50000)
valid_set = MNIST(which_set='train', start=50000, stop=60000)
test_set = MNIST(which_set='test')
model = MLP(nvis=784,
            layers=[Sigmoid(layer_name='h', dim=500, irange=0.01),
                    Softmax(layer_name='y', n_classes=10, irange=0.01)])
algorithm = SGD(batch_size=100, learning_rate=0.01,
                learning_rule=Momentum(init_momentum=0.5),
                monitoring_dataset={'train': train_set,
                                    'valid': valid_set,
                                    'test': test_set},
                cost=MethodCost('cost_from_X'),
                termination_criterion=EpochCounter(10))
train = Train(dataset=train_set, model=model, algorithm=algorithm,
              save_path="mnist_example.pkl", save_freq=1,
              extensions=[MomentumAdjustor(start=5, saturate=6,
                                           final_momentum=0.95)])
train.main_loop()
```

Porting numpy + MNIST + MLP to Pylearn2

Examples

Listing 2: YAML version of the previous code

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:pylearn2.datasets.mnist.MNIST {
        which_set: 'train', start: 0, stop: 50000,
    },
    model: !obj:pylearn2.models.mlp.MLP {
        nvis: 784,
        layers: [
            !obj:pylearn2.models.mlp.Sigmoid {
                layer_name: 'h', dim: 500, irange: 0.01,
            },
            !obj:pylearn2.models.mlp.Softmax {
                layer_name: 'y', n_classes: 10, irange: 0.01,
            },
        ],
    },
    extensions: [
        !obj:pylearn2.training_algorithms.learning_rule.MomentumAdjustor {
            start: 5,
            saturate: 6,
            final_momentum: .95
        },
    ],
    save_path: 'mnist_example.pkl',
    save_freq: 1,
    # ...
```

Porting numpy + MNIST + MLP to Pylearn2

Examples

Listing 3: YAML version of the previous code (continued)

```
    # ...
    algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
        batch_size: 100,
        learning_rate: 0.01,
        learning_rule: !obj:pylearn2.training_algorithms.learning_rule.Momentum {
            init_momentum: 0.5
        },
        monitoring_dataset: {
            'train': *train,
            'valid': !obj:pylearn2.datasets.mnist.MNIST {
                which_set: 'train', start: 50000, stop: 60000,
             },
            'test': !obj:pylearn2.datasets.mnist.MNIST { which_set: 'test', },
        },
        cost: !obj:pylearn2.costs.cost.MethodCost { method: 'cost_from_X', },
        termination_criterion: !obj:pylearn2.termination_criteria.EpochCounter {
            max_epochs: 10
        },
    },
}
```

Porting numpy + MNIST + MLP to Pylearn2

For a gentle introduction

http://daemonmaker.blogspot.ca/2014/10/a-first-experiment-with-pylearn2.html

YAML reference for Pylearn2

http://deeplearning.net/software/pylearn2/yaml_tutorial/index.html#yaml-tutorial

Online Pylearn2 tutorials

http://deeplearning.net/software/pylearn2/tutorial/notebook_tutorials.html#notebook-tutorials

Porting numpy + MNIST + MLP to Pylearn2

Best way to know your way around Pylearn2

`http://deeplearning.net/software/pylearn2/library/index.html#libdoc`

Useful scripts in Pylearn2

Some of the many handy scripts

- `pylearn2.scripts.plot_monitor.py`
- `pylearn2.scripts.print_monitor.py`
- `pylearn2.scripts.show_weights.py`

Writing a new model in Pylearn2

End goal

- Minimal knowledge of the inner working of Pylearn2 required
- Being able write Theano code directly

Things to implement

- `Model` subclass
- `Cost` subclass

Writing a new model in Pylearn2

Responsibilities of the `Cost` subclass

- Describe what data it needs to perform its duty and how the data should be presented
- Compute the cost expression by feeding the input to the model and receiving its output
- Differentiate the cost expression with respect to the model parameters and returns the gradients to the training algorithm

Nice to know

By subclassing `Cost` and `DefaultDataSpecsMixin`, some of the `Cost` interface is already implemented for you (differentiation and specification of the data requirements).

Writing a new model in Pylearn2

Examples

Listing 4: Cost subclass implementation mockup

```
from pylearn2.costs.cost import Cost, DefaultDataSpecsMixin


class MyCostSubclass(Cost, DefaultDataSpecsMixin):
    # Here it is assumed that we are doing supervised learning
    supervised = True

    def expr(self, model, data, **kwargs):
        space, source = self.get_data_specs(model)
        space.validate(data)

        inputs, targets = data
        outputs = model.some_method_for_outputs(inputs)
        loss = # some loss measure involving outputs and targets
        return loss
```

Writing a new model in Pylearn2

Responsibilities of the `Model` subclass

- Define what its parameters are
- Define what its data requirements are
- Do something with the input to produce an output

Protip

The `pylearn2.utils.sharedX` method initializes a shared variable with the value and an optional name you provide. This allows your code to be GPU-compatible without putting too much thought into it.

Writing a new model in Pylearn2

Examples

Listing 5: Model subclass implementation mockup

```
from pylearn2.models.model import Model

class MyModelSubclass(Model):
    def __init__(self, *args, **kwargs):
        super(MyModelSubclass, self).__init__()

        # Some parameter initialization using *args and **kwargs
        # ...
        self._params = [
            # List of all the model parameters
        ]

        self.input_space = # Some `pylearn2.space.Space` subclass
        # This one is necessary only for supervised learning
        self.output_space = # Some `pylearn2.space.Space` subclass

    def some_method_for_outputs(self, inputs):
        # Some computation involving the inputs
```

Writing a new model in Pylearn2

Examples

Listing 6: Format the data for Pylearn2

```
wget http://deeplearning.net/data/mnist/mnist.pkl.gz; \
gunzip mnist.pkl.gz; \
python -c "from pylearn2.utils import serial; \
          data = serial.load('mnist.pkl'); \
          serial.save('mnist_train_X.npy', data[0][0]); \
          serial.save('mnist_train_y.npy', data[0][1].reshape((-1, 1))); \
          serial.save('mnist_valid_X.npy', data[1][0]); \
          serial.save('mnist_valid_y.npy', data[1][1].reshape((-1, 1))); \
          serial.save('mnist_test_X.npy', data[2][0]); \
          serial.save('mnist_test_y.npy', data[2][1].reshape((-1, 1)))"
```

Writing a new model in Pylearn2

Examples

Listing 7: Cost implementation for MLP

```
import theano.tensor as T
from pylearn2.costs.cost import Cost, DefaultDataSpecsMixin


class MLPCost(DefaultDataSpecsMixin, Cost):
    supervised = True

    def expr(self, model, data, **kwargs):
        space, source = self.get_data_specs(model)
        space.validate(data)

        inputs, targets = data
        outputs = model.fprop(inputs)
        loss = -(targets * T.log(outputs)).sum(axis=1)
        return loss.mean()
```

Writing a new model in Pylearn2

Examples

Listing 8: Model implementation for MLP

```
import numpy, theano.tensor as T
from pylearn2.models.model import Model
from pylearn2.space import VectorSpace
from pylearn2.utils import sharedX


class MLP(Model):
    def __init__(self, nvis, nhid, nclasses):
        super(MLP, self).__init__()
        self.nvis, self.nhid, self.nclasses = nvis, nhid, nclasses

        self.W = sharedX(
            numpy.random.normal(scale=0.01, size=(self.nvis, self.nhid)), 'W')
        self.b = sharedX(numpy.zeros(self.nhid), name='b')
        self.V = sharedX(
            numpy.random.normal(scale=0.01, size=(self.nhid, self.nclasses)), 'V')
        self.c = sharedX(numpy.zeros(self.nclasses), name='c')
        self._params = [self.W, self.b, self.V, self.c]

        self.input_space = VectorSpace(dim=self.nvis)
        self.output_space = VectorSpace(dim=self.nclasses)

    def fprop(self, inputs):
        H = T.nnet.sigmoid(T.dot(inputs, self.W) + self.b)
        return T.nnet.softmax(T.dot(H, self.V) + self.c)
```

Writing a new model in Pylearn2

Examples

Listing 9: Corresponding YAML file

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:pylearn2.datasets.dense_design_matrix.DenseDesignMatrix {
        X: !pkl: 'mnist_train_X.npy', y: !pkl: 'mnist_train_y.npy', y_labels: 10,
    },
    model: !obj:my_model.MLP { nvis: 784, nhid: 500, nclasses: 10, },
    algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
        batch_size: 200,
        learning_rate: 1e-2,
        monitoring_dataset: {
            'train' : *train,
            'valid' : !obj:pylearn2.datasets.dense_design_matrix.DenseDesignMatrix {
                X: !pkl: 'mnist_valid_X.npy', y: !pkl: 'mnist_valid_y.npy',
                y_labels: 10,
            },
            'test' : !obj:pylearn2.datasets.dense_design_matrix.DenseDesignMatrix {
                X: !pkl: 'mnist_test_X.npy', y: !pkl: 'mnist_test_y.npy',
                y_labels: 10,
            },
        },
        cost: !obj:my_model.MLPCost {},
        termination_criterion: !obj:pylearn2.termination_criteria.EpochCounter {
            max_epochs: 15
        },
    },
}
```

Writing a new model in Pylearn2

Examples

Listing 10: Train the model

```
python -c "from pylearn2.utils import serial; \
           train_obj = serial.load_train_file('my_model.yaml'); \
           train_obj.main_loop()"
```

Writing a new model in Pylearn2

Examples

Listing 11: Bonus! Monitoring

```
# Keeps things compatible for Python 2.6
from theano.compat.python2x import OrderedDict
from pylearn2.space import CompositeSpace


class MLP(Model):
    # (Your previous code)

    def get_monitoring_data_specs(self):
        space = CompositeSpace([self.get_input_space(), self.get_target_space()])
        source = (self.get_input_source(), self.get_target_source())
        return (space, source)

    def get_monitoring_channels(self, data):
        space, source = self.get_monitoring_data_specs()
        space.validate(data)

        X, y = data
        y_hat = self.fprop(X)
        error = T.neq(y.argmax(axis=1), y_hat.argmax(axis=1)).mean()

        return OrderedDict([('error', error)])
```

Writing a new model in Pylearn2

An online tutorial can be found here

`http://deeplearning.net/software/pylearn2/theano_to_pylearn2_tutorial.html#theano-to-pylearn2-tutorial`

Writing a new model in Pylearn2

Bottom line

- Monolithic blocks of Theano code are OK as a starting point
- Flexibility and modularity can be incorporated progressively
- Nobody's expected to know Pylearn2 from A to Z (even me)
- Reading the library documentation is a good way to know how to reuse existing code (*e.g.* `Layer` subclasses)