CCW: Pylearn2

Not your grandfather's machine learning library

Pascal Lamblin
slides by Vincent Dumoulin

March 5th, 2015

Objectives

- Manage an experiment using Pylearn2
  - Anatomy of a **YAML** experiment file
  - The **train.py** script
- High-level understanding of Pylearn2
  - **Train** object
  - **TrainingAlgorithm** object
  - **Model** object
  - **Dataset** object
  - **Cost** object
  - **Monitor** object
  - **TerminationCriterion** object
  - **TrainExtension** object
  - **utils** module
  - **scripts** directory
- Extend Pylearn2 to suit your needs

What is Pylearn2?

- Machine learning **prototyping** library
- Built on top of Theano
- Easy to extend

Accompanying material

- Make sure you have access to a machine that has Pylearn2 and its dependencies installed

- The whole presentation and accompanying material can be found on Github here: `https://github.com/lamblin/ccw_tutorial`

Case study: softmax regression on MNIST digits [1]

 $\Rightarrow ([0., 0., 0., \ldots, 0., 0.], [4])$

- Predict the class: learn $p(\mathbf{y} \mid \mathbf{x})$
- $\mathbf{x} \in [0, 1]^{784}$ (28×28 pixels unrolled into a 784-dimension vector)
- $\mathbf{y} \in \{0, 1, \ldots, 9\}$

[1] Adapted from Ian Goodfellow's softmax regression iPython Notebook tutorial (`http://goo.gl/qSdAjA`)

Case study: softmax regression on MNIST digits

 $\Rightarrow ([0., 0., 0., \ldots, 0., 0.], [4])$

- Model $p(\mathbf{y} \mid \mathbf{x})$ as

$$p(\mathbf{y} \mid \mathbf{x}) = \text{softmax}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{x}^T \mathbf{W} + \mathbf{b})}{\sum_i \exp(\mathbf{x}^T \mathbf{W} + \mathbf{b})_i}$$

  with $\mathbf{W}$ a 784×10 matrix and $\mathbf{b}$ a 10-dimension vector

- Measure performance using negative log-likelihood (NLL):

$$\mathcal{L}(\mathcal{D}, \mathbf{W}, \mathbf{b}) = -\sum_{\mathbf{x},\mathbf{y} \in \mathcal{D}} \log p(\mathbf{y} \mid \mathbf{x})$$

- Train by stochastic gradient descent:

$$\theta \leftarrow \theta - \eta \nabla \mathcal{L}$$

Launch an experiment: how-to

Launch

```
$ python ${PYLEARN2_LOCATION}/scripts/train.py \
> softmax_regression.yaml
```

What happened?

Launch an experiment: **train.py** script

- Takes a YAML file as argument
- Instantiates the object(s) listed in the file
- Calls its (their) **main_loop** method

Launch an experiment: **YAML** anatomy

- Object description or list of object descriptions
- Instantiate an object with
  !obj:<package>[.<subpackage>]*.<module>.<object>
- Constructor arguments specified with
  { <name>: <value>, ..., <name>: <value>}
- Objects are instantiated recursively
- Set an anchor (reference) to an object with
  &<anchorname> !obj: ...
- Refer to an anchor with ***<anchorname>**
- For more details, see `http://deeplearning.net/software/pylearn2/yaml_tutorial/index.html`

Pylearn2 overview

- Train
  - Dataset
  - Model
  - TrainingAlgorithm
    - Monitor
    - Cost
    - TerminationCriterion
  - TrainingExtension
- utils
- scripts

Pylearn2 overview: **Train** object

- Drives the main training loop
- Responsible for
  - Starting training
  - Stopping training
  - Putting together the training algorithm, the model and the dataset
  - Managing misc. tasks before and after each training epoch
  - Saving the trained model

Pylearn2 overview: **TrainingAlgorithm** object

- Drives the epoch training loop
- Responsible for
  - Setting up the model
  - Setting up the monitor
  - Compiling the Theano function for parameter updates
  - Doing one epoch's worth of parameter updates
  - Save information about a training epoch via the monitor

Pylearn2 overview: **Model** object

- Represents the mathematical model you want to optimize
- Responsible for
  - Implementing the mapping from input to output that's described by the mathematical model
  - Describing the format of the data it expects to receive
  - Storing the model's parameters
- There are multiple model frameworks (e.g. **MLP** and **DBM**, each is specialized in a different way)

Pylearn2 overview: **Dataset** object

- Wraps around the dataset on which you train
- Common interface for all data
- Responsible for
  - Storing the data
  - Describing the format of the data it stores
  - Instantiating iterators to loop over the data
- Main subclasses are **DenseDesignMatrix** and **SparseDataset**

Pylearn2 overview: **Cost** object

- Represents a performance metric you want to maximize for the model
- Responsible for
  - Mapping the input to the cost expression as a Theano expression
  - Mapping the input to the cost gradient as a Theano expression
  - Describing the format of the input data it expects
  - Describing cost-related quantities that are to be monitored during training
- Possible to combine multiple costs using **SumOfCosts**

Pylearn2 overview: **Monitor** object

- Holds information relative to training
- Responsible for
  - Aggregating monitored quantities during training
  - Compiling Theano function mapping input data to monitored quantities
- Monitored quantities are called *channels* and are implemented in the **MonitoringChannel** class
- Can monitor over multiple datasets (e.g. training, validation and test sets)

Pylearn2 overview: **TerminationCriterion** object

- Determines when training has to stop
- Gets called between each training epoch

Pylearn2 overview: **TrainExtension** object

- Represents a misc. task to be performed during training
- Gets called through **on_monitor** (after the monitor has been called), **on_save** (after the model has been saved) and **on_setup** (right after the model has been instantiated)
- Use case: do early stopping (see **MonitorBasedSaveBest**)

Pylearn2 overview: **utils** module

- Lots of convenience functions: see
  - **utils.sharedX**
  - **utils.safe_update**
  - **utils.safe_{,i}zip**
  - **utils.safe_update**
  - **utils.function**
  - **utils.grad**
- **utils.serial**: meet your new best friend
  - **serial.load**: handles pretty much everything related to loading files in various formats
  - **serial.save**: handles pretty much everything related to saving files in various formats
  - Other serialization convenience functions are available, you are encouraged to check them out on your own

Pylearn2 overview: **scripts** directory

- **plot_monitor.py**: interactively lets you plot channels of a trained model's monitor
- **print_monitor.py**: show all channel values of a trained model's monitor after training
- **show_weights.py**: visually show a model's weights
- Once again, you are encouraged to explore the **scripts** directory on your own, lots of useful scripts are stored there

Extending the library

Pylearn2 doesn't do what you want?

- Look at the **pylearn-users** mailing list (`https://groups.google.com/d/forum/pylearn-users`), the question might have been asked before
- If nothing answers your question, ask it; there probably is something implemented but well-hidden
- If nothing suits your needs, most of the time subclassing one element of the Pylearn2 library and overriding a few methods is sufficient

Exercise 1

Softmax regression

- Have a look at `ccw_tutorial/softmax_regression.yaml`
- Launch the training of that model
- Plot some training curves
- Plot the weights of the trained model

Exercise 2

Convolutional Net

- Edit `ccw_tutorial/conv_net.yaml`
- Use only 1,000 examples for train, valid and test
- Reduce the training phase to 5 epochs maximum
- Launch the training of the model

Exercise 3

Convolutional Net with Dropout

- Edit `ccw_tutorial/conv_net.yaml`
- Change the cost to use Dropout
- Launch the training of the model

Exercise 4

Pre-training with Denoising Autoencoder

- Have a look at `dae_l1.yaml`, `dae_l2.yaml`, `dae_mlp.yaml`
- Edit `dae_mlp.yaml` to use only 1000 samples, and reduce fine-tuning to 5 epochs
- Launch the training of `dae_l1.yaml`, then `dae_l2.yaml`, then `dae_mlp.yaml`