EE–559: Information sheet

François Fleuret

https://fleuret.org/dlc/

February 15, 2018

Draft, do not distribute

1 Introduction

1.1 Content of the courses

1. What is deep learning, introduction to `torch.Tensor` and linear regression.
2. Standard machine-learning concepts and tools. Empirical risk minimization, bias-variance dilemma, PCA and k-means.
3. Perceptron, linear predictors, linear separability, feature design, multi-layer perceptrons, back-propagation, gradient descent.
4. Generalized networks, autograd, batch processing, convolutional networks, `torch.module`
5. Cross-entropy, stochastic gradient descent, weight initialization, L^1 and L^2 regularization, `torch.autograd.function`.
6. Going deeper, benefits of deep models, better initialization, drop-out, activation normalization, residual networks, benefits and usage of GPU.
7. Deep models for Computer Vision. Data-sets, performance measures and networks for classification, detection, and semantic segmentation.
8. Under the hood. Filters and inner activations. Sensitivity maps, deconvolution, guided-backprop. Maximum-activation and adversarial examples.
9. Auto-encoders, embeddings, and generative models.
10. Generative Adversarial Networks.
11. Recurrent and memory models, natural language processing.
12. **Invited lecture** Facebook.
13. **Invited lecture** Google, presentation of TensorFlow.
14. **Invited lecture** Google.

1.2 Pre-requisites

- Linear algebra (vector and Euclidean spaces),
- differential calculus (Jacobian, Hessian, chain rule),

- Python,
- basics in probabilities and statistics (discrete and continuous distributions, law of large numbers, conditional probabilities, Bayes, PCA),
- basics in optimization (notion of minima, gradient descent),
- basics in algorithmic (computational costs),
- basics in signal processing (Fourier transform, wavelets).

2 Practical sessions

2.1 Documentation

You may have to look at the python 3 and PyTorch documentations at

- https://docs.python.org/3/
- http://pytorch.org/docs/

2.2 Helper prologue for the practical sessions

You can download a python file at

https://fleuret.org/dlc/dlc_practical_prologue.py

to facilitate the loading of the data in the practical sessions.

2.2.1 Command line arguments

This prologue parses command-line arguments as follows

```
usage: dummy.py [-h] [--full] [--tiny] [--force_cpu] [--seed SEED]
                [--cifar] [--data_dir DATA_DIR]

DLC prologue file for practical sessions.

optional arguments:
  -h, --help           show this help message and exit
  --full               Use the full set, can take ages (default
                       False)
  --tiny               Use a very small set for quick checks
                       (default False)
  --force_cpu          Keep tensors on the CPU, even if cuda is
                       available (default False)
  --seed SEED          Random seed (default 0, < 0 is no seeding)
  --cifar              Use the CIFAR data-set and not MNIST
                       (default False)
  --data_dir DATA_DIR  Where are the PyTorch data located (default
                       $PYTORCH_DATA_DIR or './data')
```

It sets the default `Tensor` to `torch.cuda.FloatTensor` if cuda is available (and `--force_cpu` is not set).

2.2.2 Loading data

The prologue provides the function

`load_data(cifar = None, one_hot_labels = False, normalize = False, flatten = True)`

which downloads the data when required, reshapes the images to 1d vectors if `flatten` is `True`, narrows to a small subset of samples if `--full` is not selected, moves the Tensors to the GPU if cuda is available (and `--force_cpu` is not selected).

It returns a tuple of four tensors: `train_data`, `train_target`, `test_data`, and `test_target`.

If `cifar` is `True`, the data-base used is CIFAR10, if it is `False`, MNIST is used, if it is None, the argument `--cifar` is taken into account.

If `one_hot_labels` is `True`, the targets are converted to 2d `torch.Tensor` with as many columns as there are classes, and -1 everywhere except the coefficients $[n, y_n]$, equal to 1.

If `normalize` is `True`, the data tensors are normalized according to the mean and variance of the training one.

If `flatten` is `True`, the data tensors are flattened into 2d tensors of dimension $N \times D$, discarding the image structure of the samples. Otherwise they are 4d tensors of dimension $N \times C \times H \times W$.

2.2.3 Minimal example

The following minimal example

```
import dlc_practical_prologue as prologue

train_input, train_target, test_input, test_target = prologue.load_data()

print('train_input', train_input.size(), 'train_target', train_target.size())
print('test_input', test_input.size(), 'test_target', test_target.size())
```

prints

```
data_dir ./data
* Using MNIST
** Reduce the data-set (use --full for the full thing)
** Use 1000 train and 1000 test samples
train_input torch.Size([1000, 784]) train_target torch.Size([1000])
test_input torch.Size([1000, 784]) test_target torch.Size([1000])
```