Deep-Learning-TensorFlow Documentation

Release latest

October 09, 2016

Contents

This project is a collection of various Deep Learning algorithms implemented using the TensorFlow library. This package is intended as a command line utility you can use to quickly train and evaluate popular Deep Learning models and maybe use them as benchmark/baseline in comparison to your custom models/datasets.

CHAPTER 1

Requirements

- python 2.7
- tensorflow >= 0.8 (tested on tf 0.8 and 0.9)

CHAPTER 2

Installation

Through pip::

```
pip install yadlt
```

Through github:

- cd in a directory where you want to store the project, e.g. `/home/me`
- clone the repository: `git clone https://github.com/blackecho/Deep-Learning-TensorFlow.git`
- `cd Deep-Learning-TensorFlow/dlmodels`
- now you can configure (see below) the software and run the models!

Command line utility Configuration

- **command_line/config.py: Configuration file, used to set the path to the data directories:**
  - models_dir: directory where trained model are saved/restored
  - data_dir: directory to store data generated by the model (for example generated images)
  - summary_dir: directory to store TensorFlow logs and events (this data can be visualized using TensorBoard)

CHAPTER 4

Available models

Below you can find a list of the available models along with an example usage from the command line utility. Please note that the parameters are not optimized in any way, I just put random numbers to show you how to use the program.

Convolutional Networks

Cmd example usage::

```
python command_line/run_conv_net.py --dataset custom --main_dir convnet-models --model_name my.Awesor
```

This command trains a Convolutional Network using the provided training, validation and testing sets, and the specified training parameters. The architecture of the model, as specified by the –layer argument, is:

- 2D Convolution layer with 5x5 filters with 32 feature maps and stride of size 1
- Max Pooling layer of size 2
- 2D Convolution layer with 5x5 filters with 64 feature maps and stride of size 1
- Max Pooling layer of size 2
- Fully connected layer with 1024 units
- Softmax layer

For the default training parameters please see command_line/run_conv_net.py. The TensorFlow trained model will be saved in config.models_dir/convnet-models/my.Awesome.CONVNET.

CHAPTER 6

Recurrent Neural Network (LSTM)

Cmd example usage::

```
python command_line/run_lstm.py --dataset ptb --main_dir lstm-models --ptb_dir /path/to/ptb/dataset
```

Instructions to download the ptb dataset:

- download the dataset from here: http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz
- extract it
- provide the path to the data/ directory

CHAPTER 7

Restricted Boltzmann Machine

Cmd example usage::

```
python command_line/run_rbm.py --dataset custom --main_dir rbm-models --model_name my.Awesome.RBM --t
```

This command trains a RBM with 250 hidden units using the provided training and validation sets, and the specified training parameters. For the default training parameters please see command_line/run_rbm.py. The TensorFlow trained model will be saved in config.models_dir/rbm-models/my.Awesome.RBM.

Deep Belief Network

Stack of Restricted Boltzmann Machines used to build a Deep Network for supervised learning.

Cmd example usage::

```
python command_line/run_dbn.py --dataset mnist --main_dir dbn-models --model_name my-deeper-dbn  --ve
```

This command trains a DBN on the MNIST dataset. Two RBMs are used in the pretraining phase, the first is 784-512 and the second is 512-256. The training parameters of the RBMs can be specified layer-wise: for example we can specify the learning rate for each layer with: –rbm_learning_rate 0.005,0.1. In this case the fine-tuning phase uses dropout and the ReLU activation function.

CHAPTER 9

Deep Autoencoder

Stack of Restricted Boltzmann Machines used to build a Deep Network for unsupervised learning.

Cmd example usage::

```
python command_line/run_deep_autoencoder.py --dataset cifar10 --cifar_dir path/to/cifar10 --main_dir
```

This command trains a Deep Autoencoder built as a stack of RBMs on the cifar10 dataset. The layers in the finetuning phase are 3072 -> 8192 -> 2048 -> 512 -> 256 -> 512 -> 2048 -> 8192 -> 3072, that's pretty deep.

Denoising Autoencoder

Cmd example usage::

```
python command_line/run_autoencoder.py --n_components 1024 --batch_size 64 --num_epochs 20 --verbose
```

This command trains a Denoising Autoencoder on MNIST with 1024 hidden units, sigmoid activation function for the encoder and the decoder, and 50% masking noise. You can also initialize an Autoencoder to an already trained model by passing the parameters to its `build_model()` method. If you are using the command line, you can add the options `--weights /path/to/file.npy`, `--h_bias /path/to/file.npy` and `--v_bias /path/to/file.npy`. If you want to save the reconstructions of your model, you can add the option `--save_reconstructions /path/to/file.npy` and the reconstruction of the test set will be saved. You can also save the parameters of the model by adding the option `--save_paramenters /path/to/file`. Three files will be generated: `file-enc_w.npy`, `file-enc_b.npy` and `file-dec_b.npy`.

Stacked Denoising Autoencoder

Stack of Denoising Autoencoders used to build a Deep Network for supervised learning.

Cmd example usage::

```
python command_line/run_stacked_autoencoder_supervised.py --dae_layers 1024,784,512,256 --dae_batch_s
```

This command trains a Stack of Denoising Autoencoders 784 <-> 1024, 1024 <-> 784, 784 <-> 512, 512 <-> 256, and then performs supervised finetuning with ReLU units. This basic command trains the model on the training set (MNIST in this case), and print the accuracy on the test set. If in addition to the accuracy you want also the predicted labels on the test set, just add the option `--save_predictions /path/to/file.npy`. You can also get the output of each layer on the test set. This can be useful to analyze the learned model and to visualized the learned features. This can be done by adding the `--save_layers_output /path/to/file`. The files will be saved in the form `file-layer-1.npy`, `file-layer-n.npy`.

Stacked Deep Autoencoder

Stack of Denoising Autoencoders used to build a Deep Network for unsupervised learning.

Cmd example usage::

```
python command_line/run_stacked_autoencoder_unsupervised.py --dae_layers 512,256,128 --dae_batch_size
```

This command trains a Stack of Denoising Autoencoders 784 <-> 512, 512 <-> 256, 256 <-> 128, and from there it constructs the Deep Autoencoder model. The final architecture of the model is 784 <-> 512, 512 <-> 256, 256 <-> 128, 128 <-> 256, 256 <-> 512, 512 <-> 784. If you want to get the reconstructions of the test set performed by the trained model you can add the option `--save_reconstructions /path/to/file.npy`. Like for the Stacked Denoising Autoencoder, you can get the layers output by calling `--save_layers_output_test /path/to/file` for the test set and `--save_layers_output_train /path/to/file` for the train set. The Deep Autoencoder accepts, in addition to train validation and test sets, reference sets. These are used as reference samples for the model. For example, if you want to reconstruct frontal faces from non-frontal faces, you can pass the non-frontal faces as train/valid/test set and the frontal faces as train/valid/test reference. If you don't pass reference sets, they will be set equal to the train/valid/test set.

CHAPTER 13

MultiLayer Perceptron

Just train a Stacked Denoising Autoencoder of Deep Belief Network with the –do_pretrain false option.

CHAPTER 14

Utilities

Each model has the following utilities:

- `--seed n`: set numpy and tensorflow random number generators to n
- `--restore_previous_model`: restore a previously trained model with the same `model_name` and model architecture of the current model. Note: when using this feature with models that support pretraining (e.g. stacked_denoising_autoencoder) you should keep the `--do_pretrain` option to true and set the `--num_epochs` option to 0.

CHAPTER 15

TODO list

- Add Performace file with the performance of various algorithms on benchmark datasets
- Variational Autoencoders
- Reinforcement Learning implementation (Deep Q-Learning)