Deep-Learning-TensorFlow Documentation

Release stable

April 20, 2016

Contents

This repository 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

tensorflow >= 0.6 (tested on tensorflow 0.6, 0.7.1 and 0.8)

CHAPTER 2

Configuration

- 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 3

Available models

CHAPTER 4

Convolutional Networks

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 5

Restricted Boltzmann Machine

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.

CHAPTER 6

Deep Belief Network

Example usage:

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

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 7

Deep Autoencoder

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.

CHAPTER 8

Denoising Autoencoder

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. The –weight_images 200 save 200 random hidden units as images in config.data_dir/dae-models/img/ so that you can visualize the learned filters.

CHAPTER 9

Stacked Denoising Autoencoder

Example usage:

> python command_line/run_stacked_autoencoder.py –layers 1024,784,512,256 –batch_size 25 –num_epochs 5 –verbose 1 –corr_type masking –corr_frac 0.0 –finetune_learning_rate 0.002 –finetune_num_epochs 25 –opt momentum –momentum 0.9 –learning_rate 0.05 –enc_act_func sigmoid –finetune_act_func relu –dropout 0.7

This command trains a Stack of Denoising Autoencoders 784 <-> 1024, 1024 <-> 784, 784 <-> 512, 512 <-> 256, and then performs supervised finetuning with ReLUs.

CHAPTER 10

MultiLayer Perceptron

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

CHAPTER 11

TODO list

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