

adv-learning

This repository contains code, datasets, and writing related to the "adversarial learning" project.

Code

At the moment, the two most important bodies of code are contained in `code/pylearn-train.py` and `code/pylearn-attack.py`. As suggested by the filenames, each relies heavily on **pylearn2**, which is a Python library with extensive support for deep learning architectures. `pylearn-train.py` supports training a new model, described in a YAML file. `pylearn-attack.py` runs the attack on a trained model and an image.

Installation

All of the code so far is written in Python 2. It requires three primary packages to run: **opencv-python**, **theano**, and **pylearn2**. **opencv-python** contains Python bindings for OpenCV, a library with image processing and manipulation routines. **theano** is a Python library that allows one to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It has support for compilation to GPU code. **pylearn2** is a Python machine learning library built on top of Theano, which supports rapid development and customization of new deep learning architectures.

First, make sure Python 2.x is installed and working properly. The code in the repository has been tested most extensively with Python 2.7.5.

Installing the dependencies on a Mac

opencv-python

I have had the best luck installing **opencv-python** via Homebrew. If Homebrew isn't already installed, make it so with the following command:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/homebrew/go/install)"
```

Optionally, run `brew doctor` to check for errors. Next install the Homebrew science add-on:

```
brew tap homebrew/science
```

Now to actually install OpenCV:

```
brew install opencv
```

If it gives you problems, try the following alternate command:

```
brew install opencv --env=std
```

To test that everything installed correctly, fire up Python and run the following:

```
>>> import cv
>>> import cv2
```

If you experience any other problems, check out the comments section at <http://www.jeffreythompson.org/blog/2013/08/22/update-installing-opencv-on-mac-mountain-lion/>

theano

theano is available on PyPI, at <https://pypi.python.org/pypi/Theano>. Download the tarball (the code in this repository was developed with Theano 0.6.0), unpack it, and run

```
python setup.py install
```

You may need to run the setup under `sudo` if you wish for the packages to install globally.

Note: If you have `easy_install` or `pip` on your system, you can simply run `easy_install Theano` or `pip Theano`, although if you do this after version 0.6.0 has been superseded, you might not get a compatible version.

You can test the installation by opening Python from the command line, and executing:

```
>>> import theano
```

If it doesn't complain, everything so far should work.

pylearn2

pylearn2 is not on PyPI yet, so you need to check out the latest development version from the main repository.

```
git clone git://github.com/lisa-lab/pylearn2.git
```

Note that it is probably best to do this from outside of the `adv-learning` repository directory tree. Once everything has downloaded, enter the main directory and run

```
python setup.py develop
```

Again, you may need to do this as superuser, depending on your preferences. You can test the

installation by opening Python from the command line, and executing:

```
>>> import pylearn2
```

If it doesn't complain, everything so far should work.

Installing the dependencies on Linux

First, a disclaimer. This has only been tested on Ubuntu 14.04 Desktop. If you are using a different Linux, then you are on your own. Most of the packages are in managers, so it shouldn't be too difficult.

opencv-python

OpenCV is by far the most difficult of the dependencies to install. Several choices made by the always-brilliant Ubuntu maintainers have made this more painful than it should be. We'll walk through getting it running from a fresh install; the commands should be "idempotent", so if you already have some of the packages installed, nothing bad will happen.

First, remove any old versions of `ffmpeg` and `x264`:

```
sudo apt-get -qq remove ffmpeg x264 libx264-dev
```

Now install a bunch of packages needed to build and run OpenCV:

```
sudo apt-get -qq install libopencv-dev build-essential checkinstall cmake  
pkg-config yasm libjpeg-dev libjasper-dev libavcodec-dev libavformat-dev  
libswscale-dev libdc1394-22-dev libxine-dev libgstreamer0.10-dev  
libgstreamer-plugins-base0.10-dev libv4l-dev python-dev python-numpy libtbb-dev  
libqt4-dev libgtk2.0-dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev  
libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-utils
```

Notice that `ffmpeg` wasn't part of this bundle. Recent Ubuntu releases do not include it anymore, but rather opt for an obscure fork of this popular package. We must get it from an alternate package archive:

```
sudo add-apt-repository ppa:mc3man/trusty-media  
sudo apt-get update  
sudo apt-get install ffmpeg gstreamer0.10-ffmpeg
```

Now download OpenCV:

```
wget -O OpenCV-2.4.9.zip  
http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/$version/opencv-2.4.  
9.zip/download
```

When it completes, unzip and enter `opencv-2.4.9`, then prepare the `build` directory:

```
mkdir build
cd build
```

Construct the build system:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON
-D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D
INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
```

and finish building:

```
make -j2
sudo checkinstall
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
```

If you did not hit any bumps, consider yourself lucky. You can test that everything installed correctly by opening `python`, and executing:

```
>>> import cv
>>> import cv2
```

You may need to reboot your machine before this works.

If, when building OpenCV, you see an error message along the lines of,

```
No rule to make target `/usr/lib/x86_64-linux-gnu/libGL.so'
```

Then check to see if `/usr/lib/x86_64-linux-gnu/libGL.so` is a link which (eventually) points to a non-existent file:

```
ls -alh /usr/lib/x86_64-linux-gnu/libGL.so
```

(note: you might have to modify the filename above if your error message refers to a different path). If it does, then try to find a valid `libGL.so` somewhere on your machine:

```
sudo find / -name libGL.so*
```

On my system, a valid `libGL.so` was at

`/usr/lib/x86_64-linux-gnu/mesa/libGL.so.9.0.24229.991745`; the exact filename will depend on your environment. Update `/usr/lib/x86_64-linux-gnu/libGL.so` to point to the file you found, which we will denote `VALID_LIBGL`:

```
sudo rm /usr/lib/x86_64-linux-gnu/libGL.so
sudo ln -s VALID_LIBGL /usr/lib/x86_64-linux-gnu/libGL.so
```

After fixing this issue, try building again:

```
make -j2
```

This problem usually arises in virtualized environments, or when Nvidia drivers are present.

theano

Installing **theano** is mostly the same as on Mac, but we're assuming a clean Ubuntu installation, so install all the needed dependencies to be safe:

```
sudo apt-get install python-numpy python-scipy python-dev python-pip python-nose
g++ libopenblas-dev
```

Then simply use `pip` to install **theano**:

```
sudo pip install Theano
```

Test by importing `theano` in Python.

pylearn2

Installing **pylearn2** on Linux is exactly the same as on Mac. Follow the instructions given above.

Running the code

Once you have all of the dependencies installed, try learning a new model. Enter the `code` directory of this repository, and train a Softmax regression model on the AT&T faces database:

```
python pylearn-train.py yaml/softmax_att_faces.yaml
```

You should see quite a bit of output, and if everything worked correctly, a new model should be trained in the `models` directory (which resides at the same level as `code`) as `softmax_att_faces.pkl`. You can now run the attack on this model, using the familiar faces from our examples:

```
python pylearn-attack.py ../models/softmax_att_faces.pkl
../images/att_faces/s1/3.pgm 1 --victim ../images/att_faces/s10/1.pgm --blur
--status_scale 2
```

Here, the first parameter is the model we trained in the previous step. The second parameter is the image that the attack will attempt to morph into an image accepted by the model as another class. The third parameter is the class label that the attack will target. The fourth parameter is optional, and contains an image from the targeted class; it is used for display purposes only. `--blur` tells the script to apply a Gaussian smoothing filter to the noise mask at each iteration. `--status_scale 2` tells the script to scale the size of the status images by a factor of two (the AT&T faces are quite small).

As the attack runs, you will see several lines of output indicating the current status of the attack, e.g.,

```
1.72992253304 1 0.0603811311426 0.170474 -0.133065
```

These lines have the following format:

	objective function value	label predicted by model	absolute mean value of noise mask	max noise mask value	min noise mask value
	1.72992253304	1	0.0603811311426	0.170474	-0.133065

When the attack completes, you should see a line that says something like,

```
... final label is 1
```