First tutorial session

Vincent Dumoulin

January 15, 2015

Outline

1. Solution to the numpy + MNIST + MLP assignment
2. Git primer
3. Theano primer
4. Porting numpy + MNIST + MLP to theano + MNIST + MLP

Solution to numpy + MNIST + MLP

Setup

$$\mathbf{x} = [x_1, \cdots, x_A]^T, \qquad \mathbf{t} = [t_1, \cdots, x_C]^T,$$

$$\mathbf{W} = \begin{pmatrix} W_{1,1} & \cdots & W_{1,A} \\ \cdots & \cdots & \cdots \\ W_{H,1} & \cdots & W_{H,A} \end{pmatrix}, \qquad \mathbf{b} = [b_1, \cdots, b_H]^T,$$

$$\mathbf{V} = \begin{pmatrix} V_{1,1} & \cdots & V_{1,H} \\ \cdots & \cdots & \cdots \\ V_{C,1} & \cdots & V_{C,H} \end{pmatrix}, \qquad \mathbf{c} = [c_1, \cdots, c_C]^T,$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \qquad \mathbf{y} = \mathrm{softmax}(\mathbf{V}\mathbf{h} + \mathbf{c})$$

$$\mathcal{L} = -\mathbf{t}^T \log \mathbf{y}$$

Solution to numpy + MNIST + MLP

Derivatives

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= (\mathbf{y} - \mathbf{t})\mathbf{h}^T, \\
\frac{\partial \mathcal{L}}{\partial \mathbf{c}} &= \mathbf{y} - \mathbf{t} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= [\mathbf{V}^T(\mathbf{y} - \mathbf{t}) \odot \mathbf{h} \odot (\mathbf{1} - \mathbf{h})]\mathbf{x}^T, \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= \mathbf{V}^T(\mathbf{y} - \mathbf{t}) \odot \mathbf{h} \odot (\mathbf{1} - \mathbf{h})
\end{aligned}
$$

A complete derivation can be found here

`https://raw.githubusercontent.com/vdumoulin/ift6266h15/master/assignments/01/solution.pdf`

Solution to numpy + MNIST + MLP

The full python implementation can be found here

https://raw.githubusercontent.com/vdumoulin/
ift6266h15/master/assignments/01/solution.py

Git primer

Why version control?

- A nice and clean alternative to maintaining multiple versions of the same file using some sort of custom naming scheme (*e.g.* `myfile_9.py`)
- A way to end the fear of saving and quitting
- Keep a trace how your files change throughout development
- Revert back to older versions
- Manage multiple versions (*branches*) of your code at the same time

Going further

`http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`

Git primer

What is git?

- Distributed version control system
  - No checking out single files: local version fully mirrors the repository
  - No central authority on what is the *true* codebase
- Takes *snapshots* of the state of a repository at a given time
- Intelligent about not duplicating information from one snapshot to another

Going further

`http://git-scm.com/book/en/v2/Getting-Started-Git-Basics`

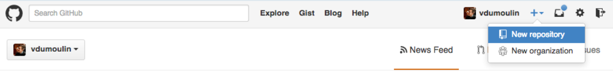
Git primer

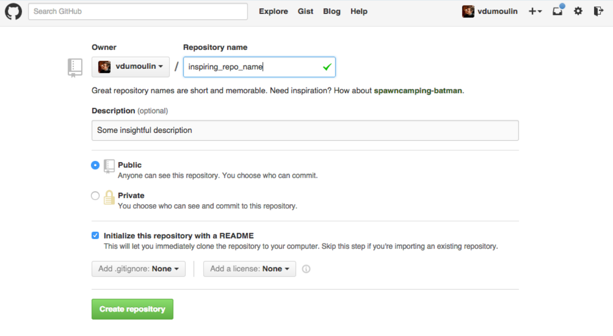
What is Github?

- A place to host your git repositories
- Makes it easy to
  - Share code with others
  - Keep track of other people's code
  - Modify other people's code (*forking*)
  - Collaborate with other people on common code
- Technically no different from your own machine:
  - Both can pull and push changes
  - Both host a fully functional version of your repository

Git primer

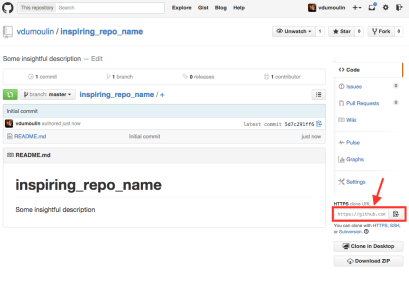
Creating a repository on Github





Git primer

Identifying the repository URL



Cloning a git repository

```
> git clone <REPO URL>
```

Git primer

Putting a file under version control

- Create a dummy file
- Check the status of the repository:

```
> git status
```

- Add the file to version control:

```
> git add my_dummy_file.py
```

- Commit the newly-added file:

```
> git commit -m "Add␣new␣dummy␣file␣to␣repository"
```

Git primer

Commit changes to a file

- Change something in your file
- Stage the changes:

```
> git add my_dummy_file.py
```

- Commit the changes:

```
> git commit -m "Add␣stuff␣to␣dummy␣file"
```

Git primer

Pull changes on Github

- Run

```
> git pull origin master
```

Push changes on Github

- Pull the latest changes from your Github repo:

```
> git pull origin master
```

- Push your changes to Github:

```
> git push origin master
```

Git primer

Going further with Git

http://git-scm.com/book/en/v2

Theano primer

What is Theano?

- From Theano's online documentation:

  > *Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.*

- Does *symbolic* computation **and** differentiation (*i.e.* the end result of differentiation is itself a symbolic expression)
- Very similar to numpy with respect to its interface
- Allows doing numerical computation in a high-level language (Python) while still retaining the speed of low-level languages (like C)
- Allows the generation of efficient CPU and GPU code transparently

Theano primer

Typical Theano workflow

1. Instantiate symbolic variables
2. Build a computation graph out of those variables
3. Compile a function with the symbolic variables as input and the output of the computation graph as output
4. Call the compiled function with numerical inputs

Theano primer

Theano vs. numpy

- Theano interface is *very* similar to numpy interface
- numpy arrays are automatically converted to constant symbolic variables when used inside a computation graph
- You can manipulate Theano symbolic variables in the same way you'd manipulate numpy arrays

Going further: Theano's basic interface

`http://deeplearning.net/software/theano/library/tensor/basic.html`

Theano primer

Types of symbolic variables

TensorVariable Its value is unspecified at graph creation and can change from one call of the compiled function to another (*e.g.* x and y in $y = 3x - 2$). **Not persistent across function calls**

TensorConstant Its value is specified at graph creation and does **not** change from one call of the compiled funtion to another (*e.g.* 3 and -2 in $y = 3x - 2$)

TensorSharedVariable Its value is specified at graph creation but is bound to change from one call of the compiled function to another (*e.g.* a and b in $y = ax + b$ in a regression setting where some x and y pairs have been observed). **Persistent across fuction calls**

Theano primer

Examples

Listing 1: Simple algebra

```
import theano
import theano.tensor as T

# 1. Instantiate symbolic variables
x = T.vector(name='x')
y = T.vector(name='y')

# 2. Build a computation graph
z = x + y

# 3. Compile a callable function
f = theano.function(inputs=[x, y], outputs=z)

# 4. Call the function using numerical inputs
print f([1, 2], [3, 4])
```

Theano primer

Examples

Listing 2: Gradient computation

```
import theano
import theano.tensor as T

# 1. Instantiate symbolic variables
x = T.vector(name='x')

# 2. Build a computation graph
z = (x ** 2).sum()
d_z_d_x = T.grad(z, x)

# 3. Compile a callable function
f = theano.function(inputs=[x], outputs=d_z_d_x)

# 4. Call the function using numerical inputs
print f([1, 2])
```

Theano primer

Examples

Listing 3: Linear regression

```
import theano
import theano.tensor as T

x = T.scalar(name='x'); t = T.scalar(name='t')
a = theano.shared(-1.0, name='a')
b = theano.shared(0.0, name='b')

y = a * x + b
mse = (y - t) ** 2
grad_a, grad_b = T.grad(mse, [a, b])

f = theano.function(inputs=[x, t], outputs=mse,
                    updates={a: a - 0.01 * grad_a,
                             b: b - 0.01 * grad_b})

print [f(1, 5)) for i in xrange(10)]
```

Theano primer

Going further: online Theano tutorial

http://deeplearning.net/software/theano/tutorial/index.html#tutorial