## How to contribute

In addition to writing code, joining in on discussions in the issues is a great way to get involved. Another great way to get started is to write additional testing and benchmarking scripts.

We are using Python and Keras because we believe they are beginner-friendly and easy to read. This is, to some extent, at the expense of speed. One of the biggest ways to help is to run some benchmarking scripts to see where the bottlenecks are and fix them!

## Git guide

1. keep `upstream` functional

1. write useful commit messages

1. `commit --amend` or `rebase` to avoid publishing a series of "oops" commits (better done on your own branch) ([read this](https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History))

1. ..but don't modify published history

1. prefer `rebase master` to `merge master`, again for the sake of keeping histories clean. Don't do this if you're not totally comfortable with how `rebase` works.

1. ..but don't modify published history

1. keep pull requests at a manageable size

## Coding guide

We are using Python 2.7. It is recommended you use [virtualenv](http://docs.python-guide.org/en/latest/dev/virtualenvs/) to set up an environment. Once you've done so, you can install all necessary dependencies using

pip install -r requirements.txt

A good way to test if this worked is to run the tests

python -m unittest discover

Lastly, follow these guidelines:

1. remember that ["code is read more often than it is written"](https://www.python.org/dev/peps/pep-0008)

1. avoid premature optimization. instead, be pedantic and clear with code and we will make targeted optimizations later using a profiler

1. write [tests](https://docs.python.org/2/library/unittest.html) in the `tests/` directory. These are scripts that essentially try to break your own code and make sure your classes and functions can handle what is thrown at them

1. [document](http://epydoc.sourceforge.net/docstrings.html) and comment liberally