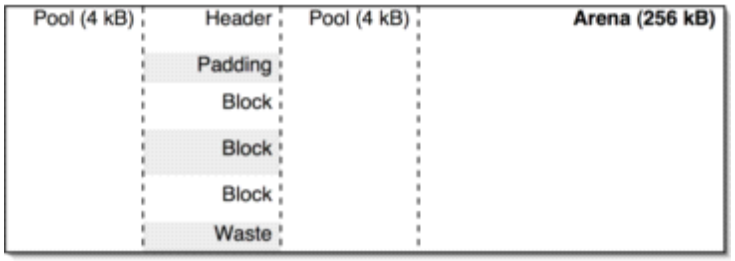
Python Memory Management

Friday, January 22, 2016 7:51 PM

- Python structures use substantially more(3-6x) memory than C++ ones
- Inefficient memory allocation, does not remove de-allocated memory of objects
    - 256kb Arena divided into 4kb pools, with fixed size blocks
        - Objects go into various 4kb pools depending on the specified block sizes
        - freepools, usedpools, partially_allocated_arena are some of the variables which keep track
        - Once created, the memory in the pools and arenas is reused when necessary, but the created chunks aren't deleted - this builds up when different pools needed
        - 

- Pickling also uses an unnecessarily large chunk of memory
    - More efficient to serialize certain objects into a file
    - To control what is pickled within a class, use(long term storage):
        - _getstate_(self): assigns what is passed(variables and such) to the pickle dump
        - _setstate_(self): assigns what pickle retrieves and variables it sets
        - Can also pass the entire dictionary and remove or set the necessary variables(not very robust)
    - Use protocol=pickle.HIGHEST_PROTOCOL as parameter in a pickle dump to get best efficiency(check which protocols are compatible with the appropriate version of Python)
    - Use the theano pkl_utils module to pickle Theano objects in numpy readable form
        - Good for sharing
- Theano memory model
    - Manages its memory apart from Python
    - Shared variables in the buffer are unique(never aliased to another shared variable)
    - borrow=True flag allows aliasing of variables(faster, but more prone to bugs)
        - Shared variables by default are not aliasable(borrow=False)
            - deep copy of an object is made when passed to shared, otherwise, the shared variable uses the same object that's passed to it
        - Use for large objects
        - Useless with GPUs
        - Also can be used with get_value() - both may create copies
            - Returns a value that may be aliased if borrow=True
            - Returns a value that may not be aliased if borrow=False
            - return_internal_type=True returns the actual internal representation unlike borrow=True with the GPU(may use a different internal representation)
            - get_value() always returns the same object that is created as the shared variable
            - Safe to use return_internal_type=True when the value won't be modified
        - Usage with set_value() - reuses provided buffer
            - Pattern:

```
s.set_value(
    some_inplace_fn(s.get_value(borrow=True)),
    borrow=True)
```

            - Use contiguous values when assigning to CudaNdarraySharedVariable

- Works with defining a function

```
import theano, theano.tensor

x = theano.tensor.matrix()
y = 2 * x
f = theano.function([theano.In(x, borrow=True)], theano.Out(y, borrow=True))
```

  - theano.In and theano.Out arguments as the arguments and the function values
  - Reuses the same buffer