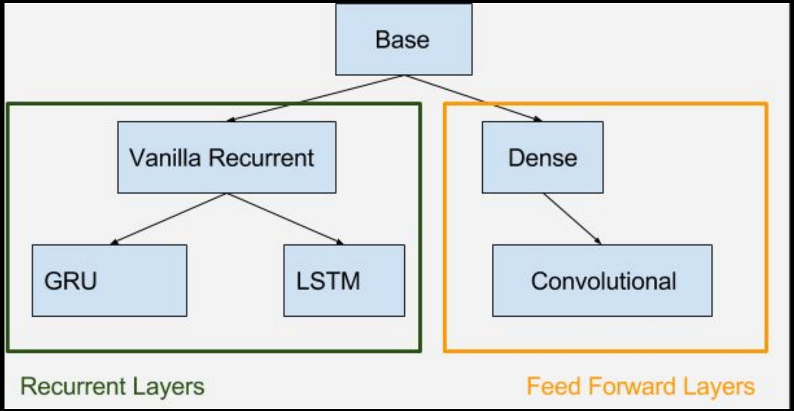
mlStax

Stackable Layers for Deep Learning

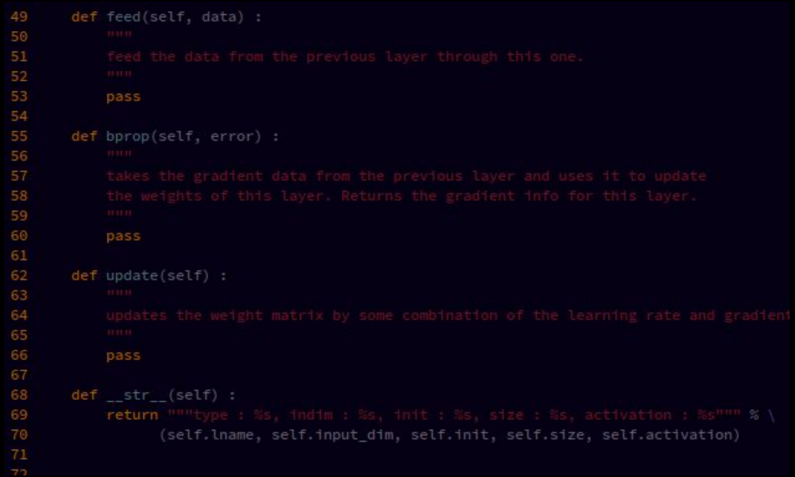
in Python, C++, Haskell

Stackable Layers

Initialize a model
stack on some layers
train, test, and predict



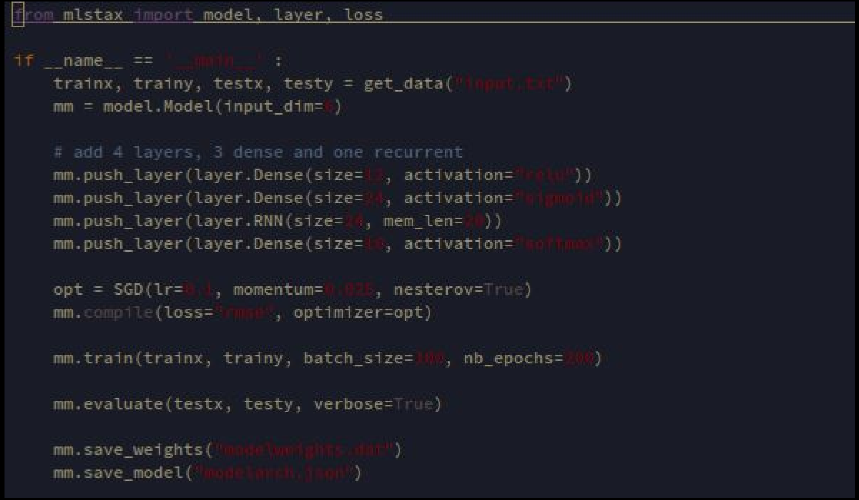
Base Class :



```
    def feed(self, data) :
        """
        feed the data from the previous layer through this one.
        """
        pass

    def bprop(self, error) :
        """
        takes the gradient data from the previous layer and uses it to update
        the weights of this layer. Returns the gradient info for this layer.
        """
        pass

    def update(self) :
        """
        updates the weight matrix by some combination of the learning rate and gradient
        """
        pass

    def __str__(self) :
        return """type : %s, indim : %s, init : %s, size : %s, activation : %s""" % \
            (self.lname, self.input_dim, self.init, self.size, self.activation)

```

Code

Building a simple layer in Python



```
from mlstax import model, layer, loss

if __name__ == '__main__' :
    trainx, trainy, testx, testy = get_data("input.txt")
    mm = model.Model(input_dim=6)

    # add 4 layers, 3 dense and one recurrent
    mm.push_layer(layer.Dense(size=12, activation="relu"))
    mm.push_layer(layer.Dense(size=24, activation="sigmoid"))
    mm.push_layer(layer.RNN(size=24, mem_len=20))
    mm.push_layer(layer.Dense(size=10, activation="softmax"))

    opt = SGD(lr=0.1, momentum=0.025, nesterov=True)
    mm.compile(loss="rmse", optimizer=opt)

    mm.train(trainx, trainy, batch_size=100, nb_epochs=200)

    mm.evaluate(testx, testy, verbose=True)

    mm.save_weights("model\weights.dat")
    mm.save_model("model\arch.json")
```

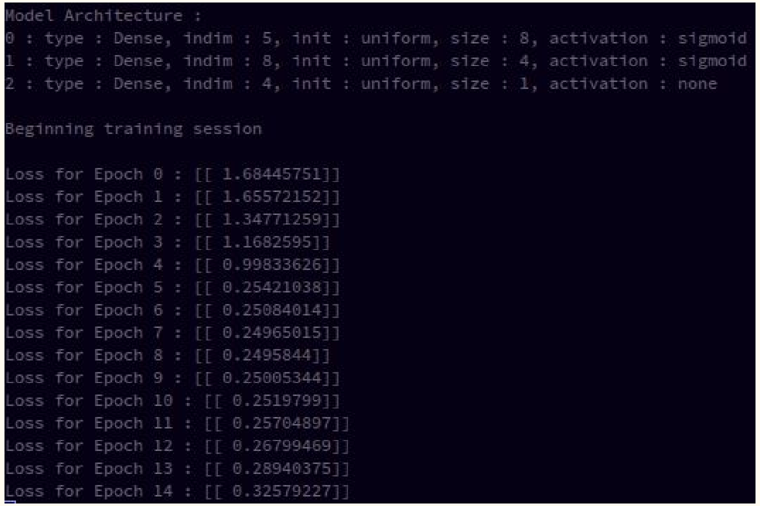
Code

Building a simple layer in C++

Examples

Character level RNN

Simple XOR



```
Model Architecture :
0 : type : Dense, indim : 5, init : uniform, size : 8, activation : sigmoid
1 : type : Dense, indim : 8, init : uniform, size : 4, activation : sigmoid
2 : type : Dense, indim : 4, init : uniform, size : 1, activation : none

Beginning training session

Loss for Epoch 0 : [[ 1.68445751]]
Loss for Epoch 1 : [[ 1.65572152]]
Loss for Epoch 2 : [[ 1.34771259]]
Loss for Epoch 3 : [[ 1.1682595]]
Loss for Epoch 4 : [[ 0.99833626]]
Loss for Epoch 5 : [[ 0.25421038]]
Loss for Epoch 6 : [[ 0.25084014]]
Loss for Epoch 7 : [[ 0.24965015]]
Loss for Epoch 8 : [[ 0.2495844]]
Loss for Epoch 9 : [[ 0.25005344]]
Loss for Epoch 10 : [[ 0.2519799]]
Loss for Epoch 11 : [[ 0.25704897]]
Loss for Epoch 12 : [[ 0.26799469]]
Loss for Epoch 13 : [[ 0.28940375]]
Loss for Epoch 14 : [[ 0.32579227]]
```

Language Comparisons

Python	C++
Very easy to start.	Slow to start, lots of types to figure out.
Numpy and Theano make all matrix operations very simple.	Had to be more careful in my choice of matrix libraries.
Small, high-entropy code base.	Verbose code, lots of template boilerplate.
Runs a bit slower than C++.	Runs faster.
Later on, bugs were subtle and hard to track down.	Catch bugs earlier because of rigid type system.