CS446: Machine Learning	**Spring 2018**
Machine Problem 6	
Handed Out: Jan. 16, 2018	*Due: Feb. 27, 2018 (11:59 AM Central Time)*

Note: The assignment will be autograded. It is important that you do not use additional libraries, or change the provided functions input and output.

Part 1: Setup

- Remove connect to a EWS machine.

```
ssh (netid)@remlnx.ews.illinois.edu
```

- Load python module, this will also load pip and virtualenv

```
module load python/3.4.3
```

- Reuse the virtual environment from mp0.

```
source ~/cs446sp_2018/bin/activate
```

- Copy mp6 into your svn directory, and change directory to mp6.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp6 .
cd mp6
```

- Install the requirements through pip.

```
pip install -r requirements.txt
```

Part 2: Exercise

In this exercise, you will implement the deep net from the written section and train it using stochastic gradient descent. The deep net should be implemented from scratch in python (i.e. forward pass and back-propagation). Do not use Tensorflow, Pytorch, Chainer etc. to implement the network. Consider the number of input layers to be 2, the number of hidden layers to be 3, the number of output layers to be 1 and the batch size to be 3. The loss to be optimized is $\frac{1}{m}\sum_m E_m$, where m is the bach size and E_m is the L2 loss from the written part, associated with the m^{th} sample.
Hint: Think how the derivatives computed in the written part will change when considering this new loss.

Description of the functions to be implemented:

- Method *sigmoidPrime* of the class *Neural_Network*: computes the derivative of a sigmoid function. The input to this method is the sigmoid's output.
- Method *forward* of the class *Neural_Network*: implements the forward propagation of the deep net described in the written section.
- Method *d_loss_o* of the class *Neural_Network*: computes the derivative of the loss with respect to the network's output c.
- Method *error_at_layer2* of the class *Neural_Network*: computes the derivative of the loss with respect to the second fully connected layer's output h (Written section, Part b).
- Method *error_at_layer1* of the class *Neural_Network*: computes the derivative of the loss with respect to the first fully connected layer's output z (Written section, Part e).
- Method *derivative_of_w* of the class *Neural_Network*: computes the derivative of the loss with respect to W (Written section, Part c).
- Method *derivative_of_f* of the class *Neural_Network*: computes the derivative of the loss with respect to f (Written section, Part d).
- Method *derivative_of_u* of the class *Neural_Network*: computes the derivative of the loss with respect to U (Written section, Part f).
- Method *derivative_of_e* of the class *Neural_Network*: computes the derivative of the loss with respect to e (Written section, Part g).
- Method *backward* of the class *Neural_Network*: computes the gradients of the parameters W, U, e and f, and performs the gradient descent update rule. The backpropagation should be implemented using the matrix form of the parameters (i.e. do not use any *for* loops).

Relevant File: *back_prop.py*

Part 3: Writing Tests

In *test.py*, we have provided basic test-cases. Feel free to write more. To test the code, run:

```
nose2
```

Part 4: Submit Submitting the code is equivalent to committing the code. This can be done with the follow command:

```
svn commit -m "Some meaningful comment here."
```

Lastly, double check on your browser that you can see your code at

```
https://subversion.ews.illinois.edu/svn/sp18-cs446/(netid)/mp6/
```