Homework-03

Neural Networks - Back-propagation pass

Background

This homework teaches how to code the fundamental training algorithm for neural networks: back-prop and stochastic gradient descent. While there are a number of tools for this, it is important to understand the main operations involved in a neural network training and code your very own SGD.

Deliverables

Small write up in *Markdown* where you will also compare the Theta you hand-crafted for `HW02` vs. the one you've learnt with back-propagation.
Source code in `NeuralNetwork.py`, `logicGates.py` description in `README.md`.

API (`NeuralNetwork`)

Points: 6

```
-- create the table of matrices 0
[nil] build(([int] in, [int] h1, [int] h2, …, [int] out))
-- returns 0(layer)
[2D FloatTensor] getLayer([int] layer)
-- feedforward pass single vector
[1D FloatTensor] forward([1D FloatTensor] input)
-- feedforward pass design matrix (should come free after transposition)
[2D FloatTensor] forward([2D FloatTensor] input)
-- back-propagation pass single target (computes ∂E/∂0)
[nil] backward([1D FloatTensor] target)
-- back-propagation pass target matrix
-- (computes the average of ∂E/∂0 across seen samples)
[nil] backward([2D FloatTensor] target)
-- update parameters
[nil] updateParams([float] eta)
```

When I import your library, I should get a class with **only** five methods specified in the API. No global variables, no other helper functions (which must be `local`, if needed).

API (`logicGates`)

Points: 4

```
[nil] AND.train()
[nil]  OR.train()
[nil] NOT.train()
[nil] XOR.train()
[boolean] AND.forward([boolean] x, [boolean] y)
[boolean]  OR.forward([boolean] x, [boolean] y)
[boolean] NOT.forward([boolean] x)
[boolean] XOR.forward([boolean] x, [boolean] y)
```

When I import your library I should see 4 `classes` with **only** two public methods for each logic gate class, specified in the API. No global variables, no other helper functions (which must be `local`, if needed).

API (`Img2Num`) (Preview of next homework)

```
[nil] train()
[int] forward([28x28 ByteTensor] img)
```

When I import your library I should get a `class` with **only** two public methods specified in the API. No global variables, no other helper functions (which must be `local`, if needed, like the `oneHot()` conversion function).

Instructions

1. Create a *Python* script NeuralNetwork.py that contains two `local` empty dictionaries `Theta` and `dE_dTheta` and the five functions described in the API.
2. Implement back-propagation with a *Mean Square Error* loss function.
3. Update the matrices `Theta` with `updateParams(eta)` based on the learning rate `eta` and the gradient of the error with respect of the parameters `dE_dTheta`.
4. Train the `AND`, `OR`, `NOT` and `XOR` networks using the `NeuralNetwork` API (calling `forward()`, `backward()` and `updateParams()` in a cycle) on a hand-crafted data set (Hint: use *Python's* `and`, `or`, `not` and combination of these in order to build your data on the fly). Compare the the `Theta`s with what you set manually in `HW02`. One may learn the new `Theta`s with the `train()` function.
5. Train the `img2num` network using the MNIST dataset using the `NeuralNetwork` API. To speed up training, you may want to use the `NeuralNetwork` API in batch mode (where you forward chunks of the design and target matrices at a time). This means that if `size(x) = n` and you have `m` examples, then `size(X) = m × n` (I know it is transpose of what you were providing in last homework) and `NeuralNetwork`'s `forward()` can take `x` or `X` and `NeuralNetwork`'s `backward()` can take `y` or `Y`. Output of `NeuralNetwork`'s `forward()` will be `m × c` (where m is number of examples and c is number of classes/categories).

Format

Submit ZIP file containing all the deliverables with instructions to run your script using. Your folder name should be as following:

`<your_purdue_username>_HW<0X>`

For example, my folder name for this homework will look like `aabhish_HW03`.

Bonus

Points: 2

Add one more parameter to `backward([1D DoubleTensor] target)` which becomes now `backward(target, [string] loss)`. By default `loss` is `MSE`. Nevertheless, `loss` can be `CE`, which stands for *cross-entropy*.

The cross-entropy loss function is defined as following:

```
loss(h, y) = -log(exp(h[y]) / (\sum_j exp(h[j])))
           = -h[y] + log(\sum_j exp(h[j]))
```

Suggestions/Recommendations

- You may want to create two additional `attributes` (dictionaries) `a` and `z` in NeuralNetwork class.
- Make your own tests to verify the output of the logic gate functions against the *Python*'s `and`, `or`, `not` and combination of these. You should build and train the `AND`, `OR`, `NOT` and `XOR` networks using your own `NeuralNetwork` library. You will need to convert `boolean`s to the integers `0` and `1` and vice versa, in order to comply with the given API.
- You will probably need to download the MNIST data set. Instruction about how to use the data set are provided here -> https://github.com/andresy/mnist.
- Your `img2num` network will need `view` the 2D input into a 1D `Tensor` so that it can be fed to the network. The labels will need a `oneHot` encoding.