Investigating Backprop, and Hopfield Networks

Due at 4:00pm on Thursday 5 March 2020

What you need to get

- `YOU_a3.ipynb`: a Python notebook (hereafter called "the notebook")
- `Network.py`: Module with `Network` and `Layer` classes

What you need to know

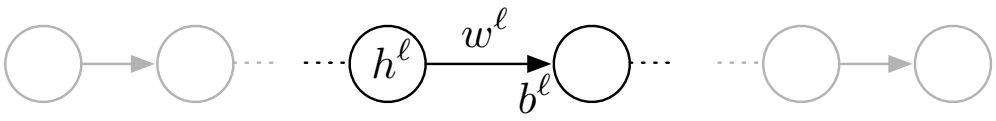
The module `Network` includes an implementation of the `Network` class as well as the `Layer` class. The `Network` class includes implementations of `FeedForward`, `BackProp`, `SGD` (stochastic gradient descent), among other methods. The module also defines a number of useful functions, such as common activation functions, their derivatives, common cost functions, etc. You can use that module, but should not alter it in any way.

The notebook includes two class, `RobustNetwork`, and `ExtNetwork`, each derived from `Network`. In this assignment, you will alter `RobustNetwork` and `ExtNetwork` so that they implement the required functionality.

What to do

1. **Vanishing Gradients [6 marks]**

   Consider a deep network that is simply a chain of nodes, as shown below.



   Assume that the activation function, $\sigma(\cdot)$, is the logistic function. Then the deep gradients for this network take the form,

$$\frac{\partial E}{\partial z} = \prod_\ell w^\ell \sigma'(z^\ell) \quad \text{where } z^\ell = w^\ell h^\ell + b^\ell \,,$$

   That is, while propagating the error gradients back through the network, each layer adds a term of the form $w^\ell \sigma'(z^\ell)$. Dropping the superscripts, consider the magnitude of one generic term in that product, $|w\sigma'(wh+b)|$.

   (a) **[1 mark]** Suppose $|w\sigma'(wh+b)| \geq 1$. Prove that this can only occur if $|w| \geq 4$.

   (b) **[5 marks]** Supposing that $|w| \geq 4$, consider the set of input currents h for which $|w\sigma'(wh+b)| \geq 1$. Show that the set of activity values for h satisfying that constraint can range over an interval no greater in width than

$$\frac{2}{|w|} \ln \left[\frac{|w|}{2} \left(1 + \sqrt{1 - \frac{4}{|w|}} \right) - 1 \right] \,.$$

   (c) **[0 marks]** ~~Plot the expression from part (b) over a range of $|w|$ values that show the expression's peak value. Approximately what value of $|w|$ yields the maximum value (to within 2 significant digits of precision)?~~

2. **Implementing Weight Decay [12 marks]**

   The `RobustNetwork` class extends the `Network` class, and overrides its `BackProp` and `SGD` methods. The `RobustNetwork` implementations have an additional argument, `decay`. By default, it is set to zero. However, when it is nonzero (and positive), then it becomes the decay coefficient for weight decay.

   For your convenience, the notebook includes a function for creating training and testing datasets. Simply call

   ```
   train, test = GenerateDatasets(P)
   ```

   to generate a training set with `P` samples, and a test set with 300 samples. Each call to that function yields data for a *different* model, so don't combine the data from multiple calls into one dataset. You can also include a seed argument to `GenerateDatasets`, such as `seed=623`, if you want to generate the same dataset repeatedly.

   (a) **[2 marks]** Alter the function `BackProp` (in the `RobustNetwork` class) so that it implements weight (and bias) decay. You can implement either L_2 or L_1 decay. Use the value of the `decay` argument as the weight for the L_2 or L_1 penalty term.

   (b) **[2 marks]** Create a regression-type `RobustNetwork` with one input node, 10 hidden nodes, and one output node. Make two identical copies of that network, one called `original_net` and one called `decay_net`. You can use `copy.deepcopy` to do that.

   (c) **[2 marks]** Generate a dataset with only 5 training samples, and use it to train `original_net` (using `SGD` with a batch size of 5) for 5000 epochs, a learning rate of 0.1, and `decay=0`. Evaluate the network on the training and test datasets (using the supplied `Evaluate` function).

   (d) **[1 mark]** Train `decay_net` on the same dataset using the same parameters as above, but with `decay=0.03`. Again, evaluate the network on the training and test datasets.

   (e) **[2 marks]** Plot the original training points (blue dots), the test points (yellow), as well as lines showing the models learned by `original_net` (blue dashed line) and `decay_net` (red dashed line). As always, label the axes.

   (f) **[3 marks]** Redo the above experiment (minus the plot) 10 times (in a loop, please), each time:

      i. create and duplicate a new `RobustNetwork` (using the architecture above)

      ii. generate a new pair of training and testing datasets (5 training samples)

      iii. train one network without decay, and one with `decay=0.03`

      iv. evaluate both networks on the test dataset

      After that, you will have two lists of 10 costs, one for each network. Compute the mean cost over the 10 runs for each network. Based on those results, which method is preferred? Explain your choice.

3. **Classifier Networks [6 marks]**

   The `Network` class in the supplied module does not have a suitable implementation for the combination of softmax and categorical cross-entropy. Instead, you will edit the `ExtNetwork` class (which stands for "Extended Network") to implement that combination. In particular, the function `BackProp` calls the `TopGradient` function. So far, `TopGradient` has implementations of logistic/cross-entropy (`type='Bernoulli'`), and identity/MSE (`type='regression'`). Add

code to compute the top gradient for the softmax/categorical cross-entropy combination, which corresponds to `type='classifier'`.

The notebook has a function called `CreateDataset`; it generates training and test datasets in which 2-dimensional points belong to four distinct classes. Your task is to create a neural network that will yield high accuracy in this classification task.

(a) **[1 mark]** Complete your implementation of `ExtNetwork.TopGradient` so that it computes the gradient for the softmax/categorical cross-entropy combination when `self.type` is `'classifier'`.

(b) **[3 marks]** Create an `ExtNetwork`, and then train it using the following code:

```
train, test = CreateDataset(params)
progress = net.SGD(train[0], train[1], epochs=400, lrate=0.5)
```

(c) **[2 marks]** Evaluate the classification accuracy of your trained network on the test dataset. You can use the supplied function `ClassificationAccuracy` to help you with that. Also, plot the test inputs, but coloured according to your network's output. You can use the supplied function `ClassPlot` for that.

4. **Hopfield Networks [14 marks]**

For this question, you will complete the implementation of a continuous Hopfield network. This network uses the Hopfield energy function

$$E = -\frac{1}{2} y^{\mathrm{T}} W y - b^{\mathrm{T}} y + \lambda_x \sum_i \int_0^{y_i} \sigma^{-1}(y) dy + \frac{\lambda_W}{2} \|W\|_{\mathrm{F}}^2 + \frac{\lambda_b}{2} \|b\|_2^2 \ ,$$

where y is a vector of the node activities ($y_i = \sigma(x_i)$ for node i), W is the (symmetric) connection weight matrix, b is the vector of costs associated with the activity of each node, and λ_x is the weight for the energy term associated with y-values that are close to ± 1. That is, it takes a lot of energy for the node's activity to approach 1 or -1. The energy function also includes penalty terms for the squared Forbenius norm of W, and the squared 2-norm of b, with weights λ_W and λ_b, respectively.

(a) **[2 marks]** Prove that

$$\frac{\partial E}{\partial y_i} = -\sum_j \left(W_{ij} y_j\right) - b_i + \lambda_x x_i \ .$$

(b) **[1 mark]** If $\sigma(x) = \tanh x$, show that

$$\frac{\partial E}{\partial x_i} = \left(-\sum_j \left(W_{ij} y_j\right) - b_i + \lambda_x x_i \right) \left(1 - y_i^2\right) \ .$$

(c) **[1 mark]** Show that

$$\frac{\partial E}{\partial W_{ij}} = -y_i y_j + \lambda_W W_{ij} \ .$$

(d) **[1 mark]** Derive an expression for $\frac{\partial E}{\partial b_i}$.

(e) **[3 marks]** Using the gradients above, complete the `Recall` function in `HopfieldNet` so that it solves the differential equation $\frac{dx_i}{dt} = -\frac{\partial E}{\partial x_i}$ numerically using Euler's method.

(f) **[5 marks]** Using the gradients above, complete the `Train` function in `HopfieldNet` so that it solves the differential equations $\frac{dW_{ij}}{dt} = -\frac{\partial E}{\partial W_{ij}}$ and $\frac{db_i}{dt} = -\frac{\partial E}{\partial b_i}$. For each epoch, hold each memory from the training set for one time step of length `dt`. You can find out more about the function by looking at its help documentation.

(g) **[1 mark]** Train and run your Hopfield network on the dataset provided in the notebook. Train it for 10 epochs using a time step of 0.01. Note that many of the training samples have errors.

Enjoy!

What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a3.ipynb`), where `<WatIAM>` is your UW WatIAM login ID (not your student number). The notebook must include solutions to **all** the questions. Submit this file to Desire2Learn. You do not need to submit any of the modules supplied for the assignment.