Investigating Backprop

Due at 4:00pm on Wednesday 6 March 2019

What you need to get

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

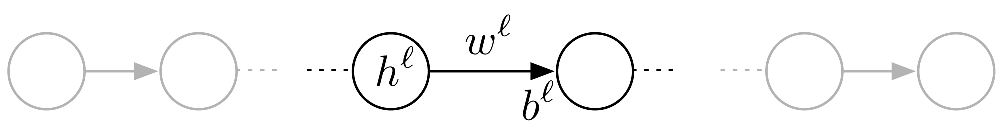
What you need to know

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 function, etc. You can use that module, but should not alter it in any way.

The notebook includes the class `RobustNetwork`, which is derived from `Network`. In this assignment, you will alter `RobustNetwork` so that it implements dropout and weight decay.

What to do

1. 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) [3 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 Dropout** [17 marks]

The `RobustNetwork` class overrides the `FeedForward` implementation in `Network`. In particular, it includes an additional (optional) argument that specifies the dropout probability for hidden nodes. For example,

```
net.FeedForward(x, dropout=0.3)
```

will randomly drop each hidden node with probability 0.3. By default, `dropout` is 0. Note that `RobustNetwork.SGD` has an additional argument, `dropout`. But, calls such as

```
net.SGD(x, t, epochs=10, lrate=0.5, dropout=0.3)
```

will include the argument `dropout=0.3` in its calls to `FeedForward`, like that shown above.

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.

(a) [5 marks] Complete the implementation of `FeedForward` in the `RobustNetwork` class to implement dropout as described above. When `dropout` is nonzero, then `FeedForward` should drop hidden nodes with the specified probability, and scale up the remaining node activities to compensate.
Note that the `FeedForward` function sets the additional class variable `dropout_nonzero` to `True` when dropout is being done, and `False` when `dropout` is zero.

(b) [1 mark] Alter `RobustNetwork.BackProp` so that it works properly when dropout is occurring (that is, when `dropout_nonzero` is `True`).

(c) [2 marks] Create a `RobustNetwork` with one input node, 10 hidden nodes, and one output node. Use MSE as the cost function. The hidden layer should use the arctan activation function, while the output node should use the identity activation function. Make two identical copies of that network, one called `original_net` and one called `dropout_net`. You can use `copy.deepcopy` to do that.

(d) [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 1, and `dropout=0`. Evaluate the network on the training and test datasets (using the supplied `Evaluate` function).

(e) [1 mark] Train `dropout_net` on the same dataset using the same parameters as above, but with `dropout=0.2`. Again, evaluate the network on the training and test datasets.

(f) [3 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 `dropout_net` (red dashed line). As always, label the axes.

(g) [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 dropout, and one with `dropout=0.2`

  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. **Implementing Weight Decay** [4 marks]

   Guess what! The `SGD` method in `RobustNetwork` also has an 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. Similar to dropout, the `decay` argument is passed as an additional parameter to the `BackProp` function.

   (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 argument as the decay coefficient (like λ in the lecture notes).

   (b) [2 marks] Redo the 10-trial experiment from question 2g, this time comparing the no-decay-no-dropout model (`original_net`) to a decay model (`decay_net`) using a decay coefficient of 0.0004. Based on their average costs, what method is preferred? Again, justify your choice.

4. **Classifier Networks** [7 marks]

   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 devise a neural-network architecture that will yield high accuracy in this classification task.

   Here's the catch: You have a fixed computational budget. You are allowed to use 10 hidden nodes, and 400 epochs with a learning rate of 0.5 and a batch size of 10 (the default). You can experiment with different cost functions, and activation functions. Find a network configuration that consistently gives the highest testing accuracy.

   (a) [3 marks] Create your network, and then train it using the following code:

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

   (b) [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.

   (c) [2 marks] Create another network that is demonstrably worse, train it, and show that its accuracy is lower than your other model.

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.