Error Backpropagation

Due at 4:00pm on Friday 14 February 2020

What you need to get

- `YOU_a2.ipynb`: a Python notebook (hereafter called "the notebook")
- `a2_solutions.pyc`: Module with solutions (optional)

What to do

1. **[2 marks]** The logistic function is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \, .$$

Prove that

$$\frac{d\sigma(z)}{dz} = \sigma(z)\big(1 - \sigma(z)\big) \, .$$

2. **[4 marks]** Consider a classification problem in which you have K classes. Suppose you have a labelled dataset containing pairs of inputs and class labels, $(\vec{x}, \ell)$, where $\vec{x} \in \mathbb{R}^X$ and $\ell \in \{1, 2, \ldots, K\}$.

Your neural network's output is a classification vector based on the softmax activation function, so that if z_k is the input current for output node k, then the activation of output node y_k is

$$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad , \quad k = 1, \ldots, K \, .$$

Thus, $\vec{y} \in [0, 1]^K$, and $y_k = P\left(\ell = k \,|\, \vec{x}\right)$.

Suppose that your loss function is categorical cross entropy,

$$E(\vec{y}, \vec{t}\,) = -\sum_{k=1}^{K} t_k \ln y_k \, .$$

Derive an expression for $\frac{\partial E}{\partial z_j}$, the gradient of the loss function with respect to the input current to the output layer.

3. **[4 marks]** Consider the network $y = f\left(\vec{x}, \theta\right)$ with output loss function $E\left(y, t\,\right)$, where $y, t \in \mathbb{R}$. Let the input current to the output layer be z, so that $y = \sigma(z)$, and $\sigma(\cdot)$ is the activation function for the nodes in the output layer.

Derive an expression for $\frac{\partial E}{\partial z}$ for the following two combinations:

(a) The logistic activation function, with cross entropy,

$$E(y, t\,) = -t \ln y - (1 - t) \ln(1 - y)$$

(b) The identity activation function, $\sigma(z) = z$, with mean squared error,

$$E(y, t\,) = (y - t)^2$$

4. **Implementing Backpropagation**

   For this question, you must complete the implementation of the `Network` class. Notice that the jupyter notebook has helper functions, a `Layer` class, and a `Network` class. Familiarize yourself with the code. Notice that a `Network` contains a series of `Layer`s, as well as connection matrices. Much of the functionality is already completed, but there are a few functions that you have to finish. Follow these directions:

   (a) Cost functions: Implement the following cost functions and their gradients. **[5 marks]**

      i. `CrossEntropy(y,t)`: Evaluates the average cross entropy between outputs `y` and targets `t`.

      ii. `gradCrossEntropy(y,t)`: Evaluates the gradient of the average cross entropy with respect to outputs `y`.

      iii. `MSE(y,t)`: Evaluates the mean squared error between outputs `y` and targets `t`.

      iv. `gradMSE(y,t)`: Evaluates the gradient of the mean squared error with respect to outputs `y`.

   (b) `FeedForward`: Complete the function `Network.FeedForward` according to the specifications in its documentation. Note that your function must work for 2D input arrays containing multiple samples. The input currents and activities for each `Layer` should be stored in the corresponding `Layer.z` and `Layer.h`, respectively. **[3 marks]**

   (c) `BackProp`: Complete the `Network.BackProp` function, which uses the network state (after a feedforward pass) and the corresponding targets to compute the error gradients, and performs an update to the network weights and biases. You only need to implement the cost-function/activation-function combinations **[7 marks]**

   (d) `Learn`: Complete the `Network.Learn` function, which performs gradient descent over the training dataset to try to find the optimal network weights and biases. The function should perform the specified number of epochs. If the input parameter `progress=True` is given, then append the cost after each epoch to the list `Network.cost_history`. **[4 marks]**

   For your convenience, the notebook includes two sample datasets for you to test your implementation on. One is a classification problem, and one is a regression problem. Your implementation should work with these lines of code. But I encourage you to tinker with them.

   There is also a pre-compiled module (`a2_solutions.pyc`) with complete implementations of all the functions you are asked to complete. This might be helpful for testing, or if you have trouble with one of the functions, and would like to move on and implement one of the dependent functions. The notebook contains instructions on how to gracefully move between your code and the solution code.

Enjoy!

What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a2.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.