Assignment 1

Due date: Friday, September 3

The back-propagation algorithm

This morning, when you asked

> Alexa, when is the 2217 homework due?

chances are your question was parsed, interpreted, etc. by **neural networks** that were fine-tuned with respect to millions of parameters. Only something of that complexity has any hope of covering the diversity of human voice, pronunciation, vocabulary, grammar, etc. that "Alexa" confronts from one user to another. Fine-tuning that many parameters seems like a nightmare — how is it done?

Fortunately for us, the creators of the enabling algorithm[1] for fine-tuning (neural network training), called *back-propagation*, had a good working knowledge of the **chain rule of multi-variable calculus**, perhaps as a result of its repeated appearance in their undergraduate E&M course. In this exercise you derive the back-propagation rule using only the tools of calculus. No familiarity with neural networks is assumed.

First let's review how the outputs of a neural network are computed from the inputs — the act of forward propagation. We'll write the output variables compactly as $\{x_i : i \in O\}$, where indices i label nodes of the network, a subset O of which are the output nodes. If the network will be used for classifying images into two types (cat vs. dog), the network would have two output nodes.

The variable x_i is the output of a neuron whose input is the variable y_i. We write the neuron input-output relationship

$$x_i = f(y_i), \tag{1}$$

where f is called the neuron's *activation function*. For simplicity, all neurons in our network have the same activation function f.

The neuron input, y_i, is a linear function of the outputs of other neurons. We write the relationship as

$$y_i = \sum_{j \to i} x_j w_{j \to i}. \tag{2}$$

The parameter $w_{j \to i}$ gives the "weight" whereby the neuron output x_j contributes to y_i, and it is these parameters that get fine-tuned when the network is trained. If

[1] Rumelhart, Hinton & Williams (1985) *Learning internal representations by error propagation*

the summation symbol seems strange, think of it as a sum over all nodes j that are connected by a network edge $j \to i$ to the given node i.

Forward propagation in the network is defined recursively by equations (1) and (2). We start with given values $\{x_j : j \in I\}$ on a set I of input nodes (say holding the pixel grayscale values of an image). By (2) these are summed with weights and forward-propagated to another set of nodes where they become inputs to neurons, y_i. The activation functions (1) then produce neuron outputs which are then forward-propagated further, etc. until we arrive at the output nodes.

A training item is a pair $\{x_i^* : i \in I\}, \{x_i^* : i \in O\}$, where the input-node values are the image pixels of a pet, and the two output-node values are $\{1, 0\}$ for cat, or $\{0, 1\}$ for dog. Training a neural network is all about modifying the weights $w_{j\to i}$ on the network edges so that the correct classification, $\{1, 0\}$ vs. $\{0, 1\}$, appears on the output nodes when forward-propagating any image in the training set.

Suppose we are at a point in training where the weights are not yet correct. We have just forward-propagated the inputs of one item from the training set, $\{x_i^* : i \in I\}$ and the values we find on the output nodes, $\{x_i : i \in O\}$, do not match what we are given in the training item, $\{x_i^* : i \in O\}$. This is where calculus enters the picture. A "soft" way to encourage the network to give the correct output is to find weights that minimize the *loss function*

$$\mathcal{L} = \frac{1}{2} \sum_{i \in O} (x_i - x_i^*)^2. \tag{3}$$

Only if the actual output values x_i exactly match the target output values x_i^* will the loss be zero. A reasonable strategy for minimizing $\mathcal{L}$, hopefully all the way to zero, is to compute the gradient of $\mathcal{L}$ with respect to all the weight parameters and take steps in the "downhill" direction. The efficient calculation of the gradient is the back-propagation algorithm. We have broken the derivation into small steps below.

1. Consider any edge $j \to i$ of the network. Show that

$$\frac{\partial \mathcal{L}}{\partial w_{j\to i}} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial w_{j\to i}} = \frac{\partial \mathcal{L}}{\partial y_i} x_j. \tag{4}$$

   Hint: Consider how the value of $w_{j\to i}$ affects variables in the forward direction (leading to the output nodes where the loss is defined).

2. Consider the case where i is not an input node and show

$$\frac{\partial \mathcal{L}}{\partial y_i} = \frac{\partial \mathcal{L}}{\partial x_i} f'(y_i), \tag{5}$$

   where f' is the derivative of the activation function.

3. Now take a deep breath and think of all the ways that x_i, where i is not an output node, affects the loss to show

$$\frac{\partial \mathcal{L}}{\partial x_i} = \sum_{i \to k} \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial x_i} \tag{6}$$

$$= \sum_{i \to k} \frac{\partial \mathcal{L}}{\partial y_k} w_{i \to k} \,. \tag{7}$$

   Note that the sum is over all the nodes k that receive input from the given node i.

4. Now combine (5) and (7) to obtain

$$\frac{\partial \mathcal{L}}{\partial y_i} = f'(y_i) \sum_{i \to k} w_{i \to k} \frac{\partial \mathcal{L}}{\partial y_k} \,. \tag{8}$$

   Rewrite this as the recursion relation

$$z_i = f'(y_i) \sum_{i \to k} w_{i \to k} \, z_k \tag{9}$$

   for the quantity

$$z_i = \frac{\partial \mathcal{L}}{\partial y_i}. \tag{10}$$

5. Explain why "back-propagation" describes the order in which the z variables are computed over the network. Propagation starts at the output nodes. Find a formula for the starting values, $\{z_k : k \in O\}$, using the loss function (3) (which you have not used up to now).

6. Once all the z's are computed by back-propagation, the gradient of the loss, by (4), is simply

$$\frac{\partial \mathcal{L}}{\partial w_{j \to i}} = x_j z_i \,. \tag{11}$$

   Explain why "taking a step in the downhill gradient direction" means making the parameter changes

$$w_{j \to i} \to w_{j \to i} - \eta \, x_j z_i \,, \tag{12}$$

   where $\eta > 0$ is the step size or "learning rate".