Working out Backpropogation

Neural Network Structure

In this meeting, we went over the math behind neural networks: feed-forwarding, derivatives, and backpropagation. This document contains what we thought you need to know for implementing back-propagation.

Say that we have a feed-forward neural network consisting of L layers, where layer L is the output layer, and layer 0 is the input layer. Let $\vec{\mathbf{a}}^{(\ell)}$ represent the activations in the ℓ-th layer of the network. So if the input to our network is the vector $\vec{\mathbf{x}}$, then $\vec{\mathbf{a}}^{(0)} = \vec{\mathbf{x}}$. For the purposes of this writeup, vectors are 1-indexed, as opposed to in code where they are 0-indexed.

Say that layer ℓ has n_ℓ neurons.

Let $w_{ij}^{(\ell)}$ represent the weight on the edge from the j-th node in layer $\ell - 1$ to the i-th node in layer ℓ. Let $W^{(\ell)}$ be the matrix defined by

$$W^{(\ell)} = \begin{bmatrix} w_{11}^{(\ell)} & w_{12}^{(\ell)} & \cdots & w_{1n_{\ell-1}}^{(\ell)} \\ w_{21}^{(\ell)} & w_{22}^{(\ell)} & \cdots & w_{2n_{\ell-1}}^{(\ell)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_\ell 1}^{(\ell)} & w_{n_\ell 2}^{(\ell)} & \cdots & w_{n_\ell n_{\ell-1}}^{(\ell)} \end{bmatrix}$$

Viewed as a linear transformation, this is $W^{(\ell)} : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$, and so its dimension is $n_\ell \times n_{\ell-1}$.

Let $b_i^{(\ell)}$ be the bias associated with the i-th node of layer ℓ. Each layer of the network has a "squishification function" written as $\sigma^{(\ell)}$, so computing the activation $a_i^{(\ell)}$ can be written as

$$a_i^{(\ell)} = \sigma^{(\ell)}\left(z_i^{(\ell)}\right)$$

where we let

$$z_i^{(\ell)} = b_i^{(\ell)} + \sum_{j=1}^{n_{\ell-1}} w_{ij}\, a_j^{(\ell-1)}$$

We can also write this more succinctly as

$$\vec{\mathbf{a}}^{(\ell)} = \sigma\left(\vec{\mathbf{z}}^{(\ell)}\right)$$

where

$$\vec{\mathbf{z}}^{(\ell)} = W^{(\ell)}\vec{\mathbf{a}}^{(\ell-1)} + \vec{\mathbf{b}}^{(\ell)}$$

and where $\sigma(\vec{\mathbf{x}})$ is applied to each element of x.

Cost Gradients

For now, we'll be using squared loss. If for training sample 1 we desire the output layer to have value $\vec{\mathbf{y}}$,

$$C_1 = \|\vec{\mathbf{a}^{(\ell)}} - \vec{\mathbf{y}}\|_2^2 = \sum_{i=1}^{n_\ell}(a_i^{(\ell)} - y_i)^2$$

The overall cost for the network over all N training samples will be the average of all costs, so

$$C = \frac{1}{N}\sum_{k=1}^{N} C_k$$

We wish to compute the gradient, ∇C, of the loss function, so that we can take a step in the "downwards" direction along the surface formed by the graph of C in order to find a minimum of C. Since we only care about the direction the gradient is pointing and not the magnitude, the factor of $\frac{1}{N}$ in front can be ignored.[1] So, we care about computing

$$\nabla C \approx \nabla C_0 + \nabla C_1 + \cdots + \nabla C_N$$

For explanation purposes, we'll go through computing ∇C_0 for a label $\vec{\mathbf{y}}$, with input $\vec{\mathbf{x}} = \vec{\mathbf{a}}^{(0)}$. The gradient is

$$\nabla C_0 = \begin{bmatrix} \partial C_0 / \partial w_{00}^{(1)} \\ \vdots \\ \partial C_0 / \partial w_{ij}^{(1)} \\ \vdots \\ \partial C_0 / \partial w_{n_1 n_0}^{(1)} \\ \vdots \\ \partial C_0 / \partial b_i^{(1)} \\ \vdots \end{bmatrix}$$

Where the dimension of this vector is the number of total parameters (weights and biases) of our network. It's components each reflect how sensitive the overall cost is to a small change in one of the parameters, so we want to take a step in the most efficient direction to decrease the cost.

1. From here on out, for two vectors $\vec{\mathbf{v}}$ and $\vec{\mathbf{u}}$, $\vec{\mathbf{v}} \approx \vec{\mathbf{u}}$ will mean that the two vectors are pointing in the same direction, but may not have the same magnitude. More formally,

$$\vec{\mathbf{v}} \approx \vec{\mathbf{u}} \iff \frac{\vec{\mathbf{v}}}{\|\vec{\mathbf{v}}\|} = \frac{\vec{\mathbf{u}}}{\|\vec{\mathbf{u}}\|}$$

Computing Partial Derivatives

Using the chain rule, we can compute the derivative with respect to one of the weights in layer ℓ.

$$\frac{\partial C_0}{\partial w_{ij}^{(\ell)}} = \frac{\partial z_i^{(\ell)}}{\partial w_{ij}^{(\ell)}} \cdot \frac{\partial a_i^{(\ell)}}{\partial z_i^{(\ell)}} \cdot \frac{\partial C_0}{\partial a_i^{(\ell)}}$$

In the same manor we can compute the derivative with respect to one of the biases.

$$\frac{\partial C_0}{\partial b_i^{(\ell)}} = \frac{\partial z_i^{(\ell)}}{\partial b_i^{(\ell)}} \cdot \frac{\partial a_i^{(\ell)}}{\partial z_i^{(\ell)}} \cdot \frac{\partial C_0}{\partial a_i^{(\ell)}}$$

We can actually simplify these computations quite a lot. Using the formula for $z_i^{(\ell)}$, we know

$$z_i^{(\ell)} = b_i^{(\ell)} + \left(\sum_{j=1}^{n_{\ell-1}} w_{ij}^{(\ell)} a_j^{(\ell-1)} \right) \implies \frac{\partial z_i^{(\ell)}}{\partial w_{ij}^{(\ell)}} = a_j^{(\ell-1)}$$

When taking the derivative with respect to bias, this becomes much simpler.

$$z_i^{(\ell)} = b_i^{(\ell)} + \left(\sum_{j=1}^{n_{\ell-1}} w_{ij}^{(\ell)} a_j^{(\ell-1)} \right) \implies \frac{\partial z_i^{(\ell)}}{\partial b_i^{(\ell)}} = 1$$

Also, because $a_i^{(\ell)} = \sigma^{(\ell)}(z_i^{(\ell)})$, $\frac{\partial a_i^{(\ell)}}{\partial z_i^{(\ell)}} = \dot{\sigma}^{(\ell)}(z_i^{(\ell)})$ where $\dot{\sigma}$ is the derivative of σ. Together, this means

$$\frac{\partial C_0}{\partial w_{ij}^{(\ell)}} = a_j^{(\ell-1)} \dot{\sigma}^{(\ell)}(z_i^{(\ell)}) \cdot \frac{\partial C_0}{\partial a_i^{(\ell)}}$$
$$\frac{\partial C_0}{\partial b_i^{(\ell)}} = \dot{\sigma}^{(\ell)}(z_i^{(\ell)}) \cdot \frac{\partial C_0}{\partial a_i^{(\ell)}}$$

Let's use matrix notation to clean this up a bit. Let $\frac{\partial C_0}{\partial W^{(\ell)}}$ represent the matrix whose (i, j)-th entry is $\frac{\partial C_0}{\partial w_{ij}^{(\ell)}}$. Likewise, $\frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(\ell)}}$ is the vector whose i-th entry is $\frac{\partial C_0}{\partial b_i^{(\ell)}}$. Now, we can write

$$\frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(\ell)}} = \dot{\sigma}^{(\ell)}(\vec{\mathbf{z}}^{(\ell)}) \odot \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell)}} \quad \text{and} \quad \frac{\partial C_0}{\partial W^{(\ell)}} = \frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(\ell)}} \left(\vec{\mathbf{a}}^{(\ell-1)} \right)^\top$$

Where $\odot$ represents the point-wise *Hadamard product*.

This leaves the question of how to compute the derivative of C_0 with respect to a_i for each layer. Notice that if $\ell = L$ (we are in the last layer) this is actually quite straightforward. Using the definition of cost,

$$C_0 = \sum_{i=1}^{n_L} (a_i^{(L)} - y_i)^2$$

we can easily compute the derivative

$$\frac{\partial C_0}{\partial a_i^{(L)}} = 2(a_i^{(L)} - y_i) \quad \text{or} \quad \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(L)}} = 2(\vec{\mathbf{a}}^{(L)} - \vec{\mathbf{y}})$$

However, if we try to find an expression for the same derivative but in a previous layer, we find

$$\frac{\partial C_0}{\partial a_k^{(\ell-1)}} = \sum_{j=1}^{n_\ell} \frac{\partial z_j^{(\ell)}}{\partial a_i^{(\ell-1)}} \cdot \frac{\partial a_j^{(\ell)}}{\partial z_j^{(\ell)}} \cdot \frac{\partial C_0}{\partial a_j^{(\ell)}} = \sum_{j=1}^{n_\ell} w_{jk}^{(\ell)} \dot{\sigma}^{(\ell)}(z_j^{(\ell)}) \cdot \frac{\partial C_0}{\partial a_j^{(\ell)}}$$

In matrix notation, this is

$$\frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell-1)}} = W^{(\ell)\top} \left(\dot{\sigma}^{(\ell)} \left(\vec{\mathbf{z}}^{(\ell)} \right) \odot \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell)}} \right)$$

Notice this formula is recursive! To compute it efficiently, we can use a dynamic programming style of approach. This gives us the following natural algorithm for computing ∇C_0.

The Backpropogation Algorithm

(Base case of the DP table.) Start by computing all ${}^{\partial C_0}/_{\partial a_i^{(L)}} = 2(a_i^{(L)} - y_i)$ for $1 \leq i \leq n_L$. With this done, we can also calculate all

$$\frac{\partial C_0}{\partial w_{ij}^{(L)}} = a_i^{(L-1)} \dot{\sigma}^{(\ell)}(z_i^{(L)}) \frac{\partial C_0}{\partial a_i^{(L)}} \quad \text{and} \quad \frac{\partial C_0}{\partial b_i^{(L)}} = \dot{\sigma}^{(\ell)}(z_i^{(\ell)}) \frac{\partial C_0}{\partial a_i^{(L)}}$$

for the last layer L. In matrix form, this means computing

$$\begin{aligned} \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(L)}} &= 2\left(\vec{\mathbf{a}}^{(L)} - \vec{\mathbf{y}}\right) \\ \frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(L)}} &= \dot{\sigma}^{(L)}(\vec{\mathbf{z}}^{(L)}) \odot \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(L)}} \\ \frac{\partial C_0}{\partial W^{(L)}} &= \frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(L)}} \left(\vec{\mathbf{a}}^{(L-1)}\right)^\top \end{aligned}$$

(Recursive case of DP table) Now, iterating ℓ from $L-1$ down to 1, compute for all $1 \leq i \leq n_\ell$ the derivatives

$$\frac{\partial C_0}{\partial a_i^{(\ell)}} = \sum_{j=1}^{n_{(\ell+1)}} w_{ij}^{(\ell)} \dot{\sigma}^{(\ell+1)}(z_j^{(\ell+1)}) \frac{\partial C_0}{\partial a_j^{(\ell+1)}}$$

Again, in matrix form, this is computing

$$\frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell)}} = W^{(\ell)} \left(\dot{\sigma}^{(\ell+1)} \left(\vec{\mathbf{z}^{(\ell)}} \right) \odot \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell+1)}} \right)$$

Once these have been computed, one can directly compute

$$\frac{\partial C_0}{\partial w_{ij}^{(\ell)}} = a_j^{(\ell-1)} \dot{\sigma}^{(\ell)}(z_i^{(\ell)}) \frac{\partial C_0}{\partial a_i^{(\ell)}} \quad \text{and} \quad \frac{\partial C_0}{\partial b_i^{(\ell)}} = \dot{\sigma}^{(\ell)}(z_i^{(\ell)}) \frac{\partial C_0}{\partial a_i^{(\ell)}}$$

which is

$$\begin{aligned} \frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(\ell)}} &= \dot{\sigma}^{(\ell)}(\vec{\mathbf{z}}^{(\ell)}) \odot \frac{\partial C_0}{\partial \vec{\mathbf{a}}^{(\ell)}} \\ \frac{\partial C_0}{\partial W^{(\ell)}} &= \frac{\partial C_0}{\partial \vec{\mathbf{b}}^{(\ell)}} \left(\vec{\mathbf{a}}^{(\ell-1)} \right)^\top \end{aligned}$$

And that's it! This gives everything you need to fully compute ∇C_0.

Stochastic Gradient Descent

Fully computing $\nabla C \approx \nabla C_0, \ldots, \nabla C_N$ is very costly, as that's a lot of gradients to compute. So instead of recomputing ∇C and taking a step in the $-\nabla C$ direction every time, we first start by randomly partitioning our training set into B "batches." We'll say that $C_{k,b}$ is the cost of the network on the b-th sample of the k-th batch of our training set, and $\nabla C_b \approx \nabla C_{1,b} + \cdots + \nabla C_{N/B,b}$ for $1 \le b \le B$. At each step of gradient descent, we iterate over $1 \le b \le B$, taking a step in the $-\nabla C_b$ direction. We repeat this iteration until some other stopping condition.