DS 102 Discussion 12

Wednesday, November 25, 2020

In this discussion, we'll practice the nuts and bolts of backpropagation presented in Lecture 24 by applying it to a two-layer neural network. This will help demonstrate how backpropagation can efficiently compute the partial gradients of complicated functions.

1. **Backpropagation for a two-layer neural network.** Consider a two-layer neural network that computes a real-valued function of the form $f_{Ab}(x) = b^T\sigma(Ax)$ where $x \in \mathbb{R}^m$, $A \in \mathbb{R}^{h\times m}$, $b \in \mathbb{R}^{h\times 1}$, and σ is the element-wise sigmoid function given by $\sigma(y) = 1/(1+\exp(-x))$ (the subscript notation in f_{Ab} is used to emphasize that A and b are the parameters of the function). In other words, the neural network has input size m, h units in the hidden layer, and a single scalar output.

   The neural network f_{Ab} can be trained to predict a real-valued output given an m-dimensional input (a regression problem). Given a dataset of n input-output pairs, $\{(x_i, y_i)\}_{i=1}^n$, a common way of training a neural network to perform this task is to find the parameter values (values of the matrix A and the vector b) that minimize the squared error loss over the dataset:

   $$\operatorname*{argmin}_{A,b} \sum_{i=1}^{n} (y_i - f_{Ab}(x_i))^2 = \operatorname*{argmin}_{A,b} \sum_{i=1}^{n} (y_i - b^T\sigma(Ax_i))^2.$$

   To perform this minimization, gradient descent is conducted on the loss with respect to the parameters A, b.

   For simplicity, here we'll just focus on the partial derivatives of the squared error loss evaluated on a single data point, (x, y):

   $$\mathcal{L}(A, b) = (y - f_{Ab}(x))^2 = (y - b^T\sigma(Ax))^2.$$

   Backpropagation leverages the chain rule, along with dynamic programming, to compute the required partial derivatives $\frac{\partial\mathcal{L}(A,b)}{\partial A}$ and $\frac{\partial\mathcal{L}(A,b)}{\partial b}$ in an efficient way. This requires first computing intermediate quantities in the computation graph in what's called a "forward pass". That is, backpropagation first computes $\mathcal{L}(A, b)$ by computing the quantities $z_1 = Ax$, $z_2 = \sigma(z_1)$, $z_3 = b^T z_2$, the error $z_4 = y - z_3$, then finally the loss $\mathcal{L}(A, b) = z_4^2$. In the following problem parts, assume these have already been computed for your use.

   (a) Backpropagation then performs a "backward pass" to compute the partial derivatives, starting with $\frac{\partial\mathcal{L}(A,b)}{\partial b}$. Using the chain rule, write down an expression for $\frac{\partial\mathcal{L}(A,b)}{\partial b}$. Use intermediate quantities from the forward pass listed above wherever possible, since these have already been computed.

   Hint: Note that b is an h-dimensional vector, so the partial derivative will be an h-dimensional vector. The expression $b^T\sigma(Ax) = b^T z_2$ is a dot product between the vector b and the vector z_2. Recall that for a dot product between two vectors $v^T w$, we have $\frac{\partial v^T w}{v} = w$.

(b) Using the chain rule, write down an expression for $\frac{\partial \mathcal{L}(A,b)}{\partial A}$. Use intermediate quantities from the forward pass wherever possible.

Hint: A is an $h \times m$-dimensional matrix, so the partial derivative will be an $h \times m$-dimensional matrix. You can approach this problem by noting that

$$\frac{\partial \mathcal{L}(A,b)}{\partial A} = 2(y - b^T\sigma(Ax)) \cdot -\frac{\partial b^T\sigma(Ax)}{\partial A}$$

and finding the partial derivative of $b^T\sigma(Ax)$ with respect to each element A_{ij} of A. Use this result to write the partial derivative of A in terms of matrices and/or vectors. Note that the derivative of the sigmoid function is $\frac{\sigma(x)}{x} = \sigma(x)(1 - \sigma(x))$.