Prof. Ryan Cotterell

Yannick Wattenberg: Assignment 01

ywattenberg@inf.ethz.ch, 19-947-464.

11/11/2022 - 09:42h

1 Question: Efficiently Computing the Hessian

a)

To prove:

$$\nabla^2 f(x) \overset{!}{=} \begin{bmatrix} (\nabla(e_1^T \nabla f(x)))^T \\ \vdots \\ (\nabla(e_n^T \nabla f(x)))^T \end{bmatrix} \tag{1}$$

Prove by simple derivation:

$$(e_i^T \nabla f(x))) \tag{2}$$

$$= [0, ..., 0, 1, 0, ..., 0] \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \quad \text{(def. of derivative } (\nabla)) \tag{3}$$

$$= \frac{\partial f(x)}{\partial x_i} \tag{4}$$

$$\Rightarrow \tag{5}$$

$$\begin{bmatrix} (\nabla(e_1^T \nabla f(x)))^T \\ \vdots \\ (\nabla(e_n^T \nabla f(x)))^T \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} (\nabla(\frac{\partial f(x)}{\partial x_1}))^T \\ \vdots \\ (\nabla(\frac{\partial f(x)}{\partial x_n}))^T \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} 2\frac{\partial f}{\partial x_1 \partial x_1} & ... & 2\frac{\partial f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ 2\frac{\partial f}{\partial x_n \partial x_1} & ... & 2\frac{\partial f}{\partial x_n \partial x_n} \end{bmatrix} \quad \text{(def. of derivative}(\nabla)) \tag{8}$$

$$= \text{Hessian(f)} \quad \text{(def. of Hessian)} \tag{9}$$

b)

The hessian is by definition the derivative of the jacobian of f, here we assume that f is twice differentiable. We can understand the jacobian as a simple function F, where F maps the input x to an output vector instead of a simple real value. As F can be calculated by the backpropagation algorithm presented in the lecture we can again use backpropagation over this algorithm, assuming that $f(x)$ is made up of predefined atomic functions for which we can precompute the derivatives.

In more concrete terms, we have shown in a) that it is enough to compute $(\nabla(e_i^T \nabla f(x)))^T$ for $i \in \{1, ..., n\}$ to be able to compute the Hessian. We can understand $e_i^T(\nabla f(x))$ as a function $F_i(x) = \frac{\partial f(x)}{\partial x_i}$ which like $f(x)$ outputs a single real value, thus we can again use the previously introduced backpropagation algorithm over $F_i(x)$ to calculate $\nabla(e_i^T \nabla f(x))$. As one would imagine some considerations have to be made if $F_i(x)$ is the derivative of a function. These considerations can be expressed in the computation graph of $F_i(x)$ which will be the computation graph of $f(x)$ augmented by the calculation made in the backpropagation algorithm. This means adding to the computation graph of $f(x)$ (forward-propagation part of the algorithm), the reverse of the graph of $f(x)$ where each node corresponds to the partial-derivative of f with respect to the variable represented by this node.

The naive algorithm which calculates each entry of the hessian individually will be in $\mathcal{O}(n^2)$ multiplied by the time it takes to calculate the second derivative of $f(x)$ with respect to some $x_i, x_j; i, j \in [0, ..., n]$, which will be bound from below by the time it takes to calculate $f(x)$. Thus the naive approach is in $\mathcal{O}(n^2 \cdot m)$ time.

From the fact that backpropagation on $f(x)$ is possible in $\mathcal{O}(m)$ time, it follows that the above-explained backpropagation over $F_i(x)$ is in $\mathcal{O}(m)$ time. This holds as the computation of $F_i(x)$ is given to be in $\mathcal{O}(m)$ this means we can calculate $f(x)$ and its derivative in $\mathcal{O}(m)$ time, further as we have learned in the lecture we can compute the derivative of a function, which consists of a composition of simple functions, in the same time that is necessary to calculate the function (forward-pass). This applies to $F_i(x)$ as its computation graph is a composition of the graph of f and the aforementioned partial derivatives. Finally, to compute the full Hessian we need to compute the derivative of $F_i(x)$ for all $i \in [0, ..., n]$ giving a runtime in $n \cdot \mathcal{O}(m) = \mathcal{O}(nm)$ time.

c)

The concept from b) can be extended to the third or higher derivatives. The method works, in the same manner, using backpropagation over the calculation of the previous derivatives, building up the computational graph of the k-1st derivatives while calculating them. This Graph will again consist of the graph of the k-2nd derivatives extended by its mirror. Important to note here is that this graph will not be connected, for example, the graph for the second derivative will consist of n different components corresponding to the n-times we execute backpropagation (for each $F_i(x)$). To calculate the k-th derivative we then need to run backpropagation for each result of the k-1st derivative. For example, to calculate the third derivative we need to run backpropagation on each element of the hessian. This algorithm can be improved assuming that second partial derivatives are all continuous for our function f, then the hessian is always symmetric which means we only need to run backpropagation on roughly half the entries of the Hessian. This algorithm will then have a runtime of $\mathcal{O}(m) + \frac{n^2}{2} \cdot \mathcal{O}(m) \in \mathcal{O}(n^2 \cdot m)$ where n is the length of the input vector X. The first addend denotes the time needed to build the hessian and subsequently, its computation graph. The second addend denotes the $\frac{n^2}{2}$ times that we need to execute backpropagation $\mathcal{O}(m)$. The execution of backpropagation is in $\mathcal{O}(m)$ as we can calculate each partial-derivative/element of the hessian in this time and thus as we know from the lecture the execution of backpropagation is also possible in the same time. With the same argument, we get a runtime for the k-th derivative:

$$\mathcal{O}(m) + \sum_{i=1}^{k} (n^{i-1} \cdot \mathcal{O}(m)) \in \mathcal{O}(n^{k-1} \cdot m) \tag{10}$$