CSED490F Lab: Autograd

Team 4: Yunkyu Lee (20210733), Hyeonu Cho (20230740)

September 14, 2025

1 Automatic Differentiation

Reverse-mode automatic differentiation performs back-to-front accumulation of local gradients based on the chain rule. Consider a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, and its $m \times n$ Jacobian $\mathbf{J}_f$. For $\mathbf{y} = f(\mathbf{x})$, suppose we know $\nabla_{\mathbf{y}}\mathcal{L}$, the gradient of some scalar value $\mathcal{L}$ w.r.t. $\mathbf{y}$. Then, we can obtain $\nabla_{\mathbf{x}}\mathcal{L}$ as a vector-Jacobian product (VJP):

$$\nabla_{\mathbf{x}}\mathcal{L} = (\nabla_{\mathbf{y}}\mathcal{L})^\top \mathbf{J}_f$$

As $\mathbf{J}_{f_1 \circ f_2} = \mathbf{J}_{f_1} \circ \mathbf{J}_{f_2}$, we can chain the above VJP in order to obtain gradients of composed functions. Ttherefore, knowing the local gradients of function outputs w.r.t. their inputs is sufficient for reverse-mode automatic differentiation. We describe the local gradients of some operators below.

1.1 Basic Operations

$$\texttt{Add}(x, y) = x + y, \quad \frac{\partial \texttt{Add}}{\partial x} = 1, \quad \frac{\partial \texttt{Add}}{\partial y} = 1 \tag{1}$$

$$\texttt{Mul}(x, y) = x \times y, \quad \frac{\partial \texttt{Mul}}{\partial x} = y, \quad \frac{\partial \texttt{Mul}}{\partial y} = x \tag{2}$$

$$\texttt{Pow}(x, y) = x^y, \quad \frac{\partial \texttt{Pow}}{\partial x} = yx^{y-1}, \quad \frac{\partial \texttt{Pow}}{\partial y} = x^y \log x \tag{3}$$

$$\texttt{Log}(x) = \log x, \quad \frac{\partial \texttt{Log}}{\partial x} = x^{-1}, \tag{4}$$

$$\texttt{ReLU}(x) = \max(x, 0), \quad \frac{\partial \texttt{ReLU}}{\partial x} = yx^{y-1}, \tag{5}$$

$$\texttt{Sum}(x, \texttt{axis}) = \sum_{i \in \texttt{axis}} x_i, \quad \frac{\partial \texttt{Sum}}{\partial x} = 1, \tag{6}$$

$$\texttt{MatMul}(x, y) = xy, \quad \frac{\partial \texttt{MatMul}}{\partial x} = y, \quad \frac{\partial \texttt{MatMul}}{\partial y} = x \tag{7}$$

1.2 Classification

Following [1], we use a shifted softmax for numerical stability:

$$\texttt{Softmax}(x_j) = \frac{\exp(x_j - s)}{\sum_i \exp(x_i - s)}$$

The negative log-likelihood loss (with log-probability input)

$$\texttt{NLLLoss}(\log \hat{y}, y) = -y \log \hat{y}$$

The cross-entropy loss (with logit inputs) can be seen as a combination of `Softmax` and `NLLLoss`.

1.3 Matrix Multiplication

1.4 Cross Entropy Loss

2 MNIST Classification

Utilizing the operators described in section 1, we perform testing on the MNIST dataset.

References

[1] Pierre Blanchard, Desmond J Higham, and Nicholas J Higham. “Accurately computing the log-sum-exp and softmax functions”. In: *IMA Journal of Numerical Analysis* 41.4 (Aug. 2020), pp. 2311–2330. ISSN: 0272-4979. DOI: 10.1093/imanum/draa038. eprint: https://academic.oup.com/imajna/article-pdf/41/4/2311/40758053/draa038.pdf. URL: https://doi.org/10.1093/imanum/draa038.