

# CSED490F Lab: Autograd

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

September 16, 2025

## 1 Automatic Differentiation

Reverse-mode automatic differentiation performs back-to-front accumulation of local gradients based on the chain rule.

We omit the definitions and the local gradients of the **Add**, **Mul**, **Pow**, **Log**, and **Sum** operations as they are trivial, and show **ReLU** as an example.

$$\text{ReLU}(x) = \max(x, 0), \quad \frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1 & x > 0, \\ 0 & x < 0 \end{cases}$$

### 1.1 Matrix Multiplication

We define **MatMul** as the matrix multiplication operator. Let  $z = \text{MatMul}(x, y)$  for  $x \in \mathbb{R}^{N \times D}$  and  $y \in \mathbb{R}^{D \times M}$ . Then, each element of  $z \in \mathbb{R}^{N \times M}$  is computed as  $z_{n,m} = \sum_d x_{n,d} y_{d,m}$ . Any element  $x_{n,m}$  of  $x$  only contributes to row  $n$  of  $z$ :

$$z_{n,*} = [\sum_d x_{n,d} y_{d,1} \quad \sum_d x_{n,d} y_{d,2} \quad \cdots \quad \sum_d x_{n,d} y_{d,M-1} \quad \sum_d x_{n,d} y_{d,M}]$$

This gives  $\partial \mathcal{L} / \partial x$  as the following:

$$\frac{\partial \mathcal{L}}{\partial x_{n,d}} = \sum_m \frac{\partial \mathcal{L}}{\partial z_{n,m}} \frac{\partial z_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial \mathcal{L}}{\partial z_{n,m}} y_{d,m} = \frac{\partial \mathcal{L}}{\partial z_{n,*}} y_{d,*}^\top$$

Finally, we obtain the backpropagation rules for **MatMul**.

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial \text{MatMul}(x, y)} y^\top, \quad \frac{\partial \mathcal{L}}{\partial y} = x^\top \frac{\partial \mathcal{L}}{\partial \text{MatMul}(x, y)}$$

### 1.2 Classification

**Softmax.** Following [1], we implement the softmax function with a shift for numerical stability. This shift has no impact on the output result or the gradient.

$$\text{Softmax}(x)_j = \frac{\exp(x_j - s)}{\sum_i \exp(x_i - s)}$$

By simple differentiation, we can obtain  $\partial \text{Softmax}(x)_i / \partial x_j$  for  $i = j$  and  $i \neq j$  as the following:

$$\frac{\partial \text{Softmax}(x)_j}{\partial x_j} = \text{Softmax}(x)_j - \text{Softmax}(x)_j^2, \quad \frac{\partial \text{Softmax}(x)_i}{\partial x_j} = -\text{Softmax}(x)_i \text{Softmax}(x)_j$$

Using the above gradients, we can derive a backpropagation rule in matrix form:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_j} &= \sum_i \frac{\partial \mathcal{L}}{\partial \text{Softmax}(x)_i} \frac{\partial \text{Softmax}(x)_i}{\partial x_j} \\ &= \frac{\partial \mathcal{L}}{\partial \text{Softmax}(x)_j} (\text{Softmax}(x)_j - \text{Softmax}(x)_j^2) - \sum_{i \neq j} \frac{\partial \mathcal{L}}{\partial \text{Softmax}(x)_i} \text{Softmax}(x)_i \text{Softmax}(x)_j \\ &= \text{Softmax}(x)_j \left( \frac{\partial \mathcal{L}}{\partial \text{Softmax}(x)_j} - \sum_i \frac{\partial \mathcal{L}}{\partial \text{Softmax}(x)_i} \text{Softmax}(x)_i \right) \end{aligned}$$

**Negative log-likelihood.** The negative log-likelihood loss (with log-probability input) is defined as the following, with trivial local gradients:

$$\text{NLLLoss}(\log \hat{p}, p) = - \sum_i p_i \log \hat{p}_i, \quad \frac{\partial \text{NLLLoss}(\log \hat{p}, p)}{\partial p} = - \log \hat{p}, \quad \frac{\partial \text{NLLLoss}(\log \hat{p}, p)}{\partial \log \hat{p}} = -p$$

**Cross-entropy.** The cross-entropy loss (with logit inputs) can be seen as a composition of **Softmax** and **NLLoss**.

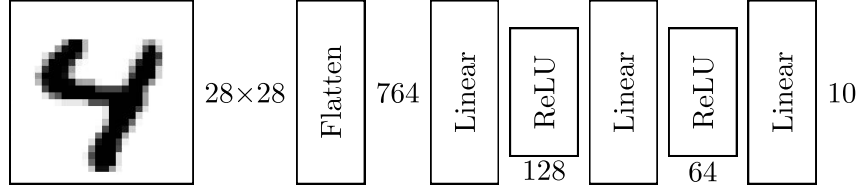
$$\text{CrossEntropyLoss}(\hat{x}, p) = \text{NLLLoss}(\log(\text{Softmax}(\hat{x})), p) = - \sum_i p_i \log(\text{Softmax}(\hat{x}_i))$$

The local gradients can be found accordingly by the chain rule on **Log**, **Softmax**, and **NLLoss**.

$$\frac{\partial \text{CrossEntropyLoss}(\hat{x}, p)}{\partial \hat{x}_j} = \text{Softmax}(\hat{x})_j - p_j, \quad \frac{\partial \text{CrossEntropyLoss}(\hat{x}, p)}{\partial p_j} = - \log(\text{Softmax}(\hat{x}))_j$$

## 2 MNIST Classification

Utilizing the operators described in section 1, we train an MLP  $f_\theta$  to perform classification on the MNIST dataset. The MLP architecture used is shown below.



The cross-entropy loss is used with L2 regularization. We found that using **NLLoss**, **Log**, and **Softmax** separately resulted in numerical instability and NaN values during training, so the fused **CrossEntropyLoss** operator was used directly.

$$\begin{aligned} \mathcal{L}(\theta; I, p) &= \mathcal{L}_{\text{class}}(\theta; I, p) + \lambda_{\text{L2}} \mathcal{L}_{\text{L2}}(\theta) \\ \mathcal{L}_{\text{class}}(\theta; I, p) &= \text{CrossEntropy}(f_\theta(I_i), p_i) = - \sum p_i \log(\text{Softmax}(f_\theta(I_i))) \\ \mathcal{L}_{\text{L2}}(\theta) &= \sum_{W \in \theta} \|W\|_2^2 \end{aligned}$$

The model is trained for 10 epochs with a batch size of 100 and a learning rate of 0.1 via SGD. We report our results on varying  $\lambda_{\text{L2}}$  values below, taking the mean of 50 runs for each setting.

$\lambda_{\text{L2}}$	0	$10^{-7}$	$10^{-6}$	$10^{-5}$	$10^{-4}$	$10^{-3}$	$10^{-2}$
Acc. % ( $\uparrow$ )	97.23	97.17	97.26	97.24	97.21	96.74	90.81

We conclude that L2 regularization is not necessary in this problem. While  $\lambda_{\text{L2}} = 10^{-6}$  did give the best results, the improvements were marginal and could be attributed to stochasticity. Large  $\lambda_{\text{L2}}$  values were significantly detrimental to training. We attribute this to the low parameter count and the short training duration.

## 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.