1 Background and Notation

Unless otherwise noted, let $\boldsymbol{A} \in \mathbb{R}^{n \times n}; \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n$ be the variables in the (sparse) system of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{1}$$

For iterative solvers, denote $\boldsymbol{x}_k$, $k \geq 0$ to be the approximate solution to eq. (1) at iteration k, along with residual $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$, and error $\boldsymbol{e}_k = \boldsymbol{x} - \boldsymbol{x}_k$.

For some sparse matrix $\boldsymbol{M}$, let $\operatorname{mask}(\boldsymbol{M})$ denote the *sparsity mask* of $\boldsymbol{M}$ like

$$[\operatorname{mask}(\boldsymbol{M})]_{ij} = \begin{cases} 1 & m_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

In other words, this has the identity $\boldsymbol{M} \odot \operatorname{mask}(\boldsymbol{M}) = \boldsymbol{M}$.

1.1 Chain Rule

For background, we begin with a refresher on the multivariate chain rule[2],

Theorem 1.1. *Given some function $f : \mathbb{R}^{n_x} \to \mathbb{R}$ and $\boldsymbol{t} \in \mathbb{R}^{n_t}, \boldsymbol{x}(\boldsymbol{t}) \in \mathbb{R}^{n_x}$, the partial derivative $\frac{\partial z}{\partial t_i}$ for $z = f(\boldsymbol{x}(\boldsymbol{t}))$ is given by*

$$\frac{\partial z}{\partial t_i} = \sum_{j=1}^{n_x} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial t_i}. \tag{3}$$

For the general case, we can extend theorem 1.1 for arbitrary tensor-valued functions, by tensors we mean n-dimensional arrays for nonnegative integer n. For a more extensive treatment on tensor computations and the notation used, see [3].

Corollary 1.1.1. *Letting $I_1, I_2, \ldots I_N$ denote the indices of an N-dimensional tensor, given a function $f : \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N_X}} \to \mathbb{R}$, and $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_{N_T}}, \boldsymbol{\mathcal{X}}(\boldsymbol{\mathcal{T}}) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N_X}}$, the partial derivative $\frac{\partial z}{\partial t_{j_1, j_2, \ldots, j_{N_T}}}$ for $z = f(\boldsymbol{\mathcal{X}}(\boldsymbol{\mathcal{T}}))$ is given by*

$$\frac{\partial z}{\partial t_{j_1, j_2, \ldots, j_{N_T}}} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_{N_X}=1}^{I_{N_X}} \frac{\partial z}{\partial x_{i_1, i_2, \ldots, i_{N_X}}} \frac{\partial x_{i_1, i_2, \ldots, i_{N_X}}}{\partial t_{j_1, j_2, \ldots, j_{N_T}}}. \tag{4}$$

Proof. (handwavey): This result comes directly from theorem 1.1. Let $\boldsymbol{t}, \boldsymbol{x}(\boldsymbol{t})$ be arbitrary (consistent) flattenings of tensors $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{X}}$ into column vectors. Use these $\boldsymbol{t}$ and $\boldsymbol{x}(\boldsymbol{t})$ directly in eq. (3) and de-flatten outputs to obtain the summations. Reshaping tensors does not affect gradient propagation, as it is just a re-ordering of elements. □

Definition 1.1. In corollary 1.1.1, denote the term being right-multiplied in the summation as the *generalized gradient*, which we will define like

$$[\nabla_{\boldsymbol{\mathcal{X}}}(z)]_{i_1, i_2, \ldots, i_{N_X}} = \frac{\partial z}{\partial x_{i_1, i_2, \ldots, i_{N_X}}}. \tag{5}$$

This is to be read like "the gradient of z with respect to $\boldsymbol{\mathcal{X}}$." This value will have the same shape as $\boldsymbol{\mathcal{X}}$ itself.

Definition 1.2. Furthermore, in corollary 1.1.1, we will refer to the term being left-multiplied in the summation as the *generalized Jacobian*, defined like

$$[J_{\boldsymbol{\mathcal{T}}}(\boldsymbol{\mathcal{X}})]_{(i_1, i_2, \ldots, i_{N_X}), (j_1, j_2, \ldots, j_{N_T})} = \frac{\partial x_{i_1, i_2, \ldots, i_{N_X}}}{\partial t_{j_1, j_2, \ldots, j_{N_T}}}, \tag{6}$$

where the parentheses in the indexing are used only to emphasize that the first N_X indices are used for the input, while the last N_T indices are used for the output. From this, we have that eq. (4) can be alternatively denoted as the tensor contraction of $\nabla_{\boldsymbol{\mathcal{X}}}(z) J_{\boldsymbol{\mathcal{X}}(\boldsymbol{\mathcal{T}})}$ over the indices $i_1, i_2, \ldots, i_{N_X}$.

1.2 Computation Graph

General familiarity with the concept of a computation graph[4, 1] will be assumed. In an automatic differentiation environment, running some set of computations in *forward mode* creates a DAG that encodes the order in which functions are composed with one another. After the forward pass is complete, one can start from a scalar *leaf* in the graph and traverse backwards in a *reverse mode* to generate gradient information for each node with respect to the leaf.

At each node when we are performing the backwards propagation, we are computing the equivalent of eq. (4) on each node with respect to that node's output edges. The implementation of this is described in the follow example code snippet.

```
class MyFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        return 2. * x

    @staticmethod
    def backward(ctx, grad_x):
        return 2. * grad_x
```

For some tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N_X}}$, this trivially computes $\boldsymbol{\mathcal{Y}} = 2\boldsymbol{\mathcal{X}}$. This gives the generalized Jacobian

$$[J_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{Y}})]_{(i_1,i_2,\ldots,i_{N_X}),(i_1,i_2,\ldots,i_{N_X})} = 2, \tag{7}$$

which is 0 at every other index. When doing the backward pass (and assuming $\boldsymbol{\mathcal{Y}}$ is further given to some function that eventually outputs a scalar z), the gradient return value is given by

$$\nabla_{\boldsymbol{\mathcal{X}}}(z)\, J_{\boldsymbol{\mathcal{X}}}(\boldsymbol{\mathcal{Y}}) = 2\nabla_{\boldsymbol{\mathcal{X}}}(z), \tag{8}$$

showing that while the generalized Jacobian is a notational convenience, it is often unnecessary and indeed can be expensive in memory usage to actually form in practice. Instead, for functions like these we are interested in computing *the action* of multiplying the gradient by the Jacobian.

2 Gradient Derivations

2.1 Spmv Product

This is implementing the basic linear algebra operation

$$\boldsymbol{w} = \alpha \boldsymbol{A}\boldsymbol{x} + \beta \boldsymbol{y}, \tag{9}$$

for $\boldsymbol{A} \in \mathbb{R}^{n\times m}, \boldsymbol{x} \in \mathbb{R}^m; \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^n; \alpha, \beta \in \mathbb{R}$. This has the component-wise forward pass of

$$w_i = \alpha \left(\sum_k a_{i,k} x_k \right) + \beta y_i. \tag{10}$$

Taking partial derivatives gives

$$\frac{\partial w_i}{\partial a_{ij}} = \alpha x_j, \tag{11}$$

$$\frac{\partial w_i}{\partial x_j} = \alpha a_{ij} \tag{12}$$

$$\frac{\partial w_i}{\partial \alpha} = \sum_k a_{ik} x_k, \tag{13}$$

$$\frac{\partial w_i}{\partial y_i} = \beta, \tag{14}$$

$$\frac{\partial w_i}{\partial \beta} = y_i. \tag{15}$$

Applying the chain rule to some function $f : \mathbb{R}^n \to \mathbb{R}$ like $z = f(\boldsymbol{w})$ results in the update rules

$$\frac{\partial z}{\partial a_{ij}} = \sum_k \frac{\partial z}{\partial w_k} \frac{\partial w_k}{\partial a_{i,j}} = \alpha \frac{\partial z}{\partial w_i} x_j \implies \nabla_{\boldsymbol{A}}(z) = \alpha \left(\nabla_{\boldsymbol{w}}(z) \boldsymbol{x}^T \right) \odot \operatorname{mask}(\boldsymbol{A}) \tag{16}$$

$$\frac{\partial z}{\partial x_j} = \sum_k \frac{\partial z}{\partial w_k} \frac{\partial w_k}{\partial x_j} = \alpha \sum_k \frac{\partial z}{\partial w_k} a_{k,j} \implies \nabla_{\boldsymbol{x}}(z) = \alpha \boldsymbol{A}^T \nabla_{\boldsymbol{w}}(z) \tag{17}$$

$$\frac{\partial z}{\partial \alpha} = \sum_k \frac{\partial z}{\partial w_j} \frac{\partial w_j}{\partial \alpha} = \sum_j \frac{\partial z}{\partial w_j} \sum_k a_{jk} x_k \implies \nabla_\alpha(z) = \nabla_{\boldsymbol{w}}(z)^T \boldsymbol{A}\boldsymbol{x} \tag{18}$$

$$\frac{\partial z}{\partial y_i} = \sum_k \frac{\partial z}{\partial w_k} \frac{\partial w_k}{\partial y_i} = \beta \frac{\partial z}{\partial w_i} \implies \nabla_{\boldsymbol{y}}(z) = \beta \nabla_{\boldsymbol{w}}(z) \tag{19}$$

$$\frac{\partial z}{\partial \beta} = \sum_k \frac{\partial z}{\partial w_k} \frac{\partial w_k}{\partial \beta} = \sum_k \frac{\partial z}{\partial w_k} y_k \implies \nabla_\beta(z) = \nabla_{\boldsymbol{w}}(z)^T \boldsymbol{y}. \tag{20}$$

2.2 Spmm Product

This will go through the derivation of the sparse matrix-matrix product defined by

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}, \tag{21}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{B} \in \mathbb{R}^{n \times o}, \boldsymbol{C} \in \mathbb{R}^{m \times o}$, and one or both of $\boldsymbol{A}, \boldsymbol{B}$ are sparse.

The forward pass is trivially defined (entrywise) like

$$c_{ij} = \sum_k^n a_{ik} b_{kj}, \tag{22}$$

which emits the partial derivatives

$$\frac{\partial c_{ij}}{\partial a_{ik}} = b_{kj}, \tag{23}$$

$$\frac{\partial c_{ij}}{\partial b_{kj}} = a_{ik}. \tag{24}$$

Applying the chain rule to some function $f : \mathbb{R}^{m \times o} \to \mathbb{R}$ like $z = f(\boldsymbol{C})$ results in the update rules

$$\frac{\partial z}{\partial a_{ij}} = \sum_{k=1}^m \sum_{l=1}^o \frac{\partial z}{\partial c_{kl}} \frac{\partial c_{kl}}{\partial a_{ij}} = \sum_{l=1}^o \frac{\partial z}{\partial c_{il}} b_{jl} \implies \nabla_{\boldsymbol{A}}(z) = \left(\nabla_{\boldsymbol{C}}(z) \boldsymbol{B}^T \right) \odot \operatorname{mask}(\boldsymbol{A}), \tag{25}$$

$$\frac{\partial z}{\partial b_{ij}} = \sum_{k=1}^m \sum_{l=1}^o \frac{\partial z}{\partial c_{kl}} \frac{\partial c_{kl}}{\partial b_{ij}} = \sum_{k=1}^m \frac{\partial z}{\partial c_{kj}} a_{ki} \implies \nabla_{\boldsymbol{B}}(z) = \left(\boldsymbol{A}^T \nabla_{\boldsymbol{C}}(z) \right) \odot \operatorname{mask}(\boldsymbol{B}). \tag{26}$$

In the case where one of $\boldsymbol{A}$ or $\boldsymbol{B}$ are dense, their respective mask becomes dense as well, i.e. $[\operatorname{mask}(\boldsymbol{X})]_{ij} = 1$ for $\boldsymbol{X}$ equal to $\boldsymbol{A}$ or $\boldsymbol{B}$.

2.3 Linear Combination of Matrices

Let $\boldsymbol{C}$ be defined as the linear combination of two matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times m}$ like

$$\boldsymbol{C} = \alpha \boldsymbol{A} + \beta \boldsymbol{B}. \tag{27}$$

From this, we easily get the Jacobian tensors

$$[J_{\boldsymbol{A}}(\boldsymbol{C})]_{(i,j),(i,j)} = \alpha, \tag{28}$$

$$[J_{\boldsymbol{B}}(\boldsymbol{C})]_{(i,j),(i,j)} = \beta, \tag{29}$$

$$J_\alpha(\boldsymbol{C}) = \boldsymbol{A}, \tag{30}$$

$$J_\beta(\boldsymbol{C}) = \boldsymbol{B}, \tag{31}$$

which for $f : \mathbb{R}^{n \times m} \to \mathbb{R}$ and $z = f(\boldsymbol{C})$, gives the update rules

$$\frac{\partial z}{\partial \boldsymbol{A}} = \nabla_{\boldsymbol{C}}(z) \, J_{\boldsymbol{A}}(\boldsymbol{C}) = \alpha \nabla_{\boldsymbol{C}}(z) \odot \mathrm{mask}(\boldsymbol{A}), \tag{32}$$

$$\frac{\partial z}{\partial \boldsymbol{B}} = \nabla_{\boldsymbol{C}}(z) \, J_{\boldsymbol{B}}(\boldsymbol{C}) = \beta \nabla_{\boldsymbol{C}}(z) \odot \mathrm{mask}(\boldsymbol{B}), \tag{33}$$

$$\frac{\partial z}{\partial \alpha} = \nabla_{\boldsymbol{C}}(z) \, J_{\alpha}(\boldsymbol{C}) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}, \tag{34}$$

$$\frac{\partial z}{\partial \beta} = \nabla_{\boldsymbol{C}}(z) \, J_{\beta}(\boldsymbol{C}) = \sum_{i=1}^{n} \sum_{j=1}^{m} b_{ij}. \tag{35}$$

2.4 LU Column Scale

For some column and starting row k for matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, let the *column scaling operation* in LU be defined entrywise like

$$b_{ij} = \begin{cases} a_{ij}/a_{kk} & i > k, j = k \\ a_{ij} & \text{otherwise} \end{cases}. \tag{36}$$

This gives entrywise partial derivatives like

$$\frac{\partial b_{ij}}{\partial a_{ij}} = \begin{cases} 1 & i = j = k \\ 1/a_{kk} & i > k, j = k \\ 1 & \text{otherwise} \end{cases}, \tag{37}$$

$$\frac{\partial b_{ij}}{\partial a_{kk}} = \begin{cases} 0 & i = j = k \\ -a_{ij}/a_{kk}^2 & i > k, j = k \\ 0 & \text{otherwise} \end{cases}, \tag{38}$$

which for $f : \mathbb{R}^{n \times m} \to \mathbb{R}$ and $z = f(\boldsymbol{B})$, gives the update rule

$$\frac{\partial z}{\partial a_{ij}} = \sum_{x} \sum_{y} \frac{\partial z}{\partial b_{xy}} \frac{\partial b_{xy}}{\partial a_{ij}} = \begin{cases} \frac{\partial z}{\partial b_{ij}} - \sum_{x=k+1}^{n} \frac{\partial z}{\partial b_{xk}} a_{xk}/a_{kk} & i = j = k \\ \frac{\partial z}{\partial b_{ij}}/a_{kk} & i > k, j = k \\ \frac{\partial z}{\partial b_{ij}} & \text{otherwise} \end{cases}. \tag{39}$$

References

[1] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, *Automatic differentiation in machine learning: a survey*, CoRR, abs/1502.05767 (2015).

[2] J. Johnson, *Derivatives, backpropagation, and vectorization.* `http://cs231n.stanford.edu/handouts/derivatives.pdf`, September 2017.

[3] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, CoRR, abs/1912.01703 (2019).