BACKPROPAGATION IN PYTHON, C++, AND CUDA

MAZIAR RAISSI

1. Outline. This is a short tutorial on backpropagation and its implementation in Python, C++, and Cuda. The full codes for this tutorial can be found in `https://github.com/maziarraissi/backprop`.

2. Feed Forward. Let us consider the following densely connected deep neural network

$$
\begin{array}{rlllll}
F = A_\ell W_\ell + b_\ell, & F \in \mathbb{R}^{n\times p_{\ell+1}}, & A_\ell \in \mathbb{R}^{n\times p_\ell}, & W_\ell \in \mathbb{R}^{p_\ell \times p_{\ell+1}}, & b_\ell \in \mathbb{R}^{1\times p_{\ell+1}},\\
A_\ell = \tanh(H_\ell), & A_\ell \in \mathbb{R}^{n\times p_\ell}, & H_\ell \in \mathbb{R}^{n\times p_\ell}, & & \\
H_\ell = A_{\ell-1} W_{\ell-1} + b_{\ell-1}, & H_\ell \in \mathbb{R}^{n\times p_\ell}, & A_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & W_{\ell-1} \in \mathbb{R}^{p_{\ell-1}\times p_\ell}, & b_{\ell-1} \in \mathbb{R}^{1\times p_\ell},\\
A_{\ell-1} = \tanh(H_{\ell-1}), & A_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & H_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & & \\
H_{\ell-1} = A_{\ell-2} W_{\ell-2} + b_{\ell-2}, & H_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & A_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & W_{\ell-2} \in \mathbb{R}^{p_{\ell-2}\times p_{\ell-1}}, & b_{\ell-2} \in \mathbb{R}^{1\times p_{\ell-1}},\\
A_{\ell-2} = \tanh(H_{\ell-2}), & A_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & H_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & & \\
\vdots & \vdots & \vdots & \vdots & \vdots\\
H_2 = A_1 W_1 + b_1, & H_2 \in \mathbb{R}^{n\times p_2}, & A_1 \in \mathbb{R}^{n\times p_1}, & W_1 \in \mathbb{R}^{p_1\times p_2}, & b_1 \in \mathbb{R}^{1\times p_2},\\
A_1 = \tanh(H_1), & A_1 \in \mathbb{R}^{n\times p_1}, & H_1 \in \mathbb{R}^{n\times p_1}, & & \\
H_1 = X W_0 + b_0, & H_1 \in \mathbb{R}^{n\times p_1}, & X \in \mathbb{R}^{n\times p_0}, & W_0 \in \mathbb{R}^{p_0\times p_1}, & b_0 \in \mathbb{R}^{1\times p_1},
\end{array}
$$

taking as input $X \in \mathbb{R}^{n\times p_0}$ and outputting $F \in \mathbb{R}^{n\times p_{\ell+1}}$. Here, n denotes the number of data while the weights $W_i \in \mathbb{R}^{p_i\times p_{i+1}}$ and biases $b_i \in \mathbb{R}^{1\times p_{i+1}}$ represent the parameters of the neural network. Moreover, let us focus on the sum of squared errors loss function

$$\mathcal{L} := \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p_{\ell+1}}(F_{i,j} - Y_{i,j})^2,$$

where $Y \in \mathbb{R}^{n\times p_{\ell+1}}$ corresponds to the output data.

3. Back Propagation [1]. The gradient of $\mathcal{L}$ with respect to F is then given by

$$G_\ell = F - Y, \quad G_\ell \in \mathbb{R}^{n\times p_{\ell+1}}.$$

Using chain rule, the gradient of $\mathcal{L}$ with respect to W_ℓ is given by

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = A_\ell^T G_\ell \in \mathbb{R}^{p_\ell\times p_{\ell+1}},$$

and the gradient of $\mathcal{L}$ with respect to b_ℓ is given by

$$\frac{\partial \mathcal{L}}{\partial b_\ell} = \mathbb{1}^T G_\ell \in \mathbb{R}^{1\times p_{\ell+1}},$$

where $\mathbb{1} \in \mathbb{R}^{n\times 1}$ is a matrix filled with ones. The gradient of $\mathcal{L}$ with respect to A_ℓ is given by

$$\frac{\partial \mathcal{L}}{\partial A_\ell} = G_\ell W_\ell^T \in \mathbb{R}^{n\times p_\ell}.$$

Consequently, the gradient of $\mathcal{L}$ with respect to H_ℓ, denoted by $G_{\ell-1}$, is given by

$$G_{\ell-1} = (1 - A_\ell \odot A_\ell) \odot (G_\ell W_\ell^T) \in \mathbb{R}^{n\times p_\ell}.$$

Here, we are using the fact that the derivative of $\tanh(x)$ with respect to x is given by $1-\tanh^2(x)$. Moreover, $\odot$ denoted the point-wise product between two matrices. The above procedure can be repeated to give us the backpropagation algorithm

$$
\begin{array}{rrr}
G_\ell = F - Y \in \mathbb{R}^{n\times p_{\ell+1}}, & \frac{\partial \mathcal{L}}{\partial W_\ell} = A_\ell^T G_\ell \in \mathbb{R}^{p_\ell\times p_{\ell+1}}, & \frac{\partial \mathcal{L}}{\partial b_\ell} = \mathbb{1}^T G_\ell \in \mathbb{R}^{1\times p_{\ell+1}},\\
G_{\ell-1} = (1 - A_\ell \odot A_\ell) \odot (G_\ell W_\ell^T) \in \mathbb{R}^{n\times p_\ell}, & \frac{\partial \mathcal{L}}{\partial W_{\ell-1}} = A_{\ell-1}^T G_{\ell-1} \in \mathbb{R}^{p_{\ell-1}\times p_\ell}, & \frac{\partial \mathcal{L}}{\partial b_{\ell-1}} = \mathbb{1}^T G_{\ell-1} \in \mathbb{R}^{1\times p_\ell},\\
G_{\ell-2} = (1 - A_{\ell-1} \odot A_{\ell-1}) \odot (G_{\ell-1} W_{\ell-1}^T) \in \mathbb{R}^{n\times p_{\ell-1}}, & \frac{\partial \mathcal{L}}{\partial W_{\ell-2}} = A_{\ell-2}^T G_{\ell-2} \in \mathbb{R}^{p_{\ell-2}\times p_{\ell-1}}, & \frac{\partial \mathcal{L}}{\partial b_{\ell-2}} = \mathbb{1}^T G_{\ell-2} \in \mathbb{R}^{1\times p_{\ell-1}},\\
\vdots & \vdots & \vdots\\
G_1 = (1 - A_2 \odot A_2) \odot (G_2 W_2^T) \in \mathbb{R}^{n\times p_2}, & \frac{\partial \mathcal{L}}{\partial W_1} = A_1^T G_1 \in \mathbb{R}^{p_1\times p_2}, & \frac{\partial \mathcal{L}}{\partial b_1} = \mathbb{1}^T G_1 \in \mathbb{R}^{1\times p_2},\\
G_0 = (1 - A_1 \odot A_1) \odot (G_1 W_1^T) \in \mathbb{R}^{n\times p_1}, & \frac{\partial \mathcal{L}}{\partial W_0} = X^T G_0 \in \mathbb{R}^{p_0\times p_1}, & \frac{\partial \mathcal{L}}{\partial b_0} = \mathbb{1}^T G_0 \in \mathbb{R}^{1\times p_1}.
\end{array}
$$

Moreover, the gradient of $\mathcal{L}$ with respect to X is given by

$$\frac{\partial \mathcal{L}}{\partial X} = G_0 W_0^T \in \mathbb{R}^{n \times p_0}.$$

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.