CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2022 Marvin Zhang

Discussion 3

This discussion will cover some background on autodiff and practice applying backpropagation.

1 Automatic Differentiation

In this section we will cover some background on different types of differentiation and motivate why we use backward autodiff (instead of forward).

In training neural networks, we are trying to find the model weights $\boldsymbol{\theta}$ that minimize a loss function $\mathcal{L}(\boldsymbol{\theta})$. To do this, recall that we typically use (stochastic) gradient descent as follows

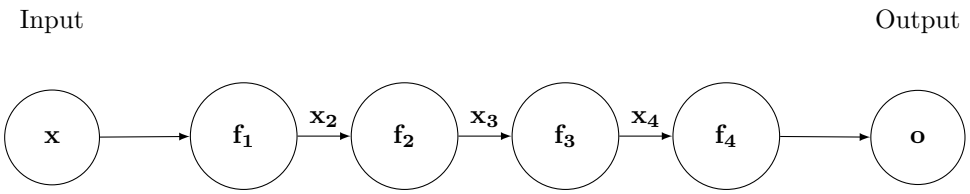
$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \alpha \boldsymbol{\nabla} L(\boldsymbol{\theta}^t).$$

which means it is important to be able to efficiently compute derivatives, especially for large and complex models. Automatic differentiation (autodiff) is a method for computing the derivative of a program-specified function.

Problem 1: Methods for Differentiation

There are three main methods for differentiation: symbolic differentiation, numerical differentiation, and automatic differentiation. How does each of them work, and what are their pros/cons?

Our neural networks are composed of a series of nested functions. Consider the feedforward network below



Here, we have an input $\mathbf{x} \in \mathbb{R}^n$, an output $\mathbf{o} \in \mathbb{R}^m$, and the Jacobian of $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ as an $n \times m$ dimensional matrix. Explicitly writing out the nested functions, we have

$$\begin{aligned}\mathbf{o} &= \mathbf{f}(\mathbf{x}) \\ &= \mathbf{f_4}(\mathbf{f_3}(\mathbf{f_2}(\mathbf{f_1}(\mathbf{x}))))\end{aligned}$$

Then by the chain rule, we can write

$$\begin{aligned}\frac{\partial \mathbf{o}}{\partial \mathbf{x}} &= \frac{\partial \mathbf{o}}{\partial \mathbf{x_4}} \frac{\partial \mathbf{x_4}}{\partial \mathbf{x_3}} \frac{\partial \mathbf{x_3}}{\partial \mathbf{x_2}} \frac{\partial \mathbf{x_2}}{\partial \mathbf{x}} \\ &= \frac{\partial \mathbf{f_4}(\mathbf{x_4})}{\partial \mathbf{x_4}} \frac{\partial \mathbf{f_3}(\mathbf{x_3})}{\partial \mathbf{x_3}} \frac{\partial \mathbf{f_2}(\mathbf{x_2})}{\partial \mathbf{x_2}} \frac{\partial \mathbf{f_1}(\mathbf{x})}{\partial \mathbf{x}} \\ &= J_{\mathbf{f_4}}(\mathbf{x_4}) J_{\mathbf{f_3}}(\mathbf{x_3}) J_{\mathbf{f_2}}(\mathbf{x_2}) J_{\mathbf{f_1}}(\mathbf{x}) \\ &= J_{\mathbf{f}}(\mathbf{x})\end{aligned}$$

Let the dimensionality of the intermediate variables be $\mathbf{x_2} \in \mathbb{R}^{m_1}, \mathbf{x_3} \in \mathbb{R}^{m_2}, \mathbf{x_4} \in \mathbb{R}^{m_3}$. The cost of computing $J_{\mathbf{f}}(\mathbf{x}) = \underbrace{J_{\mathbf{f_4}}(\mathbf{x_4})}_{m \times m_3} \underbrace{J_{\mathbf{f_3}}(\mathbf{x_3})}_{m_3 \times m_2} \underbrace{J_{\mathbf{f_2}}(\mathbf{x_2})}_{m_2 \times m_1} \underbrace{J_{\mathbf{f_1}}(\mathbf{x})}_{m_1 \times n}$ is then $O(mm_3 + m_3m_2 + m_2m_1 + m_1n)$.

Recall that $\frac{\partial \mathbf{f_i}}{\partial x_j}$ is the ith column and jth row of $J_{\mathbf{f}}(\mathbf{x})$. To build the Jacobian, we can use the **Jacobian-vector product** (JVP) or **vector-Jacobian product** (VJP) to build it up row-wise or column-wise, respectively.

Forward Differentiation. The JVP is the right-multiplication of the Jacobian with a vector $\mathbf{v} \in \mathbb{R}^n$. To find $\frac{\partial \mathbf{f}}{\partial x_j}$, we take the JVP with $\mathbf{e}_j \in \mathbb{R}^n$.

$$\frac{\partial \mathbf{f}}{\partial x_1} = J_{\mathbf{f}}(\mathbf{x})^\top \mathbf{e_1}, \frac{\partial \mathbf{f}}{\partial x_2} = J_{\mathbf{f}}(\mathbf{x})^\top \mathbf{e_2}, \cdots, \frac{\partial \mathbf{f}}{\partial x_n} = J_{\mathbf{f}}(\mathbf{x})^\top \mathbf{e_n}$$

Thus computing a gradient of $\mathbf{o}$ with respect to $\mathbf{x}$ requires n JVPs with $\mathbf{e_1}, \cdots, \mathbf{e_n}$. In total, this requires $O(n(mm_3 + m_3m_2 + m_2m_1 + m_1n))$.

Backward Differentiation. The VJP is the left-multiplication of a vector $\mathbf{u} \in \mathbb{R}^m$, and the Jacobian. To find $\nabla \mathbf{f}_i(\mathbf{x})$, we take the VJP with $\mathbf{e}_j \in \mathbb{R}^m$.

$$\nabla \mathbf{f_1}(\mathbf{x}) = \mathbf{e}_1 J_{\mathbf{f}}(\mathbf{x}), \nabla \mathbf{f_2}(\mathbf{x}) = \mathbf{e}_2 J_{\mathbf{f}}(\mathbf{x}), \cdots, \nabla \mathbf{f_m}(\mathbf{x}) = \mathbf{e}_m J_{\mathbf{f}}(\mathbf{x})$$

Thus computing a gradient of $\mathbf{o}$ with respect to $\mathbf{x}$ requires m VJPs with $\mathbf{e_1}, \cdots, \mathbf{e_m}$. In total, this requires $O(m(mm_3 + m_3m_2 + m_2m_1 + m_1n))$.

Problem 2: Motivation for Backwards Autodiff

Looking at the computational costs above, why do we use backward autodiff in machine learning?

2 Mechanical Backpropagation

In this section, we will work through some calculations used during backpropagation.

Recall the softmax function $\mathbf{p} : \mathbb{R}^k \to \mathbb{R}^k$, with entries given by

$$p_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}.$$

Each entry p_i corresponds to the probability assigned to the label i. We derived in Discussion 1 that the partial derivatives of $p_i(\mathbf{z})$ for each entry of $\mathbf{z}$ is given by,

$$\begin{aligned}\frac{\partial p_i(\mathbf{z})}{\partial z_j} &= \begin{cases} p_i(\mathbf{z})(1 - p_j(\mathbf{z})) & \text{if } i = j \\ -p_i(\mathbf{z})p_j(\mathbf{z}) & \text{if } i \neq j \end{cases} \\ &= p_i(\mathbf{z})(\delta_{ij} - p_j(\mathbf{z})).\end{aligned}$$

We can then concicsely write the full gradient with respect to $\mathbf{z}$ as

$$\nabla p_i(\mathbf{z}) = p_i(\mathbf{z})(\mathbf{e}_i - \mathbf{p}(\mathbf{z})),$$

where $\mathbf{e}_i$ is the unit vector with 1 at index i and 0 elsewhere.

In this example, we will maximize the log-likelihood of the given labels in our dataset, which motivates the following loss for a multiclass logistic regression model.

$$L(\mathbf{x}, y, W, \mathbf{b}) = -\log p_y(W\mathbf{x} + \mathbf{b}).$$

Problem 3: Gradient with respect to linear layer parameters

Utilize the chain rule to compute the gradient of $L(\mathbf{x}, y, W, \mathbf{b})$ with respect to W and $\mathbf{b}$.

Suppose now that we had a multilayer neural network and $W, \mathbf{b}$ were the the parameters of the last layer of the network. To compute gradients of the earlier parameters of the network with backpropagation, we also need to compute the gradient of the loss with respect to $\mathbf{x}$ and pass it backwards.

Problem 4: Gradient with respect to input

Utilize the chain rule to compute the gradient of $L(\mathbf{x}, y, W, \mathbf{b})$ with respect to $\mathbf{x}$.

Having computed these, one then simply needs to also compute the backwards pass through the chosen activation function to able backpropagate through fully-connected feedforward networks!

We'll now move on to a slightly more complicated example of backpropagation involving a *skip connection*, which you'll see again when we cover ResNets.

Problem 5: Gradient in a nonlinear computation graph

Suppose we have $\mathbf{y} = W_2\sigma(W_1\mathbf{x}) + \mathbf{x}$, where σ is the ReLU activation. Letting $\delta_{\mathbf{y}}$ denote the gradient of the loss with respect to $\mathbf{y}$, compute the gradient of the loss with respect to $\mathbf{x}$.