A trick to calculating partial derivatives in machine learning

MartinLwx included in category ML-DL
2023-07-26 2024-01-17 964 words 5 minutes

Intro

You may have difficulties when trying to calculate the partial derivatives in machine learning like me. Even though I found a good reference cookbook that could be used to derive the gradients, I still got confused. Today, I want to share a practical technique I recently learned from this video: **when calculating partial derivatives in machine learning, you can treat everything as if it were a scalar and then make the shapes match**

> **Disclaimer**: Calculating the partial derivatives using this trick **DO NOT** guarantee that the gradients are correct. The dimensions may match but the gradients could be wrong. Therefore, it is essential to **perform gradient checking to ensure correctness**

Application

> 💡 Uppercase bold letters represent matrices, while non-bold letters represent scalars

Backpropagation in matrix form

In my previous post, I try to derive the backpropagation equations in the scalar form because it's much easier to understand. However, if you ever try to **implement** the forward propagation or the backpropagation, you'll find that everything is done **in matrix form**. That is why it's essential to understand how the matrix form of backpropagation works. Now, I will use the technique mentioned earlier to derive it

> For simplicity, let's ignore the bias term and only consider the weight term

Consider a simple L layers MLP, where $\mathbf{Z}^l$ represents the output of layer l. We also use $\mathbf{Z}$ to represent the input. Thus, we have:

$$\mathbf{Z}^0 = \mathbf{X}$$

Here, $\mathbf{X} \in \mathrm{R}^{m \times d}$, where m is the number of samples, and d is the length of each features for each sample

The output of the model f_θ is

$$f_\theta(\mathbf{X}) = \mathbf{Z}^L$$

Here, θ represents the learnable parameters of this model

The relationship between any continuous layers is:

$$\mathbf{Z}^{l+1} = \sigma_{l+1}(\mathbf{Z}^l\mathbf{W}^{l+1}), l = 0, \dots, L-1$$

Here, σ_{l+1} is the activate function of layer $l+1$

The shapes:

$$\mathbf{Z}^l \in \mathrm{R}^{m \times n_l}$$
$$\mathbf{W}^{l+1} \in \mathrm{R}^{n_l \times n_{l+1}}$$

Here, n_l represents the number of neurons in layer l

We want to determine the gradient of the loss J with respect to any learnable parameter in the model. This gradient is essential for using gradient a descent algorithm to update the learnable parameters. Specifically, let's consider that we want to calculate the gradient of $\mathbf{W}^l$.

$$\frac{\partial J}{\partial \mathbf{W}^l} = \frac{\partial J}{\partial \mathbf{Z}^L} \cdot \frac{\partial \mathbf{Z}^L}{\partial \mathbf{Z}^{L-1}} \cdot \ldots \cdot \frac{\partial \mathbf{Z}^{l+1}}{\partial \mathbf{Z}^l} \cdot \frac{\partial \mathbf{Z}^l}{\partial \mathbf{W}^l}$$

🤔 What if we also want to calculate the gradient with respect to $\mathbf{W}^{l-1}$?

$$\frac{\partial J}{\partial \mathbf{W}^{l-1}} = \frac{\partial J}{\partial \mathbf{Z}^L} \cdot \frac{\partial \mathbf{Z}^L}{\partial \mathbf{Z}^{L-1}} \cdot \ldots \cdot \frac{\partial \mathbf{Z}^{l+1}}{\partial \mathbf{Z}^l} \cdot \frac{\partial \mathbf{Z}^l}{\partial \mathbf{Z}^{l-1}} \cdot \frac{\partial \mathbf{Z}^{l-1}}{\partial \mathbf{W}^{l-1}}$$

One thing to notice is that - **both equations share some common components**. So we can introduce an additional notation $\mathbf{G}^l$ which represents the gradient of $\mathbf{Z}^l$

$$\mathbf{G}^l = \frac{\partial J}{\partial \mathbf{Z}^l}$$

Now, let's try to figure out the relationship between $\mathbf{G}^l$ and $\mathbf{G}^{l+1}$

$$\begin{aligned}
\mathbf{G}^l &= \frac{\partial J}{\partial \mathbf{Z}^{l+1}} \cdot \frac{\partial \mathbf{Z}^{l+1}}{\partial \mathbf{Z}^l} \\
&= \mathbf{G}^{l+1} \cdot \frac{\partial \mathbf{Z}^{l+1}}{\partial \mathbf{Z}^l} \\
&= \mathbf{G}^{l+1} \cdot \frac{\partial \sigma_{l+1}(\mathbf{Z}^l \mathbf{W}^{l+1})}{\partial \mathbf{Z}^l \mathbf{W}^{l+1}} \cdot \frac{\partial \mathbf{Z}^l \mathbf{W}^{l+1}}{\partial \mathbf{Z}^l} \\
&= \mathbf{G}^{l+1} \cdot \sigma\text{'}(\mathbf{Z}^l \mathbf{W}^{l+1}) \cdot \mathbf{W}^{l+1} \ (cheat)
\end{aligned}$$

In the last line above, we are **calculating the derivatives as if they were scalars**. Now let's try to **make the shapes match**. Let's first examine the shapes of each component:

$$\mathbf{G}^{l+1} \in \mathrm{R}^{m \times n_{l+1}}$$
$$\sigma_{l+1}\text{'}(\mathbf{Z}^l \mathbf{W}^{l+1}) \in \mathrm{R}^{m \times n_{l+1}}$$
$$\mathbf{W}^{l+1} \in \mathrm{R}^{n_l \times n_{l+1}}$$

We want to get a matrix with the shape $m \times n_l$, because

$$\mathbf{G}^l \in \mathrm{R}^{m \times n_l}$$

So we can derive this

$$\mathbf{G}^l = (\mathbf{G}^{l+1} \odot \sigma_{l+1}\text{'}(\mathbf{Z}^l \mathbf{W}^{l+1}))(\mathbf{W}^{l+1})^T = (\mathbf{G}^{l+1} \odot \sigma_{l+1}\text{'}(\mathbf{Z}^{l+1}))(\mathbf{W}^{l+1})^T$$

Now, let's get back to what we originally intended to do - computing the gradient with respect to $\mathbf{W}^l$

$$\frac{\partial J}{\partial \mathbf{W}^l} = \mathbf{G}^l \cdot \frac{\partial \mathbf{Z}^l}{\mathbf{W}^l}$$

Let's expand the equation in the above

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^l} &= \mathbf{G}^l \cdot \frac{\partial \mathbf{Z}^l}{\mathbf{W}^l} \\ &= \mathbf{G}^l \cdot \frac{\partial \sigma_l(\mathbf{Z}^{l-1}\mathbf{W}^l)}{\partial \mathbf{Z}^{l-1}\mathbf{W}^l} \cdot \frac{\partial \mathbf{Z}^{l-1}\mathbf{W}^l}{\partial \mathbf{W}^l} \\ &= \mathbf{G}^l \cdot \sigma_l\text{'}(\mathbf{Z}^{l-1}\mathbf{W}^l) \cdot \mathbf{Z}^{l-1}(cheat)\end{aligned}$$

J is a scalar, so the shape of $\frac{\partial J}{\partial \mathbf{W}^l}$ should be equal to $\mathbf{W}^l$. That is, we want a matrix with the shape (n_{l-1}, n_l). Let's rearrange these components

$$\frac{\partial J}{\partial \mathbf{W}^l} = (\mathbf{Z}^{l-1})^T(\mathbf{G}^l \odot \sigma_{l+1}\text{'}(\mathbf{Z}^{l-1}\mathbf{W}^l)) \ = (\mathbf{Z}^{l-1})^T(\mathbf{G}^l \odot \sigma_{l+1}\text{'}(\mathbf{Z}^l))$$

By utilizing the relationship between $\mathbf{G}^l$ and $\mathbf{G}^{l+1}$, we can deduce $\mathbf{G}^l$ from $\mathbf{G}^{l+1}$. This allows us to compute the gradient of $\mathbf{W}^l$ by working backward, which is the essence of backpropagation

> 🤔 You may notice that calculating $\mathbf{G}^l$ and $\frac{\partial J}{\partial \mathbf{W}^l}$ involves $\mathbf{Z}^{l-1}, \mathbf{Z}^l, \mathbf{Z}^{l+1}$, which are the outputs of activation functions in different layers. This means that we need to **cache** the values from forward propagation. Caching requires memory consumption, and that's why larger models require more GPU memory for training.

The gradient of a linear regression model

Previously in this post I need to derive this equation

$$\frac{\partial}{\partial \theta} J(w,b) = \frac{\partial}{\partial \theta} \frac{1}{2m}(\mathbf{X}\theta - \vec{y})^T(\mathbf{X}\theta - \vec{y})$$

With this cool trick, we can derive like this

$$\begin{aligned}\frac{\partial}{\partial \theta} J(w,b) &= \frac{\partial}{\partial \theta} \frac{1}{2m}(\mathbf{X}\theta - \vec{y})^T(\mathbf{X}\theta - \vec{y}) \\ &= \frac{1}{2m} \frac{\partial(\ \mathbf{X}\theta - \vec{y})^T(\mathbf{X}\theta - \vec{y})}{\partial(\mathbf{X}\theta - \vec{y})} \cdot \frac{\partial(\mathbf{X}\theta - \vec{y})}{\partial \theta} \\ &= \frac{1}{2m} \cdot 2(\mathbf{X}\theta - \vec{y}) \cdot \mathbf{X}\ (cheat)\end{aligned}$$

Note the shapes here

$$(\mathbf{X}\theta - \vec{y}) \in \mathrm{R}^{m \times 1}$$
$$\mathbf{X} \in \mathrm{R}^{m \times (n+1)}$$

We want a vector whose shape is equal to $(n+1) \times 1$

$$\theta \in \mathrm{R}^{(n+1)\times 1}$$

Let's make the shapes match:

$$\frac{1}{m}\mathbf{X}^T(\mathbf{X}\theta - \vec{y})$$

Key takeaway

This trick is **practical but not rigorous**. After mastering this technique, the process of formula derivation in machine learning may become much easier. However, don't forget to use gradient checking technique to ensure correctness :)