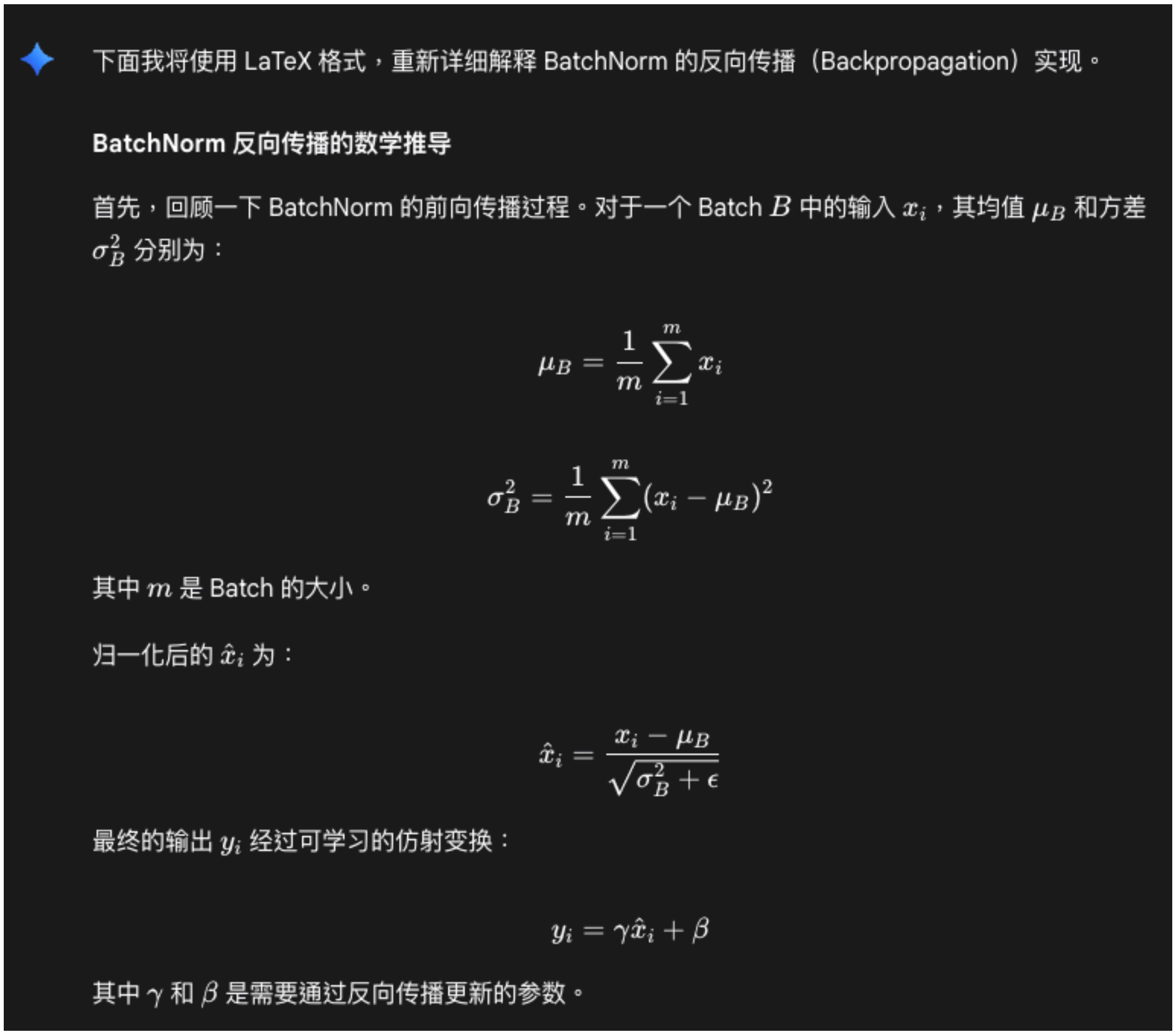


下面我将使用 LaTeX 格式，重新详细解释 BatchNorm 的反向传播（Backpropagation）实现。

BatchNorm 反向传播的数学推导

首先，回顾一下 BatchNorm 的前向传播过程。对于一个 Batch B 中的输入 x_i，其均值 μ_B 和方差 σ_B^2 分别为：

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

其中 m 是 Batch 的大小。

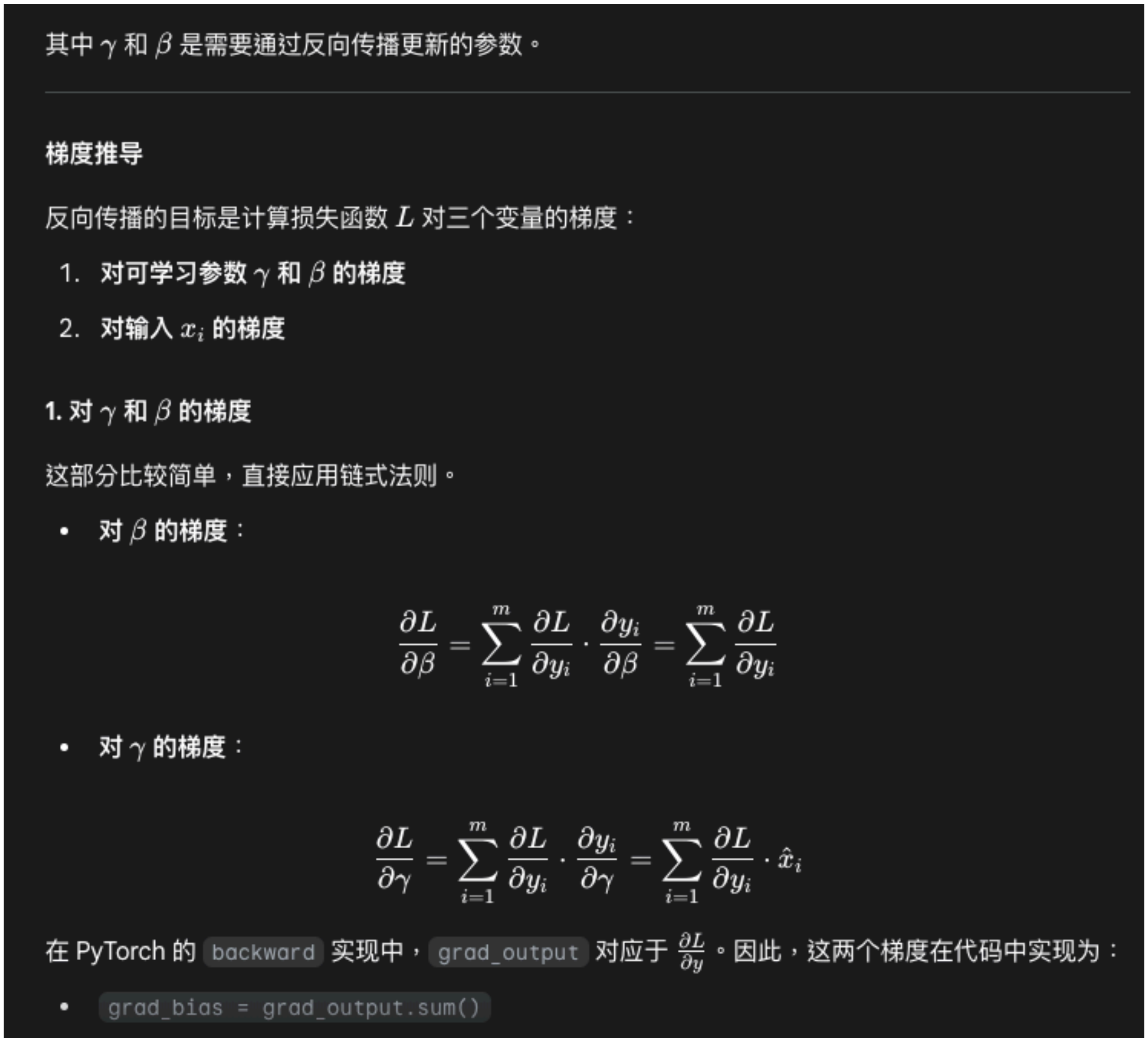
归一化后的 $\hat{x}_i$ 为：

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

最终的输出 y_i 经过可学习的仿射变换：

$$y_i = \gamma\hat{x}_i + \beta$$

其中 γ 和 β 是需要通过反向传播更新的参数。



其中 γ 和 β 是需要通过反向传播更新的参数。

梯度推导

反向传播的目标是计算损失函数 L 对三个变量的梯度：

1. **对可学习参数 γ 和 β 的梯度**
2. **对输入 x_i 的梯度**

1. 对 γ 和 β 的梯度

这部分比较简单，直接应用链式法则。

- **对 β 的梯度**：

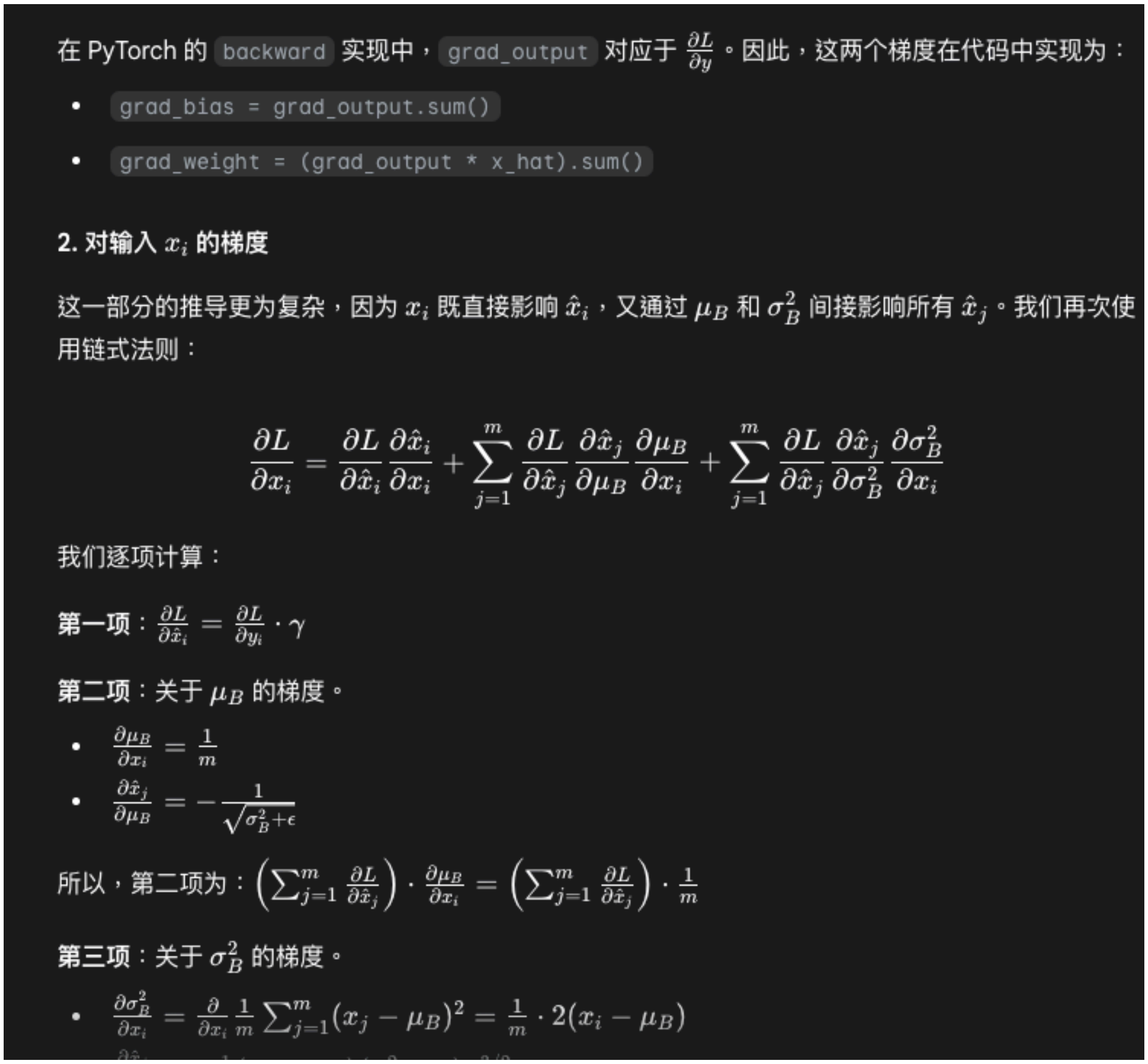
$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}$$

- **对 γ 的梯度**：

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} \cdot \hat{x}_i$$

在 PyTorch 的 `backward` 实现中，`grad_output` 对应于 $\frac{\partial L}{\partial y}$。因此，这两个梯度在代码中实现为：

- `grad_bias = grad_output.sum()`



在 PyTorch 的 `backward` 实现中，`grad_output` 对应于 $\frac{\partial L}{\partial y}$。因此，这两个梯度在代码中实现为：

- `grad_bias = grad_output.sum()`
- `grad_weight = (grad_output * x_hat).sum()`

2. 对输入 x_i 的梯度

这一部分的推导更为复杂，因为 x_i 既直接影响 $\hat{x}_i$，又通过 μ_B 和 σ_B^2 间接影响所有 $\hat{x}_j$。我们再次使用链式法则：

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial x_i} + \sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j}\frac{\partial \hat{x}_j}{\partial \mu_B}\frac{\partial \mu_B}{\partial x_i} + \sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j}\frac{\partial \hat{x}_j}{\partial \sigma_B^2}\frac{\partial \sigma_B^2}{\partial x_i}$$

我们逐项计算：

第一项：$\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \gamma$

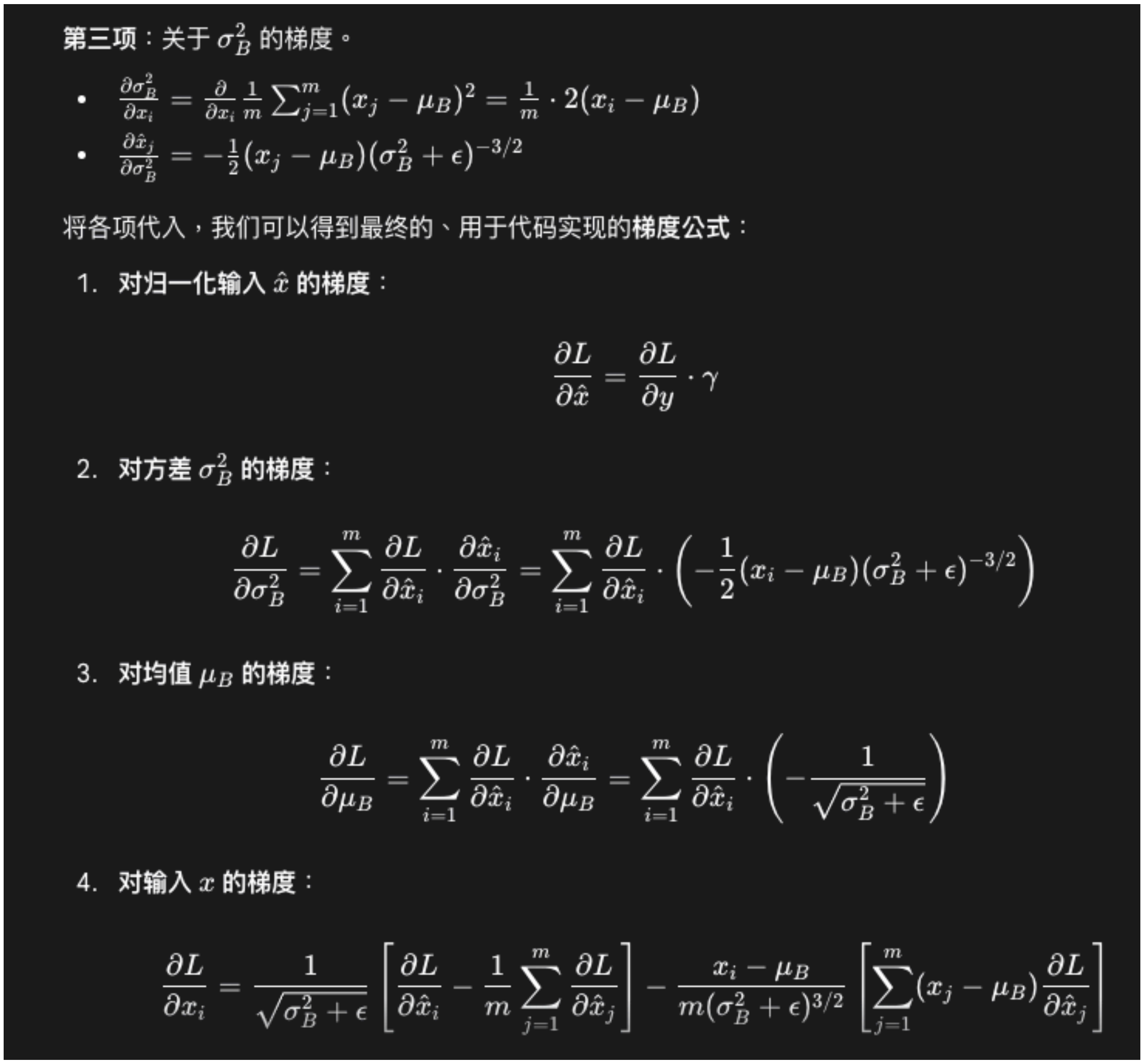
第二项：关于 μ_B 的梯度。

- $\frac{\partial \mu_B}{\partial x_i} = \frac{1}{m}$
- $\frac{\partial \hat{x}_j}{\partial \mu_B} = -\frac{1}{\sqrt{\sigma_B^2+\epsilon}}$

所以，第二项为：$\left(\sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j}\right) \cdot \frac{\partial \mu_B}{\partial x_i} = \left(\sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j}\right) \cdot \frac{1}{m}$

第三项：关于 σ_B^2 的梯度。

- $\frac{\partial \sigma_B^2}{\partial x_i} = \frac{\partial}{\partial x_i}\frac{1}{m}\sum_{j=1}^{m}(x_j-\mu_B)^2 = \frac{1}{m}\cdot 2(x_i-\mu_B)$



第三项：关于 σ_B^2 的梯度。

- $\frac{\partial \sigma_B^2}{\partial x_i} = \frac{\partial}{\partial x_i} \frac{1}{m} \sum_{j=1}^{m} (x_j - \mu_B)^2 = \frac{1}{m} \cdot 2(x_i - \mu_B)$
- $\frac{\partial \hat{x}_j}{\partial \sigma_B^2} = -\frac{1}{2}(x_j - \mu_B)(\sigma_B^2 + \epsilon)^{-3/2}$

将各项代入，我们可以得到最终的、用于代码实现的**梯度公式**：

1. **对归一化输入 $\hat{x}$ 的梯度**：

$$\frac{\partial L}{\partial \hat{x}} = \frac{\partial L}{\partial y} \cdot \gamma$$

2. **对方差 σ_B^2 的梯度**：

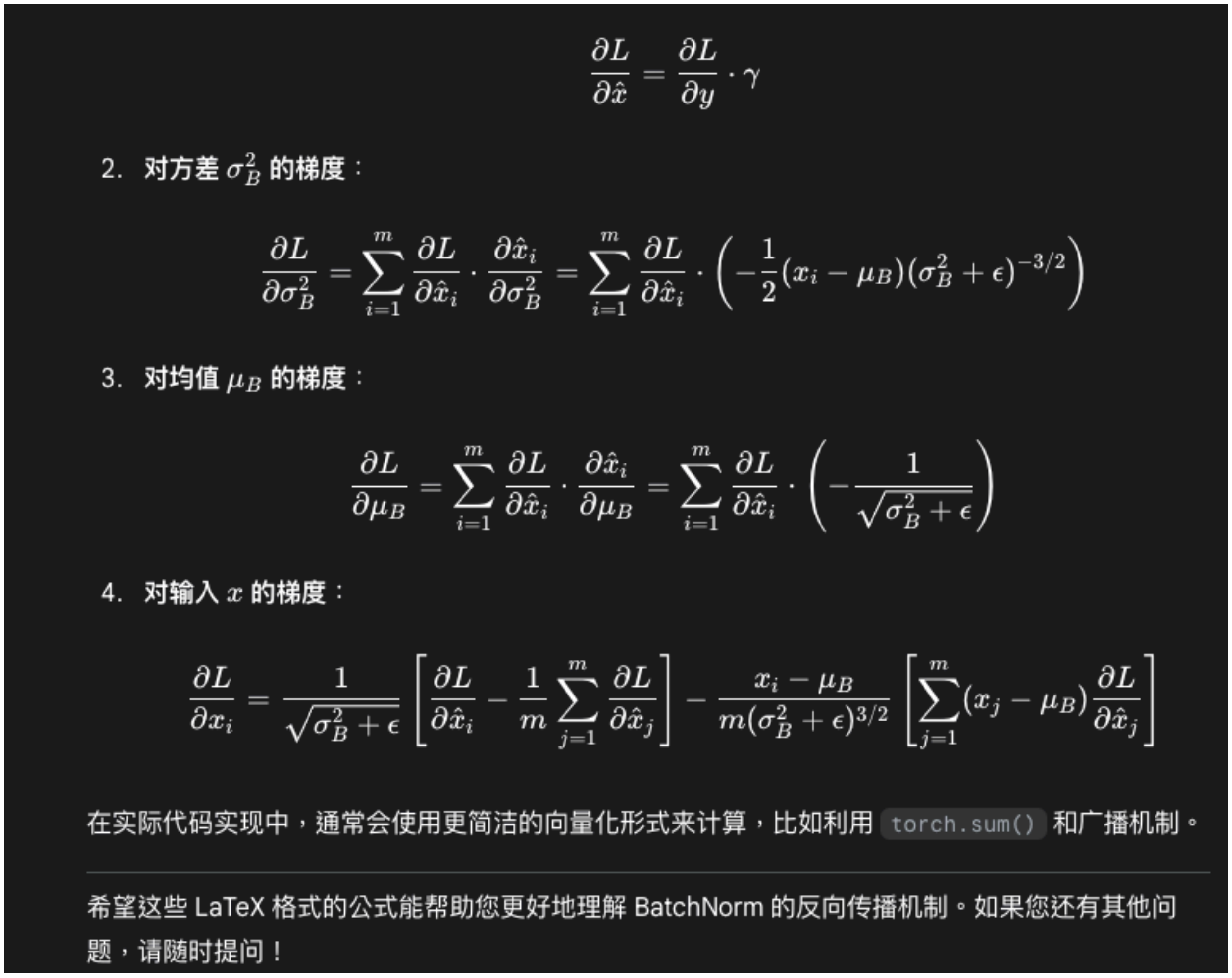
$$\frac{\partial L}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \left(-\frac{1}{2}(x_i - \mu_B)(\sigma_B^2 + \epsilon)^{-3/2} \right)$$

3. **对均值 μ_B 的梯度**：

$$\frac{\partial L}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \left(-\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \right)$$

4. **对输入 x 的梯度**：

$$\frac{\partial L}{\partial x_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \left[\frac{\partial L}{\partial \hat{x}_i} - \frac{1}{m} \sum_{j=1}^{m} \frac{\partial L}{\partial \hat{x}_j} \right] - \frac{x_i - \mu_B}{m(\sigma_B^2 + \epsilon)^{3/2}} \left[\sum_{j=1}^{m} (x_j - \mu_B) \frac{\partial L}{\partial \hat{x}_j} \right]$$



$$\frac{\partial L}{\partial \hat{x}} = \frac{\partial L}{\partial y} \cdot \gamma$$

2. **对方差 σ_B^2 的梯度：**

$$\frac{\partial L}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \left(-\frac{1}{2}(x_i - \mu_B)(\sigma_B^2 + \epsilon)^{-3/2} \right)$$

3. **对均值 μ_B 的梯度：**

$$\frac{\partial L}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \mu_B} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{x}_i} \cdot \left(-\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \right)$$

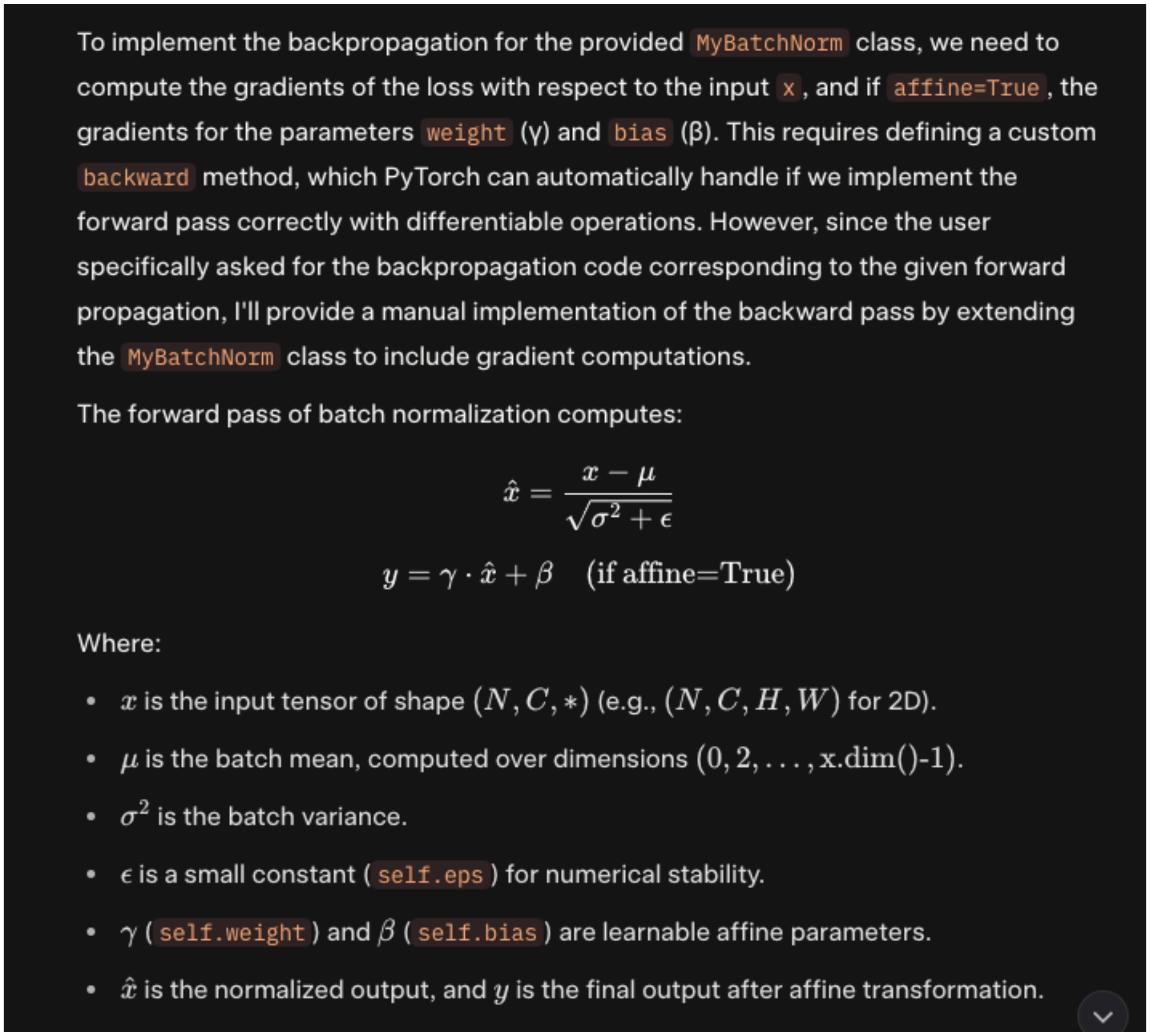
4. **对输入 x 的梯度：**

$$\frac{\partial L}{\partial x_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \left[\frac{\partial L}{\partial \hat{x}_i} - \frac{1}{m} \sum_{j=1}^{m} \frac{\partial L}{\partial \hat{x}_j} \right] - \frac{x_i - \mu_B}{m(\sigma_B^2 + \epsilon)^{3/2}} \left[\sum_{j=1}^{m} (x_j - \mu_B) \frac{\partial L}{\partial \hat{x}_j} \right]$$

在实际代码实现中，通常会使用更简洁的向量化形式来计算，比如利用 `torch.sum()` 和广播机制。

希望这些 LaTeX 格式的公式能帮助您更好地理解 BatchNorm 的反向传播机制。如果您还有其他问题，请随时提问！

//————————————————————————————————————-



To implement the backpropagation for the provided `MyBatchNorm` class, we need to compute the gradients of the loss with respect to the input `x`, and if `affine=True`, the gradients for the parameters `weight` (γ) and `bias` (β). This requires defining a custom `backward` method, which PyTorch can automatically handle if we implement the forward pass correctly with differentiable operations. However, since the user specifically asked for the backpropagation code corresponding to the given forward propagation, I'll provide a manual implementation of the backward pass by extending the `MyBatchNorm` class to include gradient computations.

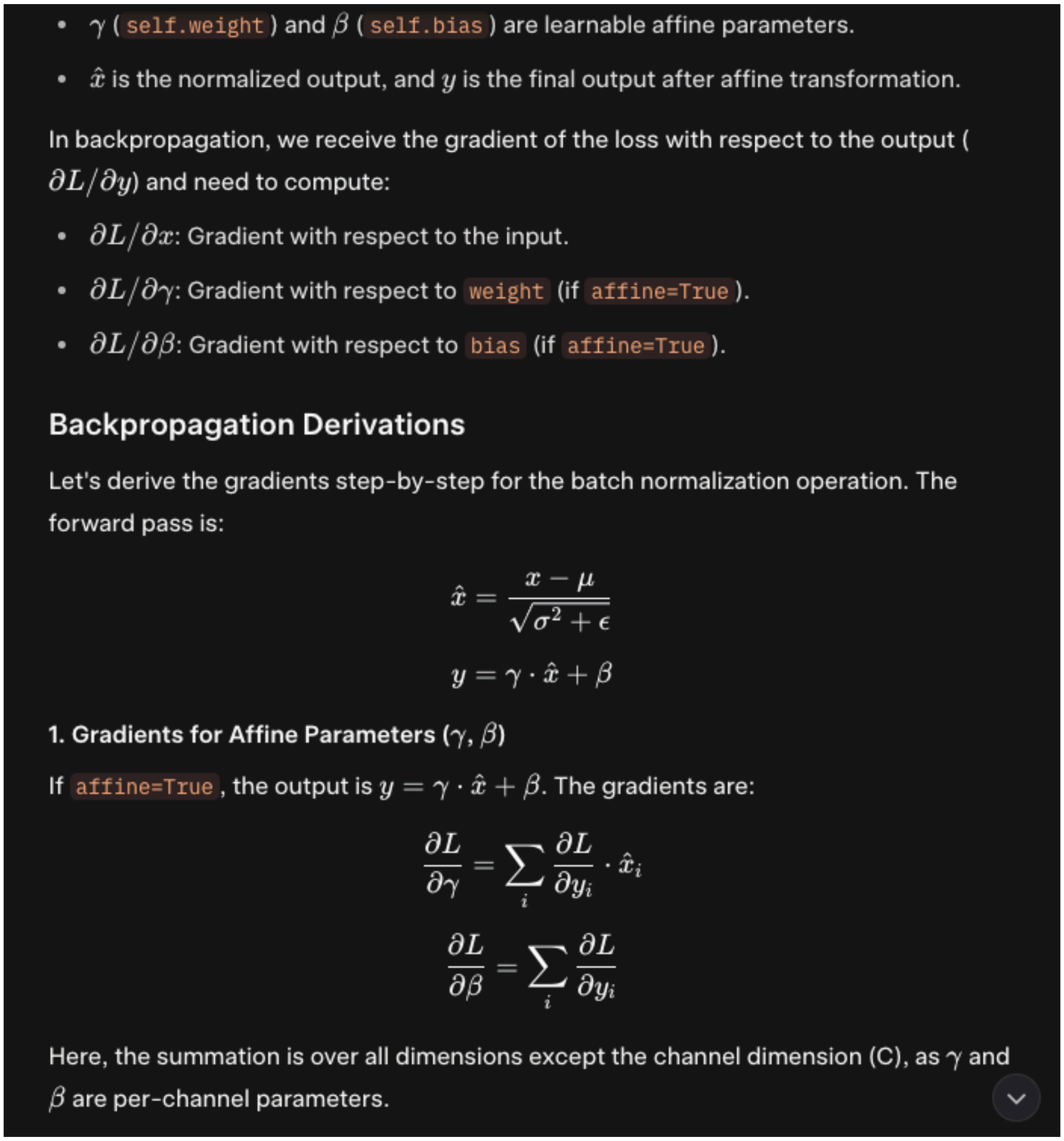
The forward pass of batch normalization computes:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma \cdot \hat{x} + \beta \quad \text{(if affine=True)}$$

Where:

- x is the input tensor of shape $(N, C, *)$ (e.g., (N, C, H, W) for 2D).
- μ is the batch mean, computed over dimensions $(0, 2, \ldots, \text{x.dim()-1})$.
- σ^2 is the batch variance.
- ϵ is a small constant (`self.eps`) for numerical stability.
- γ (`self.weight`) and β (`self.bias`) are learnable affine parameters.
- $\hat{x}$ is the normalized output, and y is the final output after affine transformation.



- γ (`self.weight`) and β (`self.bias`) are learnable affine parameters.
- $\hat{x}$ is the normalized output, and y is the final output after affine transformation.

In backpropagation, we receive the gradient of the loss with respect to the output ($\partial L/\partial y$) and need to compute:

- $\partial L/\partial x$: Gradient with respect to the input.
- $\partial L/\partial \gamma$: Gradient with respect to `weight` (if `affine=True`).
- $\partial L/\partial \beta$: Gradient with respect to `bias` (if `affine=True`).

Backpropagation Derivations

Let's derive the gradients step-by-step for the batch normalization operation. The forward pass is:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma \cdot \hat{x} + \beta$$

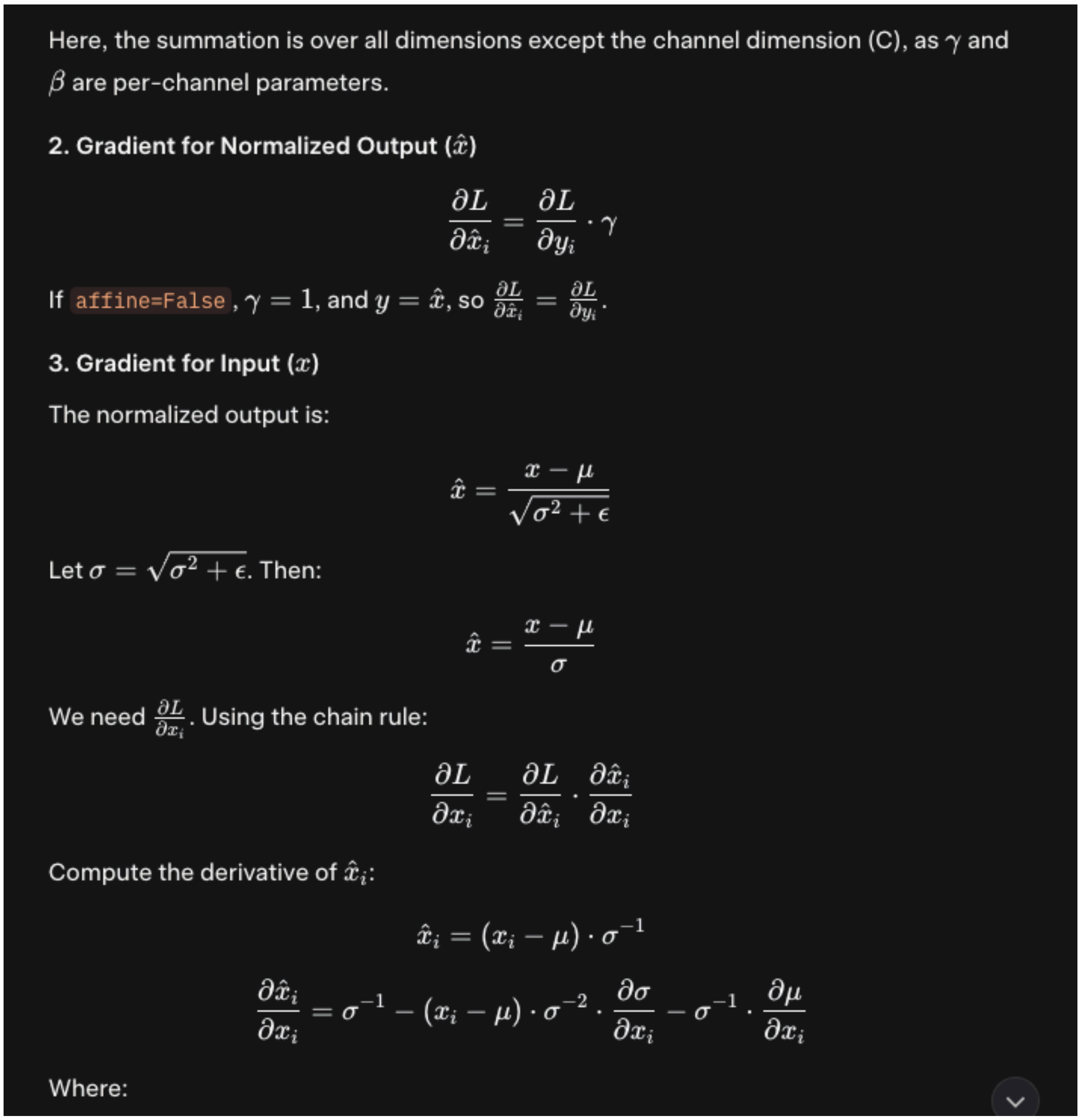
1. Gradients for Affine Parameters (γ, β)

If `affine=True`, the output is $y = \gamma \cdot \hat{x} + \beta$. The gradients are:

$$\frac{\partial L}{\partial \gamma} = \sum_i \frac{\partial L}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial L}{\partial \beta} = \sum_i \frac{\partial L}{\partial y_i}$$

Here, the summation is over all dimensions except the channel dimension (C), as γ and β are per-channel parameters.



Here, the summation is over all dimensions except the channel dimension (C), as γ and β are per-channel parameters.

2. Gradient for Normalized Output ($\hat{x}$)

$$\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i} \cdot \gamma$$

If `affine=False`, $\gamma = 1$, and $y = \hat{x}$, so $\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial y_i}$.

3. Gradient for Input (x)

The normalized output is:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Let $\sigma = \sqrt{\sigma^2 + \epsilon}$. Then:

$$\hat{x} = \frac{x - \mu}{\sigma}$$

We need $\frac{\partial L}{\partial x_i}$. Using the chain rule:

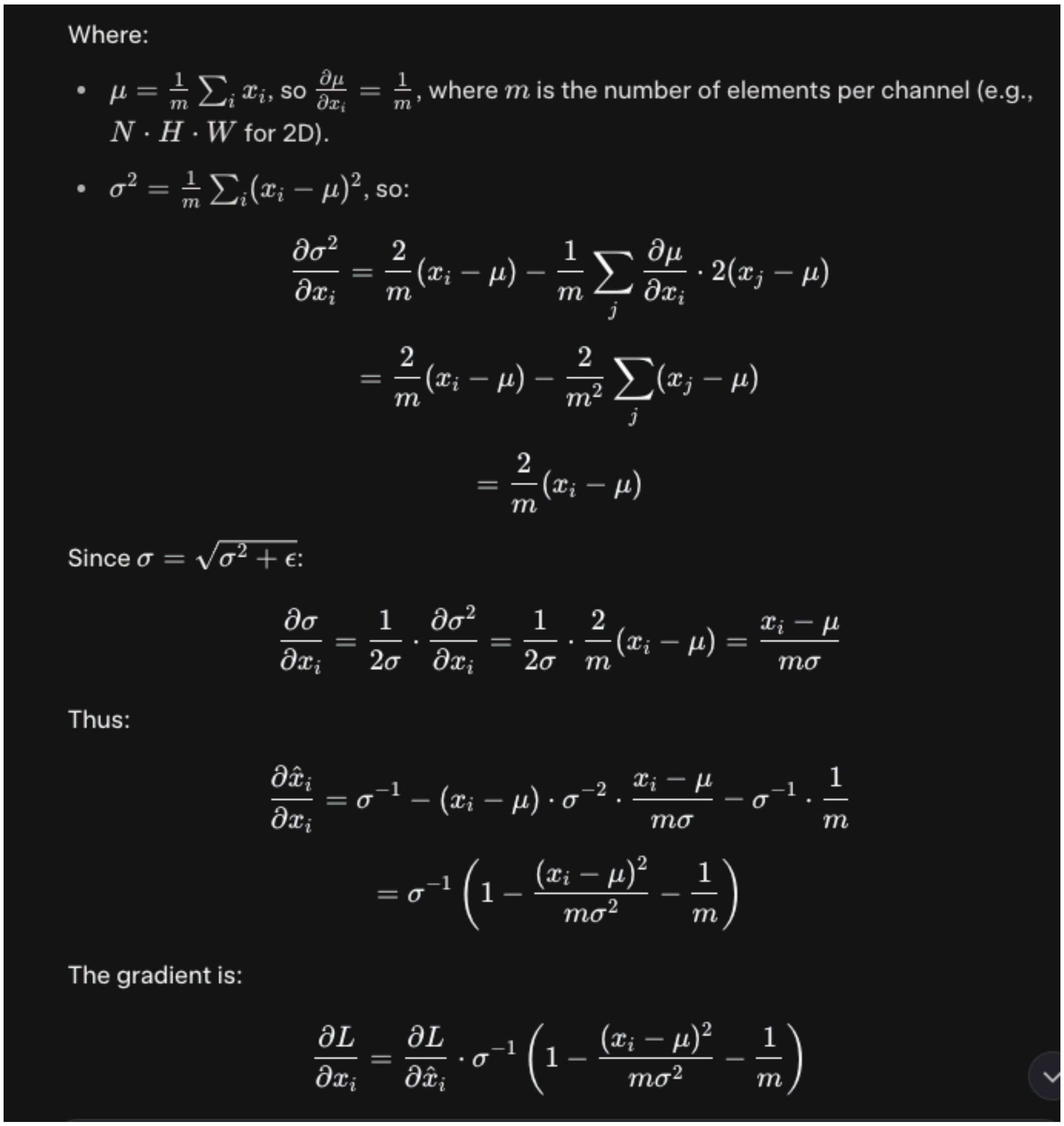
$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial x_i}$$

Compute the derivative of $\hat{x}_i$:

$$\hat{x}_i = (x_i - \mu) \cdot \sigma^{-1}$$

$$\frac{\partial \hat{x}_i}{\partial x_i} = \sigma^{-1} - (x_i - \mu) \cdot \sigma^{-2} \cdot \frac{\partial \sigma}{\partial x_i} - \sigma^{-1} \cdot \frac{\partial \mu}{\partial x_i}$$

Where:



Where:

- $\mu = \frac{1}{m}\sum_i x_i$, so $\frac{\partial \mu}{\partial x_i} = \frac{1}{m}$, where m is the number of elements per channel (e.g., $N \cdot H \cdot W$ for 2D).
- $\sigma^2 = \frac{1}{m}\sum_i (x_i - \mu)^2$, so:

$$
\begin{aligned}
\frac{\partial \sigma^2}{\partial x_i} &= \frac{2}{m}(x_i - \mu) - \frac{1}{m}\sum_j \frac{\partial \mu}{\partial x_i} \cdot 2(x_j - \mu) \\
&= \frac{2}{m}(x_i - \mu) - \frac{2}{m^2}\sum_j (x_j - \mu) \\
&= \frac{2}{m}(x_i - \mu)
\end{aligned}
$$

Since $\sigma = \sqrt{\sigma^2 + \epsilon}$:

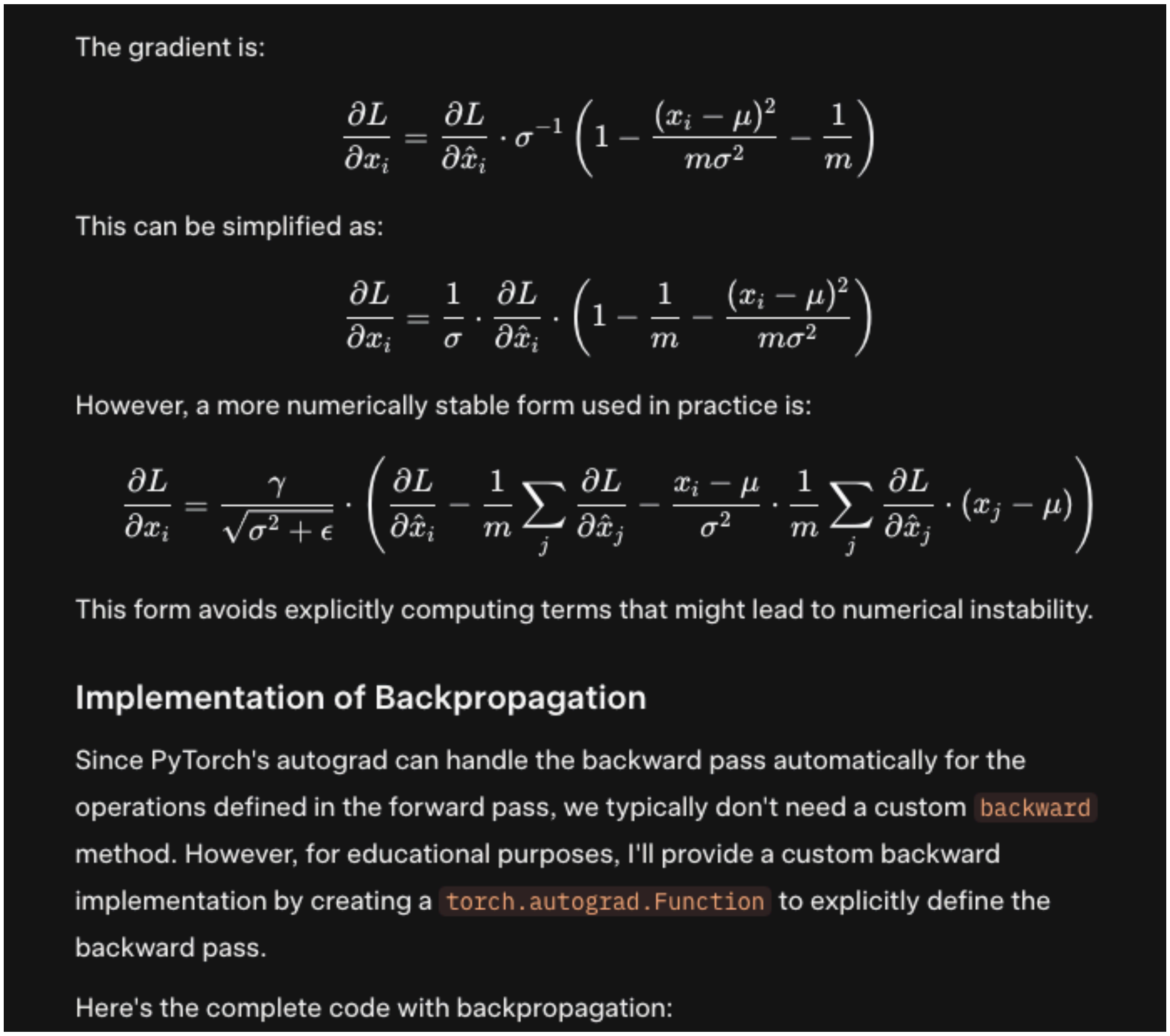
$$
\frac{\partial \sigma}{\partial x_i} = \frac{1}{2\sigma} \cdot \frac{\partial \sigma^2}{\partial x_i} = \frac{1}{2\sigma} \cdot \frac{2}{m}(x_i - \mu) = \frac{x_i - \mu}{m\sigma}
$$

Thus:

$$
\begin{aligned}
\frac{\partial \hat{x}_i}{\partial x_i} &= \sigma^{-1} - (x_i - \mu) \cdot \sigma^{-2} \cdot \frac{x_i - \mu}{m\sigma} - \sigma^{-1} \cdot \frac{1}{m} \\
&= \sigma^{-1}\left(1 - \frac{(x_i - \mu)^2}{m\sigma^2} - \frac{1}{m}\right)
\end{aligned}
$$

The gradient is:

$$
\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \sigma^{-1}\left(1 - \frac{(x_i - \mu)^2}{m\sigma^2} - \frac{1}{m}\right)
$$



The gradient is:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \sigma^{-1} \left(1 - \frac{(x_i - \mu)^2}{m\sigma^2} - \frac{1}{m}\right)$$

This can be simplified as:

$$\frac{\partial L}{\partial x_i} = \frac{1}{\sigma} \cdot \frac{\partial L}{\partial \hat{x}_i} \cdot \left(1 - \frac{1}{m} - \frac{(x_i - \mu)^2}{m\sigma^2}\right)$$

However, a more numerically stable form used in practice is:

$$\frac{\partial L}{\partial x_i} = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \cdot \left(\frac{\partial L}{\partial \hat{x}_i} - \frac{1}{m}\sum_j \frac{\partial L}{\partial \hat{x}_j} - \frac{x_i - \mu}{\sigma^2} \cdot \frac{1}{m}\sum_j \frac{\partial L}{\partial \hat{x}_j} \cdot (x_j - \mu)\right)$$

This form avoids explicitly computing terms that might lead to numerical instability.

Implementation of Backpropagation

Since PyTorch's autograd can handle the backward pass automatically for the operations defined in the forward pass, we typically don't need a custom `backward` method. However, for educational purposes, I'll provide a custom backward implementation by creating a `torch.autograd.Function` to explicitly define the backward pass.

Here's the complete code with backpropagation: