


真实环境中不是W1按列切，W2按行切吗？上面这个code是按照W1按列切，W2按行切的方式来的吗？

对，你的理解是对的 。

在真实的 **Tensor Parallel (TP)** 实现里（比如 Megatron-LM），常见的切分方式是：

- **W1（输入层 → 隐层）：**
 - W1 的 列切分（**column parallelism**）
 - 每个 rank 负责一部分 输出维度。
 - 优点：输入 **X** 可以广播到所有 rank，各自算 $X @ W1_shard$ ，得到的是对应 shard 的部分 hidden。最后再拼起来。
- **W2（隐层 → 输出层）：**
 - W2 的 行切分（**row parallelism**）
 - 每个 rank 负责一部分 输入维度。
 - 优点：每个 rank 只存储部分 hidden 的参数，forward 时各自处理 shard，对输出做 AllReduce（因为最终输出要整合所有 shard 的贡献）。

你提到的我的示例代码

我之前给你的教学版代码，确实是这样切的：

- **W1** 是 `torch.randn(D_in, D_hidden)`，在构造时我用了 列切 → `torch.chunk(W1, tp_size, dim=1)`。
✓ 这就是“W1 按列切”。
- **W2** 是 `torch.randn(D_hidden, D_out)`，在构造时我用了 行切 → `torch.chunk(W2, tp_size, dim=0)`。
✓ 这就是“W2 按行切”。

所以和你说的 真实环境常见做法是一致的。

要不要我帮你在代码里加一段 图示（矩阵切分可视化），这样可以清楚看到：

- W1 如何按列切分
- W2 如何按行切分
以及 forward/backward 中各个 shard 的数据流？

好 🙌 我来帮你把 **Tensor Parallel** 的矩阵切分方式 用图示直观展示一下。

◆ 1. W1 按列切分 (Column Parallel)

设输入层维度是 D_{in} ，隐层维度是 D_{hidden} 。

权重矩阵 W1 的形状是 (D_{in}, D_{hidden}) 。

正常情况：

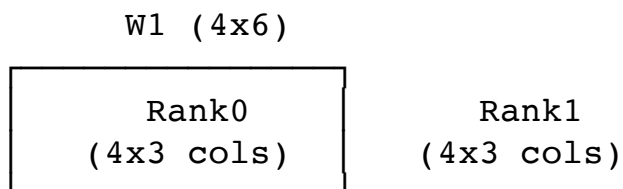
$X (N, D_{in}) @ W1 (D_{in}, D_{hidden}) \rightarrow H (N, D_{hidden})$

列切分：

W1 被按列切分 \rightarrow 每个 rank 拿一部分 hidden

例子：

假设 $D_{in} = 4, D_{hidden} = 6$, 切成 2 份：



每个 rank 算：

$H0 = X @ W1_shard0 \rightarrow (N, 3)$

$$H1 = X @ W1_shard1 \rightarrow (N, 3)$$

最后拼接：

$$H = [H0 \mid H1] \rightarrow (N, 6)$$

◆ 2. W2 按行切分 (Row Parallel)

设隐层维度是 D_hidden ，输出层维度是 D_out 。

权重矩阵 $W2$ 的形状是 (D_hidden, D_out) 。

正常情况：

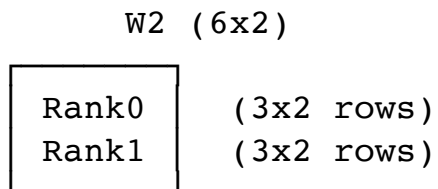
$$H (N, D_hidden) @ W2 (D_hidden, D_out) \rightarrow Y (N, D_out)$$

行切分：

$W2$ 被按行切分 \rightarrow 每个 rank 拿一部分输入 hidden

例子：

假设 $D_hidden = 6, D_out = 2$, 切成 2 份：



每个 rank 算：

$$Y0 = H[:, :3] @ W2_shard0 \rightarrow (N, 2)$$

$$Y1 = H[:, 3:] @ W2_shard1 \rightarrow (N, 2)$$

最后 加和 (AllReduce)：

$$Y = Y0 + Y1 \rightarrow (N, 2)$$

◆ 3. 总结数据流

$X \rightarrow W1 \text{ (col split)} \rightarrow \text{拼接 } H \rightarrow W2 \text{ (row split)} \rightarrow \text{AllReduce} \rightarrow Y$

- **W1**: 列切 \rightarrow 输出拼接 (Concat)
- **W2**: 行切 \rightarrow 输出求和 (Reduce)

要不要我直接把这个 可视化图解 (用 `matplotlib` 或 `ascii block`) 集成到我之前写的教学代码里, 这样你跑的时候就能看到矩阵怎么切分、数据怎么流动?

请加反向传播的 `ascii block`, 显示 `grad_W1` 和 `grad_W2` 是怎么合并的

好 

在前向我们已经画了 **W1** 列切 / **W2** 行切, 现在我来帮你补上 反向传播的 **ascii block**, 并且把它集成到代码里。这样一跑, 不光能看到 forward 的数据流, 还能看到 backward 的梯度流。

◆ 反向传播直观逻辑

1. Loss 对输出的梯度

dL/dY (N, D_out)

2. W2 (行切分) 的反传

- forward 时 W2 是行切 \rightarrow 每个 rank 负责一部分 hidden \rightarrow 反传时每个 rank 只收到对应 shard 的 `grad_H_local`。

```
grad_W2_rank = H_localT @ dL/dY
grad_H_local = dL/dY @ W2_shardT
最后:
```

```
grad_W2 = sum(grad_W2_rank)    (行拼起来)
grad_H   = concat(grad_H_local) (列拼起来)
```

3. W1 (列切分) 的反传

- forward 时 W1 是列切 → 每个 rank 只计算部分 H → backward 时要 sum 各个 shard 对输入的梯度贡献:

```
grad_W1_rank = XT @ grad_H_local
grad_X        = sum_over_ranks(grad_H_local @ W1_shardT)
最后:
```

```
grad_W1 = sum(grad_W1_rank)
```

//-----

Backpropagation is the process of calculating the gradients of a neural network's weights and biases with respect to a loss function. It uses the chain rule to efficiently compute these gradients layer by layer, starting from the output and moving backward.

Let's derive the backpropagation formulas for a 2-layer Multilayer Perceptron (MLP) with one hidden layer.

The Forward Pass

First, we need to define the network's forward pass. Let's assume the following:

- X is the input data, with dimensions $(N \times D_{in})$, where N is the batch size and D_{in} is the input dimension.
- $W^{(1)}$ and $b^{(1)}$ are the weights and biases for the hidden layer.
- $W^{(2)}$ and $b^{(2)}$ are the weights and biases for the output layer.
- Y is the ground-truth target data.
- σ is the activation function for the hidden layer (e.g., ReLU).

- Y is the ground-truth target data.
- σ is the activation function for the hidden layer (e.g., ReLU).
- g is the activation function for the output layer (e.g., linear or sigmoid).

The forward pass equations are:

Hidden Layer:

$$Z^{(1)} = XW^{(1)} + b^{(1)}$$

$$A^{(1)} = \sigma(Z^{(1)})$$

Output Layer:

$$Z^{(2)} = A^{(1)}W^{(2)} + b^{(2)}$$

$$A^{(2)} = g(Z^{(2)})$$

The **loss function** we'll use for this derivation is the Mean Squared Error (MSE), defined as:

$$L = \frac{1}{2} \|A^{(2)} - Y\|^2$$

The Backward Pass

The Backward Pass

The goal is to compute the gradients $\frac{\partial L}{\partial W^{(1)}}$, $\frac{\partial L}{\partial b^{(1)}}$, $\frac{\partial L}{\partial W^{(2)}}$, and $\frac{\partial L}{\partial b^{(2)}}$. We'll use the chain rule to work backward from the loss.

Step 1: Gradients for the Output Layer ($W^{(2)}$ and $b^{(2)}$)

We start by computing the gradient of the loss with respect to the pre-activation output $Z^{(2)}$. This is also known as the output layer's **error term**, denoted as $\delta^{(2)}$.

$$\delta^{(2)} = \frac{\partial L}{\partial Z^{(2)}} = \frac{\partial L}{\partial A^{(2)}} \odot g'(Z^{(2)})$$

$$\delta^{(2)} = (A^{(2)} - Y) \odot g'(Z^{(2)})$$

Using this error term, we can find the gradients for the weights and biases of the output layer:

$$\frac{\partial L}{\partial W^{(2)}} = (A^{(1)})^T \delta^{(2)}$$

$$\frac{\partial L}{\partial b^{(2)}} = \sum_{\text{batch}} \delta^{(2)}$$

$$\frac{\partial L}{\partial b^{(2)}} = \sum_{\text{batch}} \delta^{(2)}$$

Step 2: Gradients for the Hidden Layer ($W^{(1)}$ and $b^{(1)}$)

Now we propagate the error backward from the output layer to the hidden layer. First, we compute the hidden layer's **error term**, $\delta^{(1)}$, by using the error from the layer ahead of it ($\delta^{(2)}$).

$$\delta^{(1)} = \frac{\partial L}{\partial Z^{(1)}} = (\delta^{(2)} (W^{(2)})^T) \odot \sigma'(Z^{(1)})$$

Using this new error term, we can compute the final gradients for the hidden layer's weights and biases:

$$\frac{\partial L}{\partial W^{(1)}} = X^T \delta^{(1)}$$

$$\frac{\partial L}{\partial b^{(1)}} = \sum_{\text{batch}} \delta^{(1)}$$

$$\frac{\partial L}{\partial b^{(1)}} = \sum_{\text{batch}} \delta^{(1)}$$

Summary of Formulas

To train the 2-layer MLP, you would compute the forward pass to get $A^{(1)}$ and $A^{(2)}$, then use these values to compute the gradients in the backward pass.

1. Output Layer Gradients:

- **Output Layer Error:** $\delta^{(2)} = (A^{(2)} - Y) \odot g'(Z^{(2)})$
- **Weight Gradient:** $\frac{\partial L}{\partial W^{(2)}} = (A^{(1)})^T \delta^{(2)}$
- **Bias Gradient:** $\frac{\partial L}{\partial b^{(2)}} = \sum_{\text{batch}} \delta^{(2)}$

2. Hidden Layer Gradients:

- **Hidden Layer Error:** $\delta^{(1)} = (\delta^{(2)} (W^{(2)})^T) \odot \sigma'(Z^{(1)})$
- **Weight Gradient:** $\frac{\partial L}{\partial W^{(1)}} = X^T \delta^{(1)}$
- **Bias Gradient:** $\frac{\partial L}{\partial b^{(1)}} = \sum_{\text{batch}} \delta^{(1)}$

//-----
 //-----
 //-----

A handwriting practice sheet featuring 20 rows of three horizontal lines (top, middle dashed, bottom) and a double slash '/' at the start of each row. The lines are evenly spaced and extend across the width of the page.