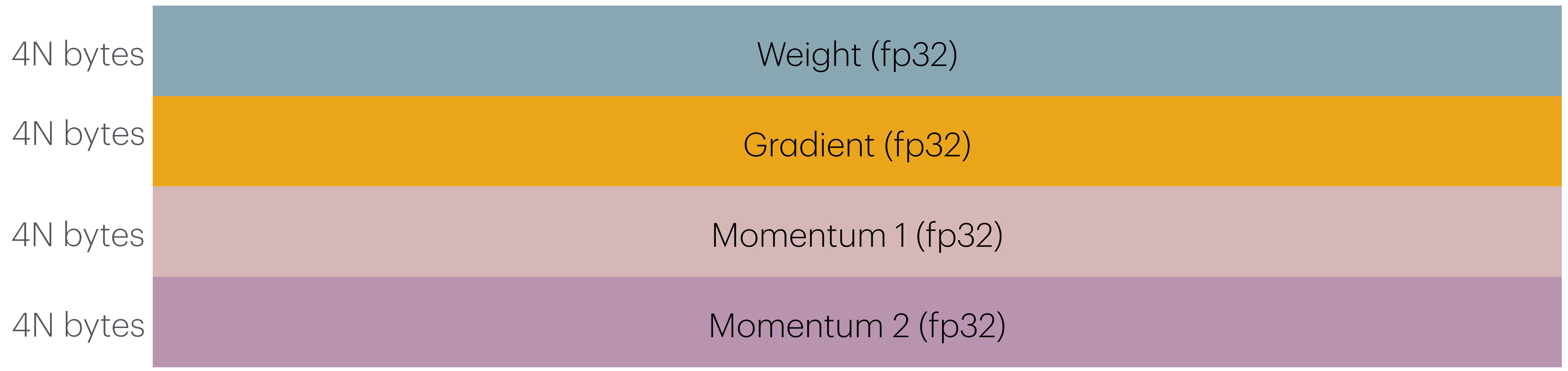
Low-rank adapters

Philipp Krähenbühl, UT Austin

Training large models

Memory requirements

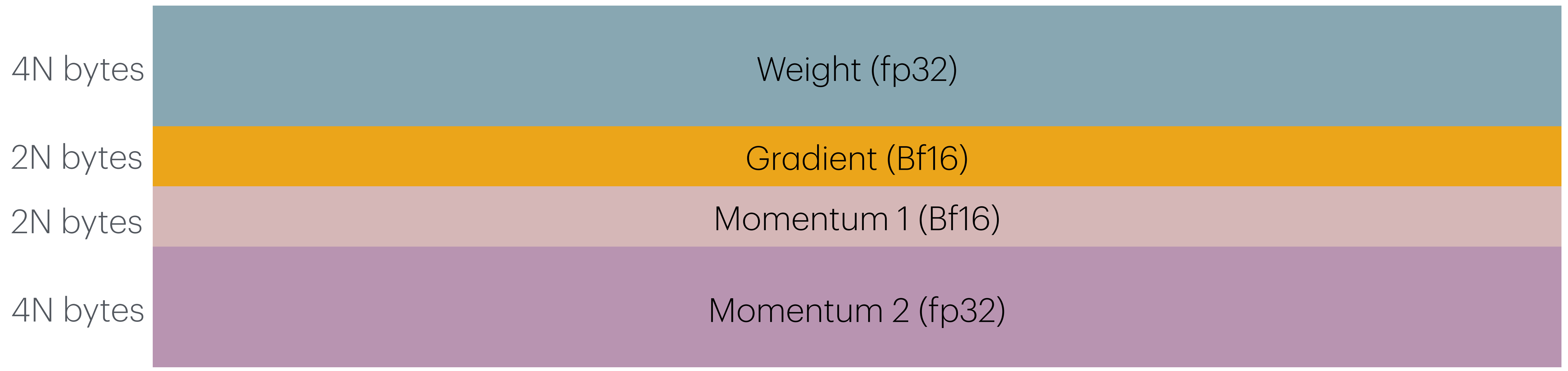
- Without optimization:
  - Model parameters: N
  - Weights: N floats
  - Gradients: N floats
  - Momentum: N floats
  - 2nd momentum (ADAM): N floats
- 16N bytes without counting activations



Training large models

Memory requirements

- Mixed precision
  - Model parameters: N
  - Weights: N floats
  - Gradients: N bfloat16
  - Momentum: N bfloat16
  - 2nd momentum (ADAM): N floats
- 12N bytes without counting activations

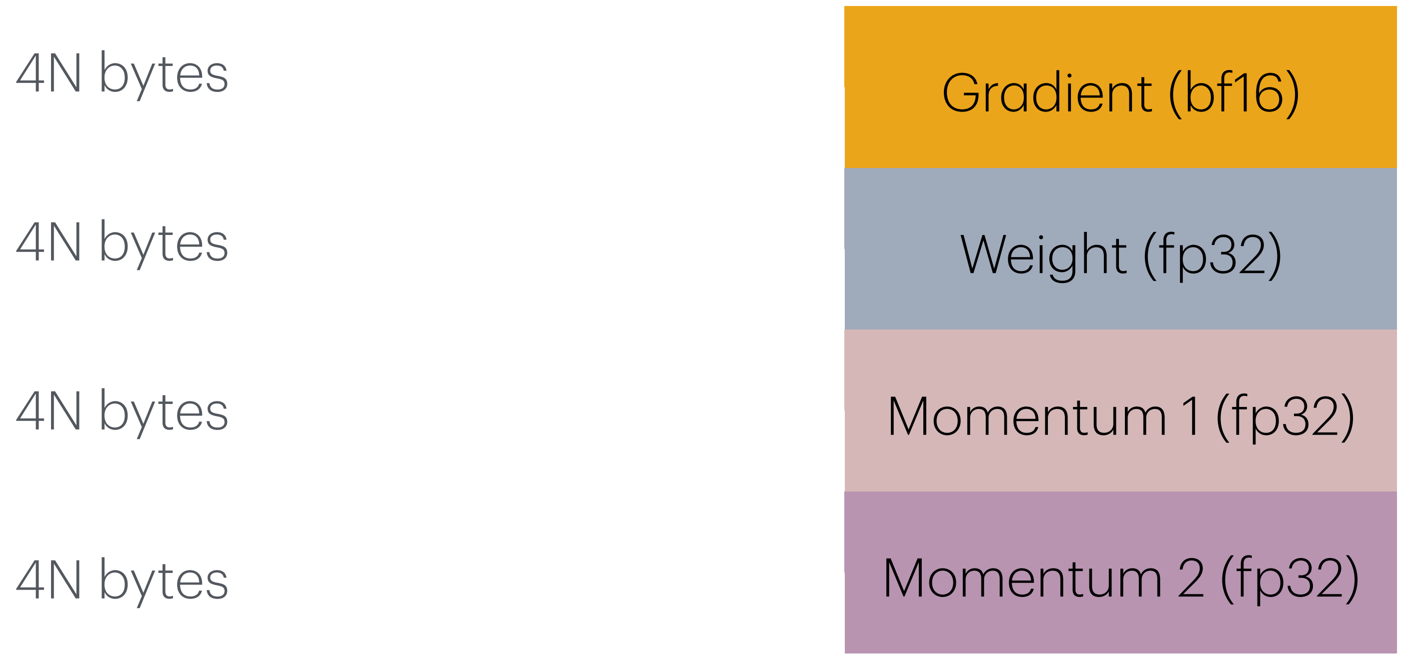


Training large models

Memory requirements

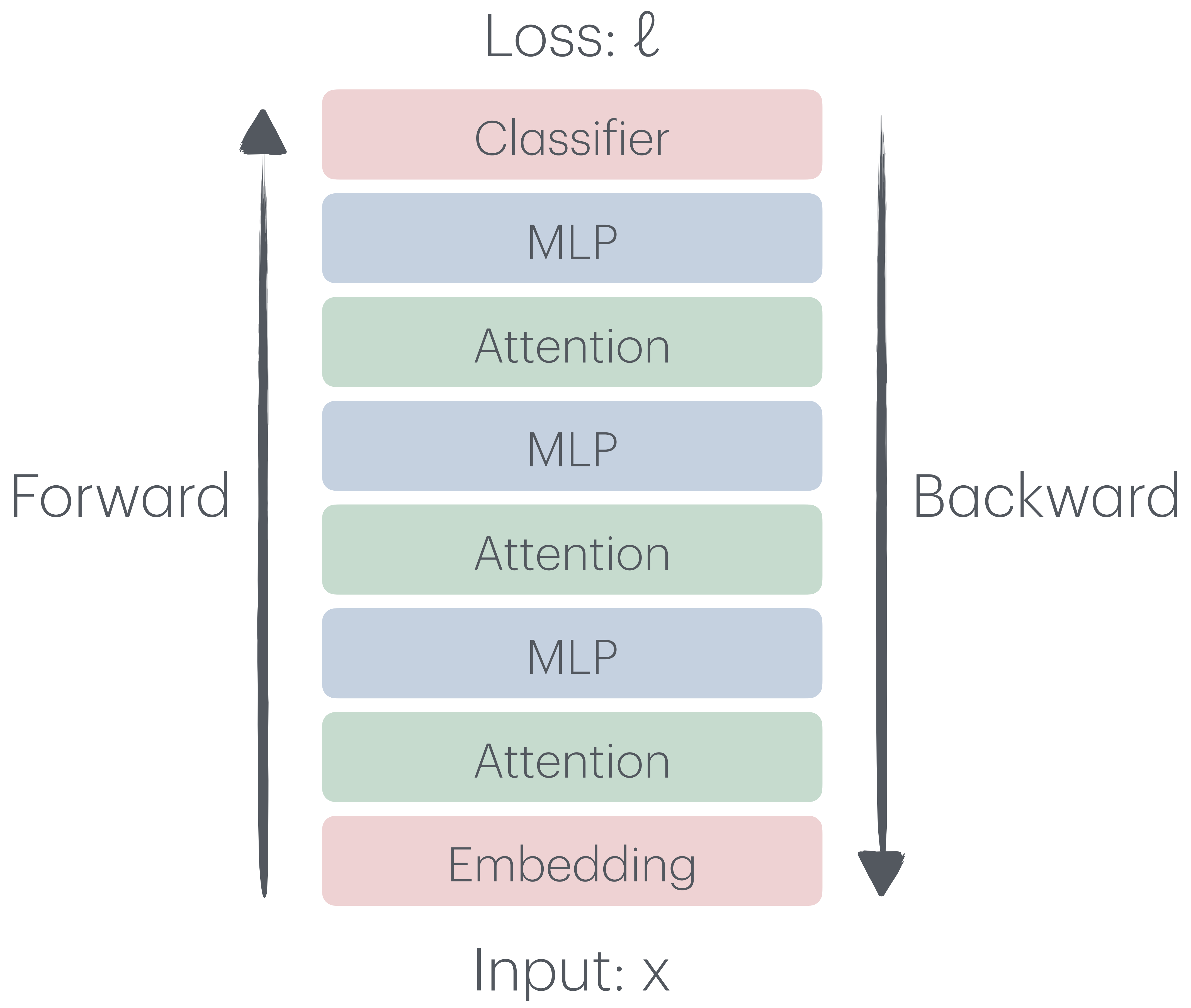
- Zero / FSDP
- 16N / M bytes without counting activations
- For M GPUs
  - Good solution for GPU-rich people

FSDP



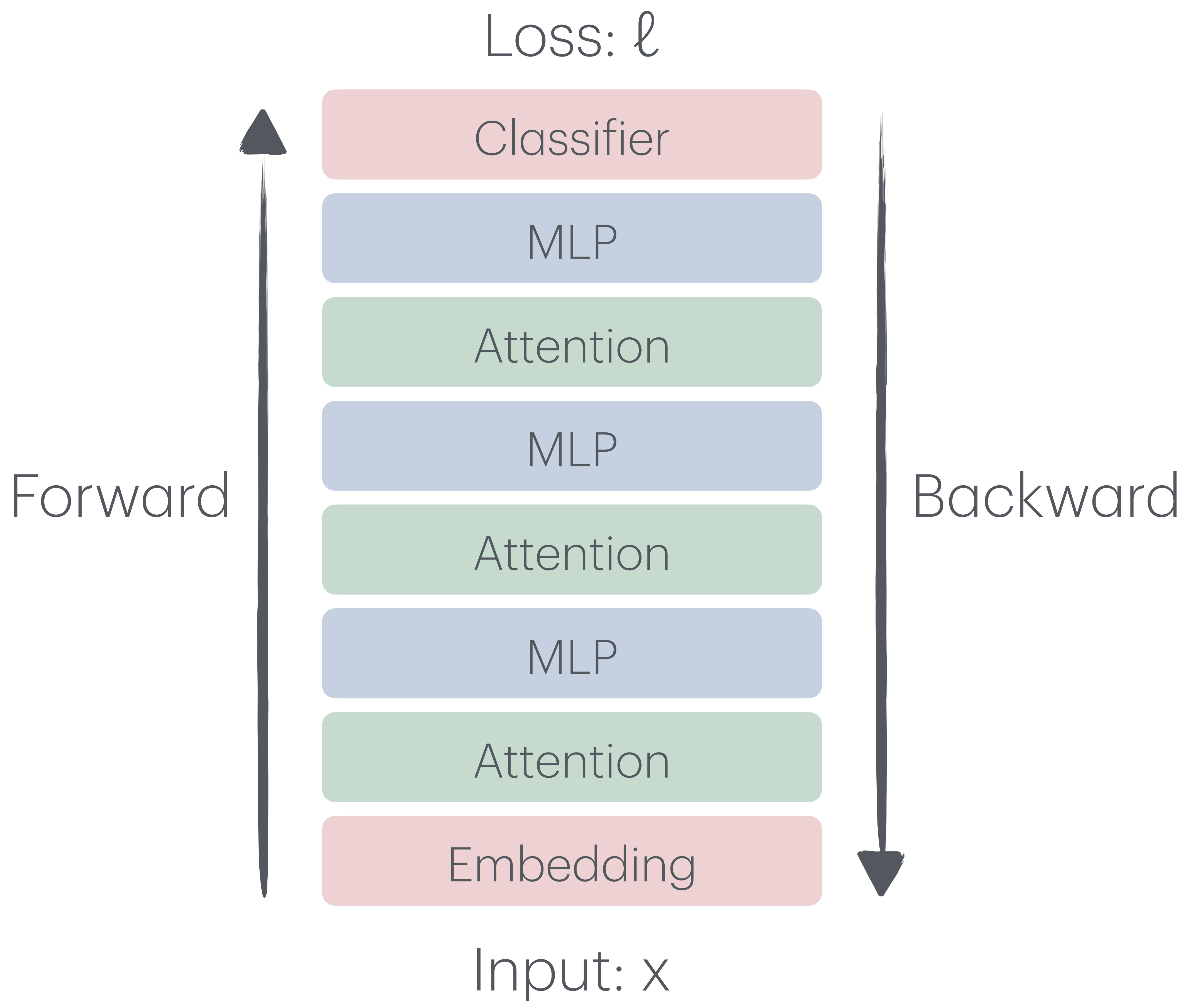
Memory Use

- What takes up GPU memory during training?
  - Model weights
  - Gradients
  - Momentum
  - Activations (more later in class)



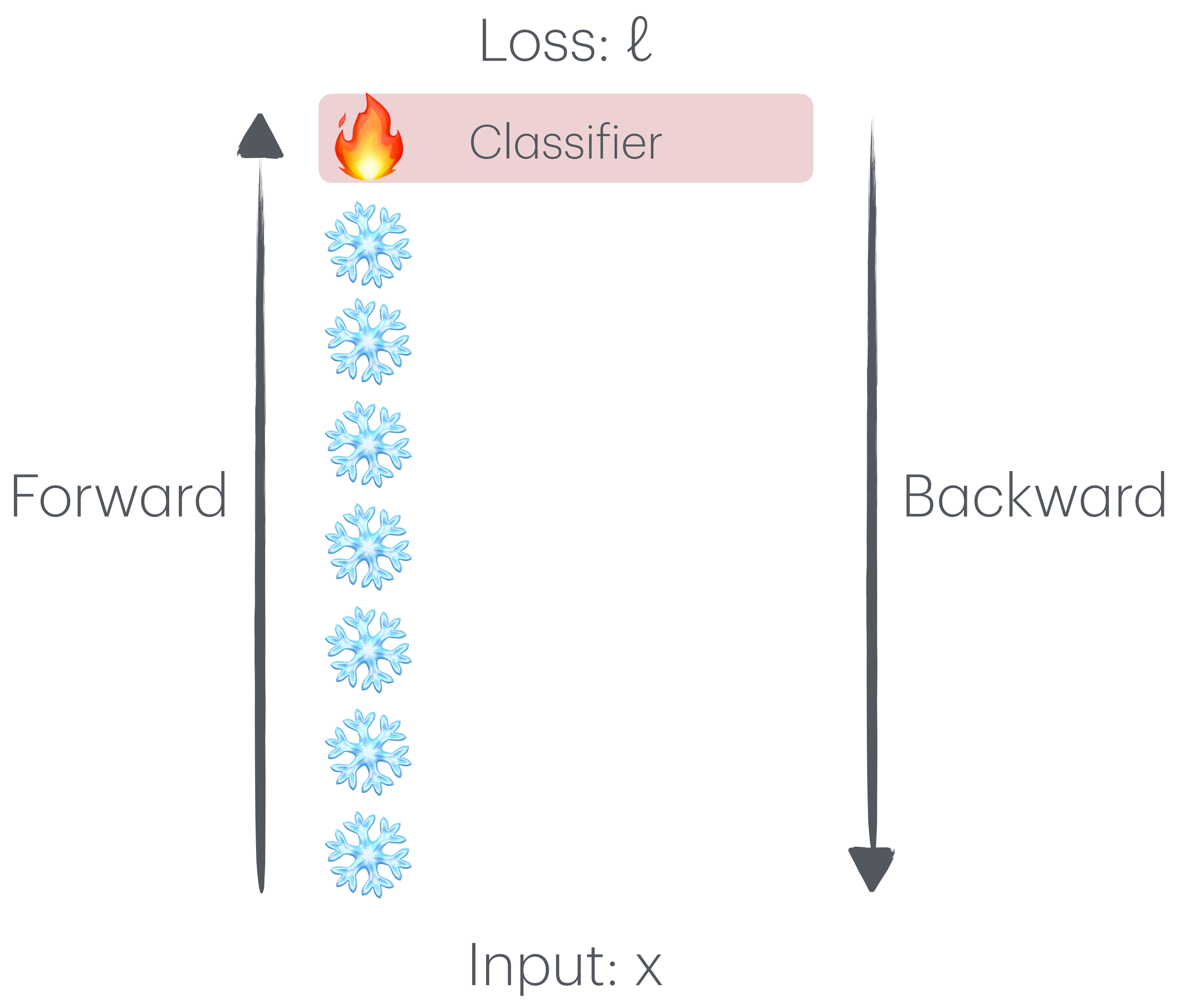
Reducing Memory Use

- Idea: Train fewer parameters
  - Keep most parameters frozen
    - No gradient, no momentum
  - Train a small subset
    - With gradient and momentum



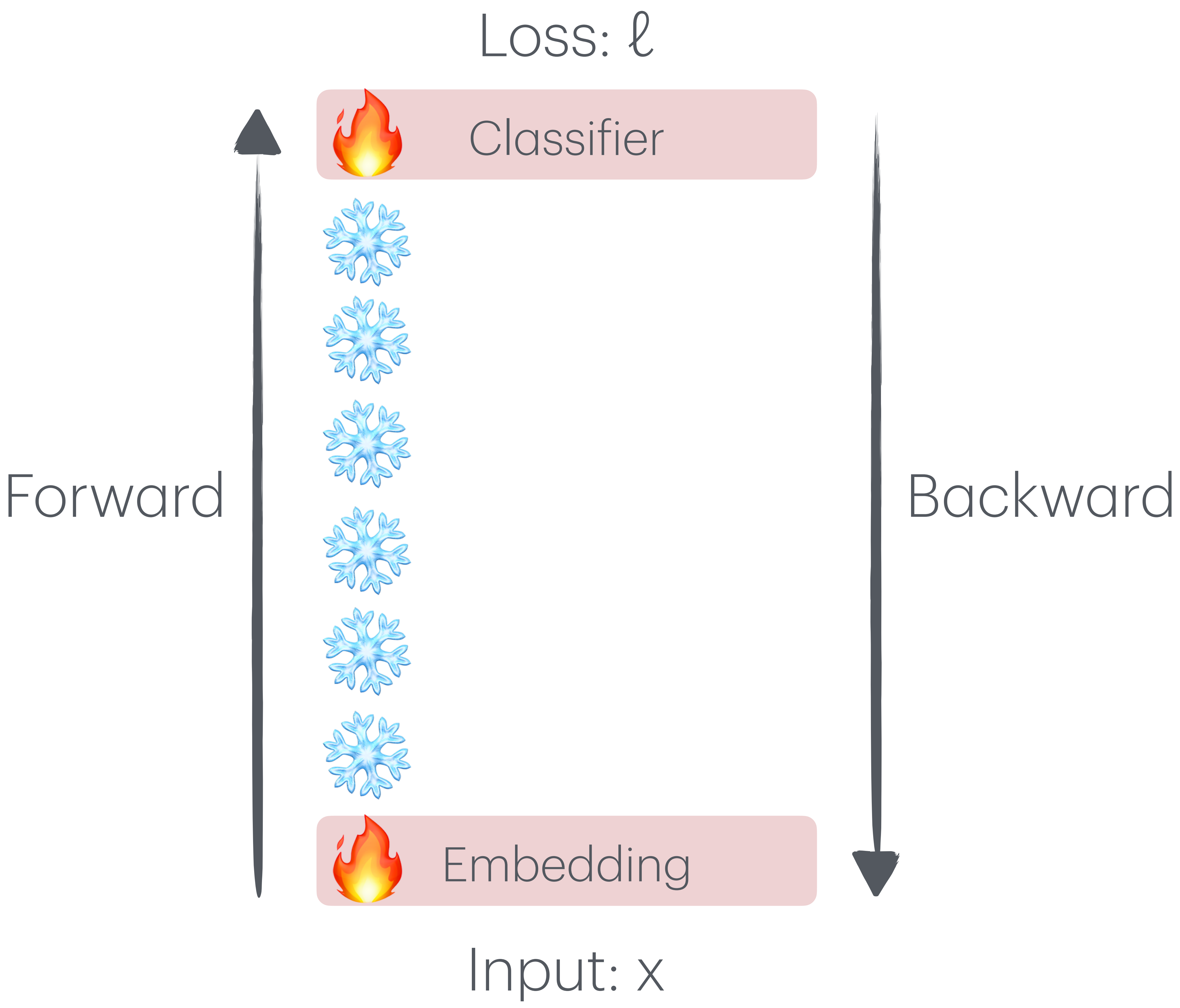
Fine-tuning classifier

- Freeze backbone
- Train classifier
- Most memory efficient
  - No backprop
  - Very few learnable parameters
- Not very expressive



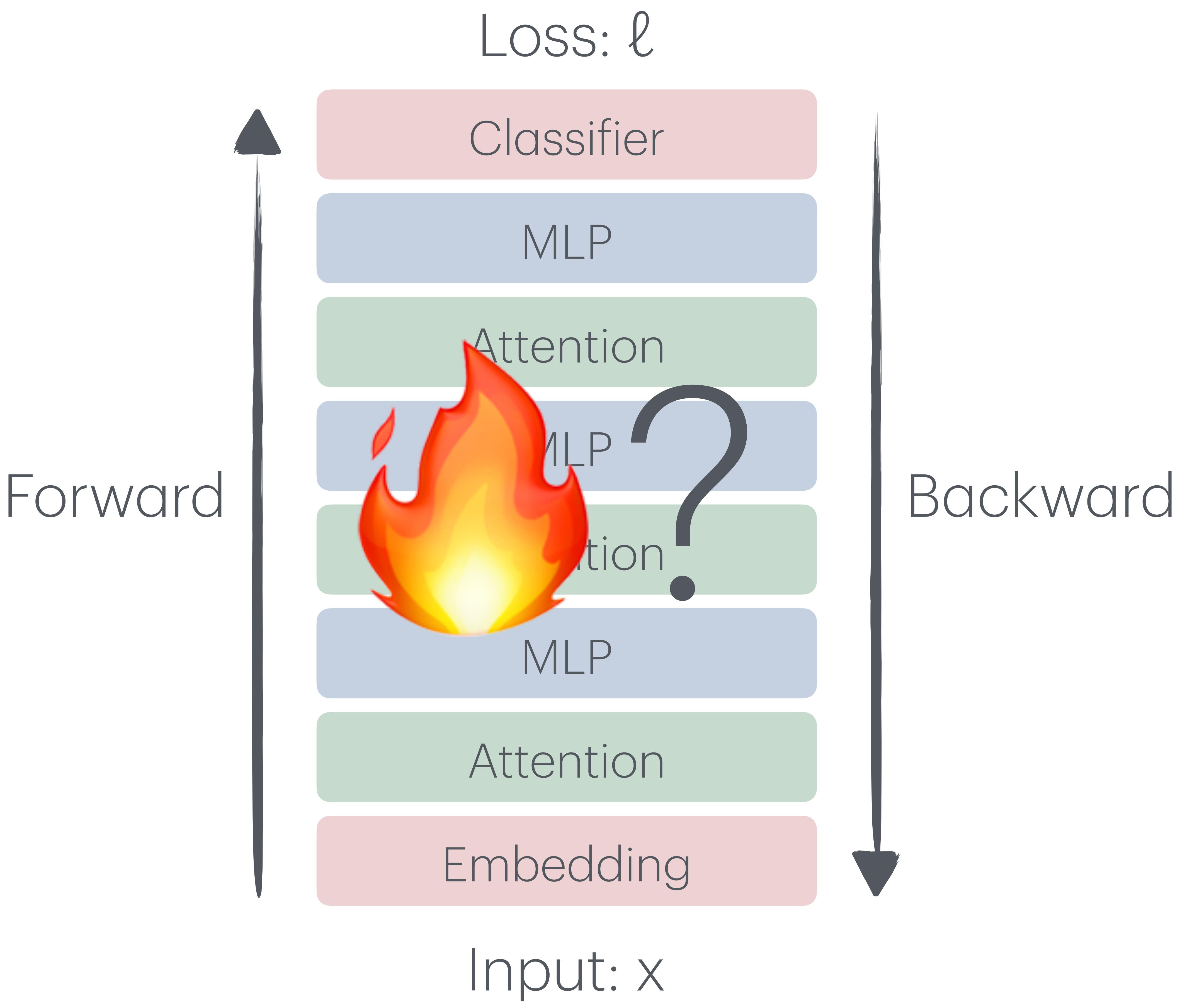
Input adapters

- Freeze backbone
- Train input embedding (maybe classifier)
- Fairly memory efficient
  - Very few learnable parameters
- Popular with LLMs
  - Soft-prompting
  - Adapters for new inputs



Intermediate layers

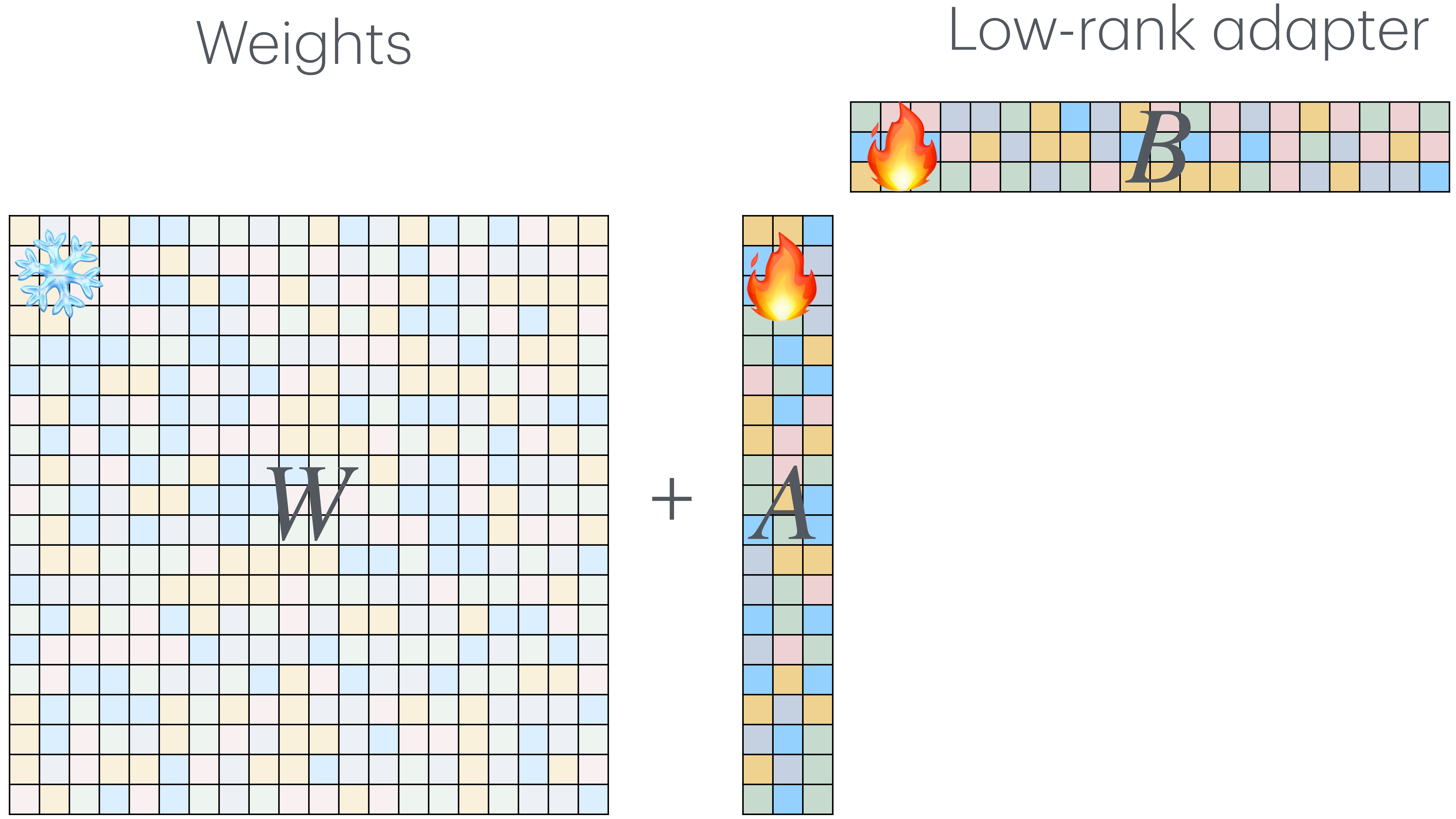
- Fine-tuning input and output does not change computation inside network significantly
  - Cannot learn new "internal computation"
- Can we learn a subset of intermediate layer parameters?



Low Rank Adapters

LoRA

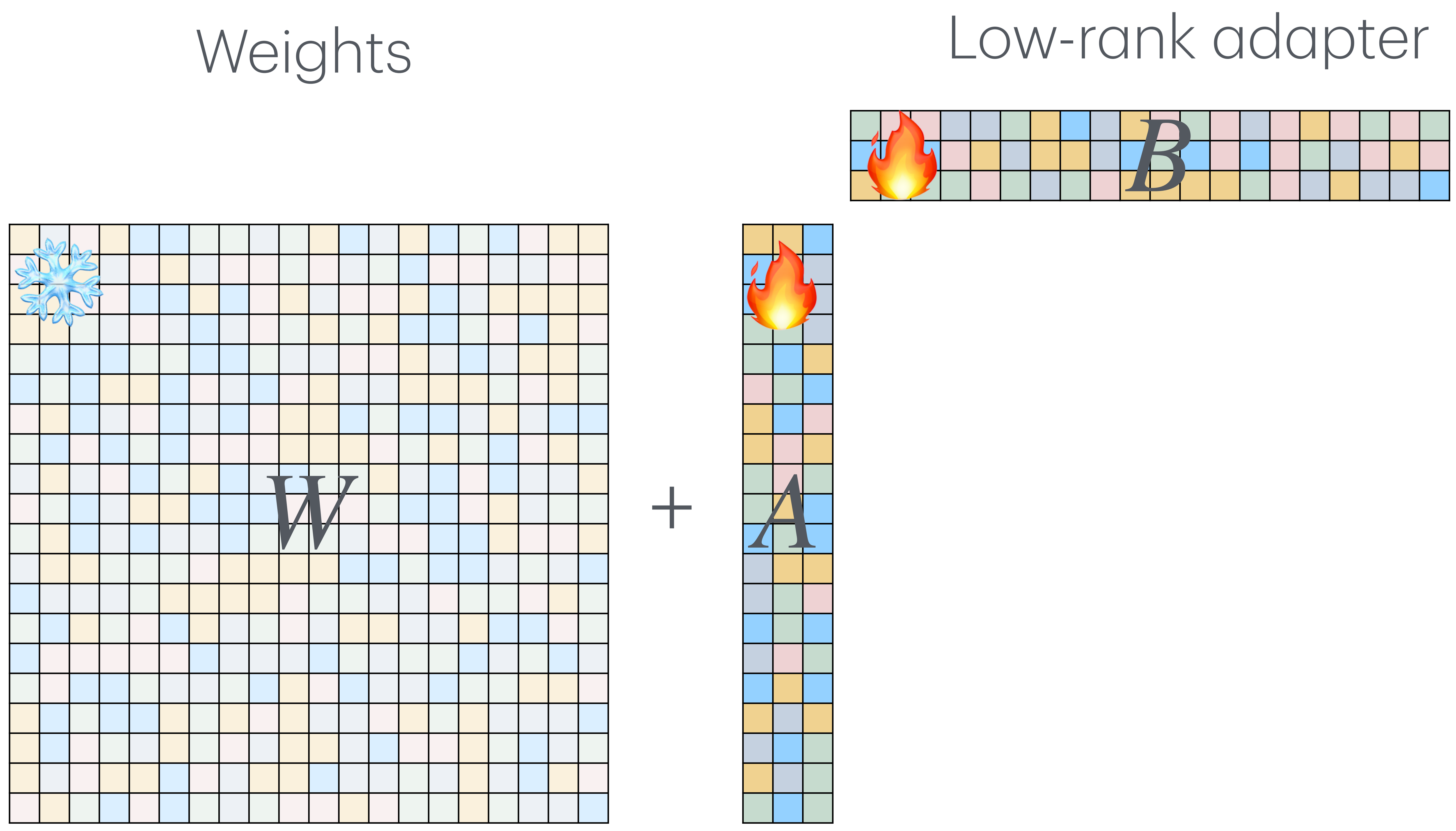
- Keep weights $W \in \mathbb{R}^{N \times M}$ frozen
- Learn adapter AB
    - $A \in \mathbb{R}^{N \times R}$, $B \in \mathbb{R}^{R \times M}$
    - Rank $R \ll min(M, N)$
- Total parameters: $R(N + M)$



[1] Edward J. Hu, et al. Lora: Low-rank adaptation of large language models. 2021

Low Rank Adapters

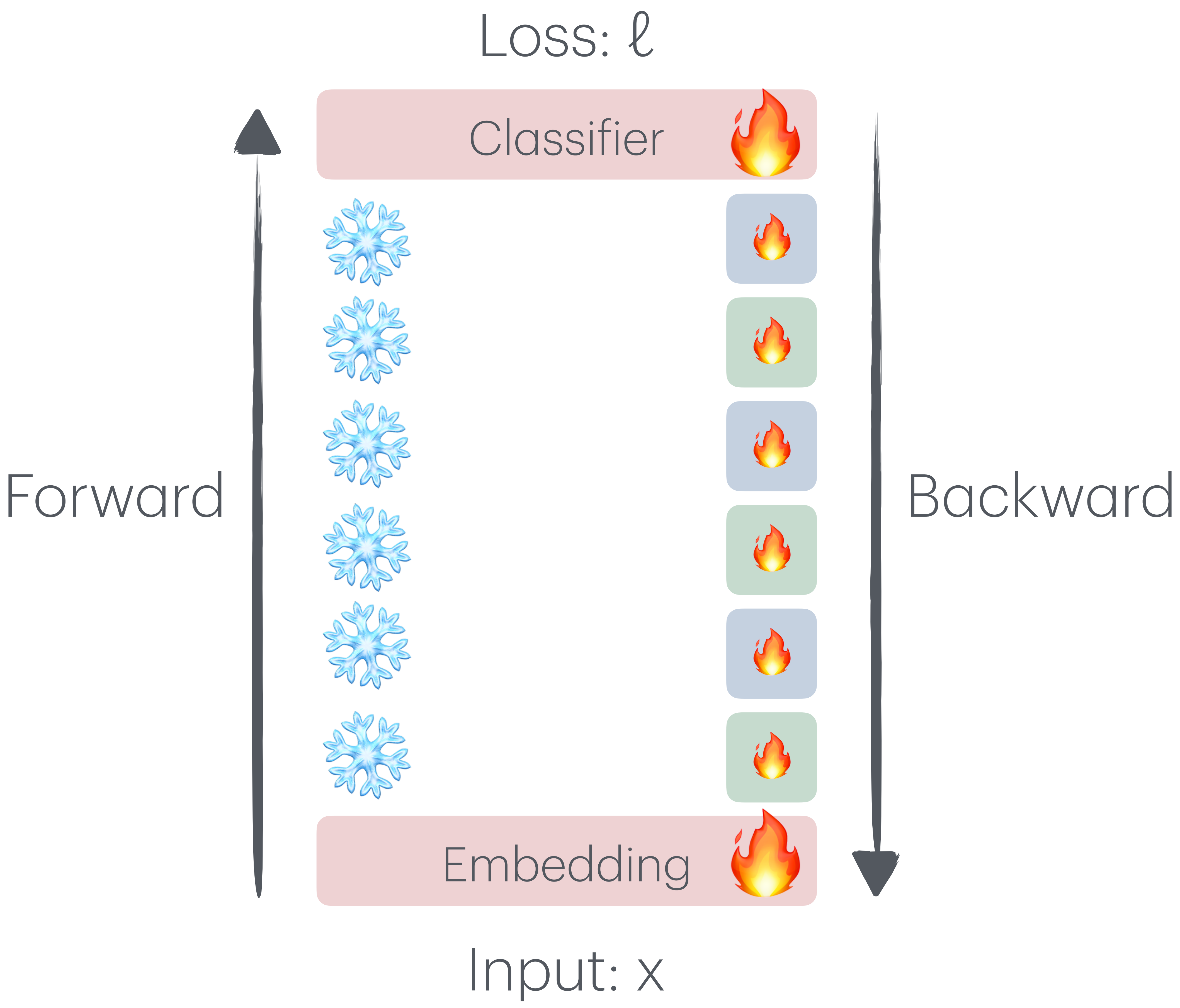
- How do we train W?
  - We use a pre-trained model
- How do we initialize A and B?
  - A small random (normal)
  - B zero



[1] Edward J. Hu, et al. Lora: Low-rank adaptation of large language models. 2021

LoRA models

- Most weights frozen
- Train adapter for
  - All linear layers
  - Just MLPs
  - Just Attention
- Optionally train full input and output embedding



LoRA in practice

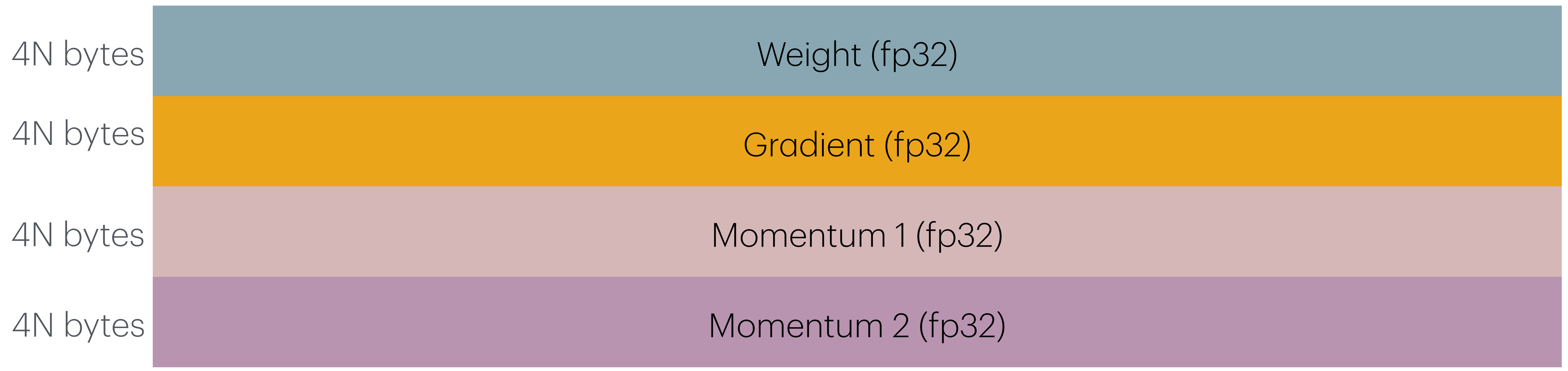
- Download weights of a pre-trained model
- Define LoRALinear layer
  - $Wx + b \rightarrow Wx + b + \alpha \frac{AB}{R} x$
- Rewrite model with LoRALinear layers
  - If careful, `load_state_dict` just works

```python
class LoRALinear(nn.Linear):
    def __init__(self, in_features: int, out_features: int, rank: int,
                 alpha: float, bias: bool = True, device=None,
                 dtype=None):
        super().__init__(in_features, out_features, bias, device, dtype)

        self.lora_a = nn.Linear(in_features, rank, bias=False,
                                device=device, dtype=dtype)
        self.lora_b = nn.Linear(rank, out_features, bias=False,
                                device=device, dtype=dtype)
        self.alpha_div_rank = alpha / rank

        nn.init.kaiming_uniform_(self.lora_a.weight)
        nn.init.zeros_(self.lora_b.weight)

    def forward(self, x: Tensor) -> Tensor:
        return super().forward(x) +
            self.alpha_div_rank / self.lora_b(self.lora_a(x))
```

Training large models

Memory requirements

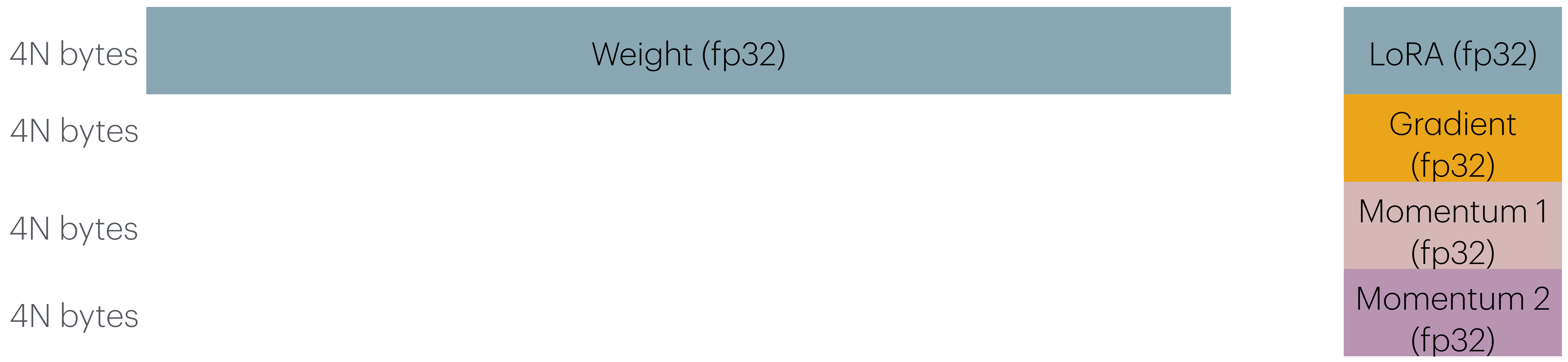
- Without optimization:
  - Model parameters: N
  - Weights: N floats
  - Gradients: N floats
  - Momentum: N floats
  - 2nd momentum (ADAM): N floats
- 16N bytes without counting activations



Training LoRA models

Memory requirements

- LoRA
  - Model parameters: N, LoRA param M
  - Weights: N+M floats
  - Gradients: M floats
  - Momentum: M floats
  - 2nd momentum (ADAM): M floats
- 4N+16M bytes without activations
- M often ~1-5% of N



References

- [1] Edward J. Hu, et al. Lora: Low-rank adaptation of large language models. 2021 (link)