Computational Graphs

Recap: Gradients

Gradient of simple functions

Fine to compute

Gradient of regular functions

Quickly get complicated

$$
\begin{aligned}
&\nabla_{\mathbf{w}} L(\theta|\mathcal{D}) \\
&= \nabla_{\mathbf{w}} E_{\mathbf{x},y\sim\mathcal{D}}\left[l(\theta|\mathbf{x},y)\right] \\
&= E_{\mathbf{x},y\sim\mathcal{D}}\left[\nabla_{\mathbf{w}} l(\theta|\mathbf{x},y)\right] \\
&= E_{\mathbf{x},y\sim\mathcal{D}}\left[\nabla_{\mathbf{w}} (\mathbf{w}^\top\mathbf{x}+b-y)^2\right] \\
&= 2E_{\mathbf{x},y\sim\mathcal{D}}\left[(\mathbf{w}^\top\mathbf{x}+b-y)\nabla_{\mathbf{w}}\mathbf{w}^\top\mathbf{x}\right] \\
&= 2E_{\mathbf{x},y\sim\mathcal{D}}\left[(\mathbf{w}^\top\mathbf{x}+b-y)\mathbf{x}^\top\right]
\end{aligned}
$$

General Linear Regression Model:

$$l(\theta|\mathbf{x},\mathbf{y}) = (\mathbf{W}\mathbf{x}+\mathbf{b}-\mathbf{y})^2$$

Binary logistic regression:

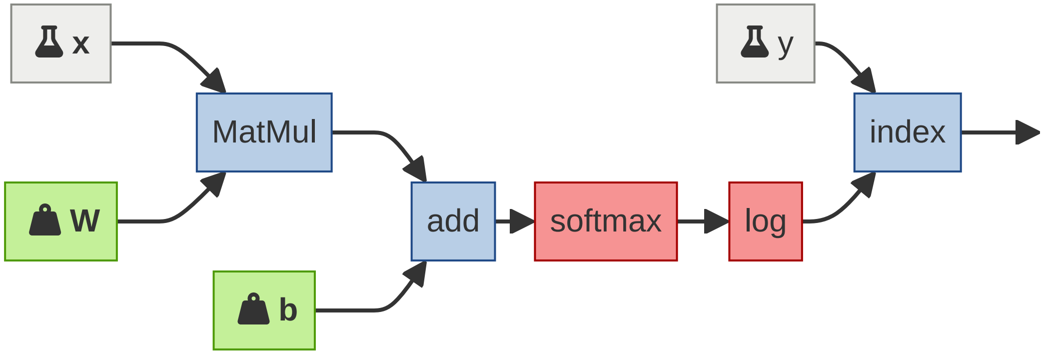
$$
\begin{aligned}
l(\theta|\mathbf{x},\mathbf{y}) = {} & y\log\sigma(\mathbf{W}\mathbf{x}+\mathbf{b}) \\
& + (1-y)\log(1-\sigma(\mathbf{W}\mathbf{x}+\mathbf{b}))
\end{aligned}
$$

Multi-class logistic regression:

$$l(\theta|\mathbf{x},\mathbf{y}) = \log\operatorname{softmax}(\mathbf{W}\mathbf{x}+\mathbf{b})_y$$

Computation as a Graph

$$
\begin{aligned}
l(\theta|\mathbf{x},\mathbf{y}) &= \log\left(\text{softmax}(\mathbf{W}\mathbf{x}+\mathbf{b})\right)_y \\
&= \text{index}\left(\log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right), y\right)
\end{aligned}
$$



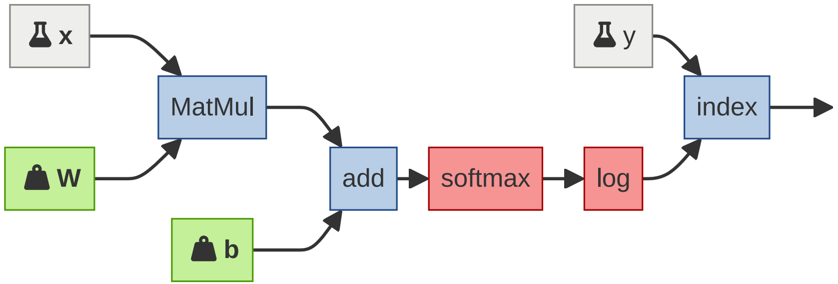
Gradients and Chain Rule

$$l(\theta|\mathbf{x},\mathbf{y}) = \text{index}\left(\log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right),y\right)$$

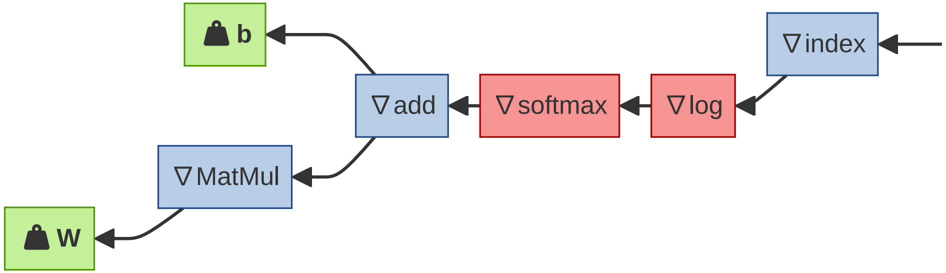
$$\begin{aligned}
&\nabla_\theta l(\theta|\mathbf{x},\mathbf{y}) = \nabla_\theta \text{index}\left(\log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right),y\right) \\
&= \underbrace{\frac{\partial}{\partial \log}\text{index}(\ldots)}_{\nabla \text{index}}\nabla_\theta \log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right) \\
&= \nabla\text{index}(\ldots)\nabla\log(\ldots)\nabla_\theta\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right) \\
&= \ldots \\
&= \nabla\text{index}(\ldots)\nabla\log(\ldots)\nabla\text{softmax}(\ldots)\nabla\text{add}(\ldots)\left(\nabla\text{matmul}(\mathbf{W},\mathbf{x})\nabla_\theta\mathbf{W} + \nabla_\theta\mathbf{b}\right)
\end{aligned}$$

Gradients on Computation Graphs

$$l(\theta|\mathbf{x},\mathbf{y}) = \text{index}\left(\log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right), y\right)$$

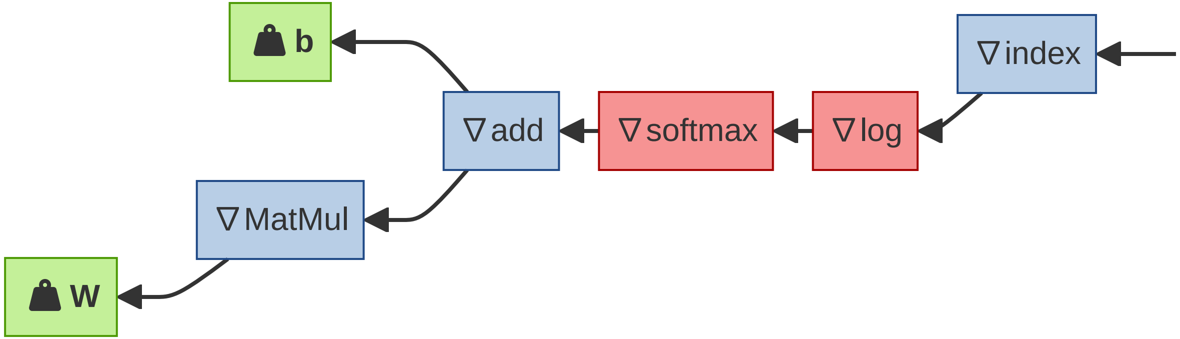


$$\nabla_\theta l(\theta|\mathbf{x},\mathbf{y}) = \nabla\text{index}(\ldots)\nabla\log(\ldots)\nabla\text{softmax}(\ldots)\nabla\text{add}(\ldots)\left(\nabla\text{matmul}(\mathbf{W},\mathbf{x})\nabla_\theta\mathbf{W} + \nabla_\theta\mathbf{b}\right)$$



Gradients - Direction of Evaluation

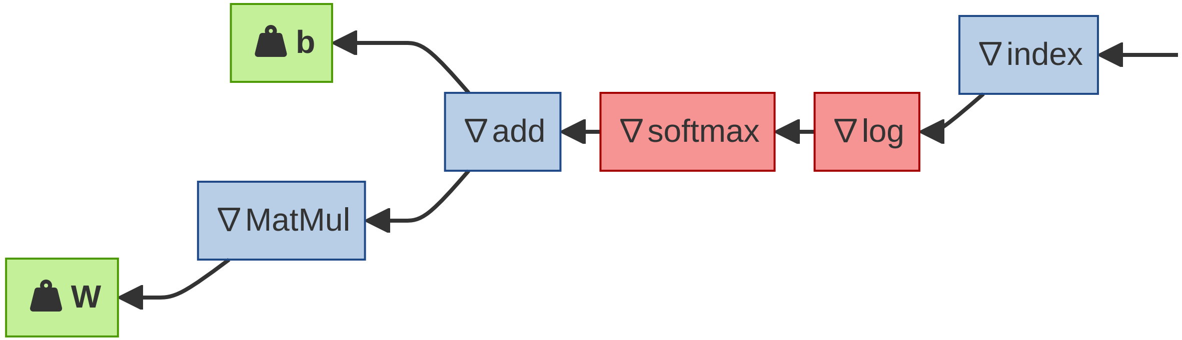
$$\underbrace{\nabla_\theta l(\theta|\mathbf{x},\mathbf{y})}_{\mathbb{R}^n} = \underbrace{\nabla \text{index}(\ldots)}_{\mathbb{R}^n}\underbrace{\nabla \log(\ldots)}_{\mathbb{R}^{n\times m}}\underbrace{\nabla \text{softmax}(\ldots)}_{\mathbb{R}^{m\times l}}\underbrace{\nabla \text{add}(\ldots)}_{\mathbb{R}^{l\times k}}\left(\underbrace{\nabla \text{matmul}(\mathbf{W},\mathbf{x})\nabla_\theta \mathbf{W}}_{\mathbb{R}^{k\times\ldots}} + \underbrace{\nabla_\theta \mathbf{b}}_{\mathbb{R}^{k\times\ldots}}\right)$$



Gradients - Backpropagation

Gradients computed backwards in graph

- Computationally more efficient
- One backward pass computes gradients of **all** parameters



Gradients: Backpropagation in Practice

Each operation in PyTorch

- Has backward-function implemented
- Graph constructed automatically

Backward pass

- Multiplies vector with Jacobian of operator
- Start by back-propagating value of 1 to loss
- Can only call backward on scalars
- Populates `Tensor.grad` for any tensor that `requires_grad=True`

```
a = torch.rand(100, requires_grad=True)
b = 0.5 * (a**2).sum()
b.backward()
a.grad
```

Computational Graphs TL;DR

PyTorch builds computational graph for automatic differentiation

Gradients are propagated backwards through computational graph: **backpropagation**

Call `Tensor.backward()` in PyTorch

No more complicated gradient math