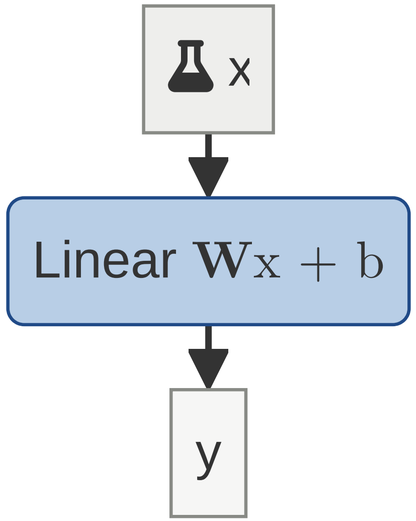
Computational Graphs

Philipp Krähenbühl, UT Austin

Recap: Training a linear network

Train a single layer deep network

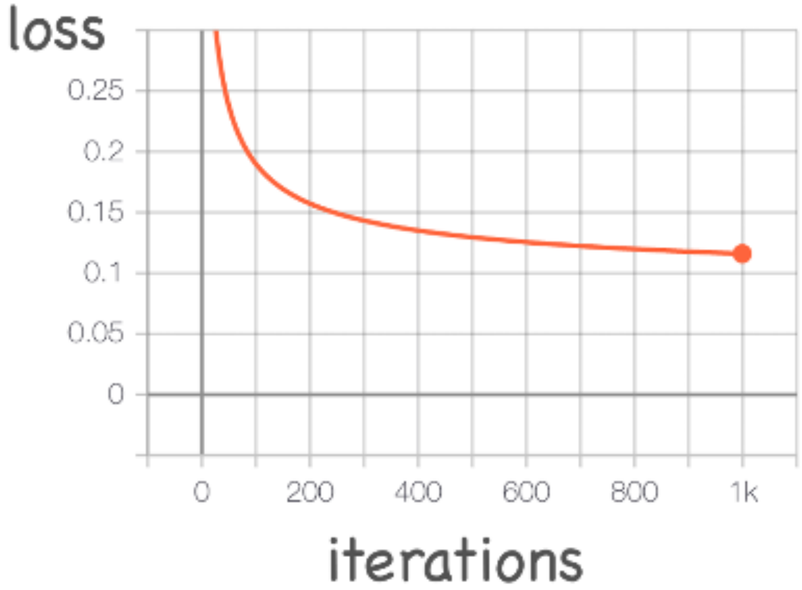
Task / Model



Dataset / Loss

$$l(\theta|\mathbf{x}, \mathbf{y})$$

Training / Optimization



Recap: Gradient descent

Gradient $\nabla L(\theta | D) = E_{x,y \sim \mathcal{D}} \left[\nabla \ell(\theta | x, y) \right]$

```
θ ~ Init
for iteration in range(n):
    J = ∇L(θ)
    θ = θ - ϵ * J.mT
```

Recap: Gradient of Linear Regression

- $$\ell(\theta \mid \mathbf{x}, y) = (f_\theta(\mathbf{x}) - y)^2 = (\mathbf{w}^\top \mathbf{x} + b - y)^2$$
  $$\nabla_b \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)$$
  $$\nabla_{\mathbf{w}} \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)\mathbf{x}^\top$$

Basic Linear Algebra

- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix} \in \mathbb{R}^n$
- Scalar functions $f : \mathbb{R}^n \to \mathbb{R} \qquad f(\mathbf{v}) = f(v_1, v_2, \ldots, v_n)$
- Vector-valued functions $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m \qquad f(\mathbf{v}) = \begin{bmatrix} f_1(v_1, v_2, \ldots, v_n) \\ f_2(v_1, v_2, \ldots, v_n) \\ \cdots \\ f_m(v_1, v_2, \ldots, v_n) \end{bmatrix}$

Derivatives (of scalar functions)

$$f : \mathbb{R}^n \to \mathbb{R} \qquad f(\mathbf{v}) = f(v_1, v_2, \ldots, v_n)$$

- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix} \in \mathbb{R}^n$
- Derivative (scalar inputs) $\frac{\partial}{\partial x} f(x) \approx \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \in \mathbb{R}$
- Gradient (vector inputs) $\nabla_{\mathbf{v}} f(\mathbf{v}) = \left[\frac{\partial}{\partial v_1} f(\mathbf{v}), \frac{\partial}{\partial v_2} f(\mathbf{v}), \ldots, \frac{\partial}{\partial v_n} f(\mathbf{v}) \right] \in \mathbb{R}^n$

Derivatives (of vector-valued functions)

- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix} \in \mathbb{R}^n$

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m \qquad f(\mathbf{v}) = \begin{bmatrix} f_1(v_1, v_2, \ldots, v_n) \\ f_2(v_1, v_2, \ldots, v_n) \\ \cdots \\ f_m(v_1, v_2, \ldots, v_n) \end{bmatrix}$$

- Derivative (scalar inputs) $\frac{\partial}{\partial x}\mathbf{f}(x) = \begin{bmatrix} \frac{\partial}{\partial x} f_1(x) \\ \frac{\partial}{\partial x} f_2(x) \\ \cdots \\ \frac{\partial}{\partial x} f_m(x) \end{bmatrix} \in \mathbb{R}^m$

- Gradient (vector inputs) $\nabla_{\mathbf{v}}\mathbf{f}(\mathbf{v}) = \mathbf{J}_{\mathbf{f}}(\mathbf{v}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{v})}{\partial v_1} & \frac{\partial f_1(\mathbf{v})}{\partial v_2} & \cdots & \frac{\partial f_1(\mathbf{v})}{\partial v_n} \\ \frac{\partial f_2(\mathbf{v})}{\partial v_1} & \frac{\partial f_2(\mathbf{v})}{\partial v_2} & \cdots & \frac{\partial f_2(\mathbf{v})}{\partial v_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{v})}{\partial v_1} & \frac{\partial f_m(\mathbf{v})}{\partial v_2} & \cdots & \frac{\partial f_m(\mathbf{v})}{\partial v_n} \end{bmatrix} = \begin{bmatrix} \nabla_{\mathbf{v}} f_1(\mathbf{v}) \\ \nabla_{\mathbf{v}} f_2(\mathbf{v}) \\ \vdots \\ \nabla_{\mathbf{v}} f_m(\mathbf{v}) \end{bmatrix} \in \mathbb{R}^{m \times n}$

Nested functions

- Function $f : \mathbb{R}^n \to \mathbb{R}^m$
- Function $g : \mathbb{R}^k \to \mathbb{R}^n$
- Nested function: $f(g(\mathbf{v}))$

Nested functions

- Function $f : \mathbb{R}^n \to \mathbb{R}^m$
- Function $g : \mathbb{R}^k \to \mathbb{R}^n$
- Nested function: $f(g(\mathbf{v})) : \mathbb{R}^k \to \mathbb{R}^m$

Gradients of nested functions

Chain rule

- $f(g(\mathbf{v})) : \mathbb{R}^k \rightarrow \mathbb{R}^m$
- $\nabla_{\mathbf{v}} f(g(\mathbf{v})) \in \mathbb{R}^{m \times k}$
- Chain rule: $\nabla_{\mathbf{v}} f(g(\mathbf{v})) = \underbrace{\nabla_{\mathbf{y}} f(\mathbf{y})}_{\in \mathbb{R}^{m \times n}} \underbrace{\nabla_{\mathbf{v}} g(\mathbf{v})}_{\in \mathbb{R}^{n \times k}}$ where $\mathbf{y} = g(\mathbf{v}))$

- Function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
- Function $g : \mathbb{R}^k \rightarrow \mathbb{R}^n$
- Nested function: $f(g(\mathbf{v})) : \mathbb{R}^k \rightarrow \mathbb{R}^m$

Exercise

- What is the gradient of $\ell = \underbrace{(\mathbf{w}^\top \mathbf{x} + b - y)}_{z}{}^2$ (in terms of $\mathbf{x}, y, z$)

$\nabla_{\mathbf{w}} \ell =$

$\nabla_{\mathbf{b}} \ell =$

Exercise

- What is the gradient of $\ell = (\underbrace{\mathbf{w}^\top \mathbf{x} + b - y}_{z})^2$ (in terms of $\mathbf{x}, y, z$)

$$\nabla_{\mathbf{w}} \ell = \underbrace{\left(\nabla_z z^2\right)}_{2z} \underbrace{\nabla_{\mathbf{w}} z}_{\mathbf{x}^\top} = 2z\mathbf{x}^\top$$

$$\nabla_{\mathbf{b}} \ell = \underbrace{\left(\nabla_z z^2\right)}_{2z} \underbrace{\nabla_b z}_{1} = 2z$$

Gradient of Linear Regression

- $$\ell(\theta \mid \mathbf{x}, y) = (f_\theta(\mathbf{x}) - y)^2 = (\mathbf{w}^\top \mathbf{x} + b - y)^2$$
  $$\nabla_b \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)$$
  $$\nabla_{\mathbf{w}} \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)\mathbf{x}^\top$$

Exercise: Gradient of Multi-class Classification

$$\ell(\theta \mid \mathbf{x}, y) = \left(\log \text{softmax}(\mathbf{W}\mathbf{x} - \mathbf{b})\right)_y$$

- $\nabla_{\mathbf{W}} \ell(\theta \mid \mathbf{x}, y) =$
  $\nabla_{\mathbf{b}} \ell(\theta \mid \mathbf{x}, y) =$

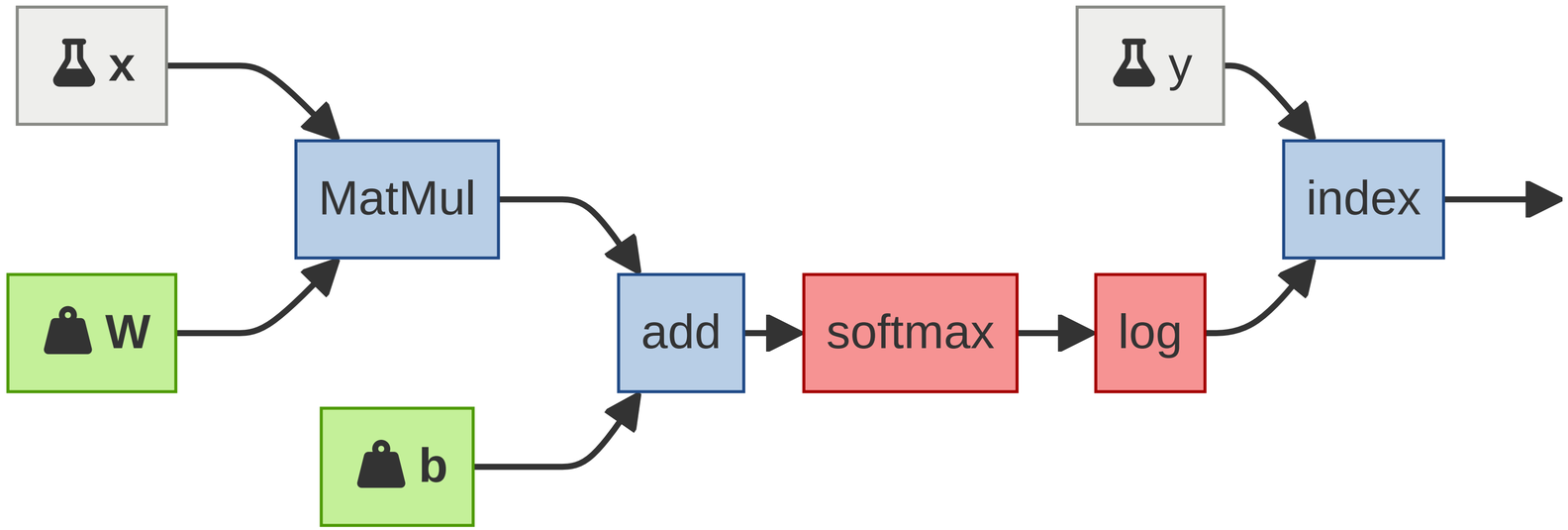
Exercise: Gradient of Multi-class Classification

$$\ell(\theta \mid \mathbf{x}, y) = \left(\log \text{softmax}(\mathbf{W}\mathbf{x} - \mathbf{b})\right)_y$$

- $\nabla_{\mathbf{W}} \ell(\theta \mid \mathbf{x}, y) =$
  $\nabla_{\mathbf{b}} \ell(\theta \mid \mathbf{x}, y) =$
- Way too hard!!!!

Computation as a graph

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



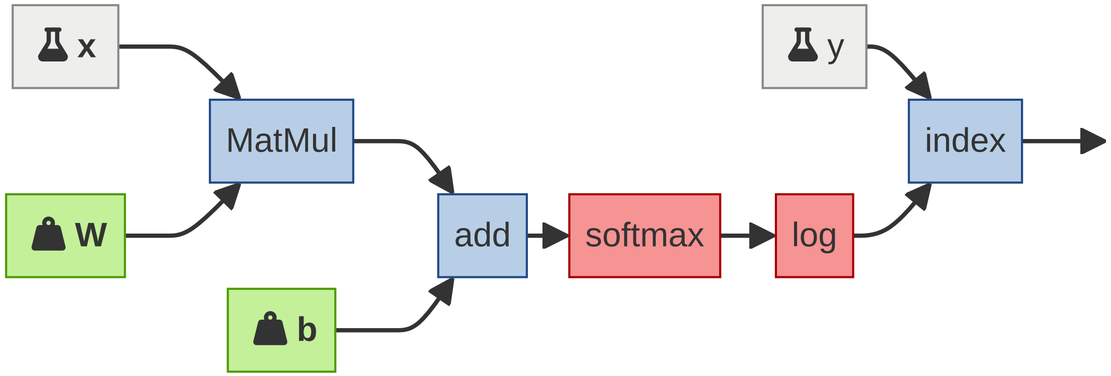
Chain-rule on a graph

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

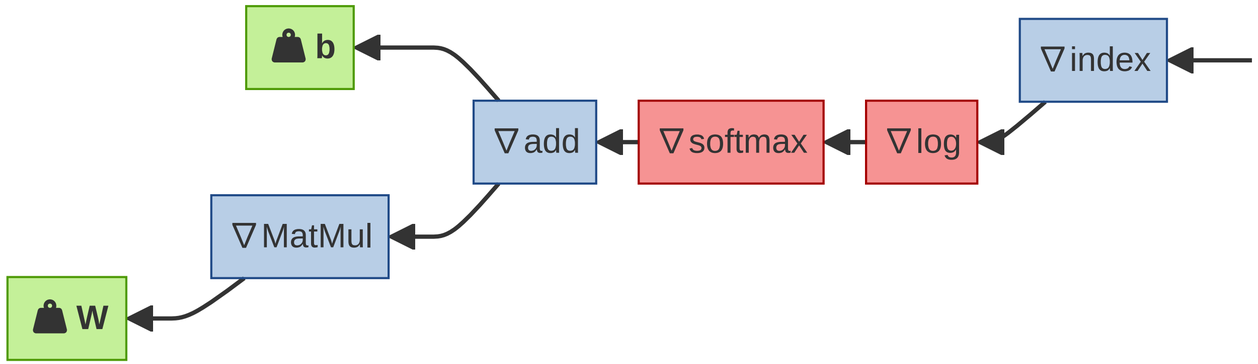
$$\begin{aligned}
\nabla_\theta l(\theta|\mathbf{x},\mathbf{y}) &= \nabla_\theta \text{index}\left(\log\left(\text{softmax}\left(\text{add}\left(\text{matmul}(\mathbf{W},\mathbf{x}),\mathbf{b}\right)\right)\right),y\right) \\
&= \underbrace{\frac{\partial}{\partial \log}\text{index}(\ldots)}_{\nabla\text{index}}\nabla_\theta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

Chain-rule on a graph

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

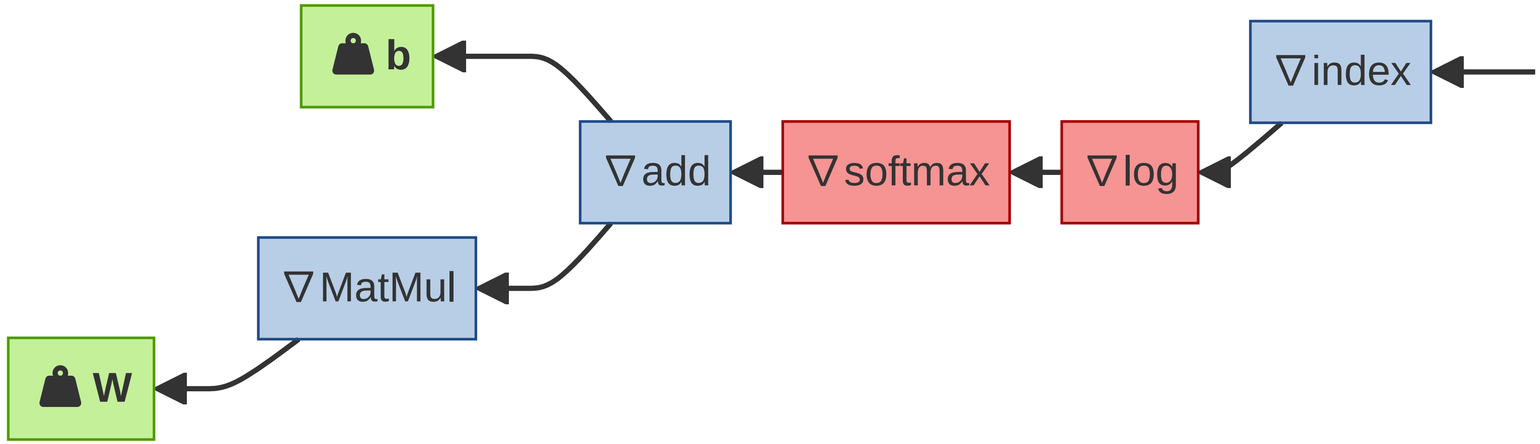


$$\nabla_\theta l(\theta|\mathbf{x},\mathbf{y}) = \nabla\text{index}(\ldots)\nabla\log(\ldots)\nabla\text{softmax}(\ldots)\nabla\text{add}(\ldots)\left(\nabla\text{matmul}(\mathbf{W},\mathbf{x})\nabla_\theta\mathbf{W} + \nabla_\theta\mathbf{b}\right)$$



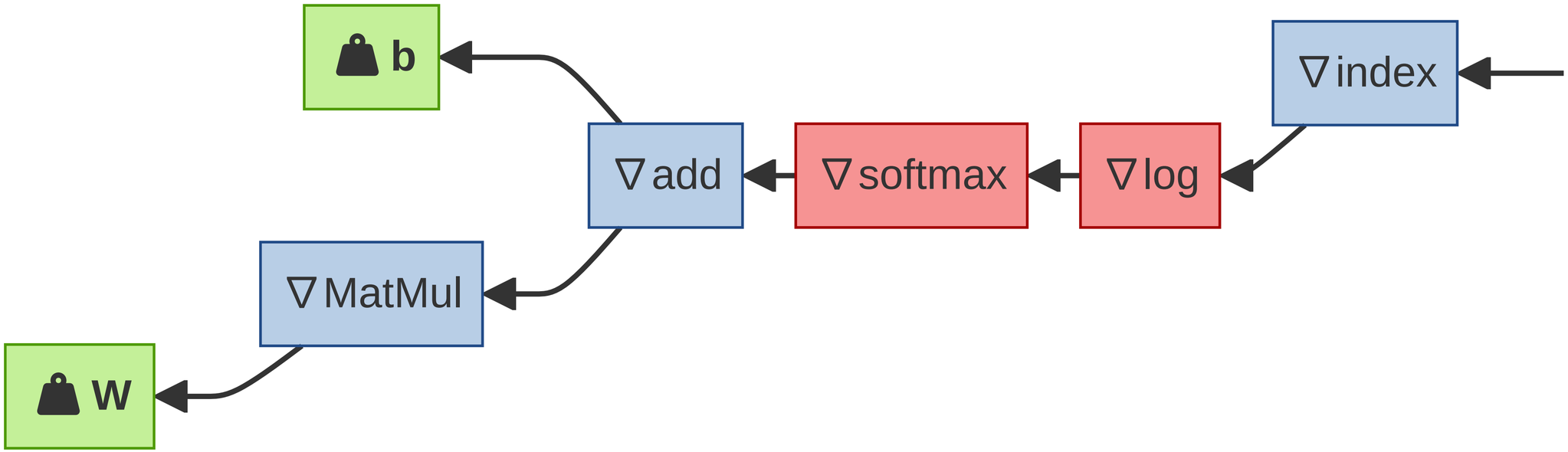
Direction of evaluation

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



Back-propagation

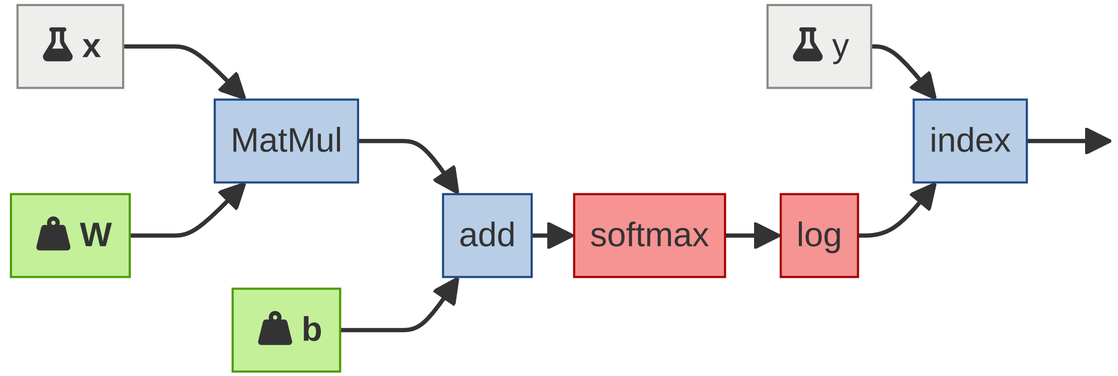
- Gradients computed backwards in graph
  - Computationally more efficient
  - Gradients of different parameters share computation

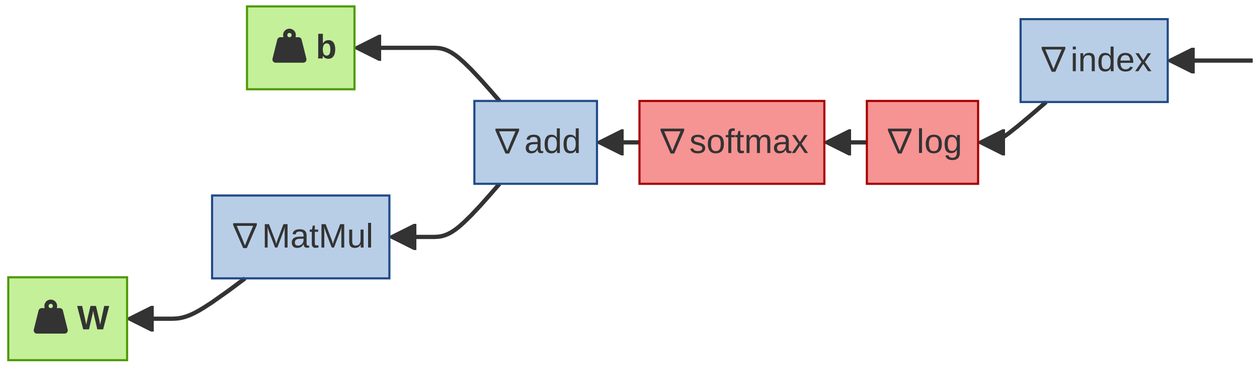


Back-propagation in PyTorch

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

- Each operation
  - Implements backprop function
  - Computation graph constructed automatically
- Backward pass
  - Multiplies “vector” with Jacobean of operation
  - Start by back-propagating value 1
  - Can only call backward on scalars
  - Populates `Tensor.grad` for all tensors with `requires_grad=True`





Computation 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

Computation Graphs in PyTorch

Linear Regression in PyTorch

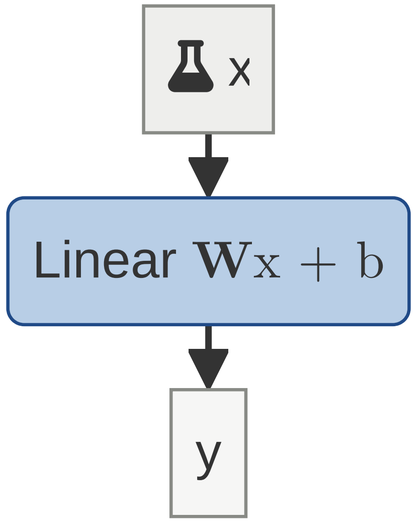
Binary Logistic Regression in PyTorch

Multi-Class Logistic Regression in PyTorch

First Example - Summary

First Example - Summary

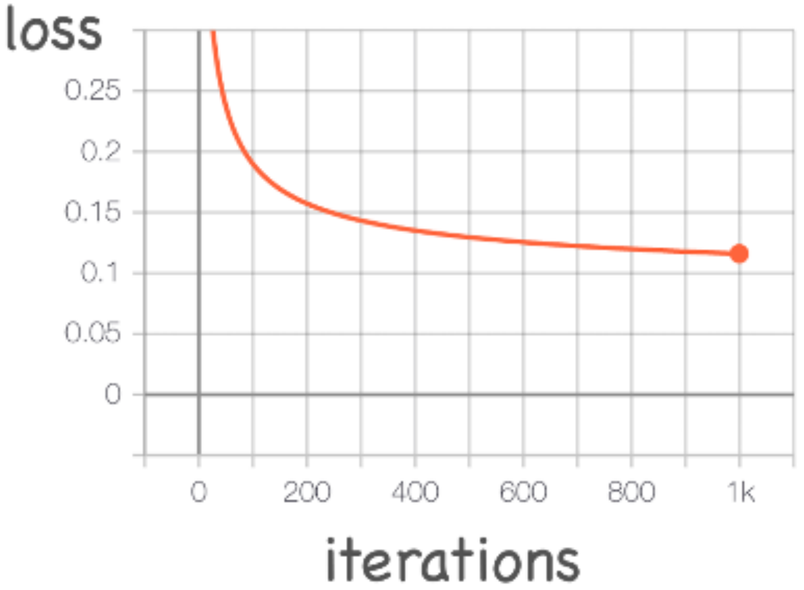
Task / Model



Dataset / Loss

$$l(\theta|\mathbf{x}, \mathbf{y})$$

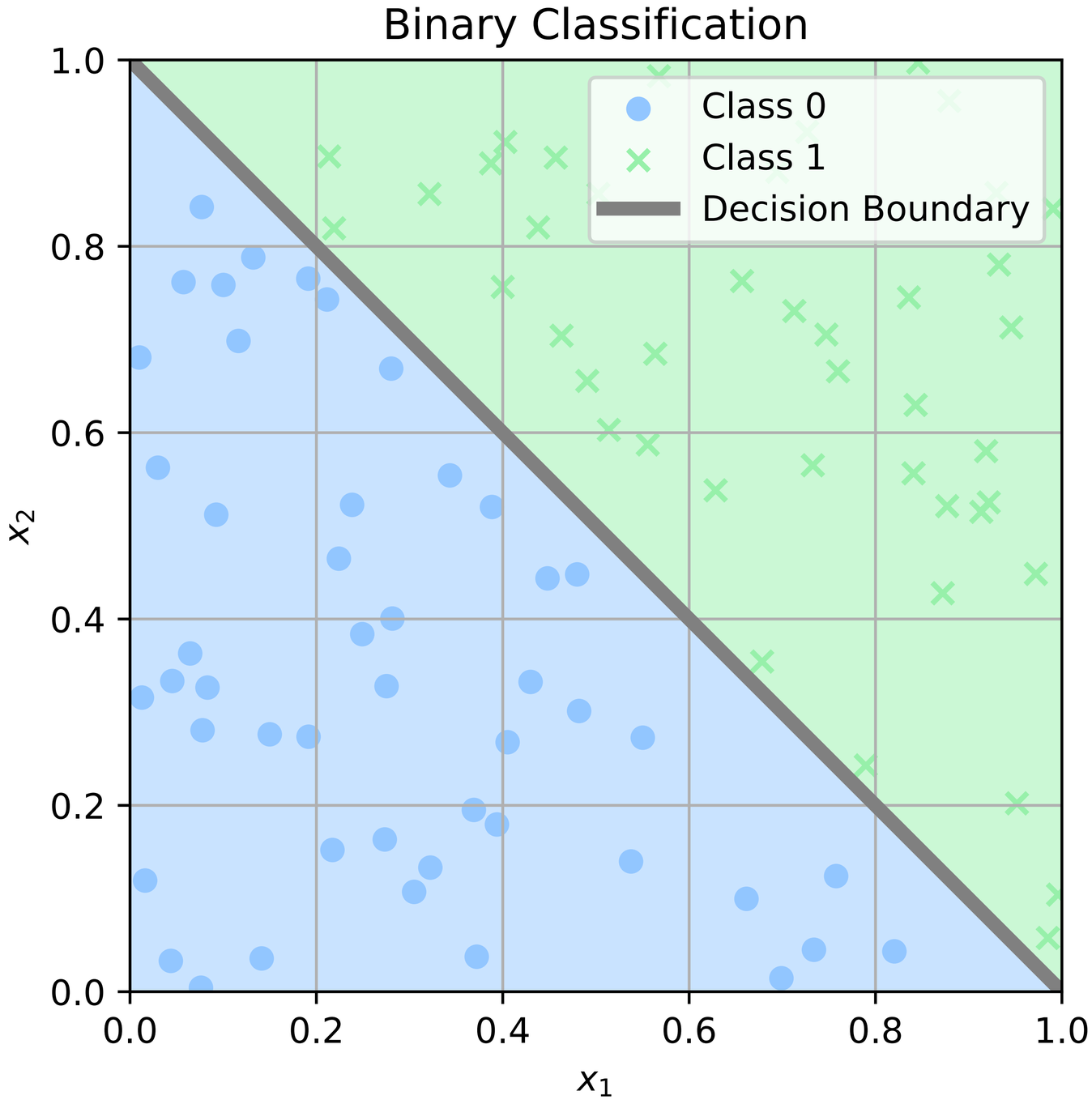
Training / Optimization



Deep Networks

Recap: Linear Binary Classification

- Binary classification model:
  - $f_\theta : \mathbb{R}^n \to [0,1]$
- Linear binary classification:
  - $f_\theta(\mathbf{x}) = \sigma(W\mathbf{x} + b)$
  - $\sigma(x) = \dfrac{1}{1 + \exp(-x)}$
- Parameters:
  - $\theta = (W, b)$



Limitations of Linear Models

- Example: Binary paw classification
  - Dog paw or not

Linear models cannot distinguish the two paw images from the gray image

Limitations of Linear Models

- Positive classes

  $\mathbf{w}^\top \mathbf{x}_{white} + b > 0$

  $\mathbf{w}^\top \mathbf{x}_{black} + b > 0$

- By definition

  $$\mathbf{x}_{gray} = \frac{1}{2}\left(\mathbf{x}_{white} + \mathbf{x}_{black}\right)$$

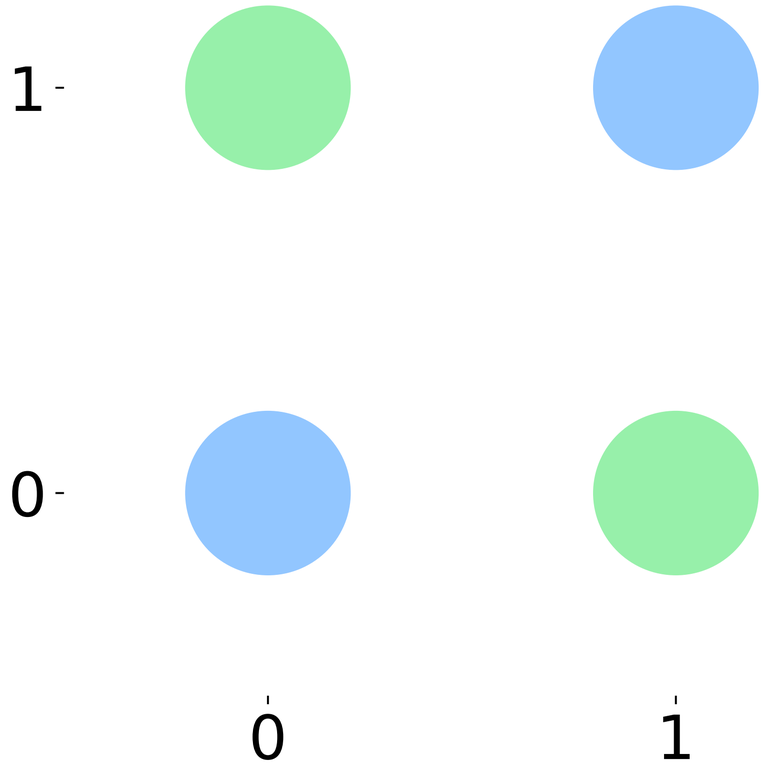
- Linearity

  $$\mathbf{w}^\top \mathbf{x}_{gray} + b = \frac{1}{2}\underbrace{\left(\mathbf{w}^\top \mathbf{x}_{white} + b\right)}_{>0} + \frac{1}{2}\underbrace{\left(\mathbf{w}^\top \mathbf{x}_{black} + b\right)}_{>0} > 0$$

Linear models cannot distinguish the two paw images from the gray image

Limitations of Linear Models

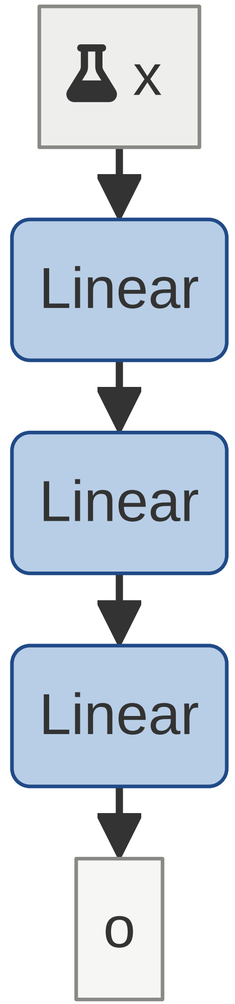
Linear models cannot learn xor



Does adding more linear layers help?

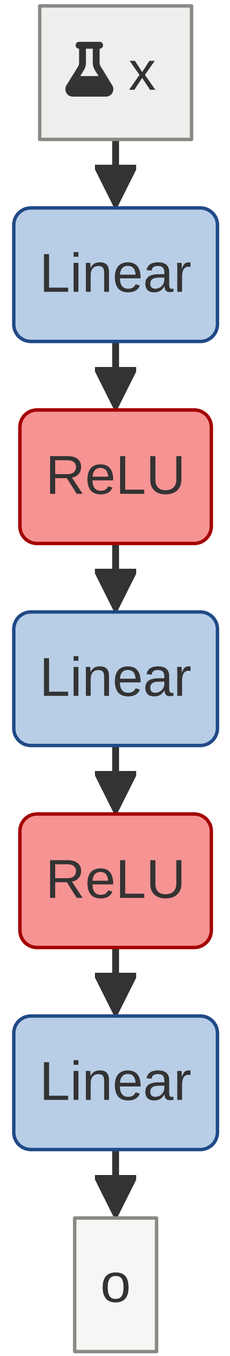
- No
- A combination of linear layers is still linear

$$\mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$
$$= (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2)$$
$$= \mathbf{W}'\mathbf{x} + \mathbf{b}'$$



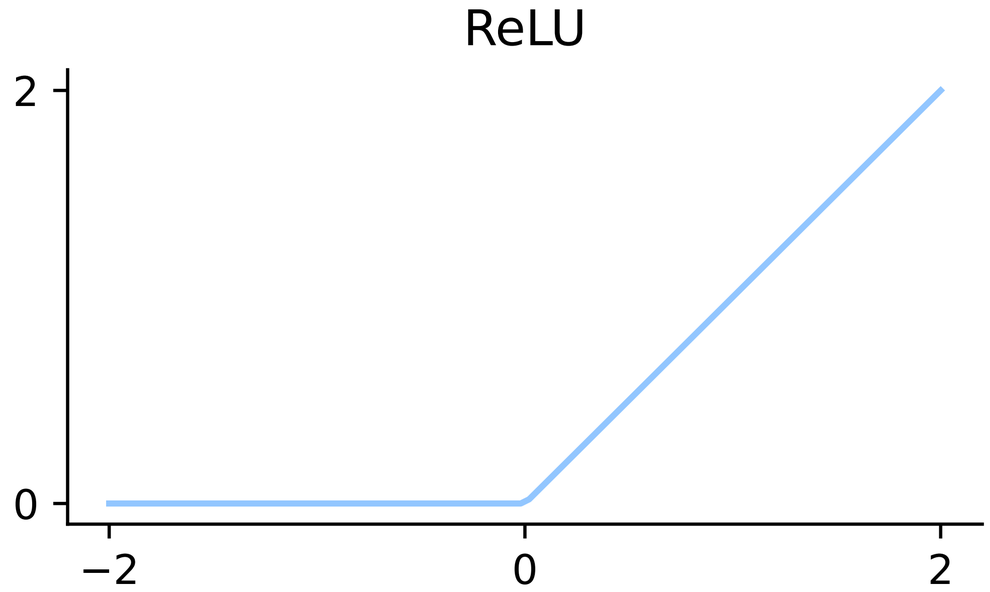
Solution: Deep Networks

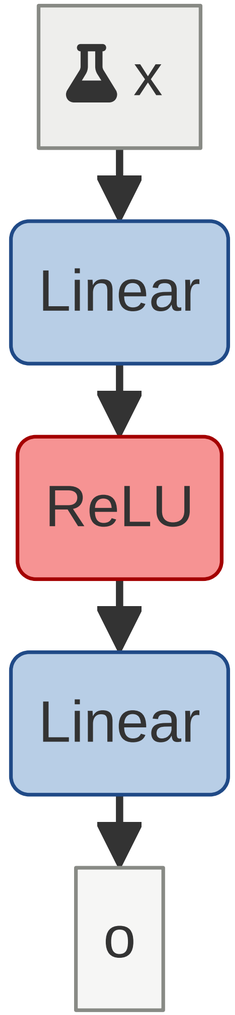
- Add non-linear functions between linear layers



Non-linearities

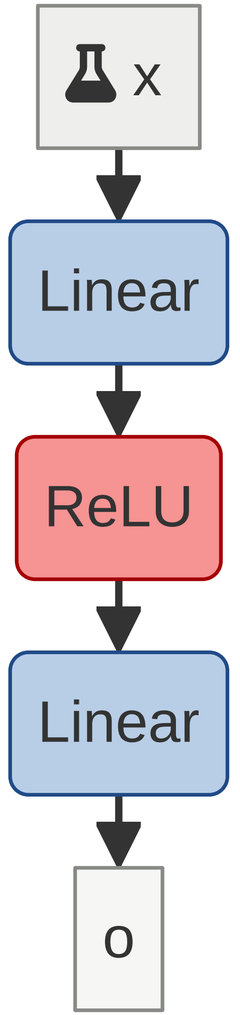
- Rectified Linear Unit (ReLU)
  $\text{ReLU}(x) = \max(x, 0)$
- Non-linear
- Differentiable almost anywhere



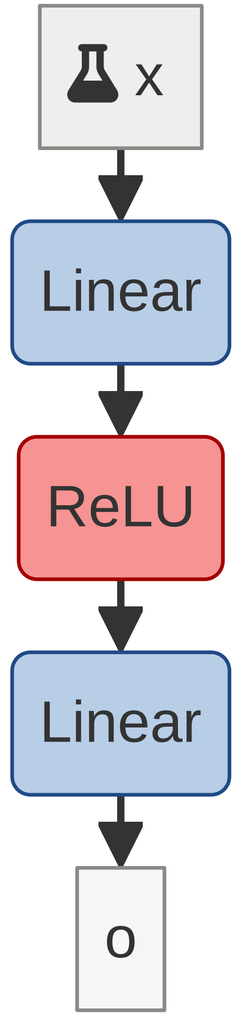


Non-linearities

- Exercise: Can we build a 2 layer network that classifies the paws
  - Layer 1: $\mathbf{z}_1 = \mathbf{W}\mathbf{x} + \mathbf{b}$
  - ReLU: $\hat{\mathbf{z}}_1 = \text{ReLU}(\mathbf{z}_1) = \max(\mathbf{z}_1, 0)$
  - Layer 2: $\mathbf{z}_2 = \mathbf{W}\hat{\mathbf{z}}_1 + \mathbf{b}$



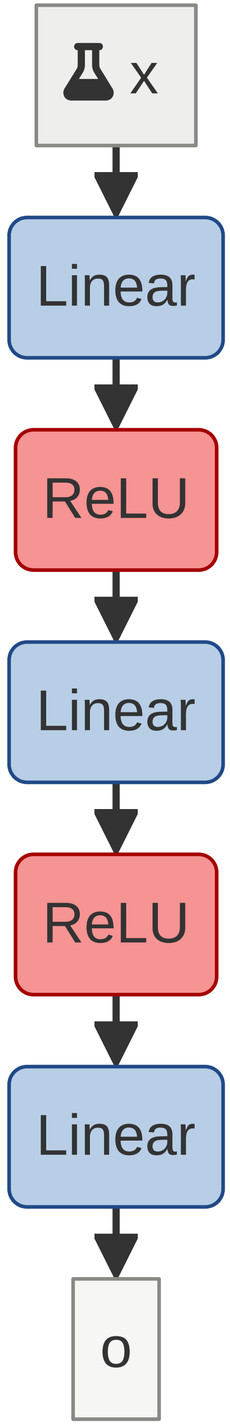
Non-linearities



- Intuition
  - first layer learns the color categories of the paw (white, black, grey)
  - second layer classifies color as paw or not
  - Layer 1: $\mathbf{z}_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot x + \frac{1}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
  - ReLU: $\hat{\mathbf{z}}_1 = \text{ReLU}(\mathbf{z}_1) = \max(\mathbf{z}_1, 0)$
  - Layer 2: $\mathbf{z}_2 = [2 \quad 2]\, \hat{\mathbf{z}}_1 - \frac{1}{2}$

Deep Networks

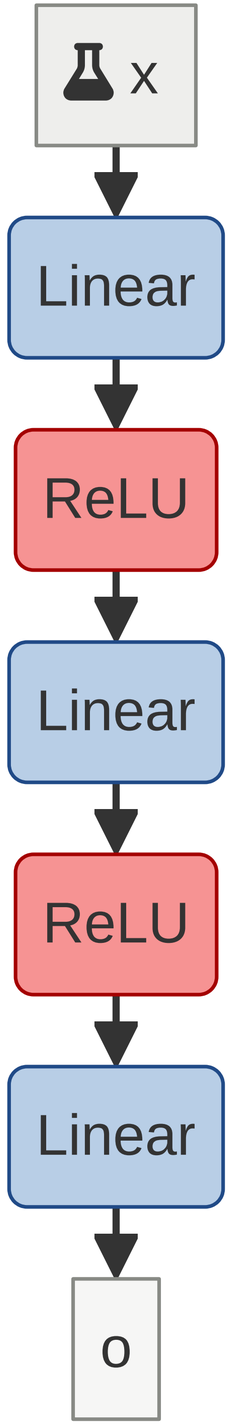
- Alternate between linear and non-linear layers



Deep Networks

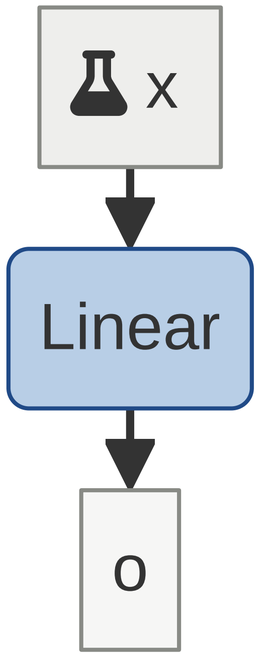
- Model: $f_\theta : \mathbb{R}^n \to \mathbb{R}^c$
- Parameters: $\theta = (\mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_N, \mathbf{b}_N)$
- Computation:

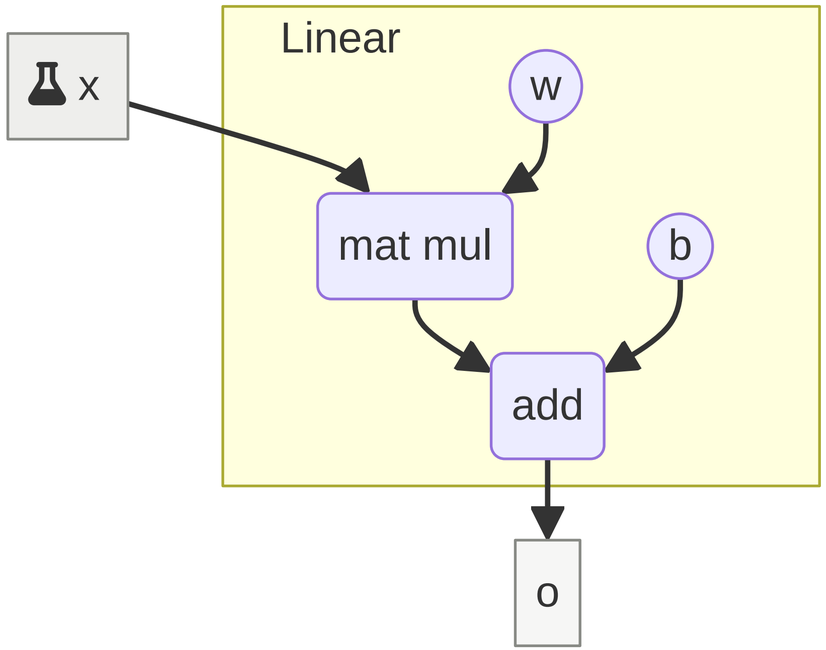
$$\mathbf{z}_1 = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{z}_2 = \text{ReLU}(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2)$$
$$\vdots$$
$$\mathbf{z}_{N-1} = \text{ReLU}(\mathbf{W}_{N-1}\mathbf{z}_{N-2} + \mathbf{b}_{N-1})$$
$$f_\theta(\mathbf{x}) = \mathbf{W}_N\mathbf{z}_{N-1} + \mathbf{b}_N$$

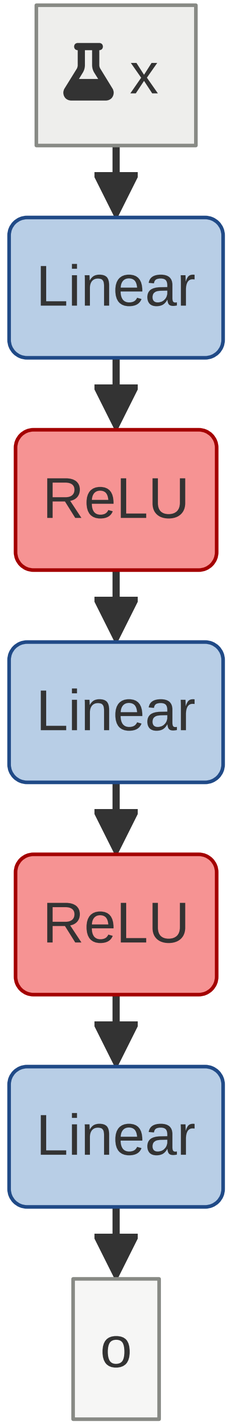


What Is a Layer?

- Largest computational unit that remains unchanged throughout different architectures

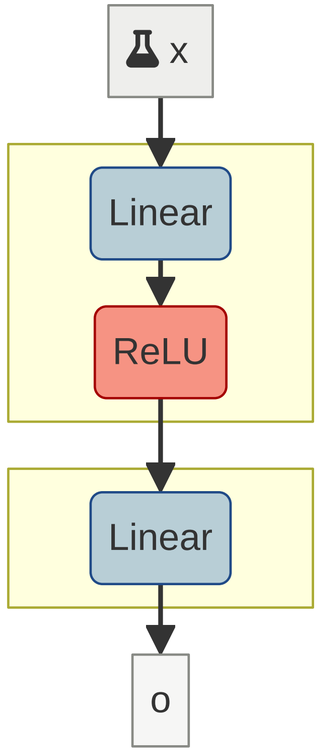


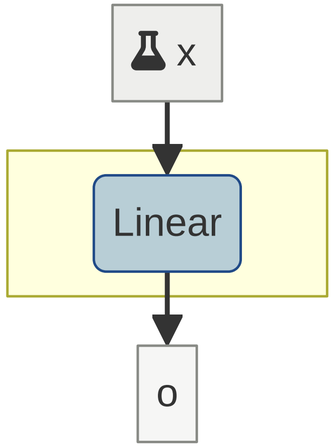




How Many Layers Does a Deep Network Have?

- We only count (consecutive) linear layers





Universal Approximation Theorem

Universal Approximation Theorem

A two-layer deep network can approximate any continuous function.

- Constructing is inefficient
- Deep learning exploit structure in data to find efficient approximations

Non-Linearities - TL;DR

- Deep networks are stacks of alternating linear and non-linear layers
- Deep networks belong to a class of continuous functions that can approximate **any** continuous function!