Deep Learning

A course about ***theory*** *&* ***practice***

Flow Graphs
& Automatic Differentiation

Marco Piastra

Learning with
Feed-Forward Neural Networks

Learning with FF Neural Networks

- Approximating a target function

$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$$

with ***feed-forward neural network***

parameters

$$\tilde{y} = \boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{\vartheta} = \left(\boldsymbol{W} \in \mathbb{R}^{h \times d},\ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}\right)$$

ReLU has been chosen in this example

Objective: *minimizing* the loss function

$$L(D, \boldsymbol{\vartheta}) = \frac{1}{N} \sum_D (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

with respect to a dataset

A set of data items

$$D := \{(\boldsymbol{x}^{(i)},\ y^{(i)})\}_{i=1}^N$$

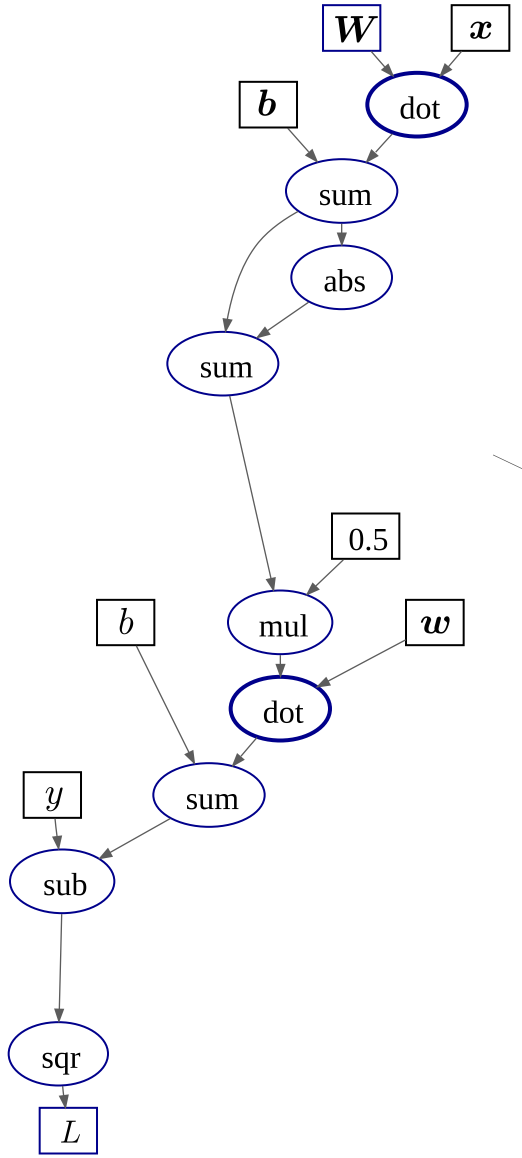
To apply *gradient descent* (any variant), we need to compute:

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} L(\tilde{y}^{(i)}, y^{(i)}, \boldsymbol{\vartheta})$$

That is, the gradient of the loss function with respect to a data item

Flow Graphs
(a.k.a. Computation Graphs)

Flow Graph

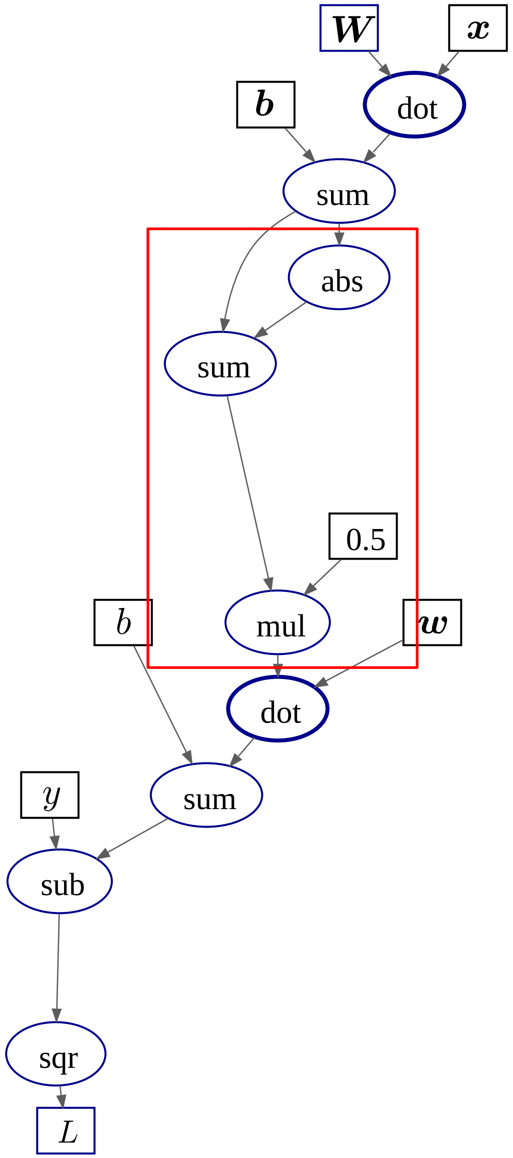


$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

The above expression translates into this flow graph

Flow Graph



$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

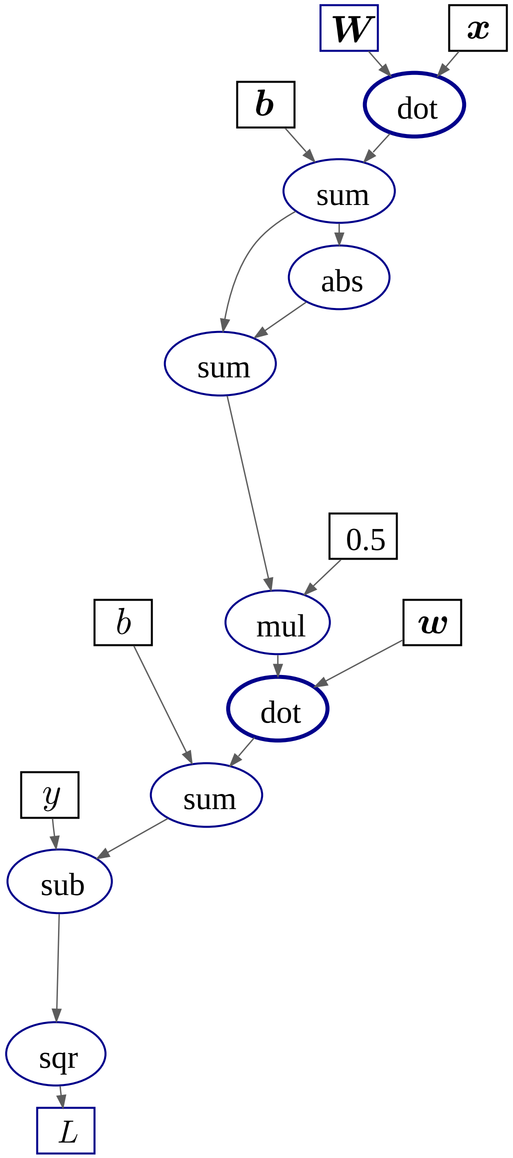
Item-wise loss function, FF neural network with ReLU as non-linearity

$$\mathrm{ReLU}(x) := \max(0, x)$$

$$\mathrm{ReLU}(x) = \frac{1}{2}(x + |x|)$$

(equivalent expression)

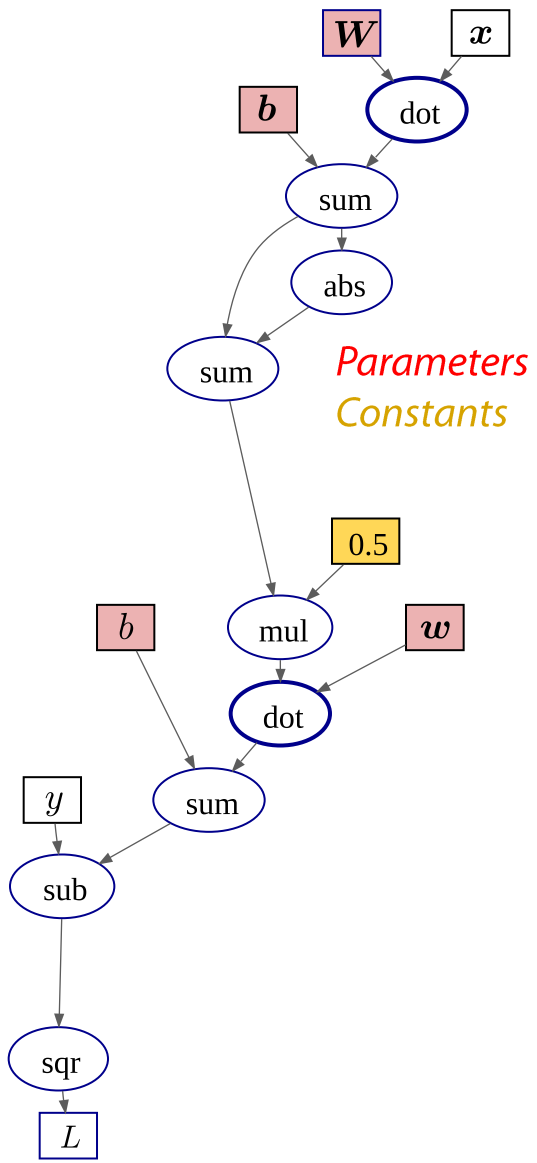
Flow Graph



$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

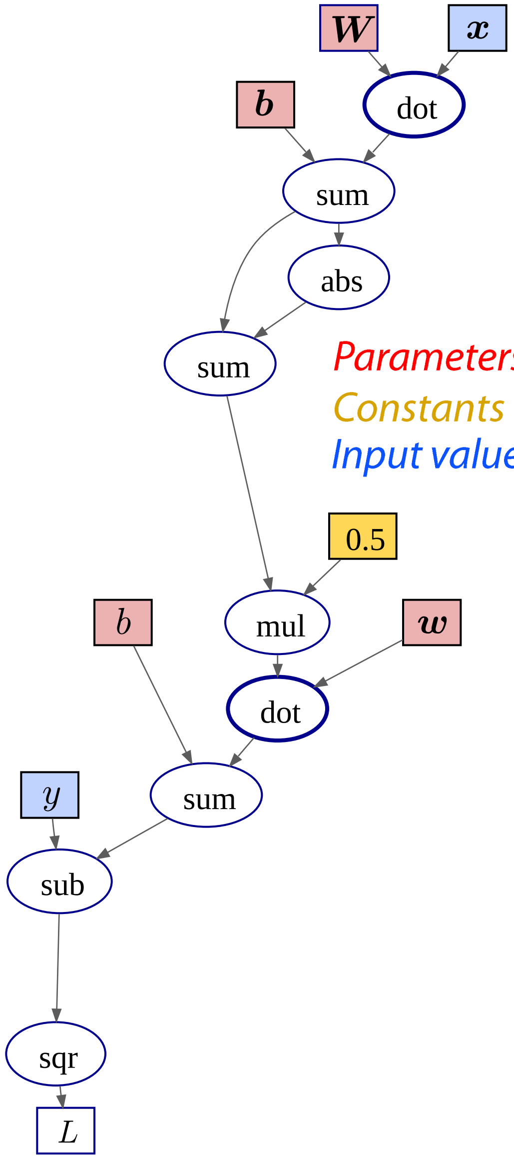
Flow Graph



$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

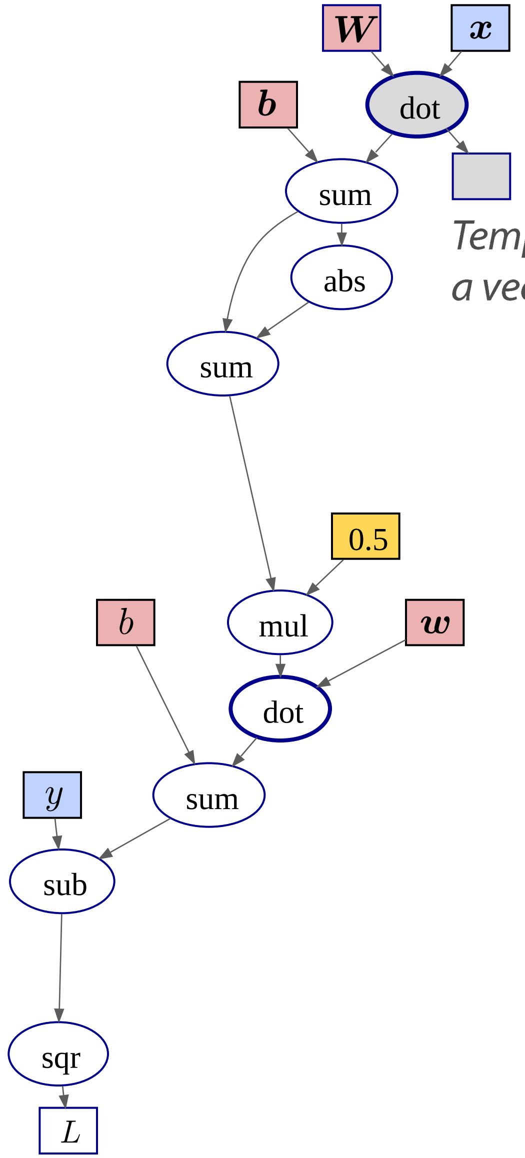


Parameters
Constants
Input values

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



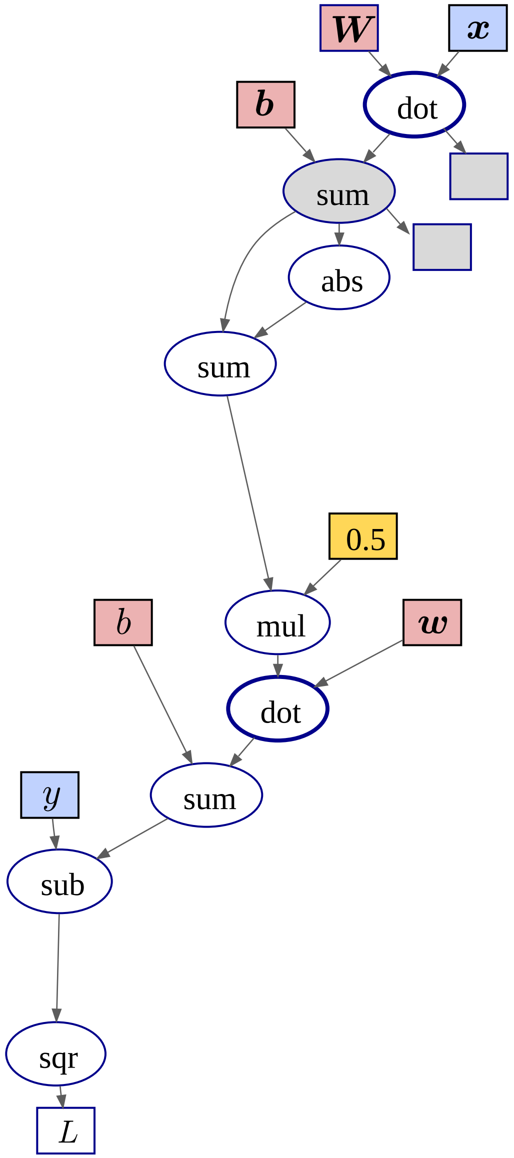
Temporary value: a vector

- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



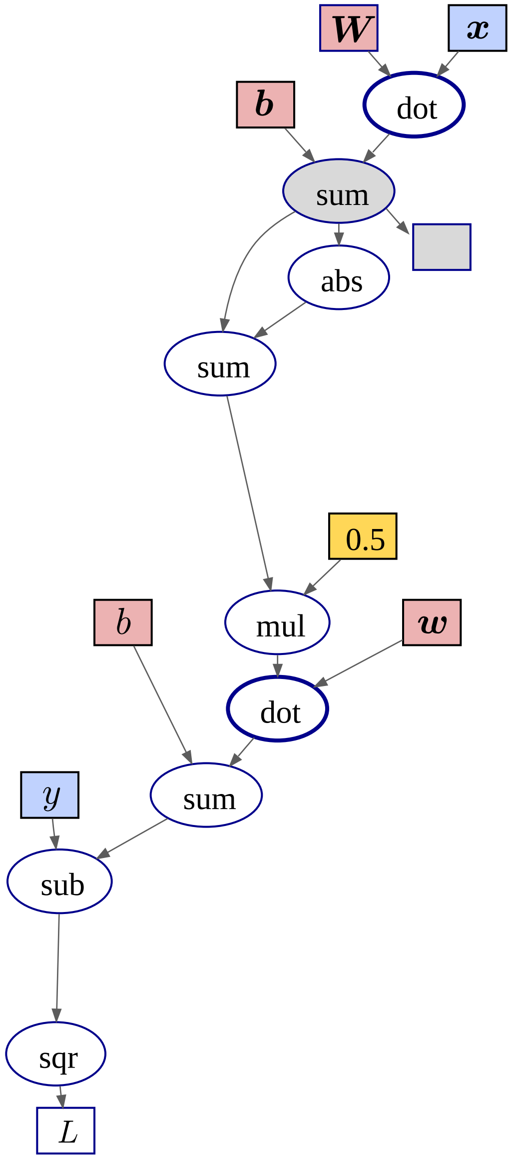
This is no longer necessary

- **Computing the Flow Graph**

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

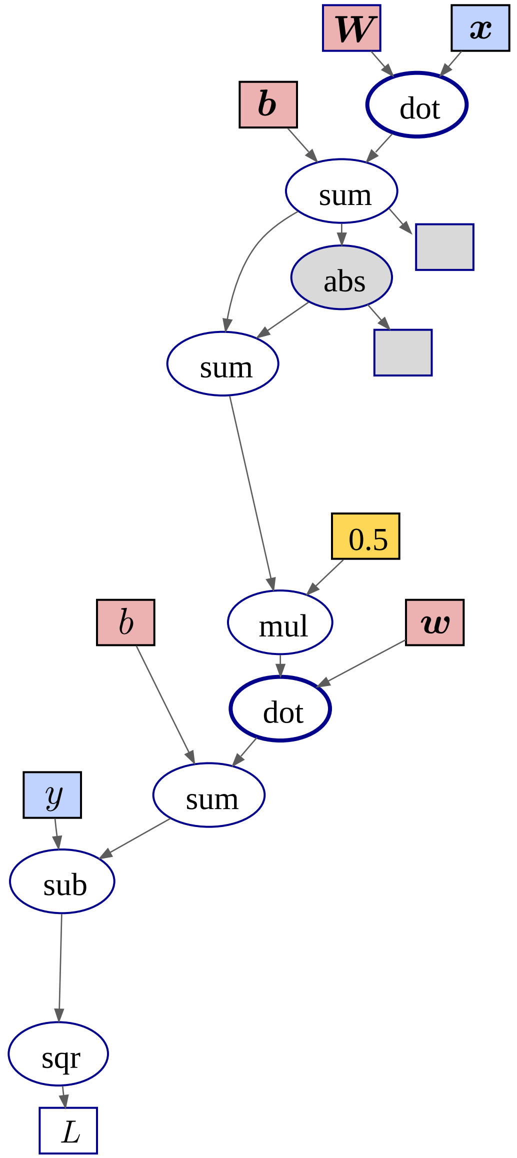


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

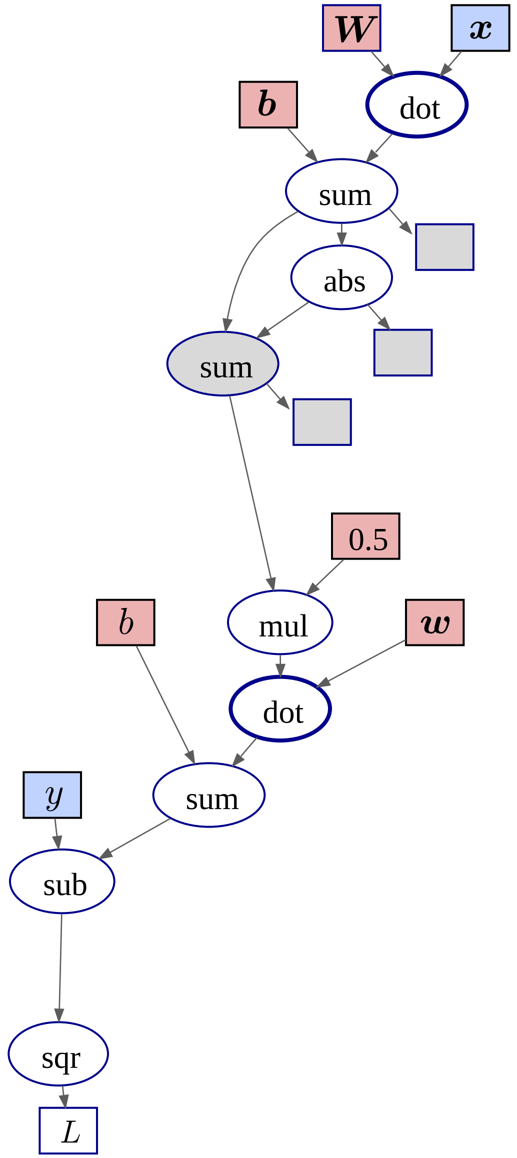


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

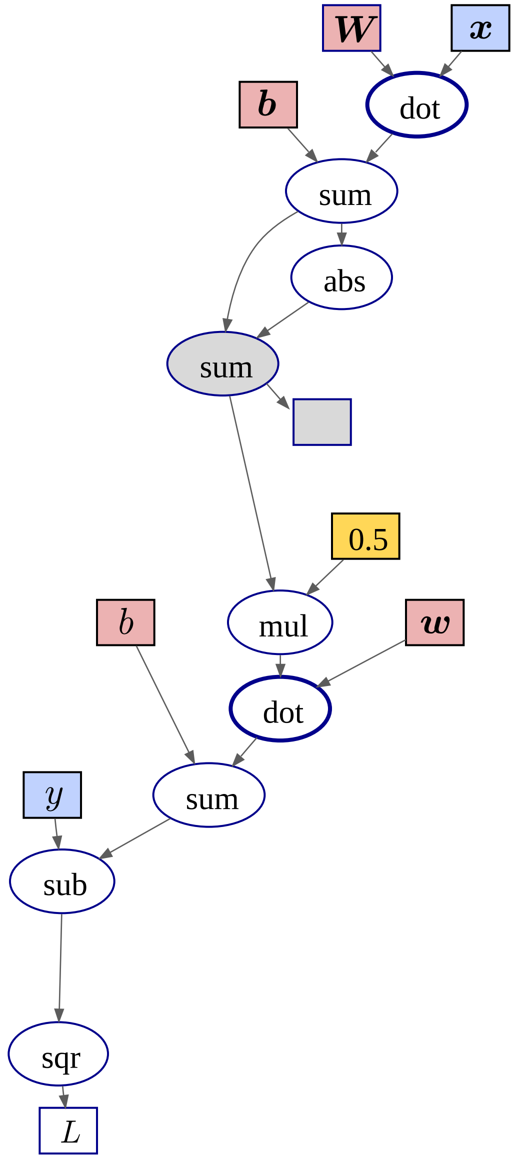


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

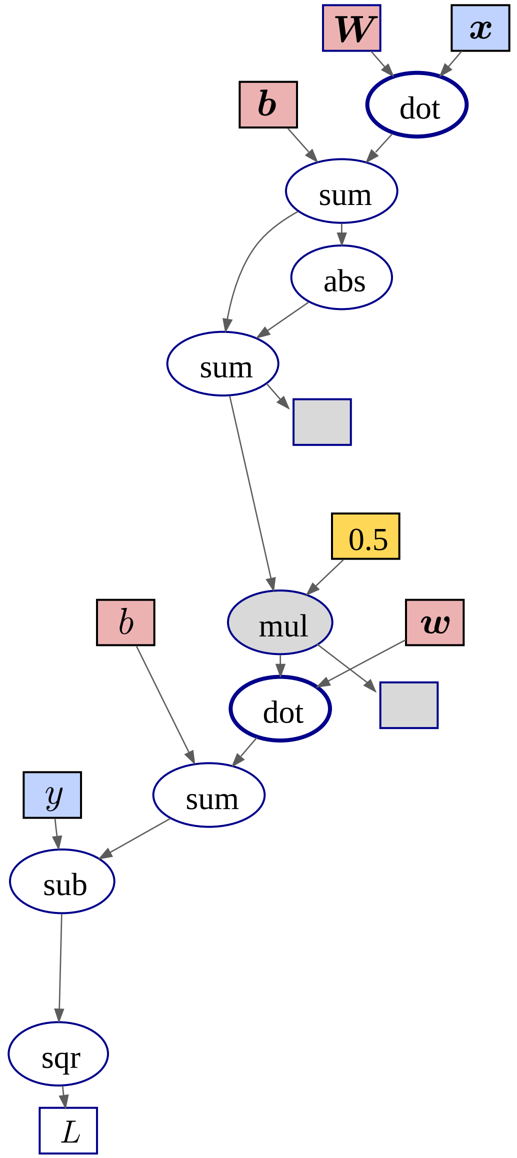


- **Computing the Flow Graph**

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

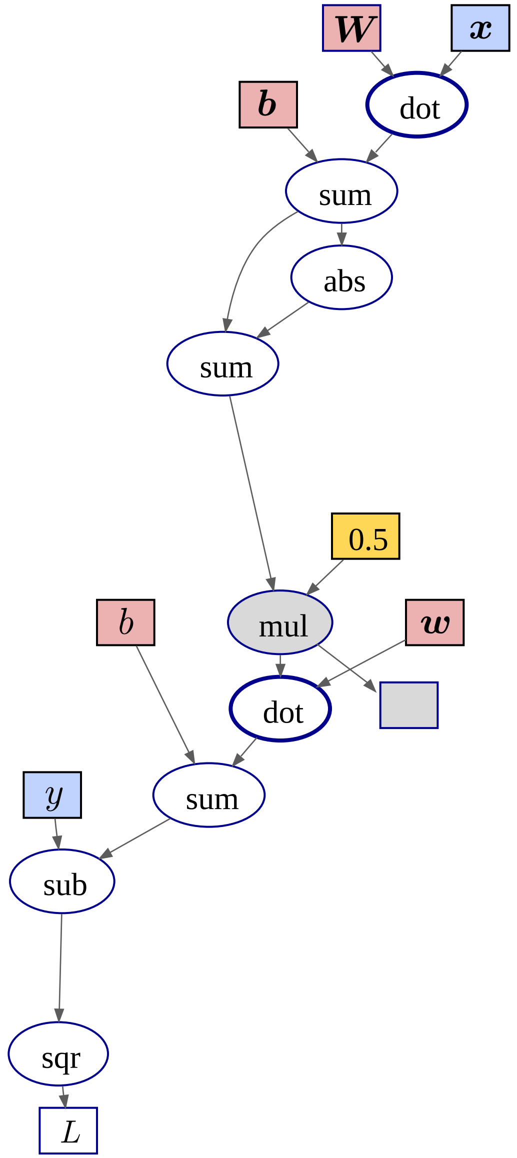


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

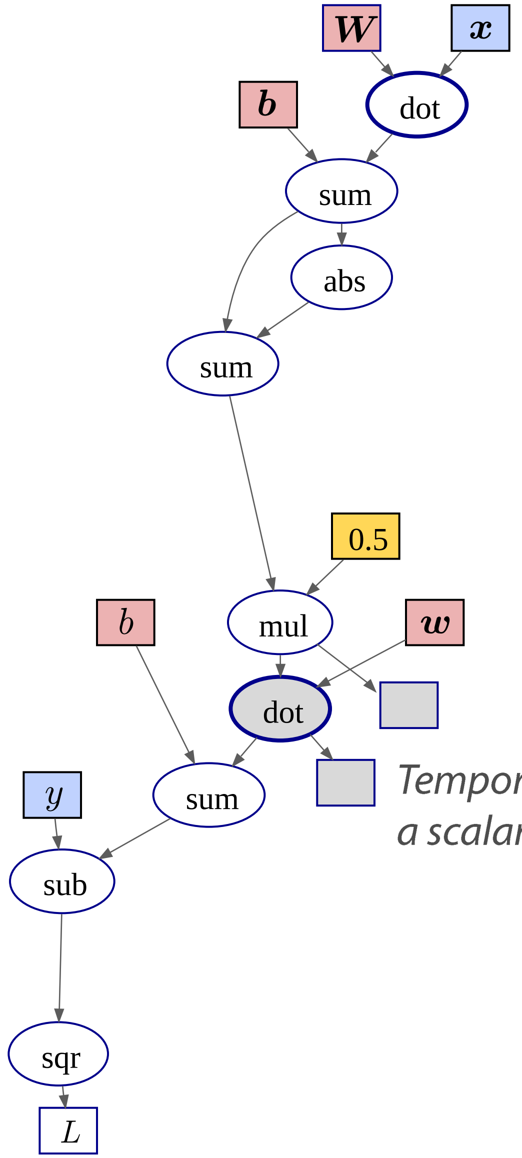


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



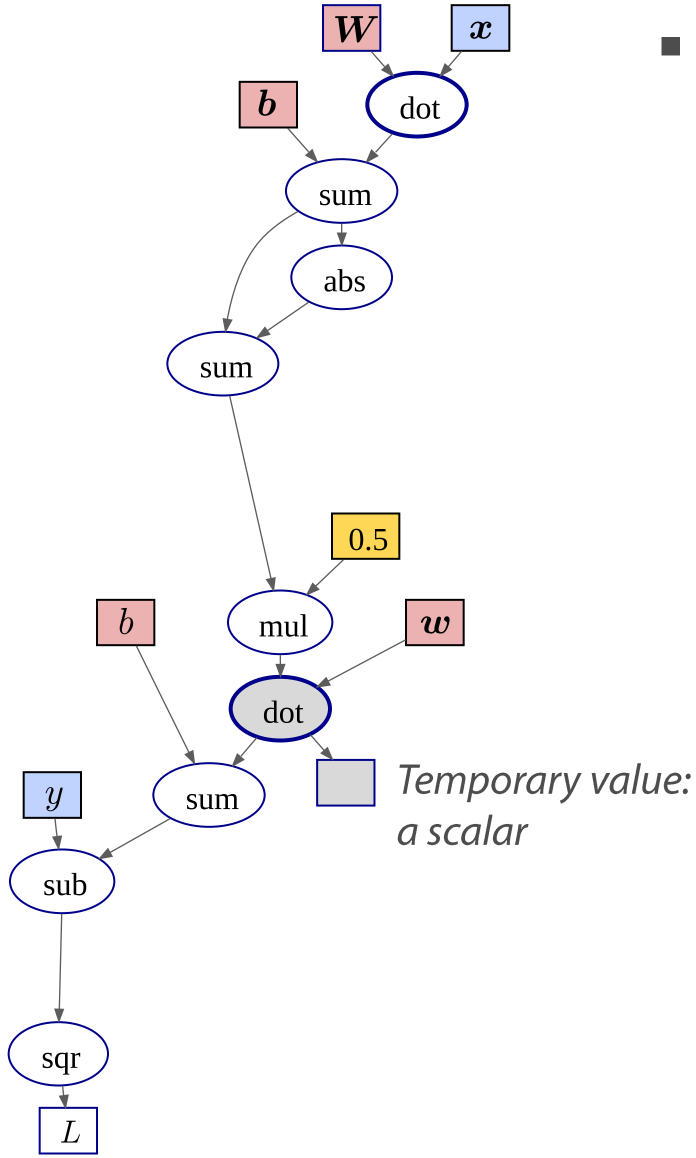
Temporary value: a scalar

- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph

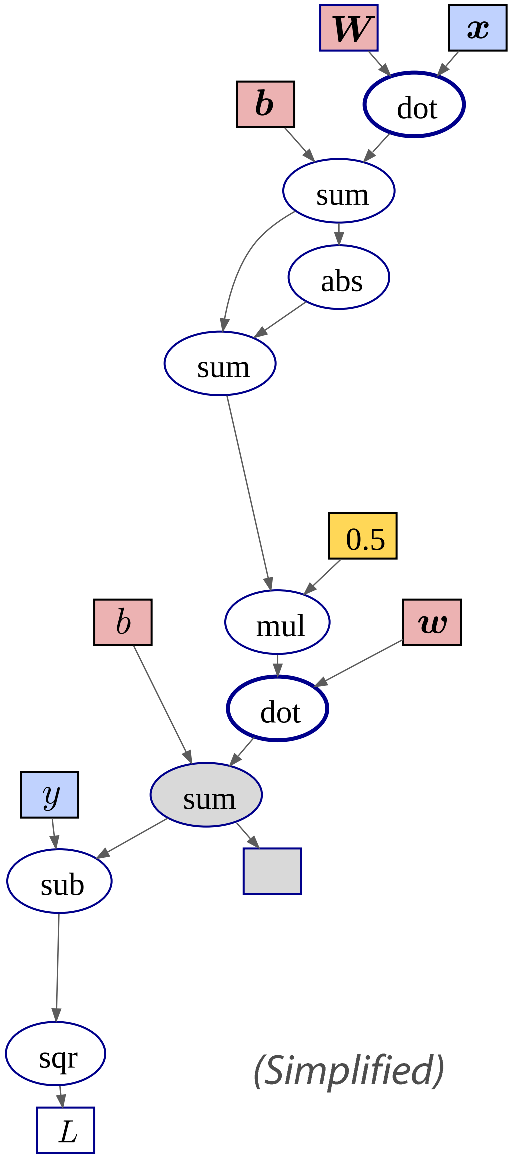


- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



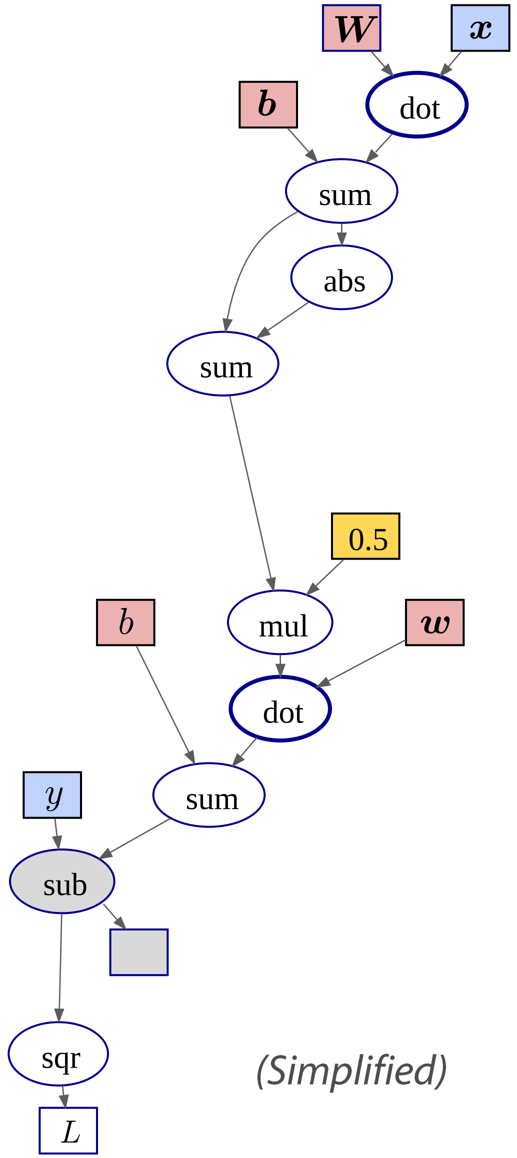
(Simplified)

- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



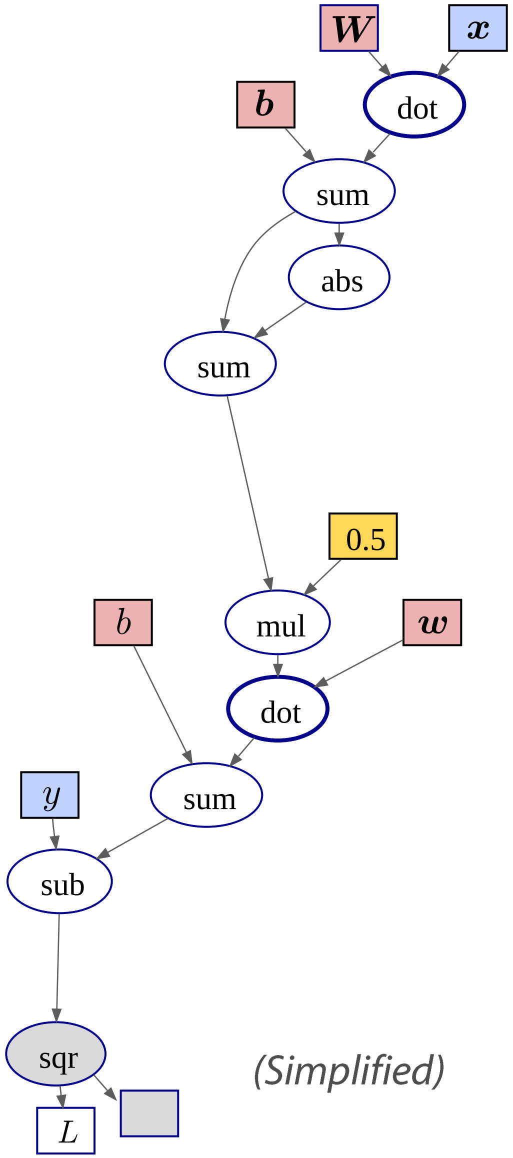
(Simplified)

- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



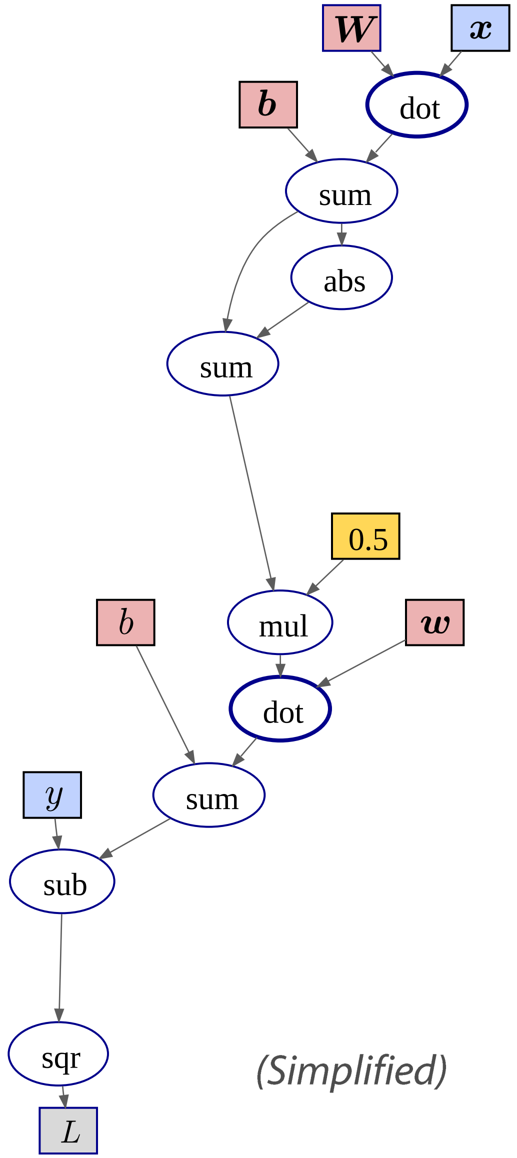
(Simplified)

- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Flow Graph



(Simplified)

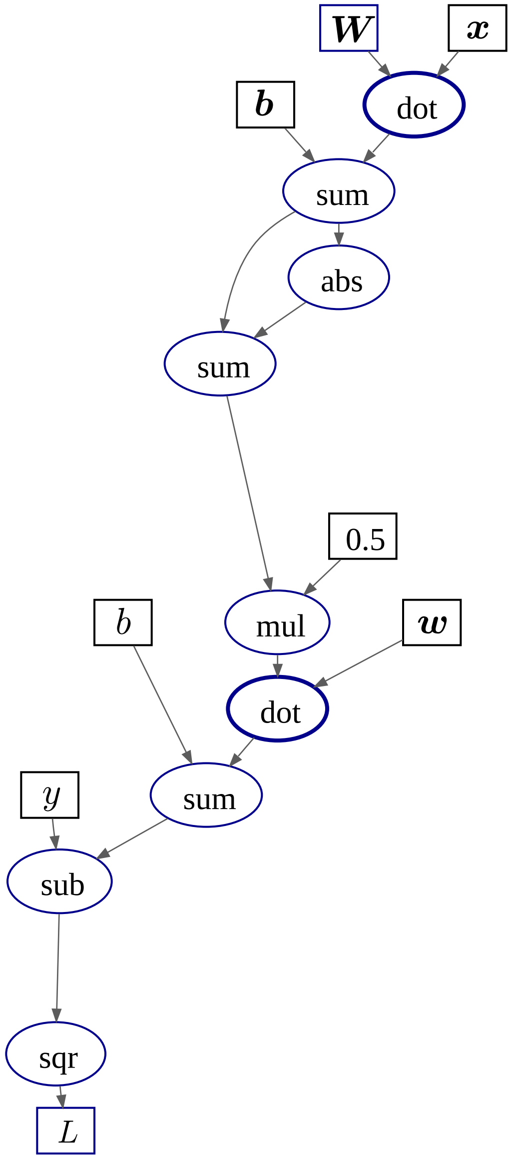
- ***Computing the Flow Graph***

$$L(\tilde{y}, y) = (\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Item-wise loss function, FF neural network with ReLU as non-linearity

Autograd:
Automatic Differentiation of Flow Graphs

Computing Gradients

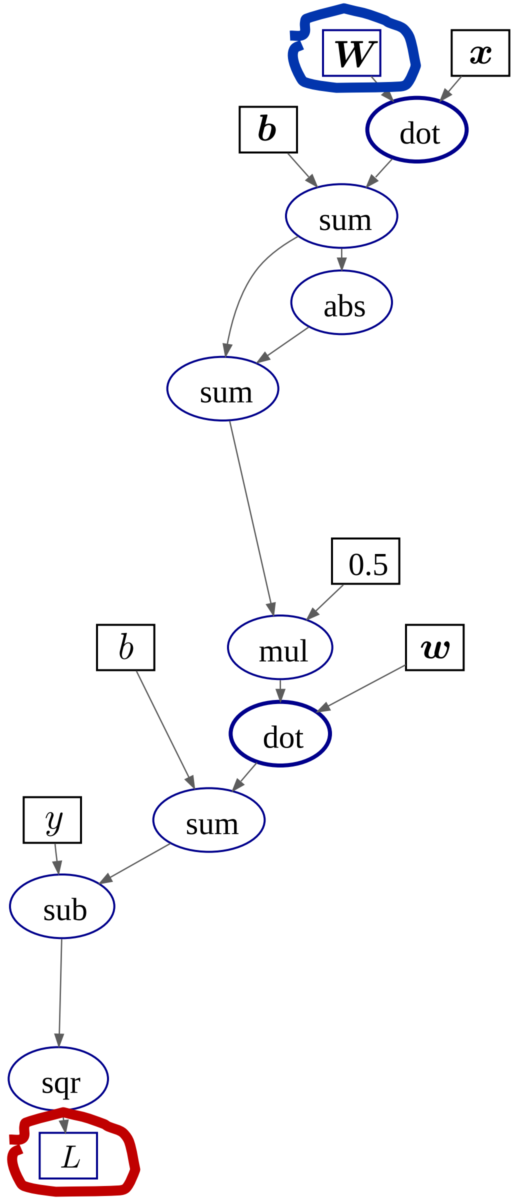


- ***Computing one gradient of the flow graph***

$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

This is the gradient we want to compute
(remember this is just one of the four)

Computing Gradients

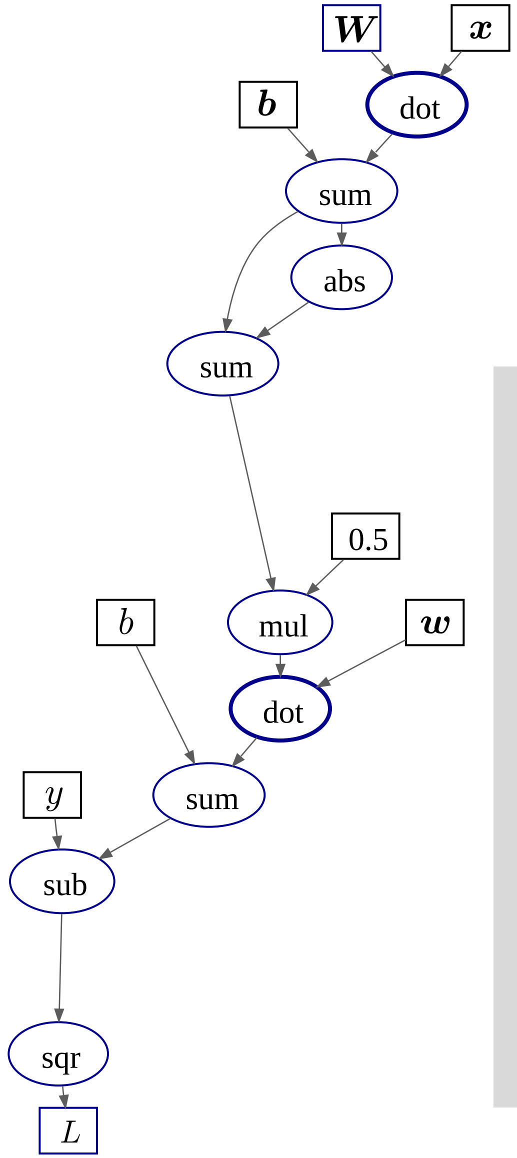


- **Computing one gradient of the flow graph**

$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

This is the gradient we want to compute (remember this is just one of the four)

Computing Gradients



- ***Computing one gradient of the flow graph***

$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

This is the gradient we want to compute
(remember this is just one of the four)

Chain rule for derivatives (*single argument*)

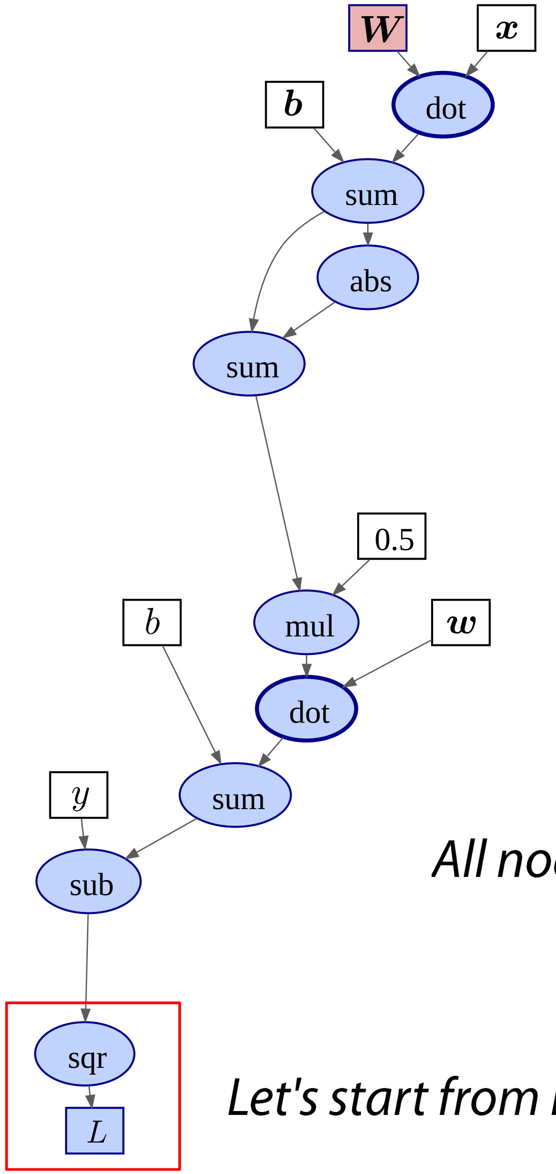
$$\frac{\partial}{\partial \boldsymbol{\vartheta}} f(g(\boldsymbol{\vartheta})) = \frac{\partial}{\partial g(\boldsymbol{\vartheta})} f(g(\boldsymbol{\vartheta})) \frac{\partial}{\partial \boldsymbol{\vartheta}} g(\boldsymbol{\vartheta})$$

Chain rule for derivatives (*multiple arguments*)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} f(g(\boldsymbol{\vartheta}), h(\boldsymbol{\vartheta})) =$$

$$\frac{\partial}{\partial g(\boldsymbol{\vartheta})} f(g(\boldsymbol{\vartheta}), h(\boldsymbol{\vartheta})) \frac{\partial}{\partial \boldsymbol{\vartheta}} g(\boldsymbol{\vartheta}) \; + \; \frac{\partial}{\partial h(\boldsymbol{\vartheta})} f(g(\boldsymbol{\vartheta}), h(\boldsymbol{\vartheta})) \frac{\partial}{\partial \boldsymbol{\vartheta}} h(\boldsymbol{\vartheta})$$

Computing Gradients

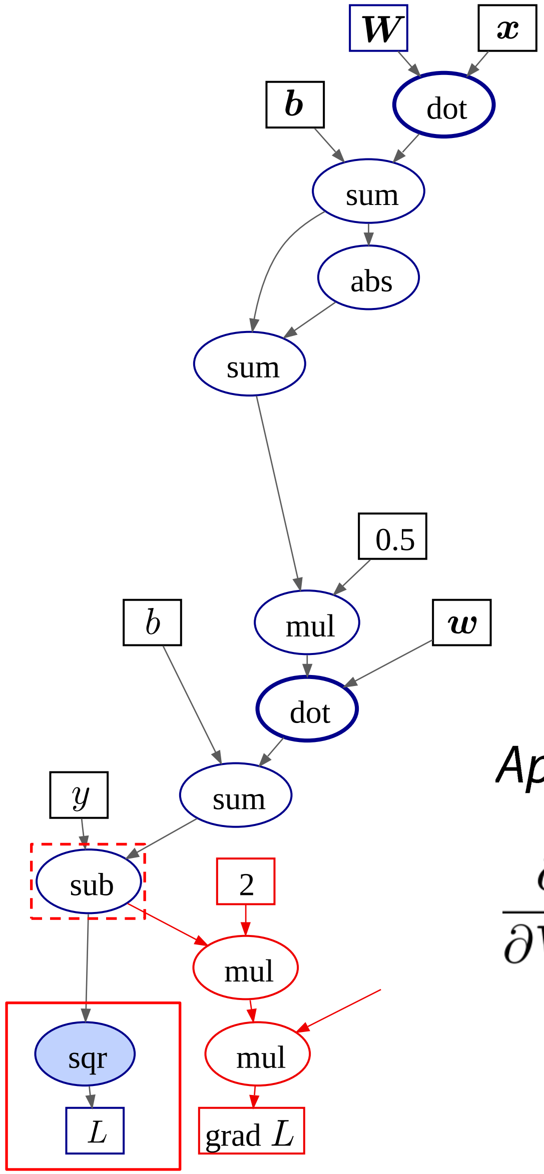


$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

All nodes depending on $\boldsymbol{W}$ *are marked in blue*

Let's start from here (i.e. ***backpropagation****, a.k.a.* ***reverse accumulation****)*

Computing Gradients



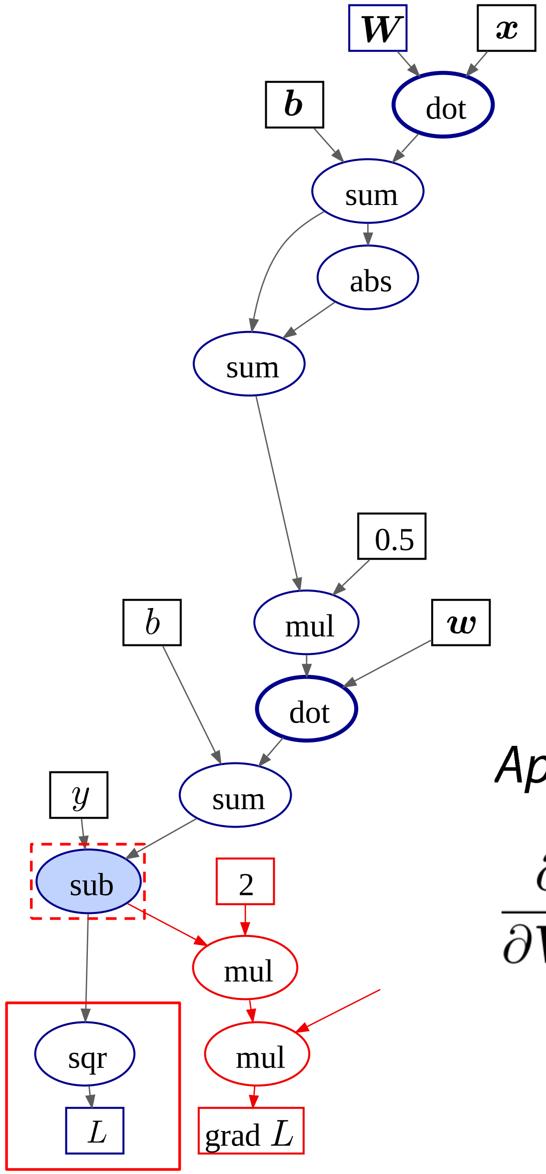
$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Apply the chain rule to the sqr *node*

$$\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})^2 = \frac{\partial}{\partial f(\boldsymbol{W})} f(\boldsymbol{W})^2 \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= 2 \cdot f(\boldsymbol{W}) \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients



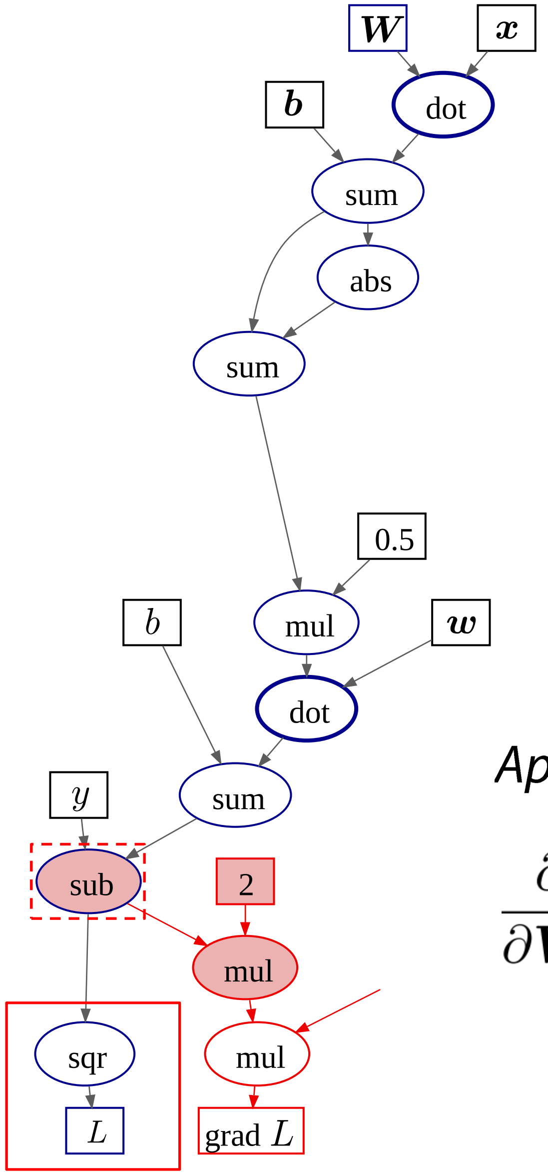
$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Apply the chain rule to the sqr *node*

$$\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})^2 = \frac{\partial}{\partial f(\boldsymbol{W})} f(\boldsymbol{W})^2 \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= 2 \cdot f(\boldsymbol{W}) \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients



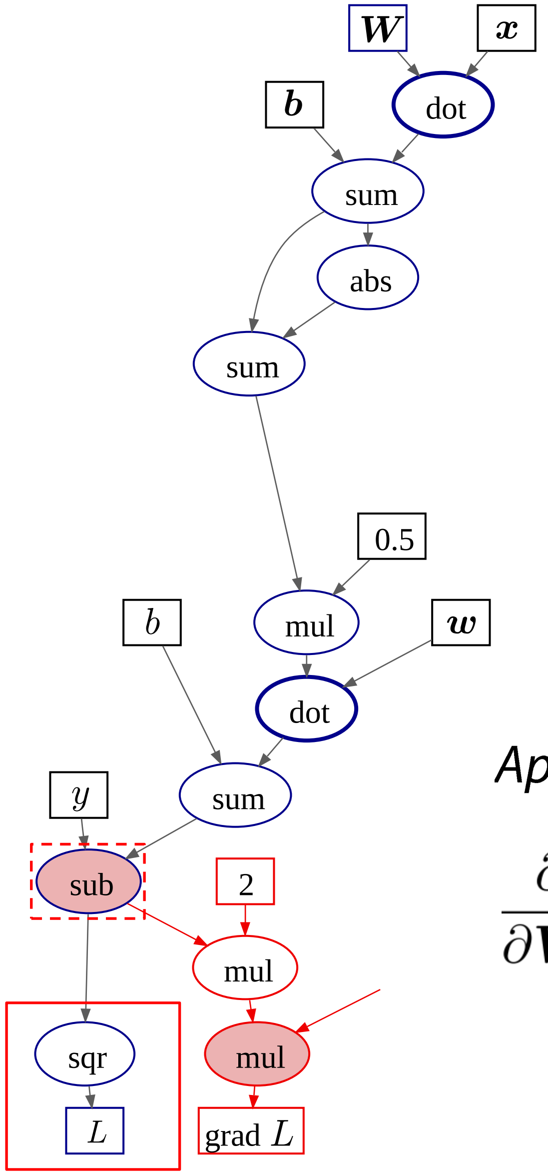
$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Apply the chain rule to the sqr *node*

$$\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})^2 = \frac{\partial}{\partial f(\boldsymbol{W})} f(\boldsymbol{W})^2 \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= 2 \cdot f(\boldsymbol{W}) \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients



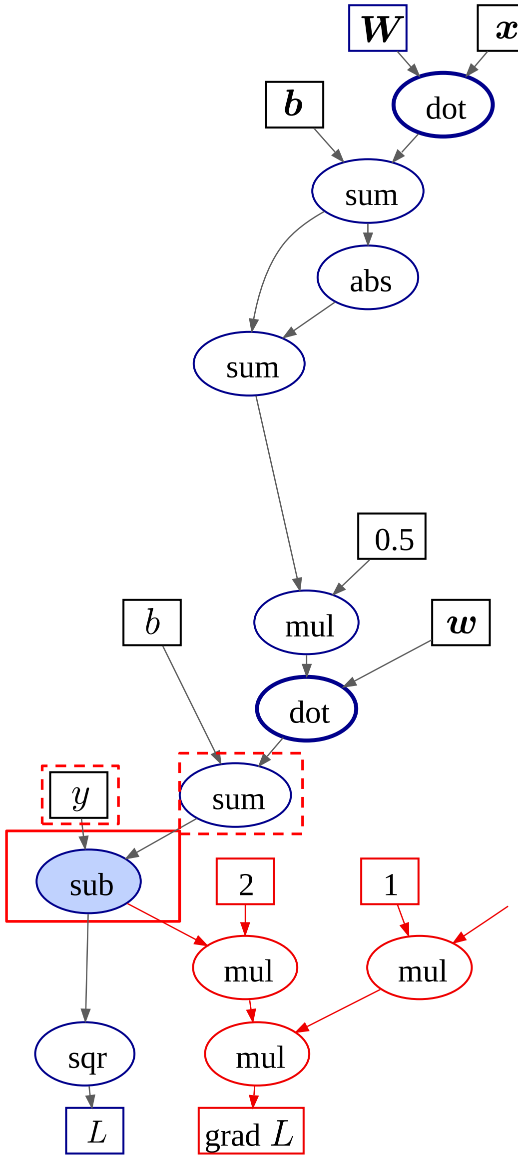
$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Apply the chain rule to the sqr *node*

$$\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})^2 = \frac{\partial}{\partial f(\boldsymbol{W})} f(\boldsymbol{W})^2 \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= 2 \cdot f(\boldsymbol{W}) \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

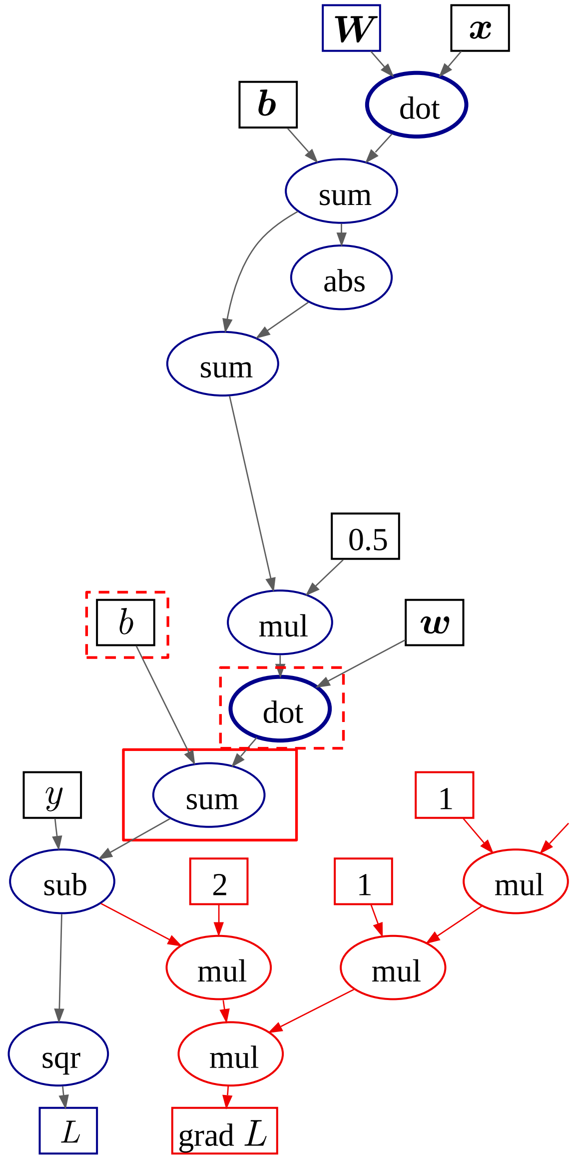
Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

$$\frac{\partial}{\partial \boldsymbol{W}}(f(\boldsymbol{W}) - y) = \frac{\partial}{\partial f(\boldsymbol{W})}(f(\boldsymbol{W}) - y)\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$
$$= 1 \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients

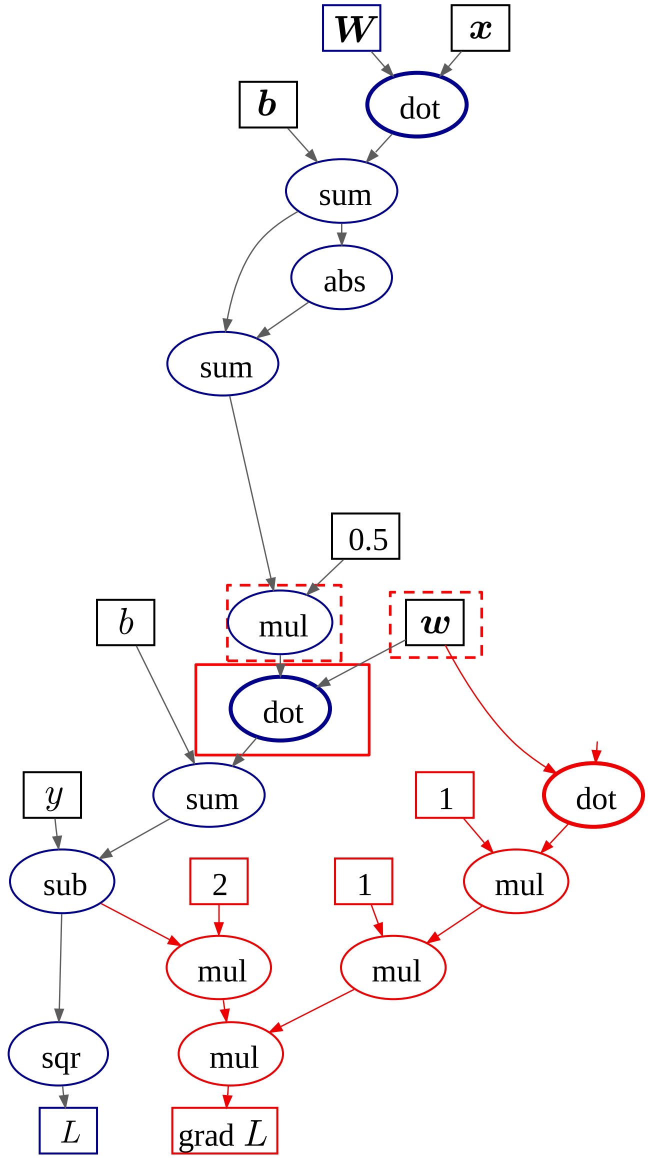


$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

$$\frac{\partial}{\partial \boldsymbol{W}}(f(\boldsymbol{W}) + b) = \frac{\partial}{\partial f(\boldsymbol{W})}(f(\boldsymbol{W}) + b)\frac{\partial}{\partial \boldsymbol{W}}f(\boldsymbol{W})$$

$$= 1 \cdot \frac{\partial}{\partial \boldsymbol{W}}f(\boldsymbol{W})$$

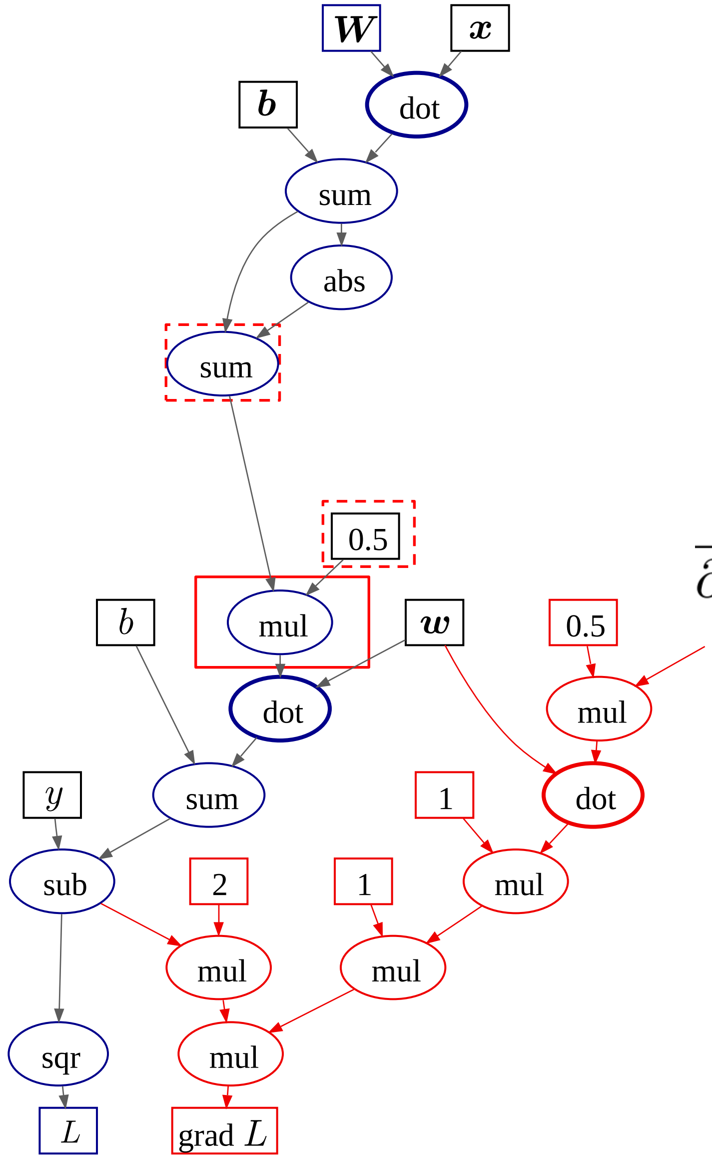
Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot f(\boldsymbol{W})) = \frac{\partial}{\partial f(\boldsymbol{W})}(\boldsymbol{w} \cdot f(\boldsymbol{W}))\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$
$$= \boldsymbol{w} \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients

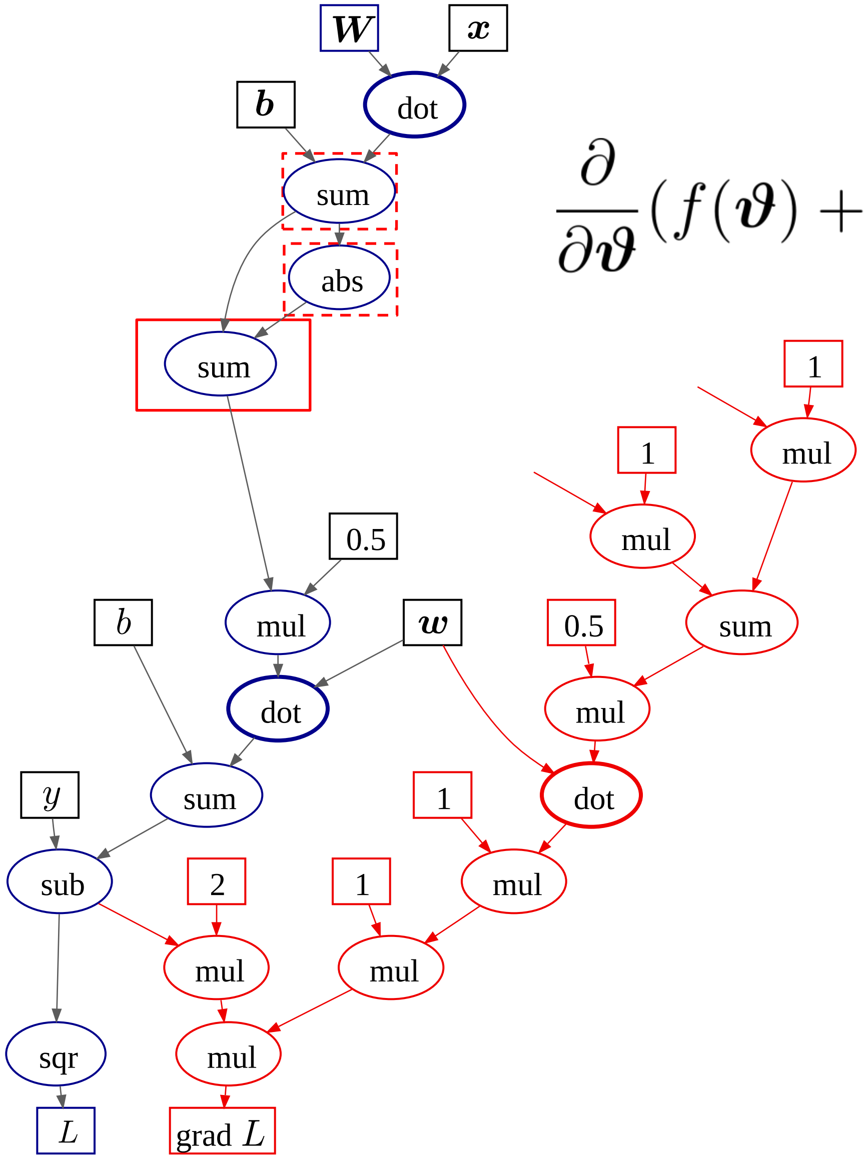


$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

$$\frac{\partial}{\partial \boldsymbol{W}}(0.5 \cdot f(\boldsymbol{W})) = \frac{\partial}{\partial f(\boldsymbol{W})}(0.5 \cdot f(\boldsymbol{W}))\frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= 0.5 \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Computing Gradients

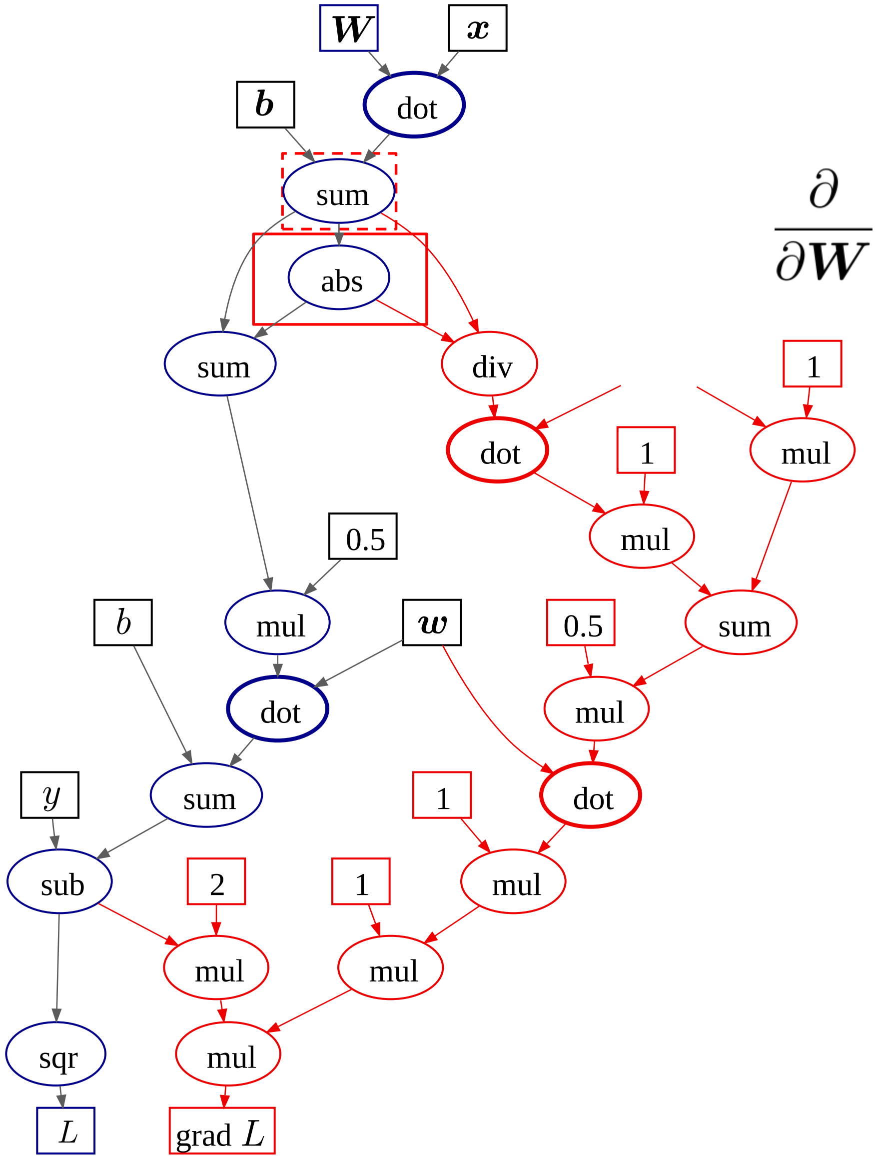


$$\frac{\partial}{\partial \boldsymbol{\vartheta}}(f(\boldsymbol{\vartheta}) + g(\boldsymbol{\vartheta})) = \frac{\partial}{\partial f(\boldsymbol{\vartheta})}(f(\boldsymbol{\vartheta}) + g(\boldsymbol{\vartheta}))\frac{\partial}{\partial \boldsymbol{\vartheta}}f(\boldsymbol{\vartheta})$$

$$+ \frac{\partial}{\partial g(\boldsymbol{\vartheta})}(f(\boldsymbol{\vartheta}) + g(\boldsymbol{\vartheta}))\frac{\partial}{\partial \boldsymbol{\vartheta}}g(\boldsymbol{\vartheta})$$

$$= 1 \cdot \frac{\partial}{\partial \boldsymbol{\vartheta}}f(\boldsymbol{\vartheta}) + 1 \cdot \frac{\partial}{\partial \boldsymbol{\vartheta}}g(\boldsymbol{\vartheta})$$

Computing Gradients



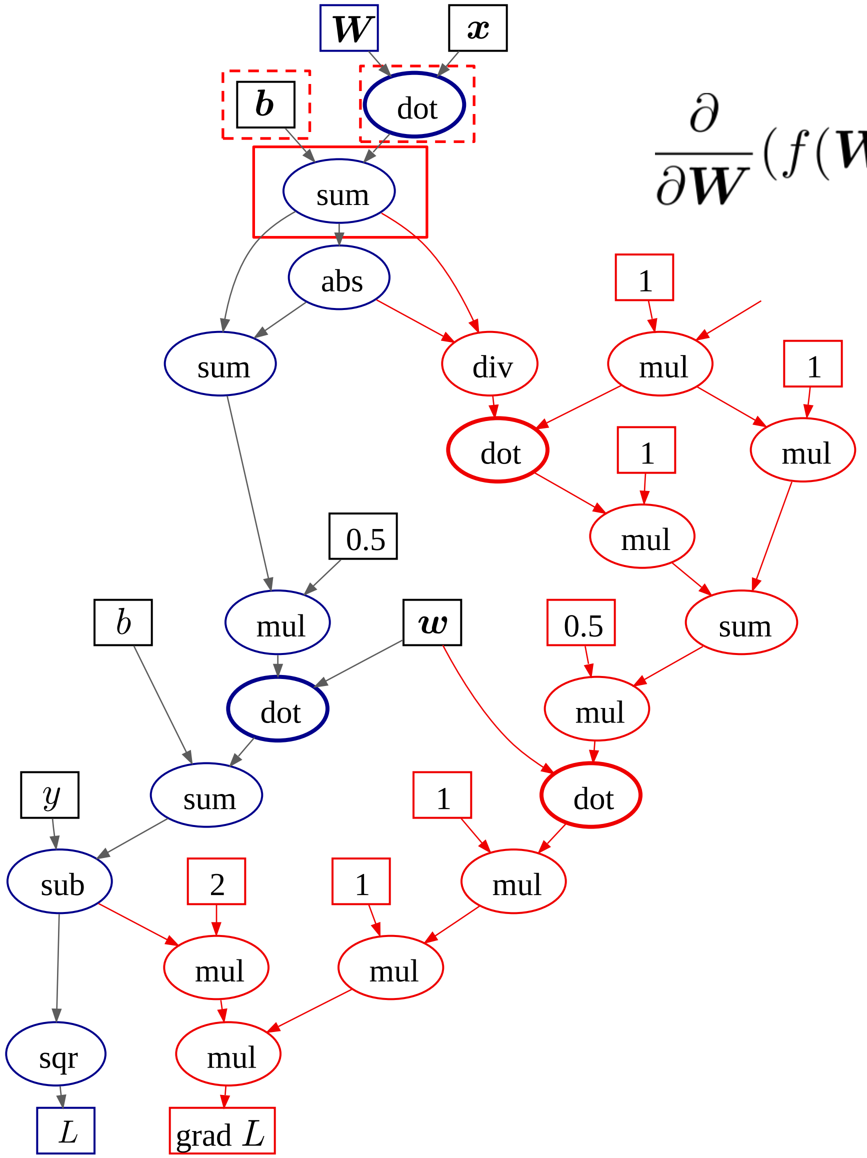
$$\frac{\partial}{\partial \boldsymbol{W}} |f(\boldsymbol{W})| = \frac{\partial}{\partial f(\boldsymbol{W})} |f(\boldsymbol{W})| \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

$$= \frac{f(\boldsymbol{W})}{|f(\boldsymbol{W})|} \cdot \frac{\partial}{\partial \boldsymbol{W}} f(\boldsymbol{W})$$

Clearly, this term is not defined for any $W_{ij} = 0$

(Typically, this is a ***protected*** *division* $\frac{x}{0} := 1$ *)*

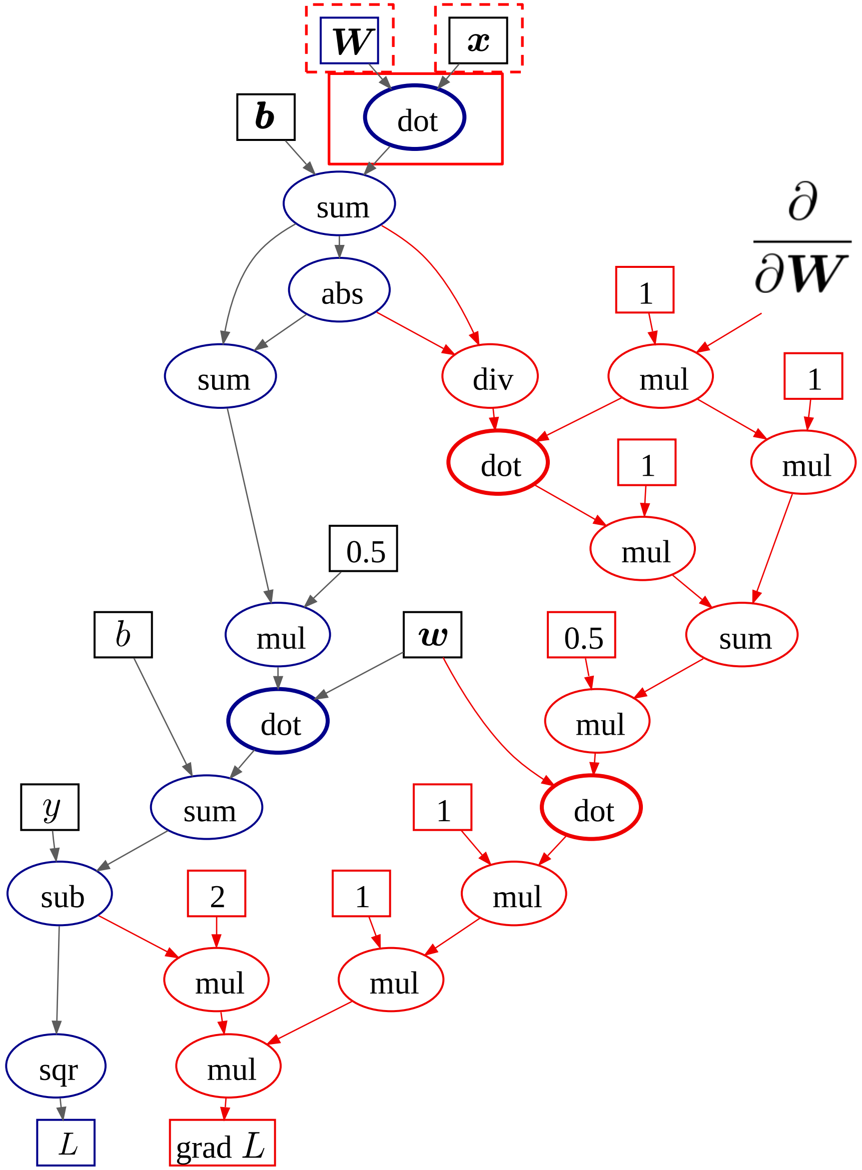
Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(f(\boldsymbol{W})+\boldsymbol{b}) = \frac{\partial}{\partial f(\boldsymbol{W})}(f(\boldsymbol{W})+\boldsymbol{b})\frac{\partial}{\partial \boldsymbol{W}}f(\boldsymbol{W})$$

$$= 1 \cdot \frac{\partial}{\partial \boldsymbol{W}}f(\boldsymbol{W})$$

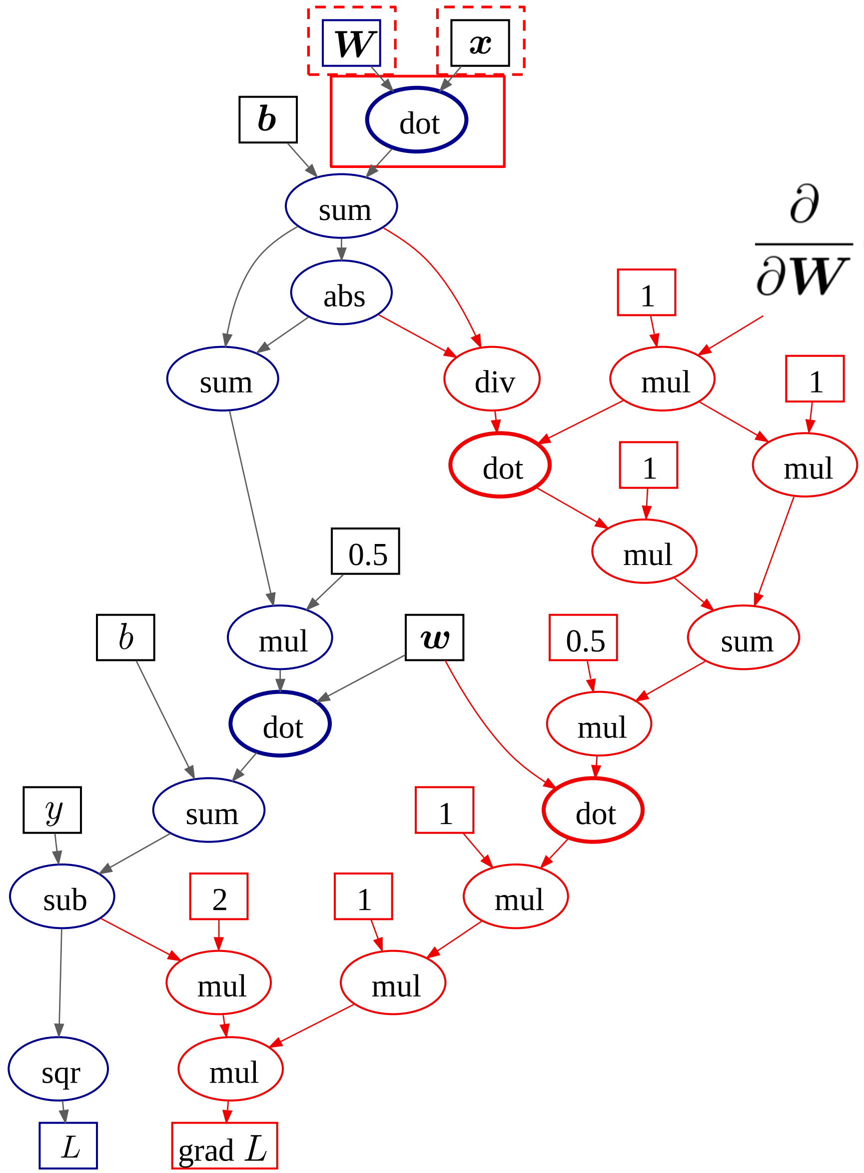
Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})$$

Well, this is tricky….

Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})$$

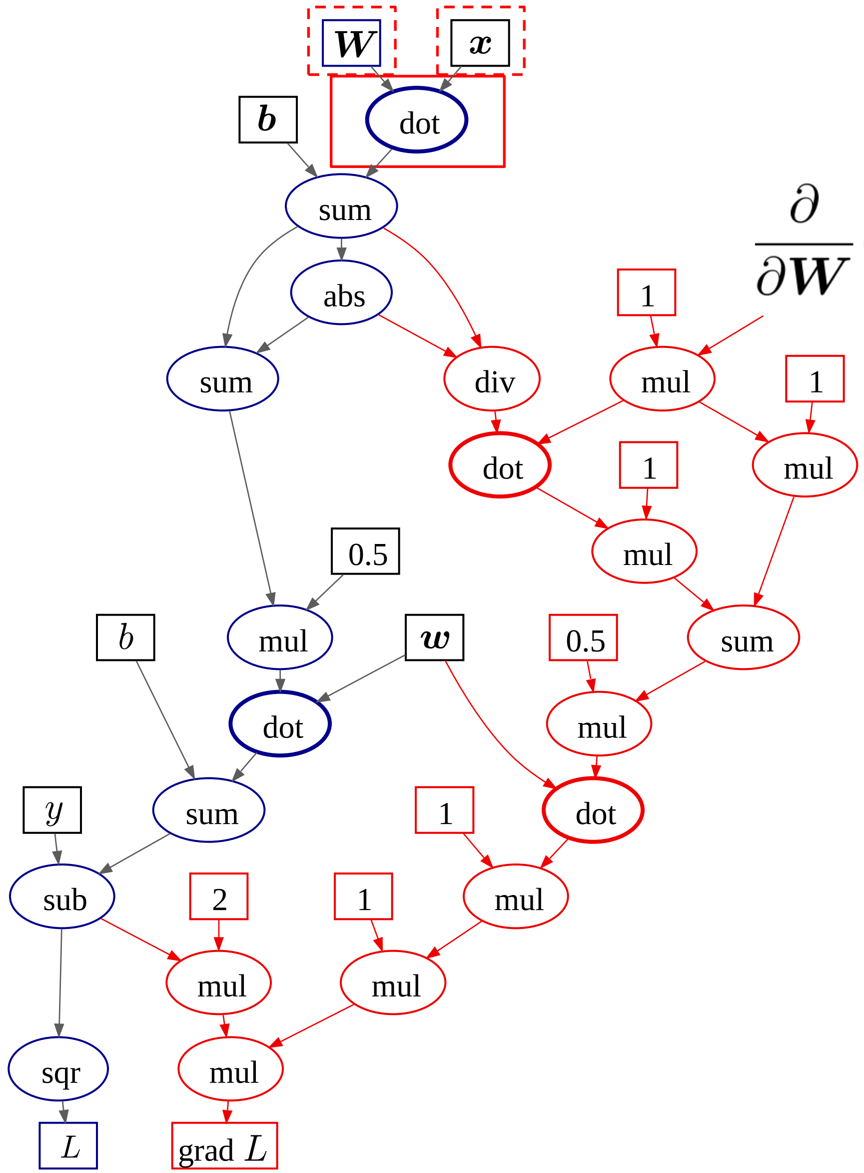
This is a third-order tensor

$$\left(\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})\right)_{ijk} = \frac{\partial}{\partial W_{kj}}(\boldsymbol{W} \cdot \boldsymbol{x})_i$$

Its ijk-th component

Note the inversion of indices

Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})$$

This is a third-order tensor

$$\left(\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})\right)_{ijk} = \frac{\partial}{\partial W_{kj}}(\boldsymbol{W} \cdot \boldsymbol{x})_i$$

Its ijk-th component

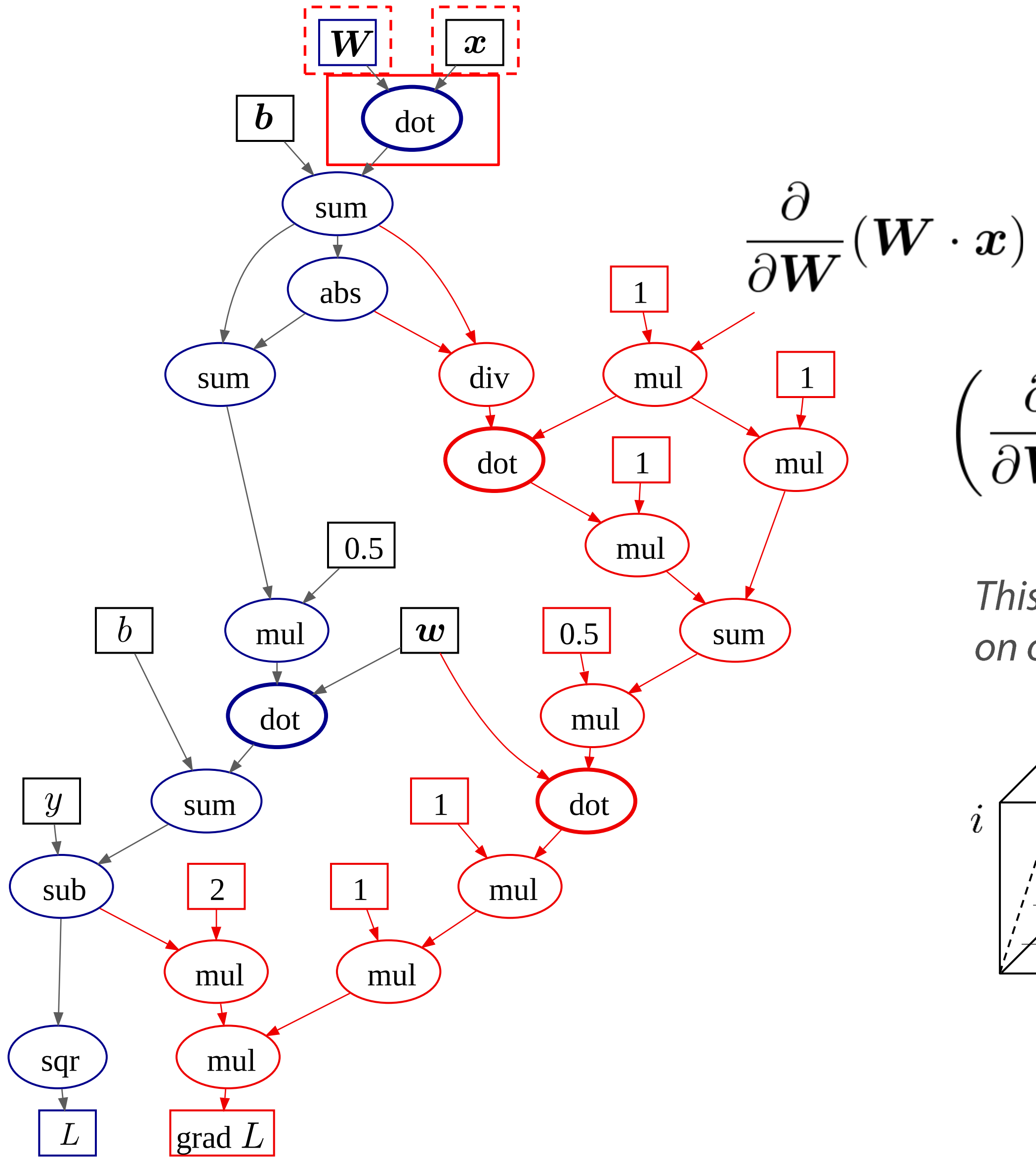
Note the inversion of indices

$$= \frac{\partial}{\partial W_{kj}}(\boldsymbol{W}_{i,:} \cdot \boldsymbol{x})$$

The i-th line in the matrix

$$= \begin{cases} 0 & k \neq i \\ x_j & k = i \end{cases}$$

Computing Gradients

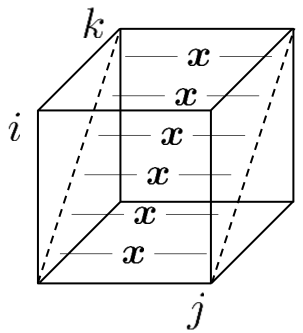


$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})$$

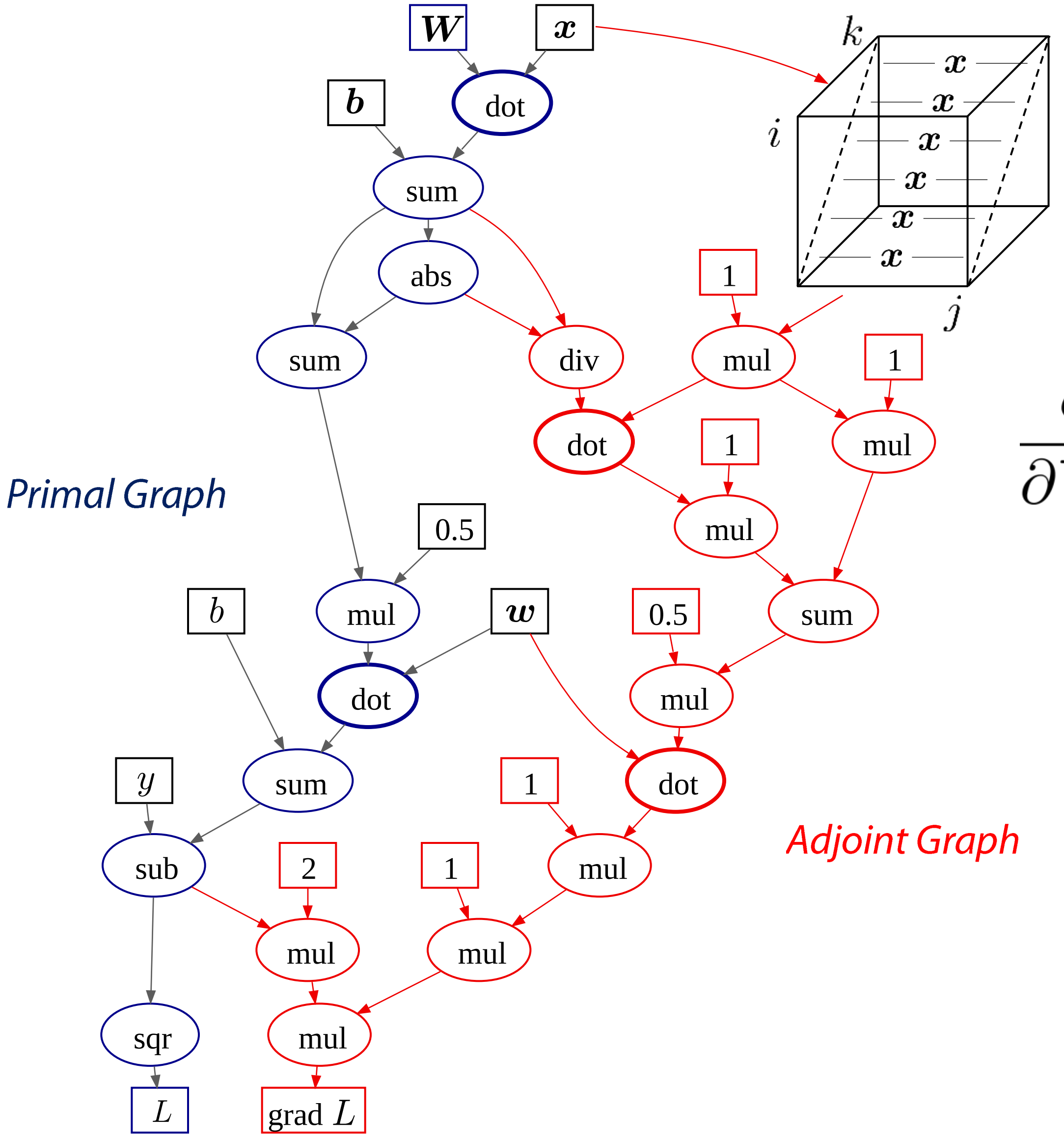
Putting it all together…

$$\left(\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{W} \cdot \boldsymbol{x})\right)_{ijk} = \begin{cases} 0 & k \neq i \\ x_j & k = i \end{cases}$$

This 'thing' (tensor) is a cube having copies of $\boldsymbol{x}$ *on one diagonal 'plane' and zeros elsewhere*

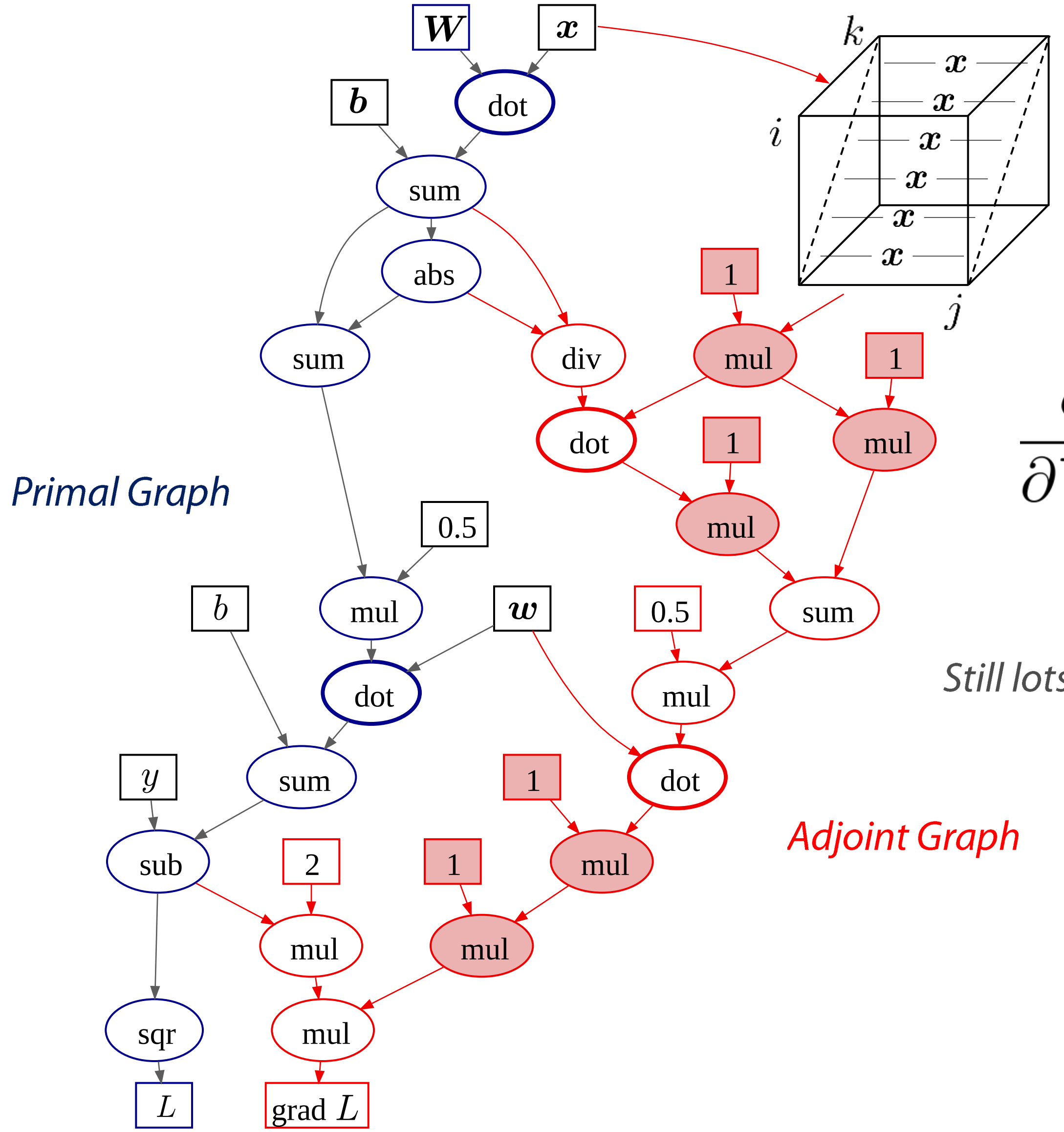


Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Computing Gradients

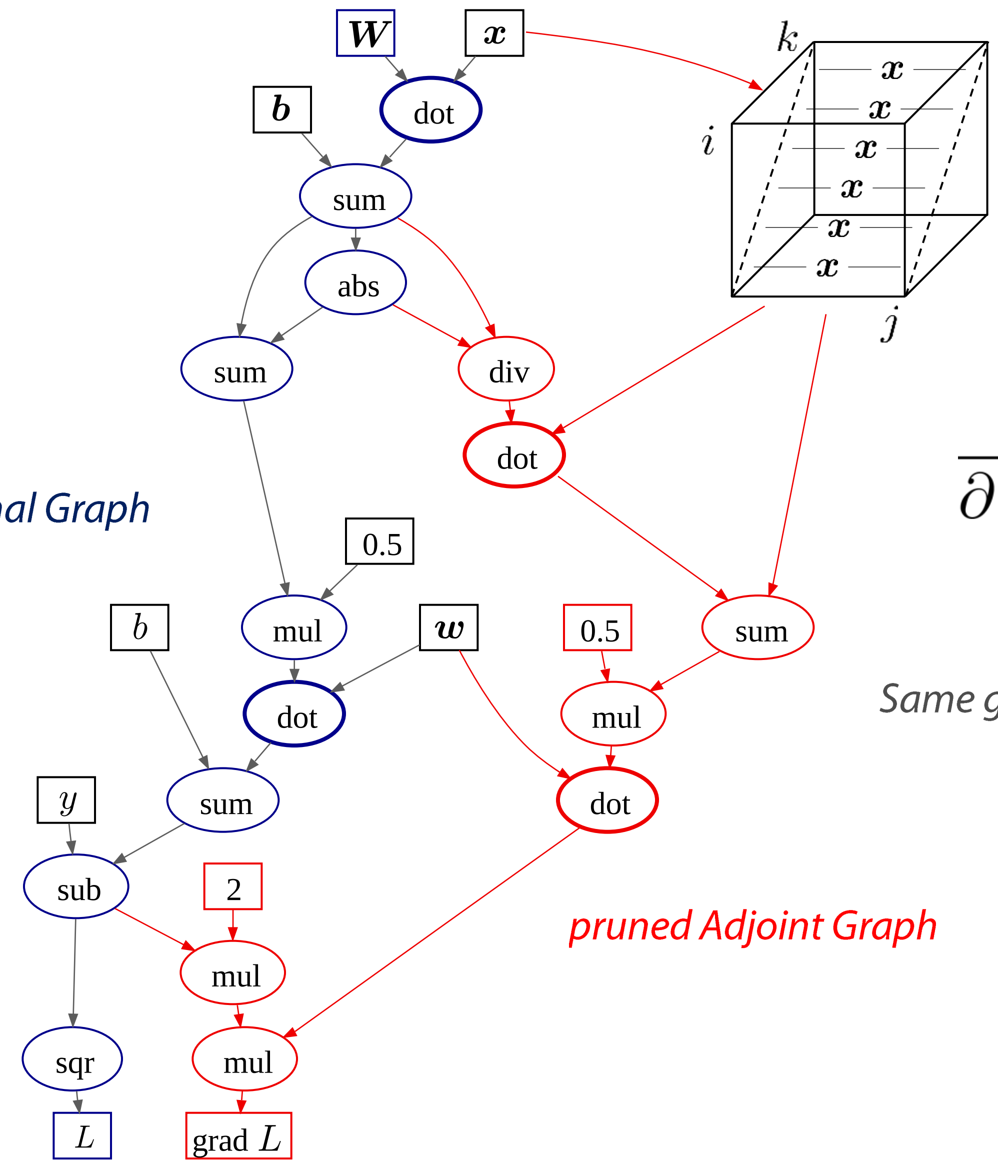


The representation of this can be optimized too

$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Still lots of useless operations

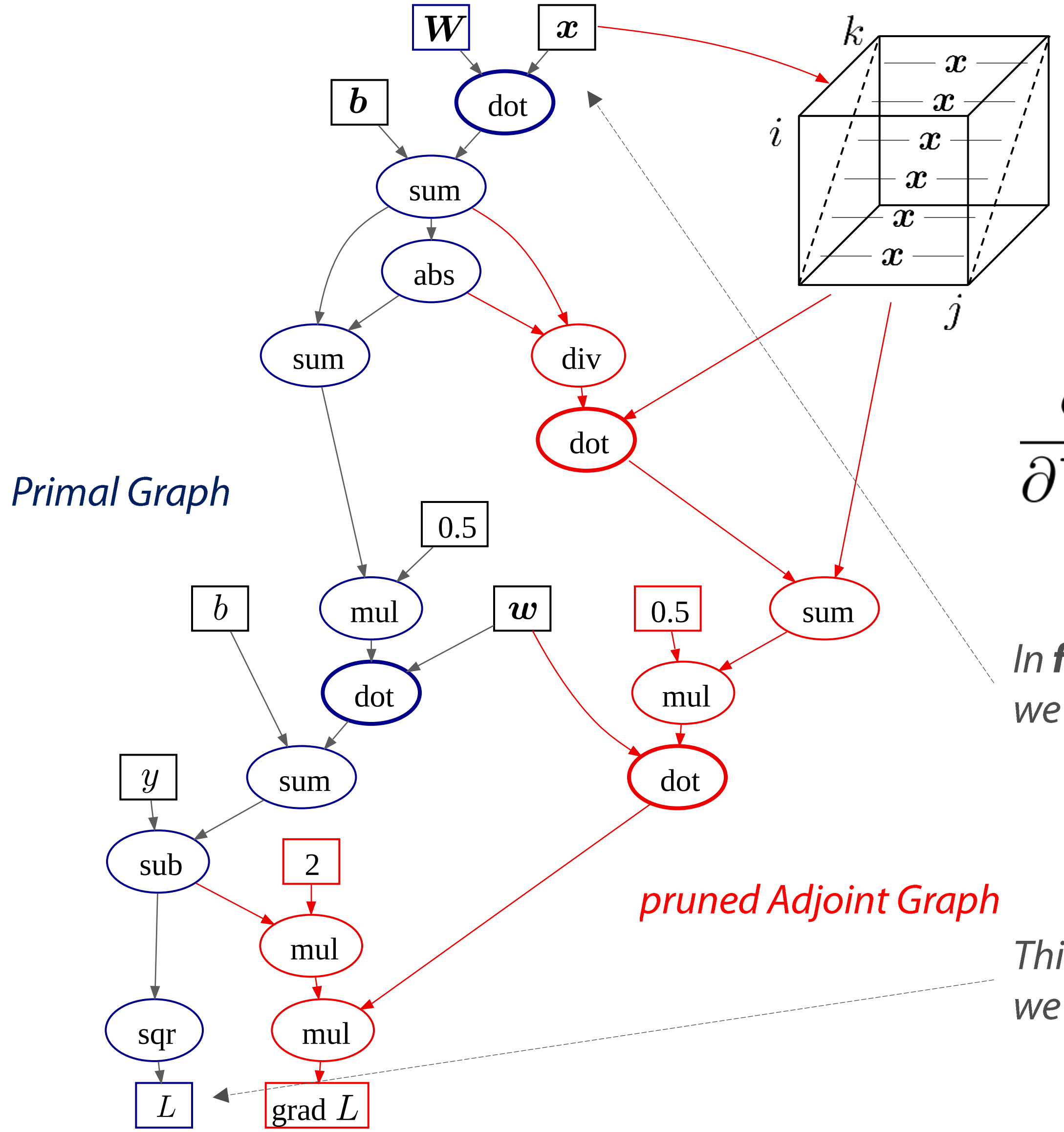
Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

Same graph, after some pruning

Computing Gradients



$$\frac{\partial}{\partial \boldsymbol{W}}(\boldsymbol{w} \cdot \mathrm{ReLU}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b - y)^2$$

*In **forward accumulation** mode we would have started from here*

*This is autograd with **reverse (backward) accumulation:** we started from here and we proceeded in reverse*

(Mini) Batches in Matrix Form

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N} \sum_{D} (\boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

Let's focus first on $\boldsymbol{W}\boldsymbol{x}$

by defining

$$\boldsymbol{X} := \begin{bmatrix} x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_d^{(N)} \end{bmatrix}$$

*input data in matrix form (**item index first**)*

Then we can write

$$\boldsymbol{W}\boldsymbol{X}^T = \begin{bmatrix} | & & | \\ \boldsymbol{W}\boldsymbol{x}^{(1)} & \dots & \boldsymbol{W}\boldsymbol{x}^{(N)} \\ | & & | \end{bmatrix}$$

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N} \sum_{D} (\boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

Consider then $(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

by defining

$$\hat{\boldsymbol{X}} := \begin{bmatrix} x_1^{(1)} & \dots & x_d^{(1)} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(N)} & \dots & x_d^{(N)} & 1 \end{bmatrix} \qquad \hat{\boldsymbol{W}} := \begin{bmatrix} \boldsymbol{W} & \begin{matrix} | \\ \boldsymbol{b} \\ | \end{matrix} \end{bmatrix}$$

Then we could write

$$\hat{\boldsymbol{W}}\hat{\boldsymbol{X}}^T = \begin{bmatrix} | & & | \\ \boldsymbol{W}\boldsymbol{x}^{(1)} + \boldsymbol{b} & \dots & \boldsymbol{W}\boldsymbol{x}^{(N)} + \boldsymbol{b} \\ | & & | \end{bmatrix}$$

Matrix $\hat{\boldsymbol{W}}$ includes two parameters: $\boldsymbol{W}$ and $\boldsymbol{b}$

this may be inconvenient for Autograd, due to the lack of modularity (more to follow)

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N}\sum_{D}(\boldsymbol{w}\cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

Consider then $(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

and let's keep the definition

$$\boldsymbol{X} := \begin{bmatrix} x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_d^{(N)} \end{bmatrix}$$

It could be convenient to redefine the operator $+$ such that is interpreted as

$$\boldsymbol{W}\boldsymbol{X}^T + \boldsymbol{b} := \begin{bmatrix} | & & | \\ \boldsymbol{W}\boldsymbol{x}^{(1)} & \dots & \boldsymbol{W}\boldsymbol{x}^{(N)} \\ | & & | \end{bmatrix} + \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{b} & \dots & \boldsymbol{b} \\ | & & | \end{bmatrix}}_{N\ \text{times}}$$

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N}\sum_{D}\left(\boldsymbol{w}\cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)}+\boldsymbol{b})+b-y^{(i)}\right)^2$$

Consider then $(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$

and let's keep the definition

$$\boldsymbol{X} := \begin{bmatrix} x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \dots & x_d^{(N)} \end{bmatrix}$$

It could be convenient to redefine the operator $+$ such that is interpreted as

$$\boldsymbol{W}\boldsymbol{X}^T + \boldsymbol{b} := \begin{bmatrix} | & & | \\ \boldsymbol{W}\boldsymbol{x}^{(1)} & \dots & \boldsymbol{W}\boldsymbol{x}^{(N)} \\ | & & | \end{bmatrix} + \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{b} & \dots & \boldsymbol{b} \\ | & & | \end{bmatrix}}_{N \text{ times}}$$

*This is called **broadcasting***

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N}\sum_{D}(\boldsymbol{w}\cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)}+\boldsymbol{b})+b-y^{(i)})^2$$

Using broadcasting, we would express the above as

$$L(D) = \frac{1}{N}(\boldsymbol{w}\cdot g(\boldsymbol{W}\boldsymbol{X}^T+\boldsymbol{b})+b-\boldsymbol{y})^2$$

But it does NOT work

*Matrix $\boldsymbol{W}\boldsymbol{X}^T \in \mathbb{R}^{h\times N}$ and vector $\boldsymbol{b}\in\mathbb{R}^h$ are not aligned (for **broadcasting**, the operands' **shapes** must be **right-aligned**)*

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N} \sum_{D} (\boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

Using broadcasting, we would express the above as

$$L(D) = \frac{1}{N} (\boldsymbol{w} \cdot g(\boldsymbol{X}\boldsymbol{W}^T + \boldsymbol{b}) + b - \boldsymbol{y})^2$$

But it does NOT work yet

Now matrix $\boldsymbol{W}\boldsymbol{X}^T \in \mathbb{R}^{N \times h}$ and vector $\boldsymbol{b} \in \mathbb{R}^h$ are right-aligned in shape
Moreover, the resulting matrix is data item index first

Vector $\boldsymbol{w} \in \mathbb{R}^h$ cannot be left-multiplied with a matrix in $\mathbb{R}^{N \times h}$

More on Matrix Forms

Say it with matrices…

We may want to get rid of the summation when computing the loss function

$$L(D) = \frac{1}{N}\sum_{D}(\boldsymbol{w}\cdot g(\boldsymbol{W}\boldsymbol{x}^{(i)} + \boldsymbol{b}) + b - y^{(i)})^2$$

Using broadcasting, we can express the above as

$$L(D) = \frac{1}{N}((g(\boldsymbol{X}\boldsymbol{W}^T + \boldsymbol{b})\boldsymbol{w} + b) - \boldsymbol{y})^2$$

The result is a vector in $\mathbb{R}^N$

Broadcasting applies

This is a vector in $\mathbb{R}^N$

A similar behavior of operators is standard in NumPy, TensorFlow, PyTorch, JAX