EE5904/ME5404: Neural Networks Lecture 04

Xingyu Liu
xyl@nus.edu.sg

Assistant Professor
Department of Electrical & Computer Engineering
National University of Singapore

Reminder: Assignments 1 & 2

Due 23:59 (SGT), Sunday, 15 February 2026.
Late submission will not be accepted unless it is well justified!

Submission instructions

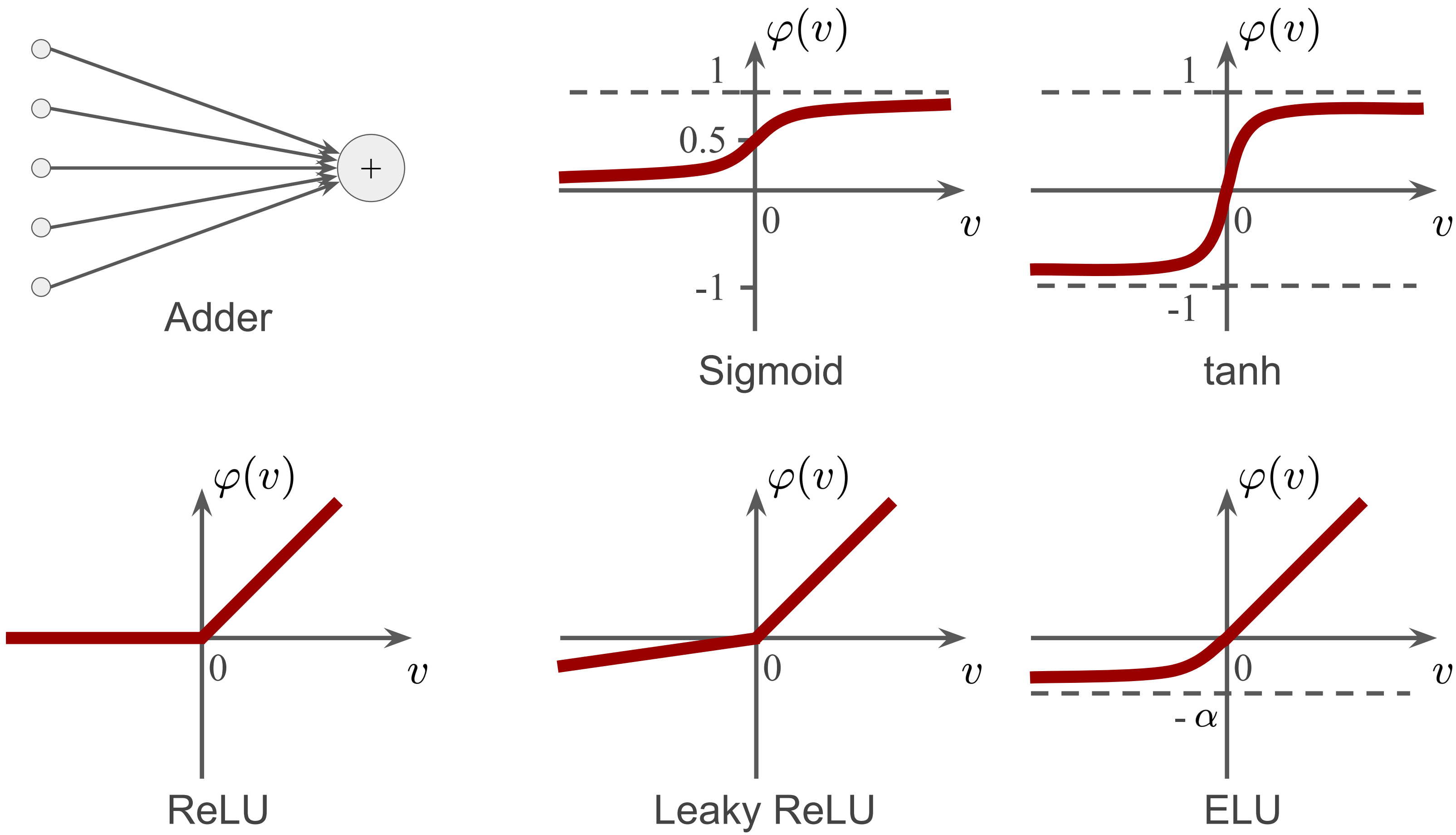
- Submit the assignment via Canvas.
- Any **Python code** and **Python code generated results** should be included as an attachment.

Handwritten submissions are encouraged!

- If all questions (except the **Python code** and **code generated results such as figures**) are handwritten, you will receive a **10% bonus** on the assignment score.
- For handwritten work, **take clear photos of the pages** and upload them to Canvas.
- Ensure that your handwriting and photos are **clear and legible**. Illegible submissions may lose marks。

Assignment 2 will be out shortly!

Recap of last lecture: activation functions

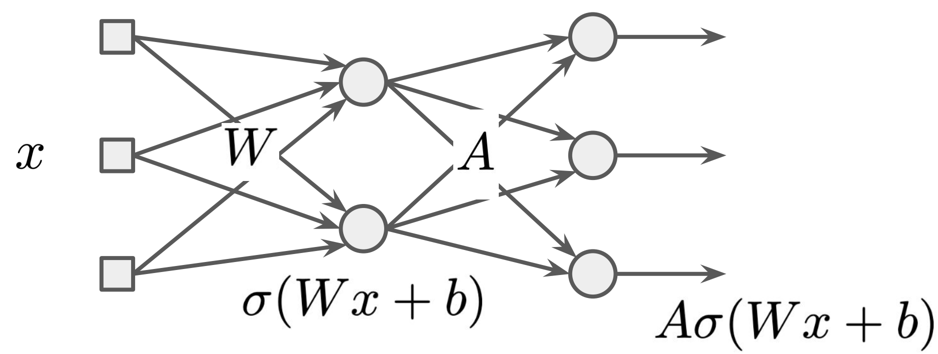


Recap of last lecture: Universal Approximation Theorem (UAT)

Theorem

For any compact set $K \subset \mathbb{R}^m$, any continuous function $f : K \to \mathbb{R}^n$ and any $\varepsilon > 0$, there exist $N, W \in \mathbb{R}^{N \times m}, b \in \mathbb{R}^N, A \in \mathbb{R}^{n \times N}$, such that

$$\sup_{x \in K} \|f(x) - A\sigma(Wx + b)\| < \varepsilon$$



It means ***a <u>two-layer</u> neural network with <u>sufficient number of hidden layer neurons</u> can approximate any continuous vector-valued function defined on a <u>bounded</u> high-dimensional region***.

Recap of last lecture: Frequently used loss functions

1. L2 loss (squared error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$$

2. L1 loss (absolute error) for regression:

$$\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_1$$

3. Logistic loss (binary cross-entropy loss) for binary classification:

$$\mathcal{L}(f_\theta(x), y) = -y \log \sigma(f_\theta(x)) - (1 - y) \log(1 - \sigma(f_\theta(x)))$$

4. Softmax cross-entropy loss for multi-class classification:

$$\mathcal{L}(f_\theta(x), y) = -\sum_{k=1}^{K} y_k \log\left(\frac{e^{f_k(x)}}{\sum_{j=1}^{K} e^{f_j(x)}}\right)$$

where $y_k = \begin{cases} 1, & k = c \\ 0, & k \neq c \end{cases}$, and $c \in \{1, 2, \ldots, K\}$ is the true class

Training of Neural Network

Big picture: What training a neural network really is

During the training of neural networks, we want to find the neural network parameters (weights and biases) that can minimize the total loss, i.e. the sum of loss over training data.

Big picture: What training a neural network really is

During the training of neural networks, we want to find the neural network parameters (weights and biases) that can minimize the total loss, i.e. the sum of loss over training data.

Today's agenda:

- **Backpropagation**: how to compute gradients efficiently
- **Batched training**: how to organize data during training
- **Optimizers**: how to use those gradients
- **Hyperparameter tuning**: how to choose the training setup

Big picture: What training a neural network really is

Neural Network computation

Suppose input is x

The result at each layer of an MLP is:

$$f_1(x) = \sigma(W_1 x + b_1)$$

Big picture: What training a neural network really is

Neural Network computation

Suppose input is x

The result at each layer of an MLP is:

$$
\begin{aligned}
f_1(x) &= \sigma(W_1 x + b_1) \\
f_2(x) &= \sigma(W_2 f_1(x) + b_2) \\
&= \sigma(W_2 \sigma(W_1 x + b_1) + b_2)
\end{aligned}
$$

Big picture: What training a neural network really is

Neural Network computation
Suppose input is x

The result at each layer of an MLP is:

$$\begin{aligned} f_1(x) &= \sigma(W_1 x + b_1) \\ f_2(x) &= \sigma(W_2 f_1(x) + b_2) \\ &= \sigma(W_2 \sigma(W_1 x + b_1) + b_2) \\ &\dots \end{aligned}$$

$$f_M(x) = W_M \sigma(W_{M-1} x + \sigma(W_{M-2} x + \ldots + b_1) + b_2) + \ldots) + b_{M-1}) + b_M$$

Big picture: What training a neural network really is

Neural Network computation

Suppose input is x

The result at each layer of an MLP is:

$$
\begin{aligned}
f_1(x) &= \sigma(W_1 x + b_1) \\
f_2(x) &= \sigma(W_2 f_1(x) + b_2) \\
&= \sigma(W_2 \sigma(W_1 x + b_1) + b_2) \\
&\dots \\
f_M(x) &= W_M \sigma(W_{M-1} x + \sigma(W_{M-2} x + \ldots + b_1) + b_2) + \ldots) + b_{M-1}) + b_M
\end{aligned}
$$

How to find $\underset{W_1, b_1, W_2, b_2, \ldots, W_M, b_M}{\arg\min} \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[\mathcal{L}(f_M(x), y)\right]$?

$$
\begin{aligned}
&= \arg\min_\theta \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[\mathcal{L}(f_M(x), y)\right] \\
&= \arg\min_\theta \mathcal{L}(\theta)
\end{aligned}
$$

Which way should I go to best vertically descent from where I am?

Follow the slope!

Gradient and gradient descent

The gradient of $\nabla_\theta L(\theta)$ is defined as

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \cdots \\ \frac{\partial L}{\partial \theta_m} \end{bmatrix}$$

Its direction is the **direction of steepest ascent**

Gradient and gradient descent

The gradient of $\nabla_\theta L(\theta)$ is defined as

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \cdots \\ \frac{\partial L}{\partial \theta_m} \end{bmatrix}$$

Its direction is the **direction of steepest ascent**

This is because according to the first-order Taylor expansion:

$$L(\theta + \delta\theta) = L(\theta) + \langle \nabla_\theta L(\theta), \delta\theta \rangle + o(\|\delta\theta\|)$$

The value of $L(\theta)$ is maximized if $\delta\theta$ aligns with $\nabla_\theta L(\theta)$ given a fixed norm of $\delta\theta$

Gradient and gradient descent

The gradient of $\nabla_\theta L(\theta)$ is defined as

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \cdots \\ \frac{\partial L}{\partial \theta_m} \end{bmatrix}$$

Its direction is the **direction of steepest ascent**

This is because according to the first-order Taylor expansion:

$$L(\theta + \delta\theta) = L(\theta) + \langle \nabla_\theta L(\theta), \delta\theta \rangle + o(\|\delta\theta\|)$$

The value of $L(\theta)$ is maximized if $\delta\theta$ aligns with $\nabla_\theta L(\theta)$ given a fixed norm of $\delta\theta$

The **direction of steepest descent** will be the **negative gradient** $-\nabla_\theta L(\theta)$

Gradient and gradient descent

Though it is difficult to find the global optima directly, we can update the parameters of the neural network along the direction of negative gradient to update the parameters to at least **locally decrease the loss**

By accumulating many such local updates of parameters, we hope to obtain a globally good solution.

Gradient and gradient descent

Though it is difficult to find the global optima directly, we can update the parameters of the neural network along the direction of negative gradient to update the parameters to at least **locally decrease the loss**

By accumulating many such local updates of parameters, we hope to obtain a globally good solution.

This is **Gradient Descent (GD)**:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

where the step size coefficient α is called **learning rate**

Gradient and gradient descent

Though it is difficult to find the global optima directly, we can update the parameters of the neural network along the direction of negative gradient to update the parameters to at least **locally decrease the loss**

By accumulating many such local updates of parameters, we hope to obtain a globally good solution.

This is **Gradient Descent (GD)**:

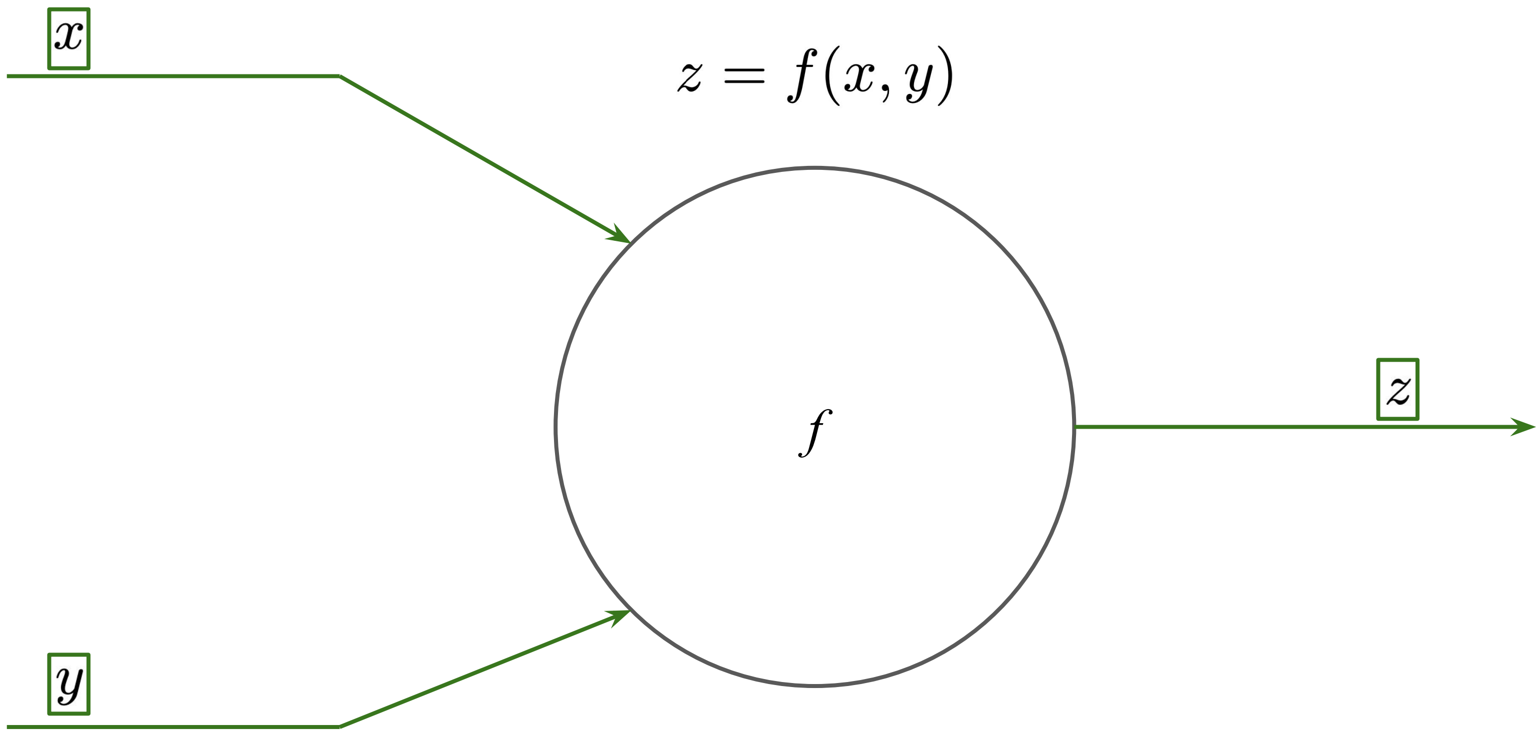
$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

where the step size coefficient α is called **learning rate**

How to compute the gradient of loss w.r.t. NN parameters $\nabla_\theta L(\theta)$?

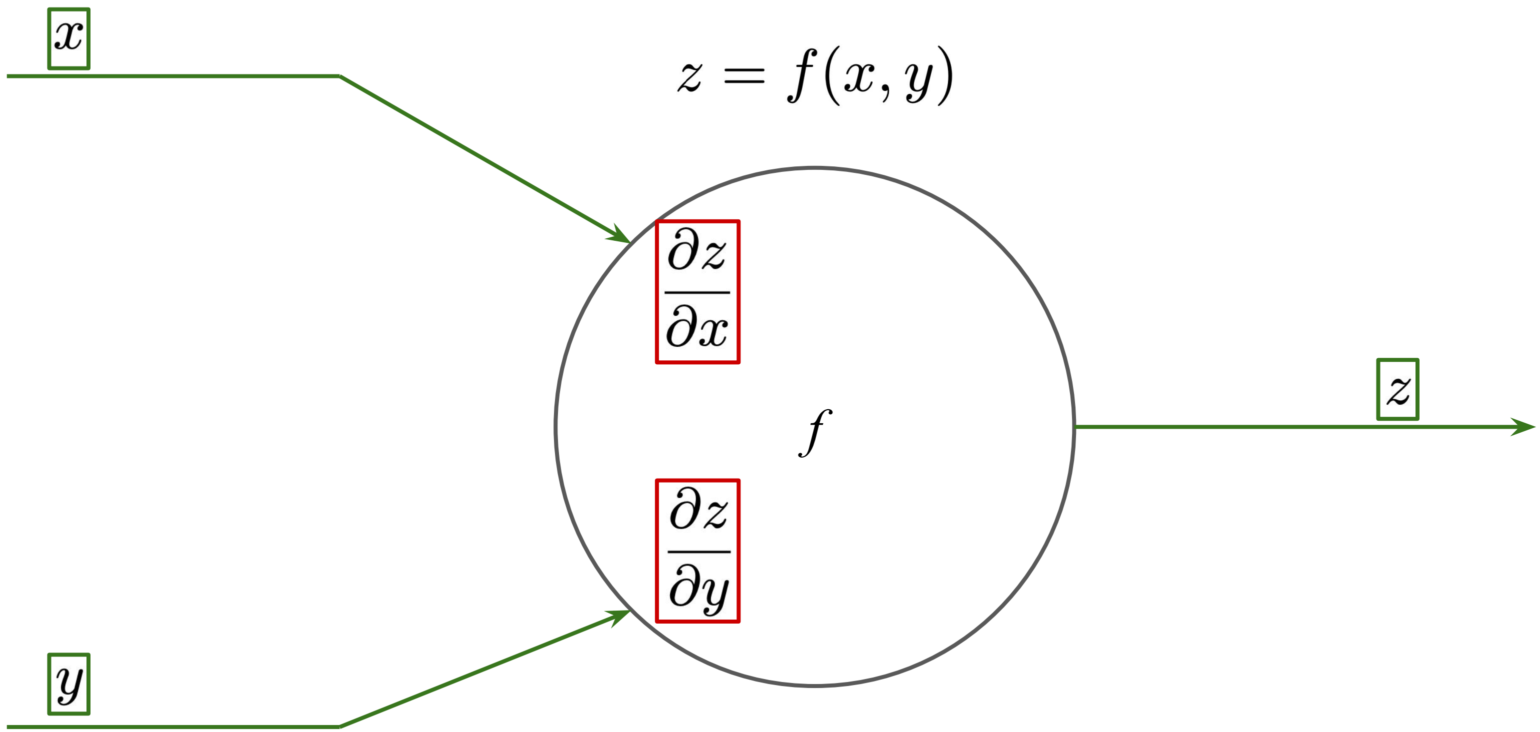
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



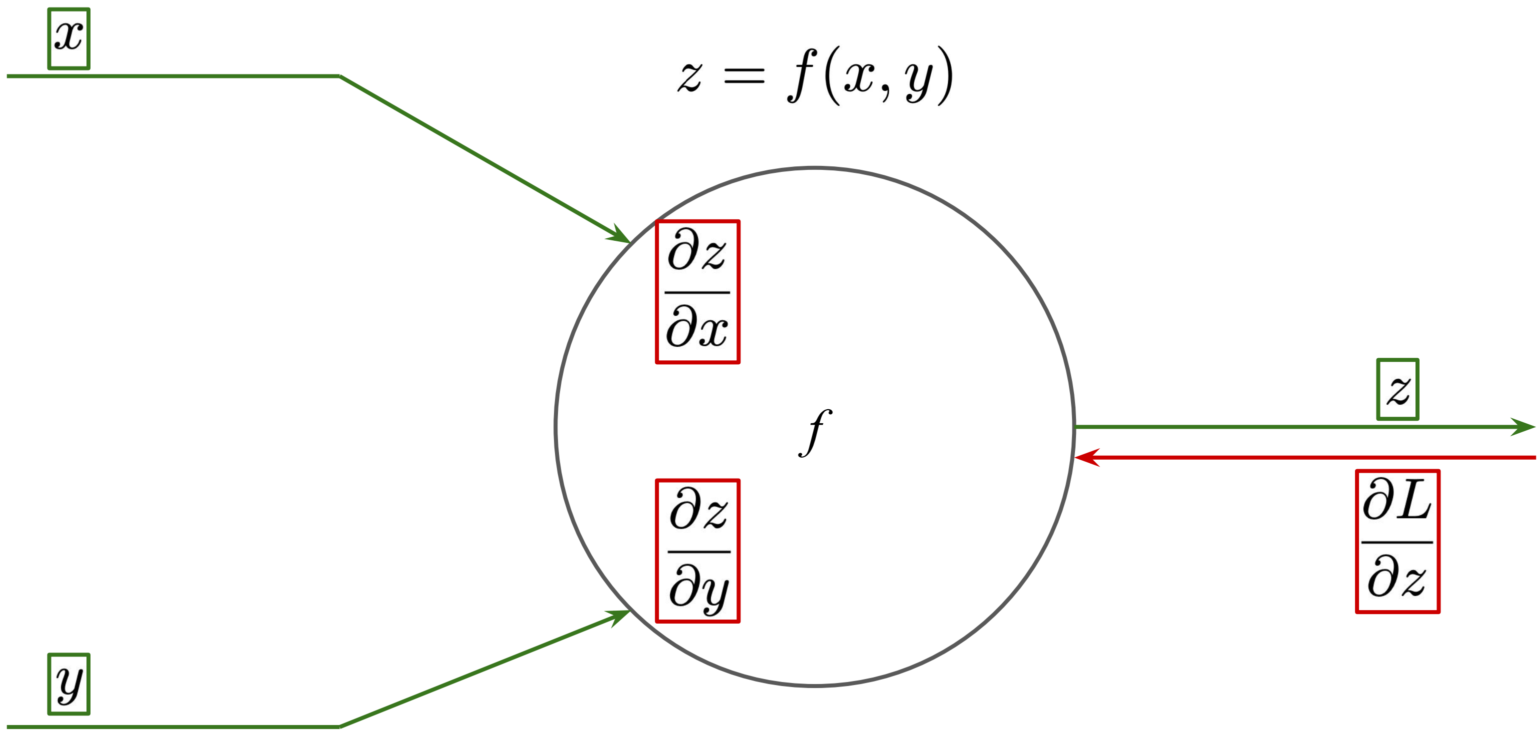
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



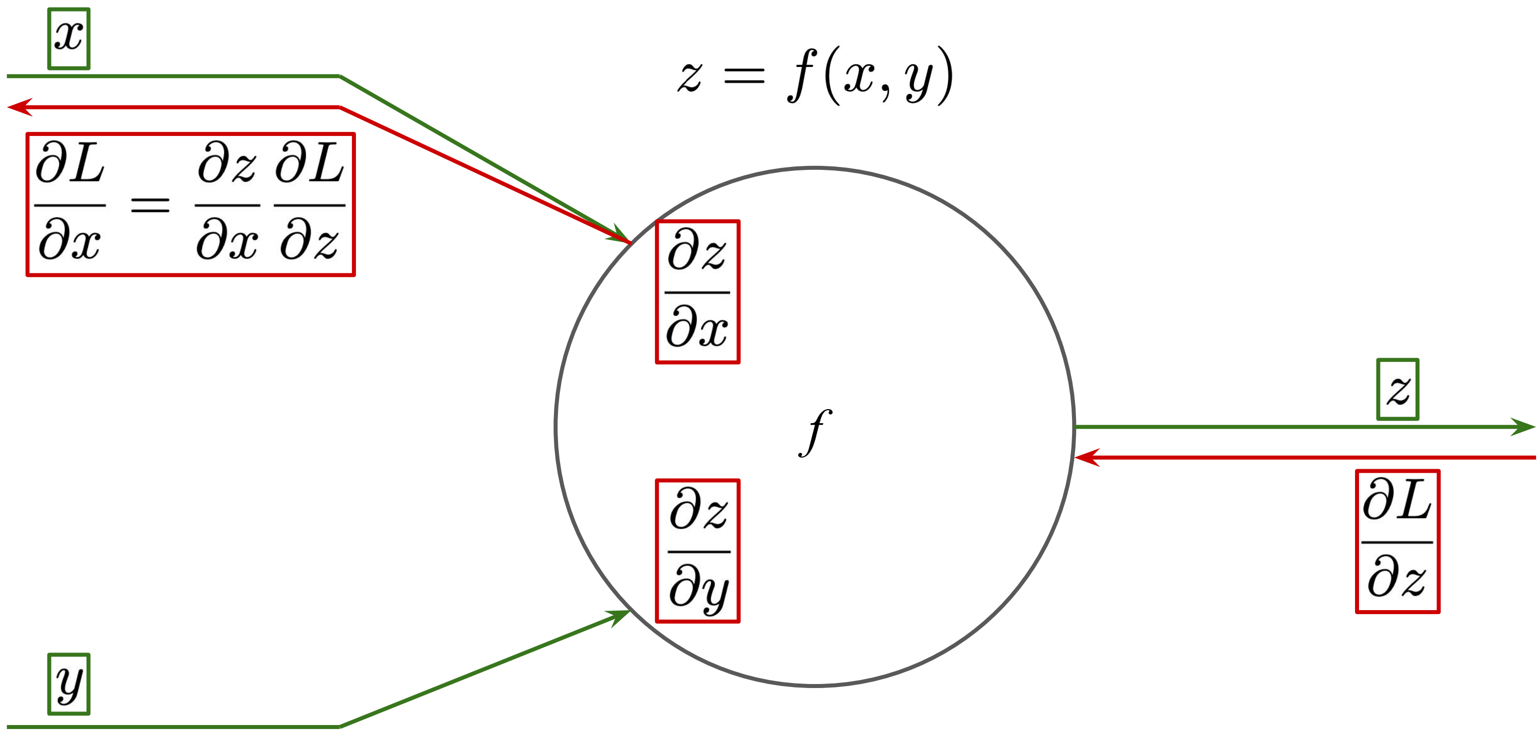
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



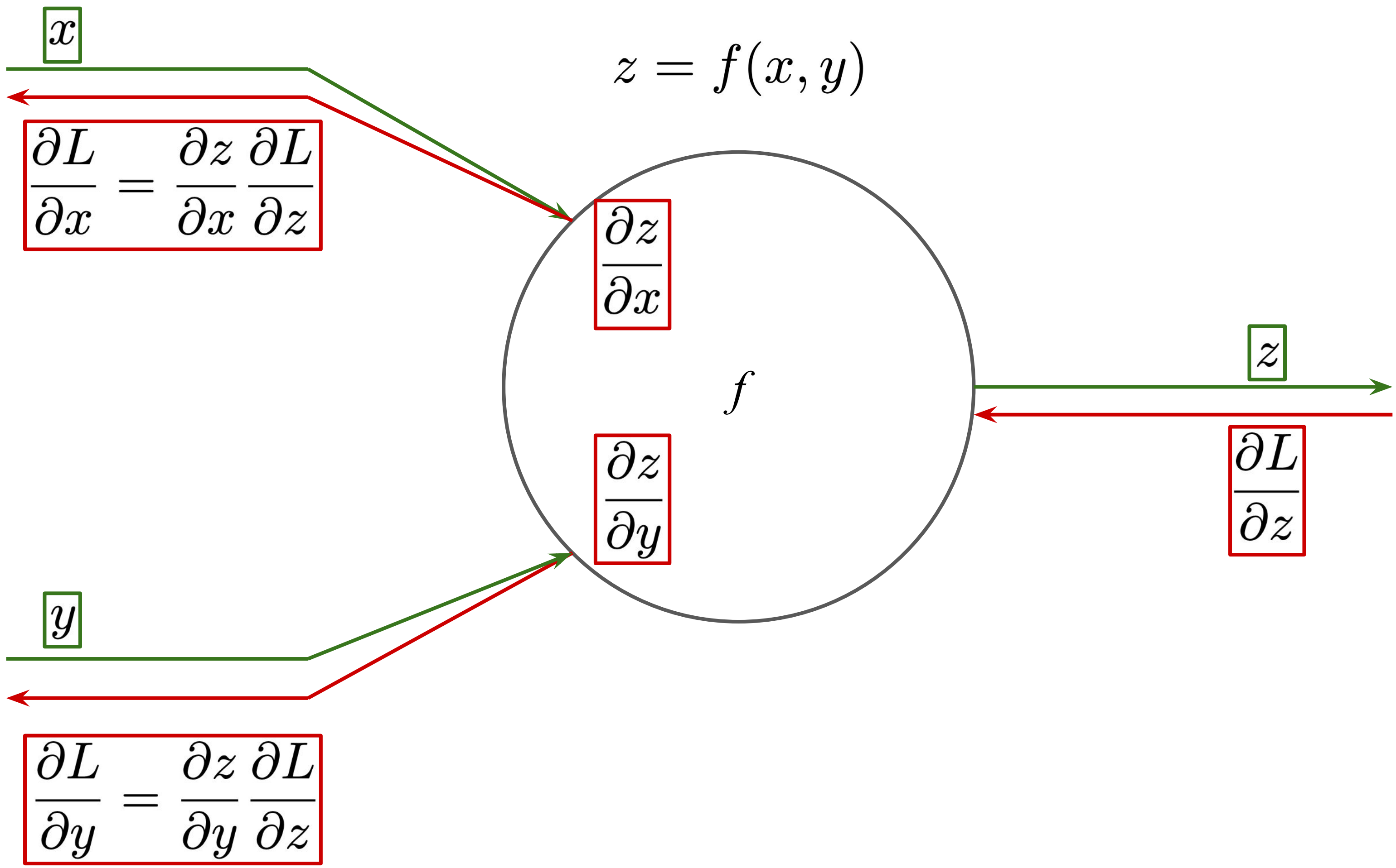
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



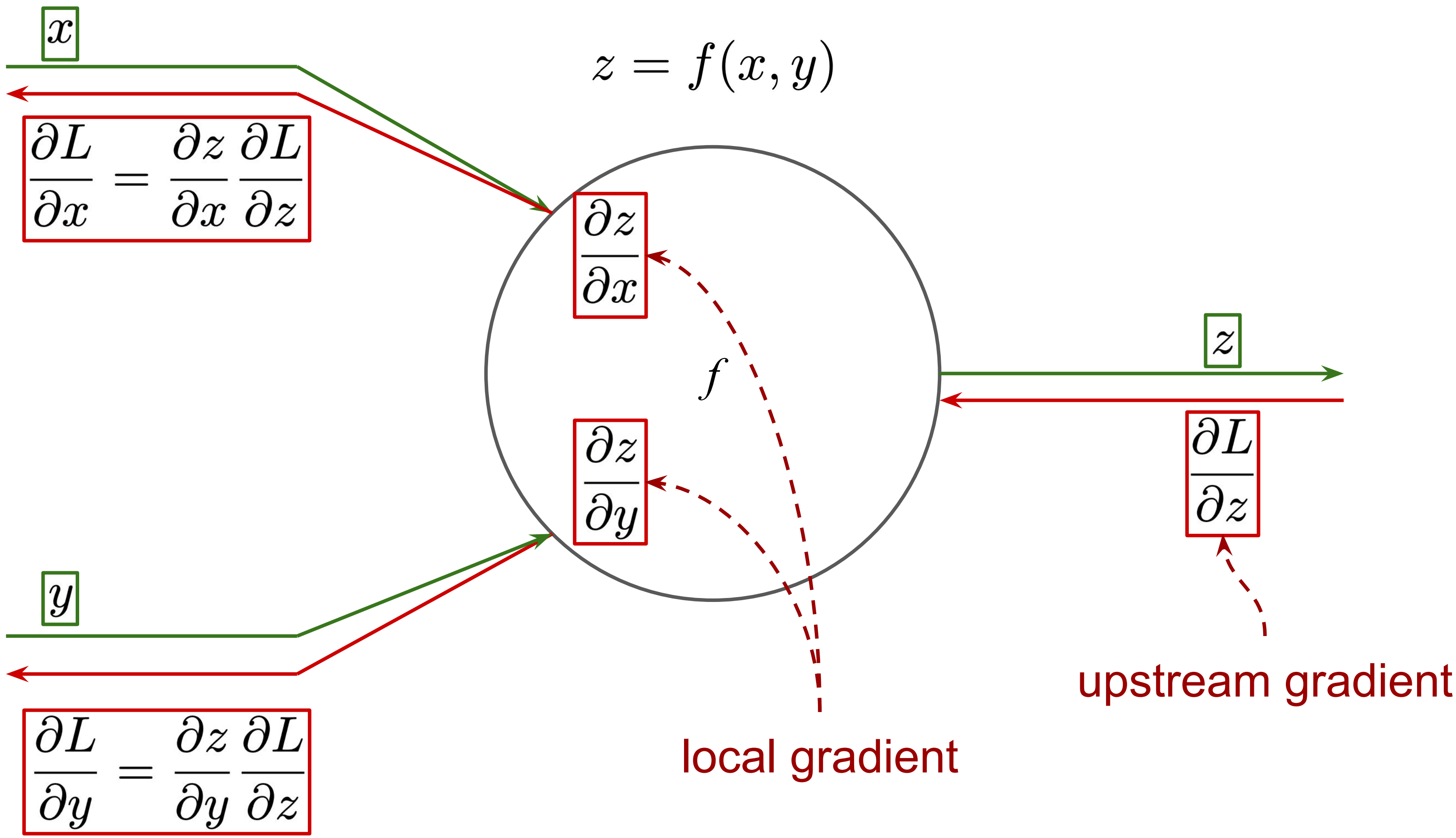
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



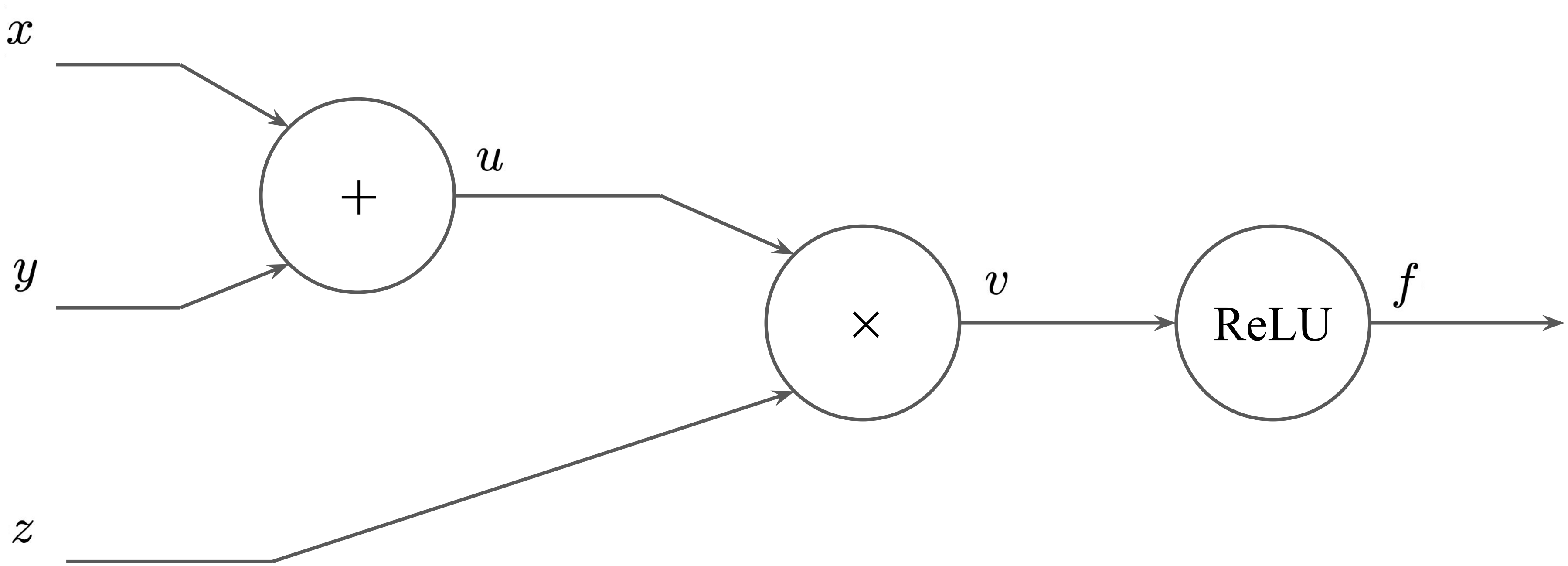
Backpropagation: scalar functions

A neural network is a parameterized computational graph.
Gradients are computed efficiently using the **chain rule**, implemented as **backpropagation** on the graph.



Backpropagation: scalar functions

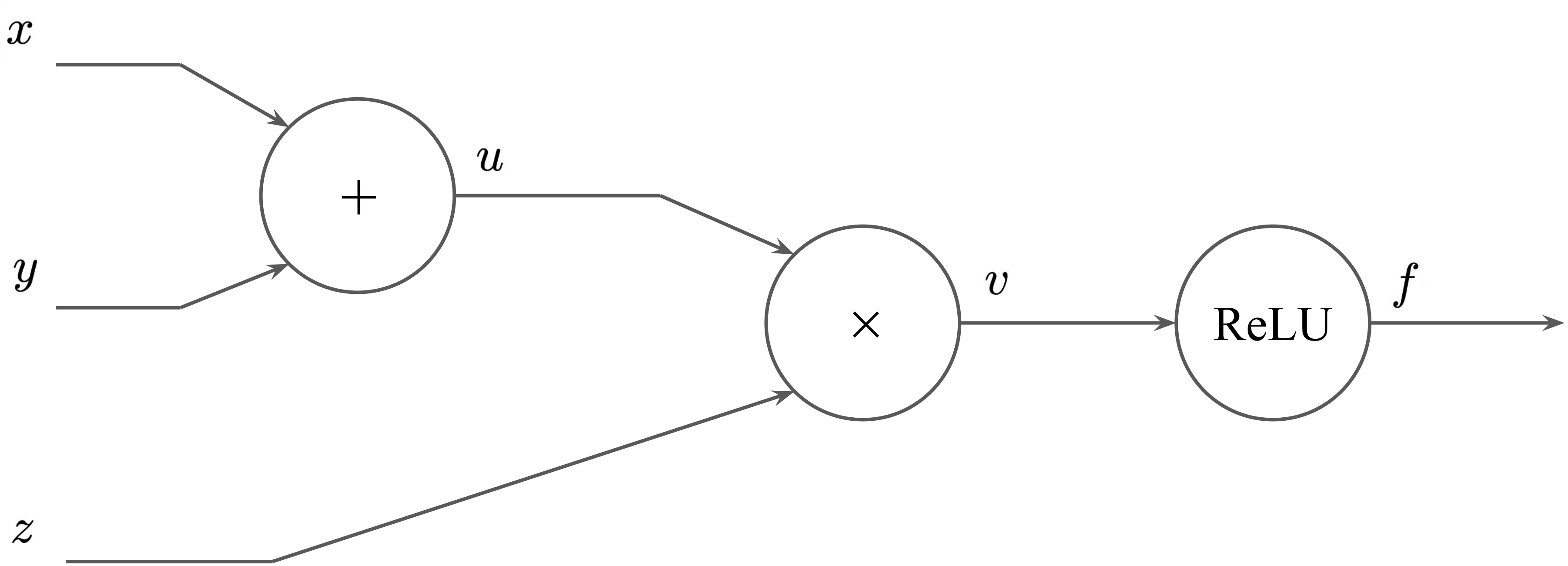
A simple example: $f(x, y, z) = \mathrm{ReLU}((x + y)z)$
Let $u = x + y$ and $v = uz$



Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$
Let $u = x + y$ and $v = uz$

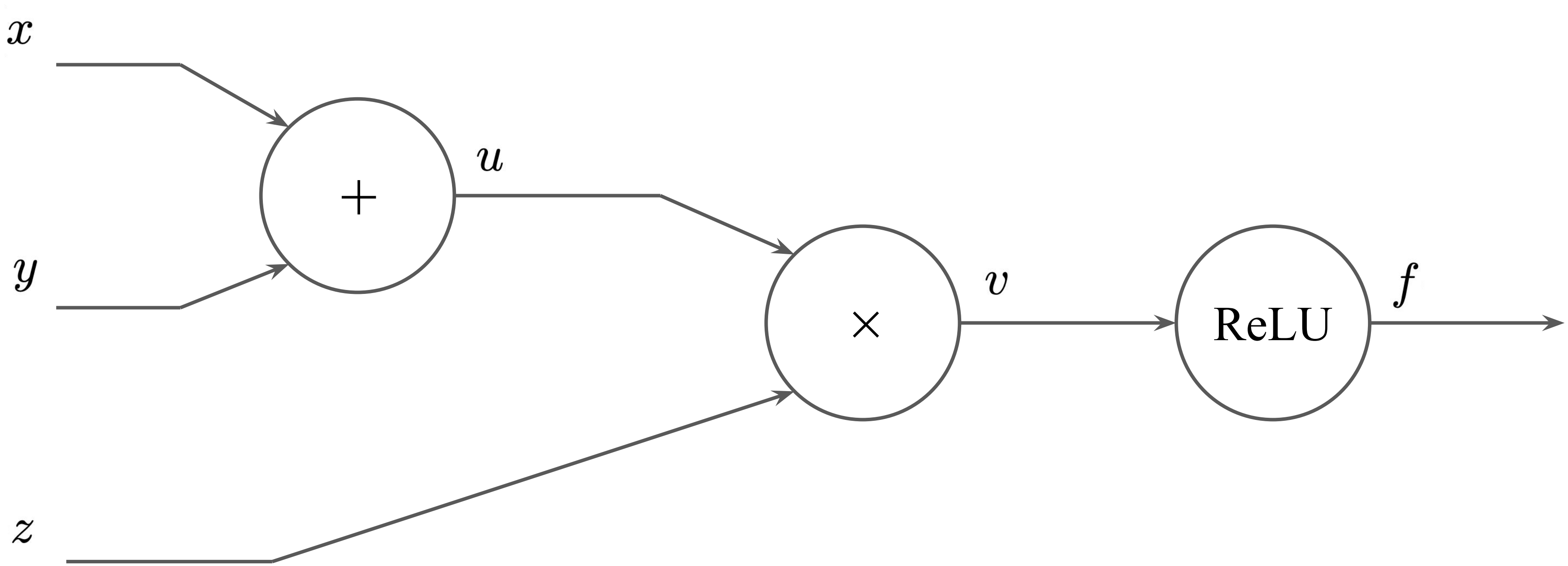
Gradients: $\frac{\partial u}{\partial x} =$ $\frac{\partial u}{\partial y} =$ $\frac{\partial v}{\partial u} =$ $\frac{\partial v}{\partial z} =$ $\frac{\partial f}{\partial v} =$



Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

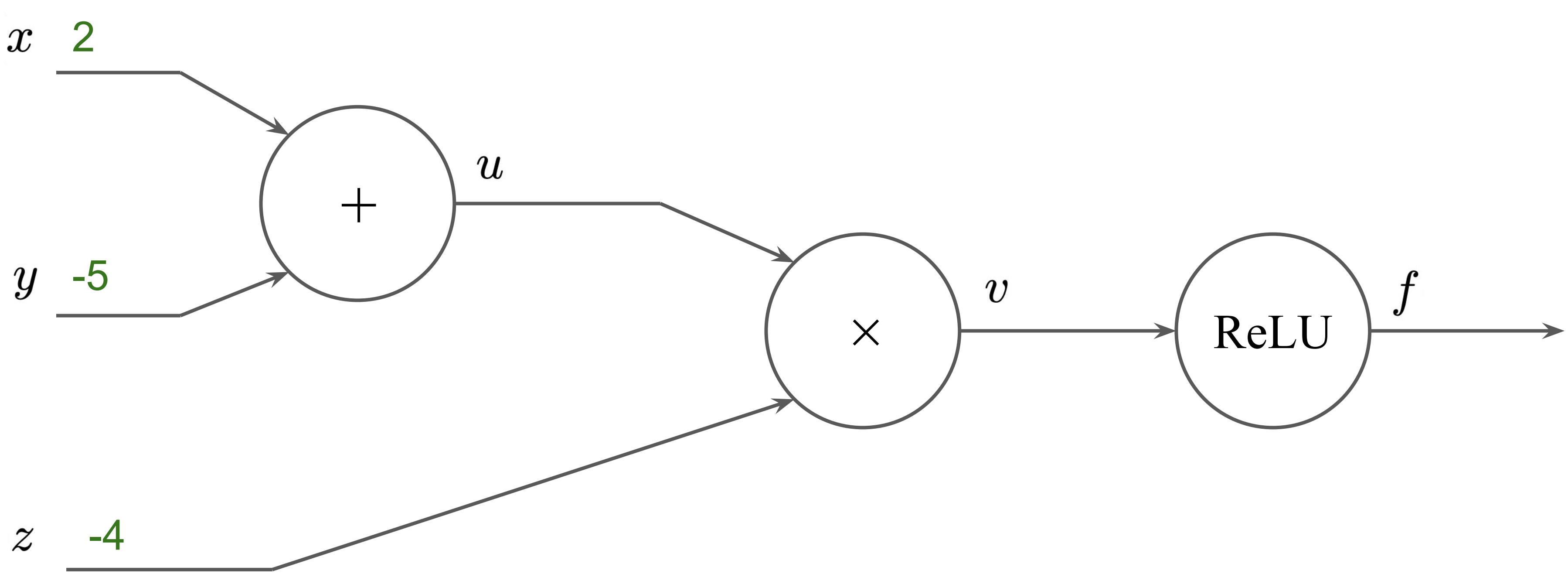


Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$

Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

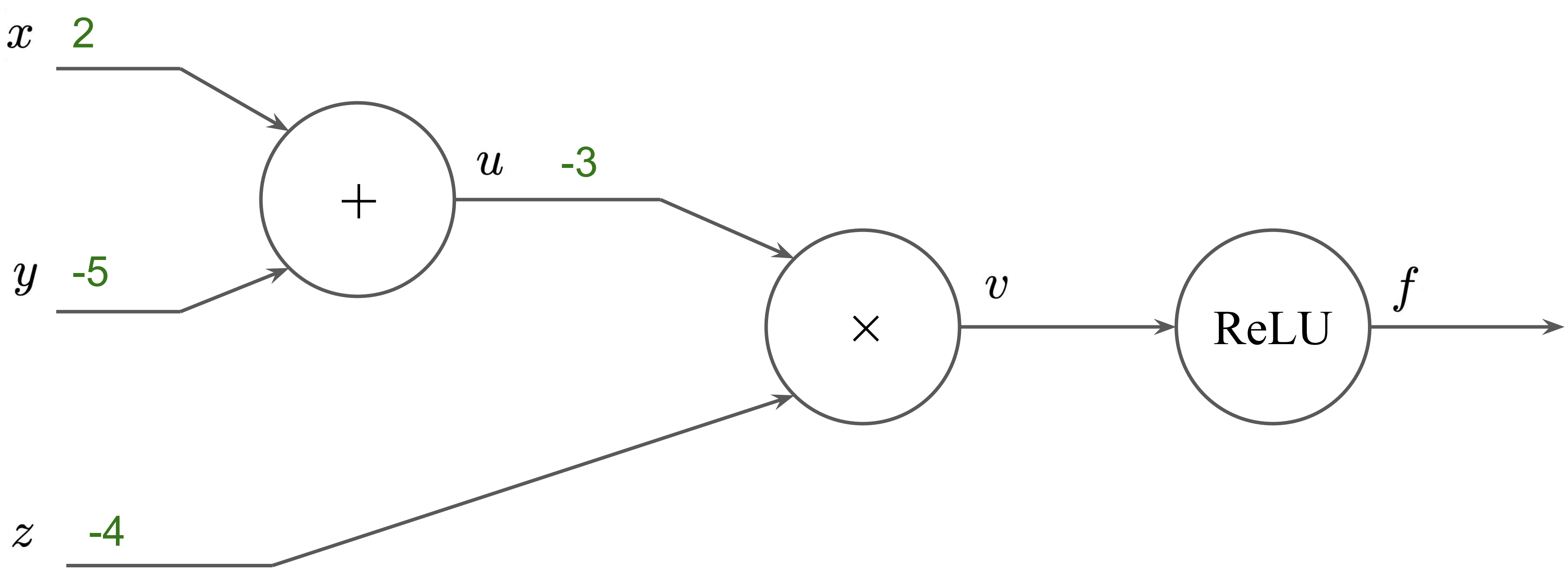


Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$

Let $u = x + y$ and $v = uz$

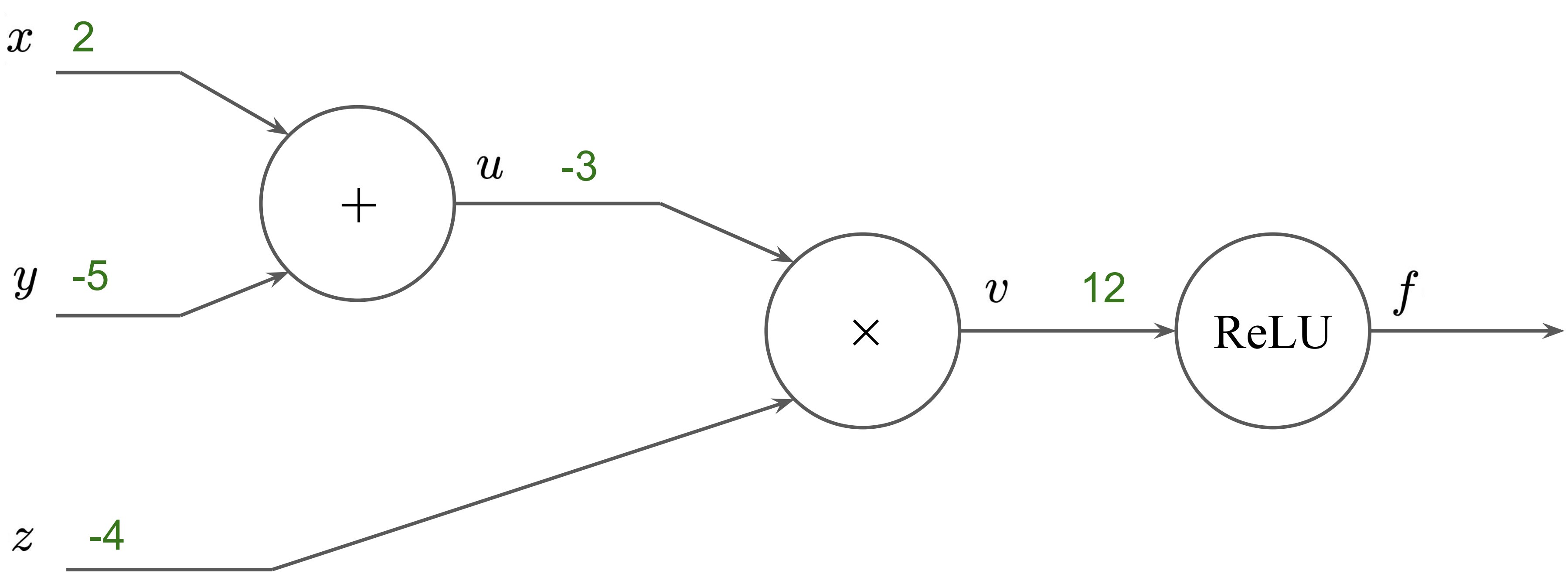
Gradients: $\dfrac{\partial u}{\partial x} = 1 \qquad \dfrac{\partial u}{\partial y} = 1 \qquad \dfrac{\partial v}{\partial u} = z \qquad \dfrac{\partial v}{\partial z} = u \qquad \dfrac{\partial f}{\partial v} = \mathbb{1}(v > 0)$



Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$
Let $u = x + y$ and $v = uz$

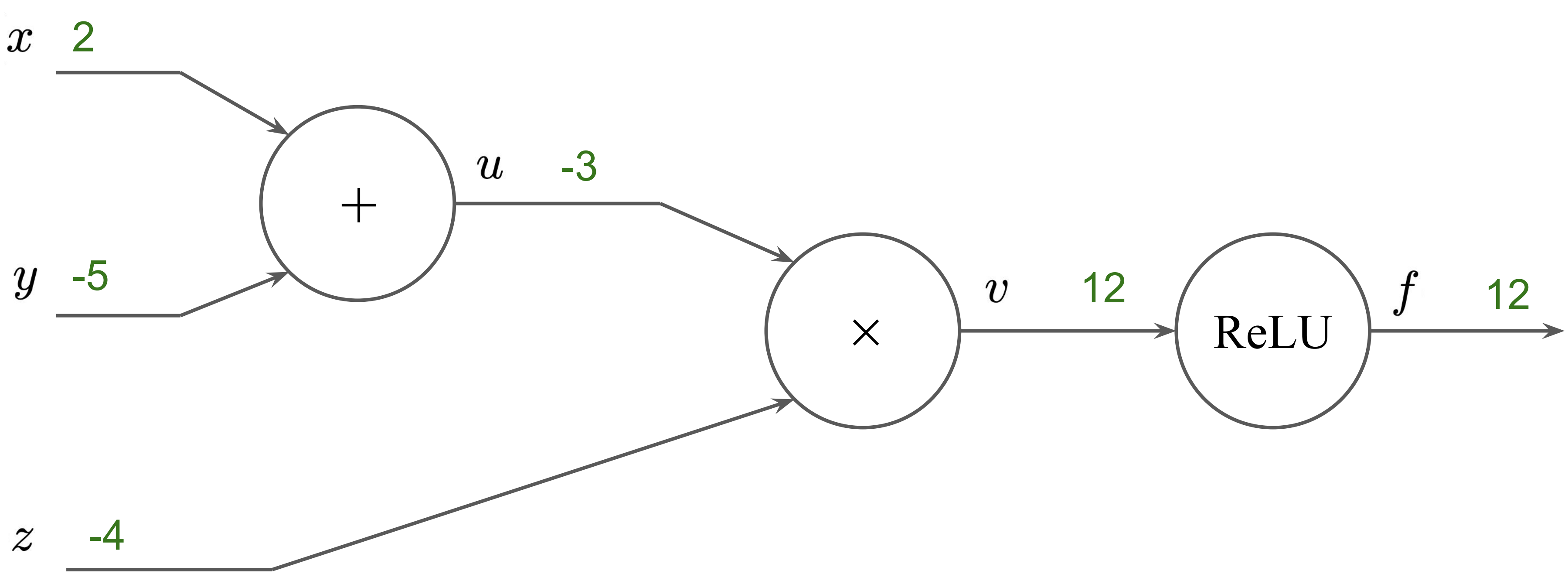
Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$



Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x + y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

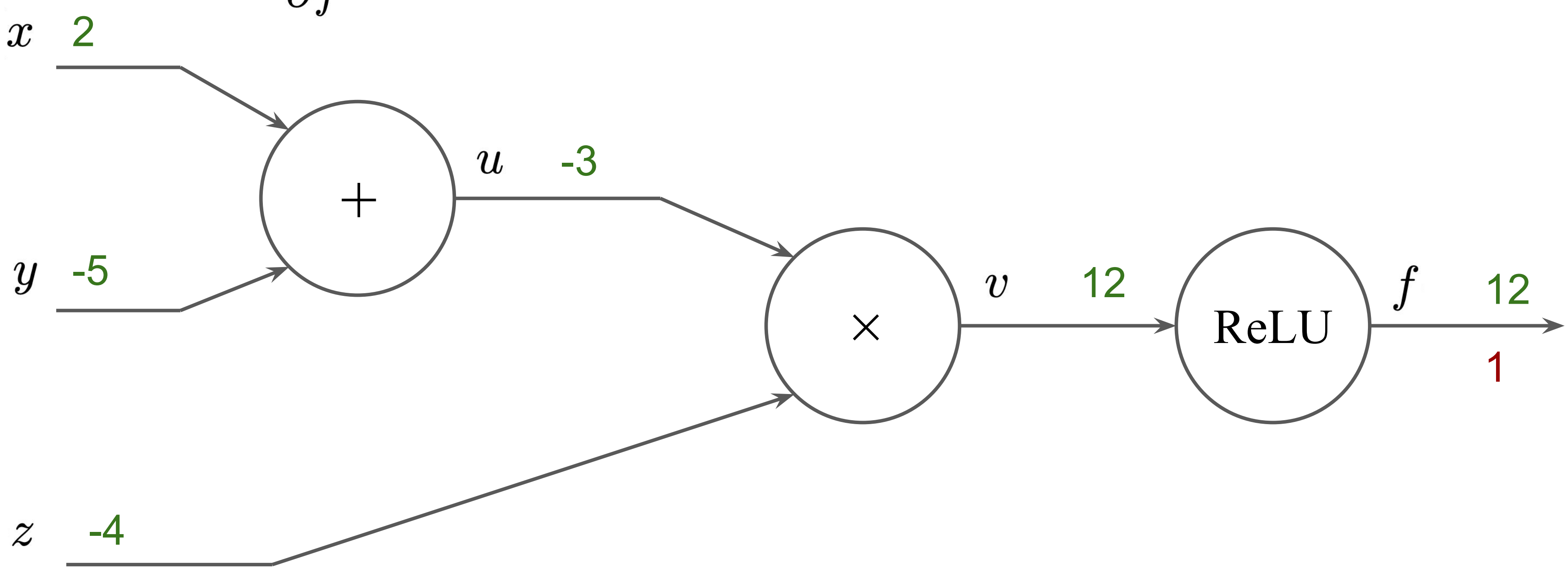


Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x + y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1$ $\frac{\partial u}{\partial y} = 1$ $\frac{\partial v}{\partial u} = z$ $\frac{\partial v}{\partial z} = u$ $\frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$

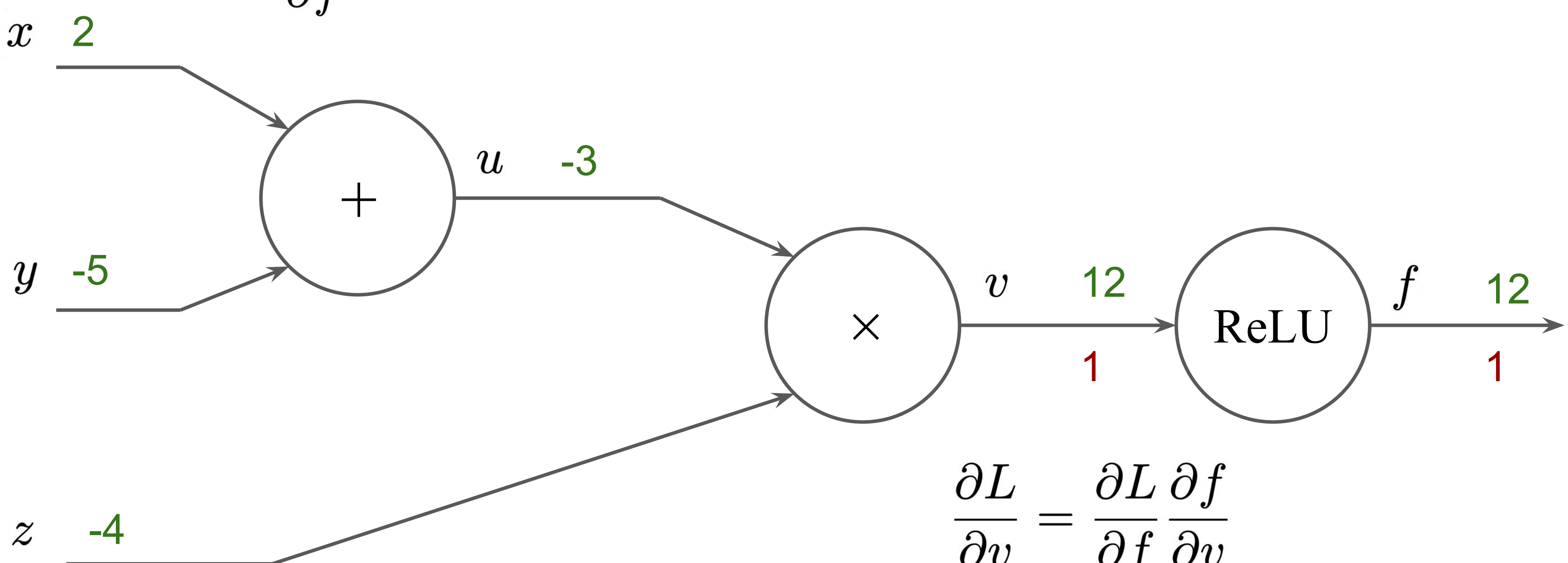


Backpropagation: scalar functions

A simple example: $f(x, y, z) = \text{ReLU}((x+y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$



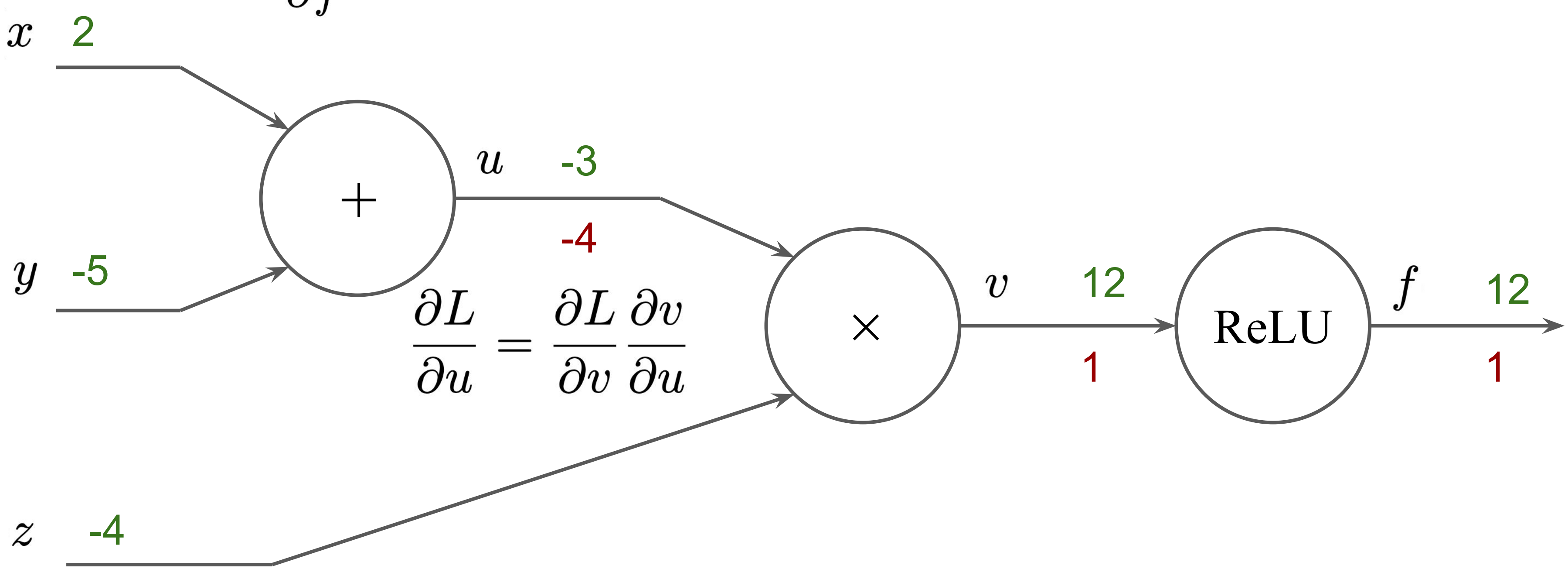
$$\frac{\partial L}{\partial v} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial v}$$

Backpropagation: scalar functions

A simple example: $f(x, y, z) = \text{ReLU}((x + y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$



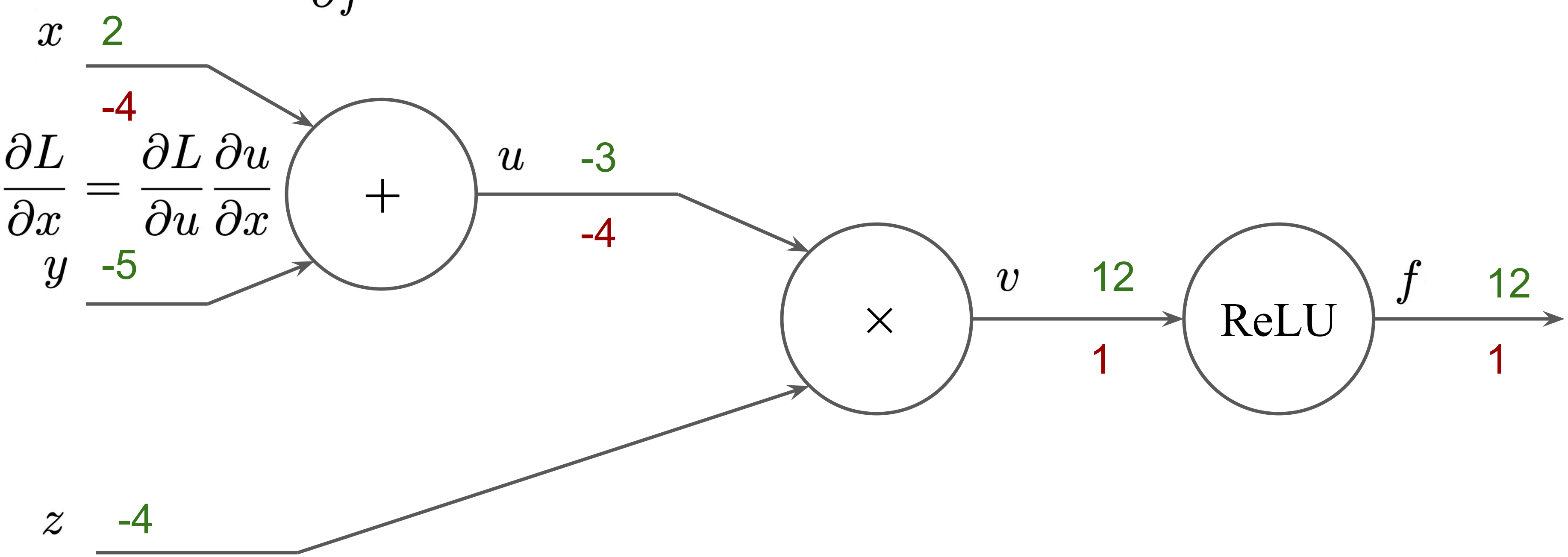
$$\frac{\partial L}{\partial u} = \frac{\partial L}{\partial v}\frac{\partial v}{\partial u}$$

Backpropagation: scalar functions

A simple example: $f(x, y, z) = \text{ReLU}((x + y)z)$
Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$



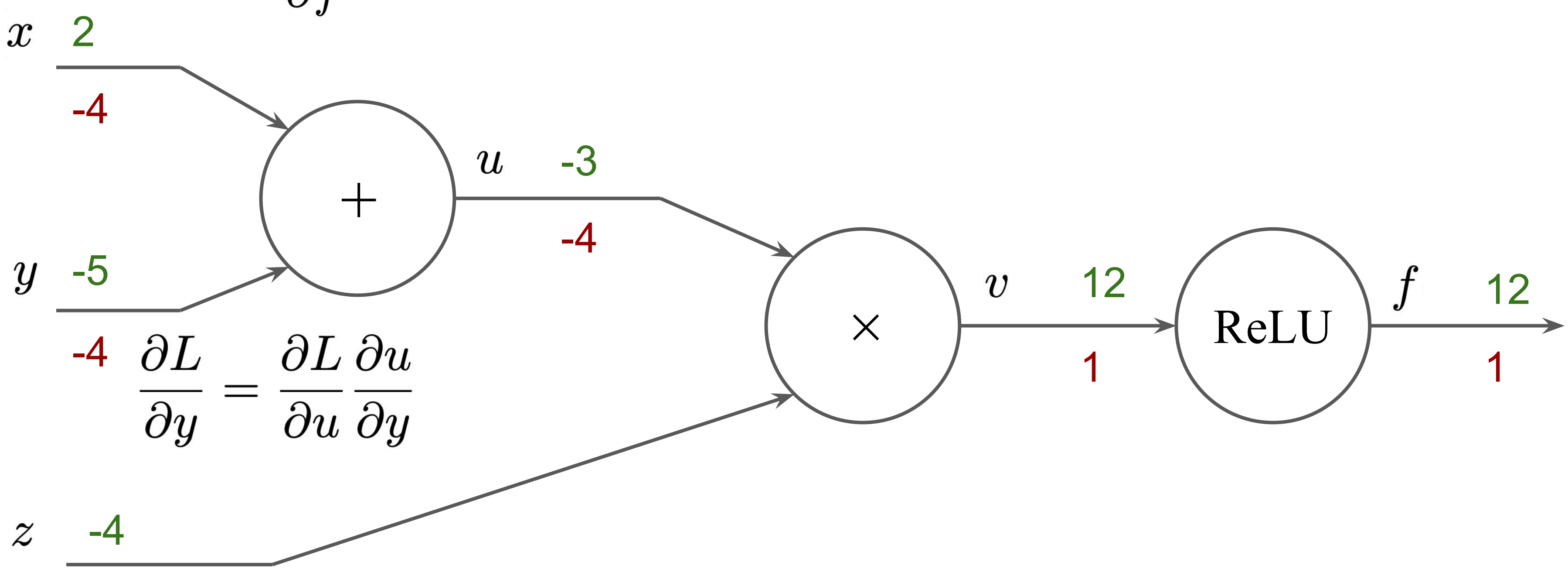
Backpropagation: scalar functions

A simple example: $f(x, y, z) = \text{ReLU}((x+y)z)$

Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1 \quad \frac{\partial u}{\partial y} = 1 \quad \frac{\partial v}{\partial u} = z \quad \frac{\partial v}{\partial z} = u \quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$



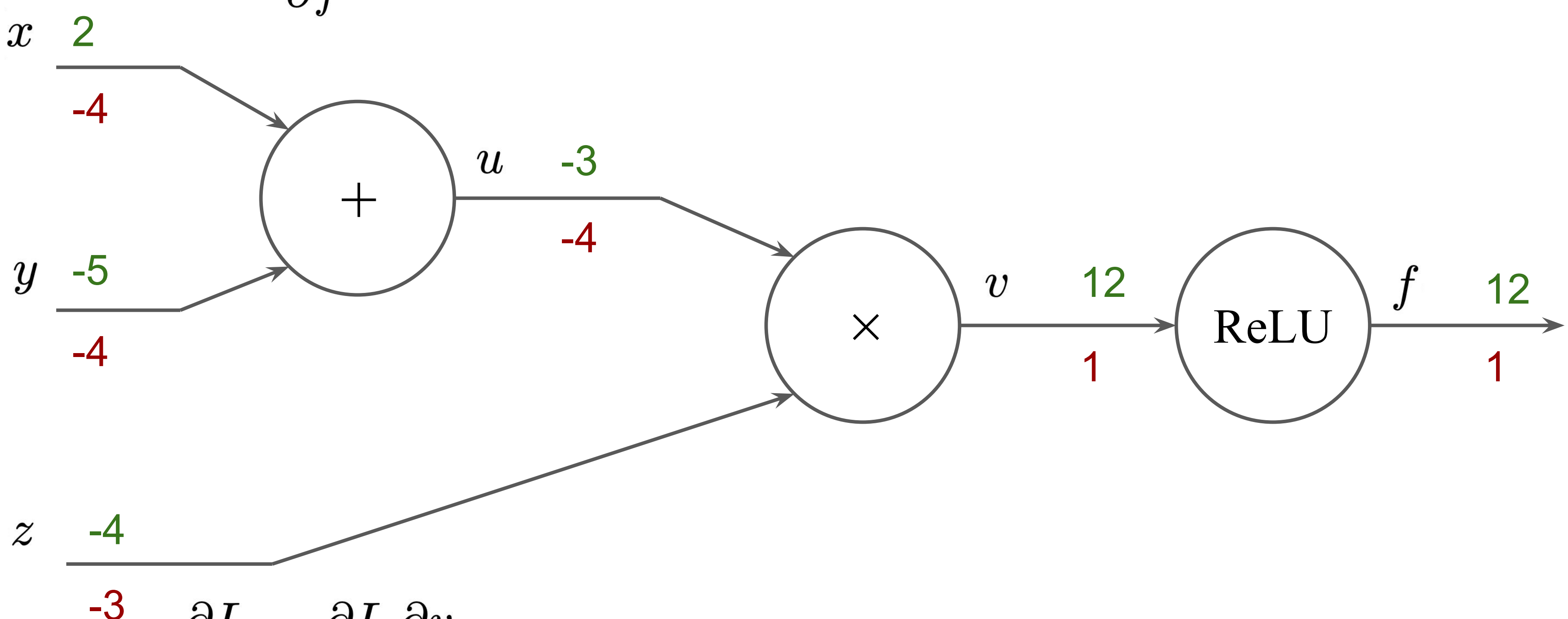
Backpropagation: scalar functions

A simple example: $f(x, y, z) = \mathrm{ReLU}((x+y)z)$

Let $u = x + y$ and $v = uz$

Gradients: $\frac{\partial u}{\partial x} = 1$ $\quad \frac{\partial u}{\partial y} = 1$ $\quad \frac{\partial v}{\partial u} = z$ $\quad \frac{\partial v}{\partial z} = u$ $\quad \frac{\partial f}{\partial v} = \mathbb{1}(v > 0)$

Suppose $\frac{\partial L}{\partial f} = 1$



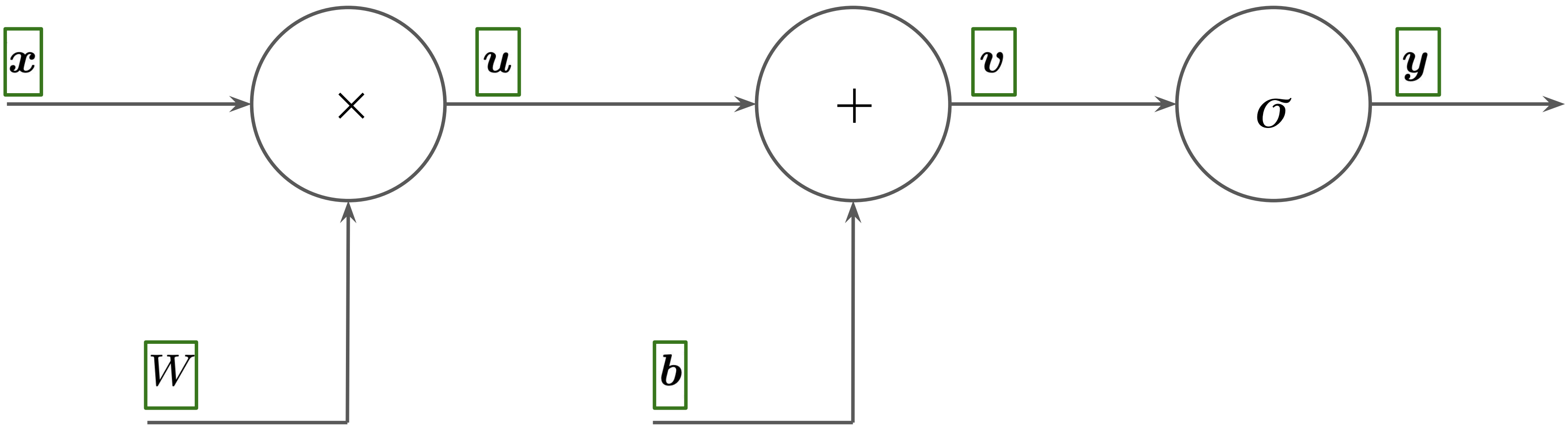
$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial v}\frac{\partial v}{\partial z}$$

Backpropagation: vector-valued functions

A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$
Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$

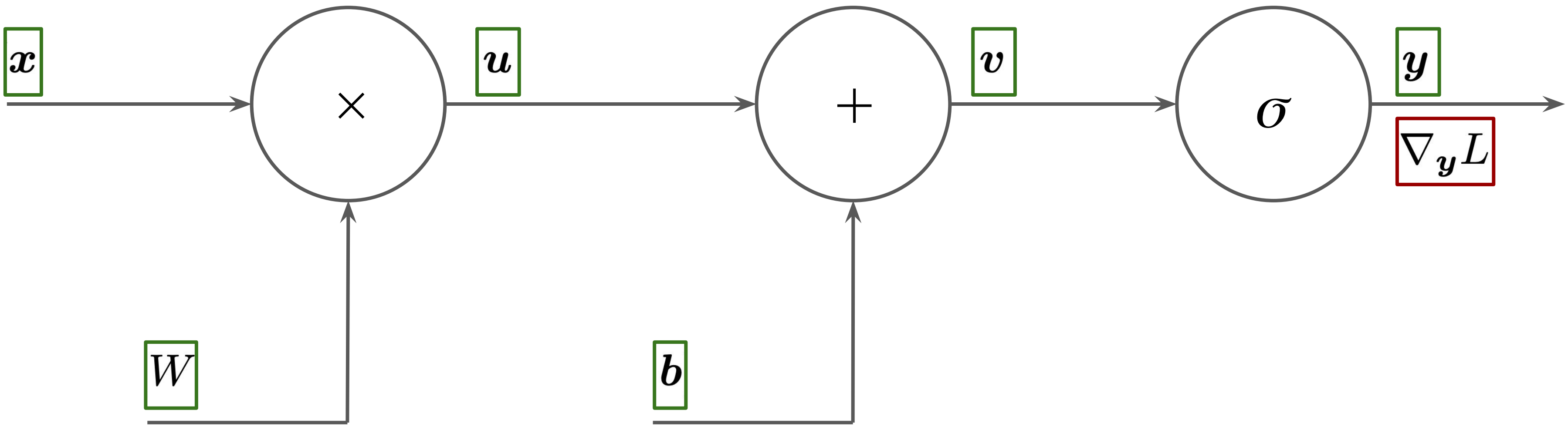
Backpropagation: vector-valued functions

A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$
Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$



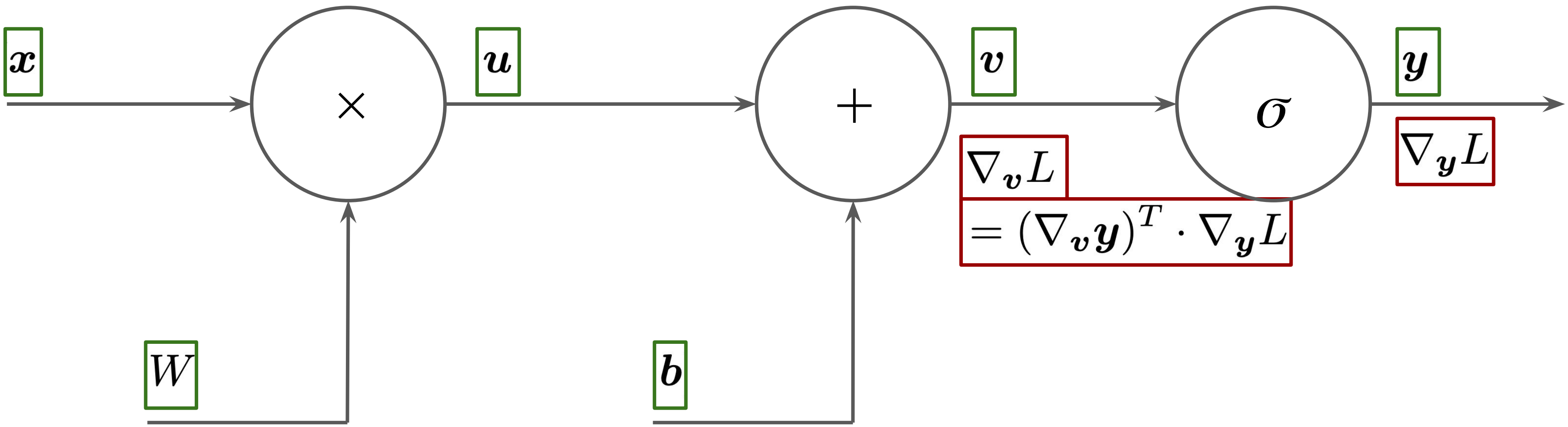
Backpropagation: vector-valued functions

A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$
Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$



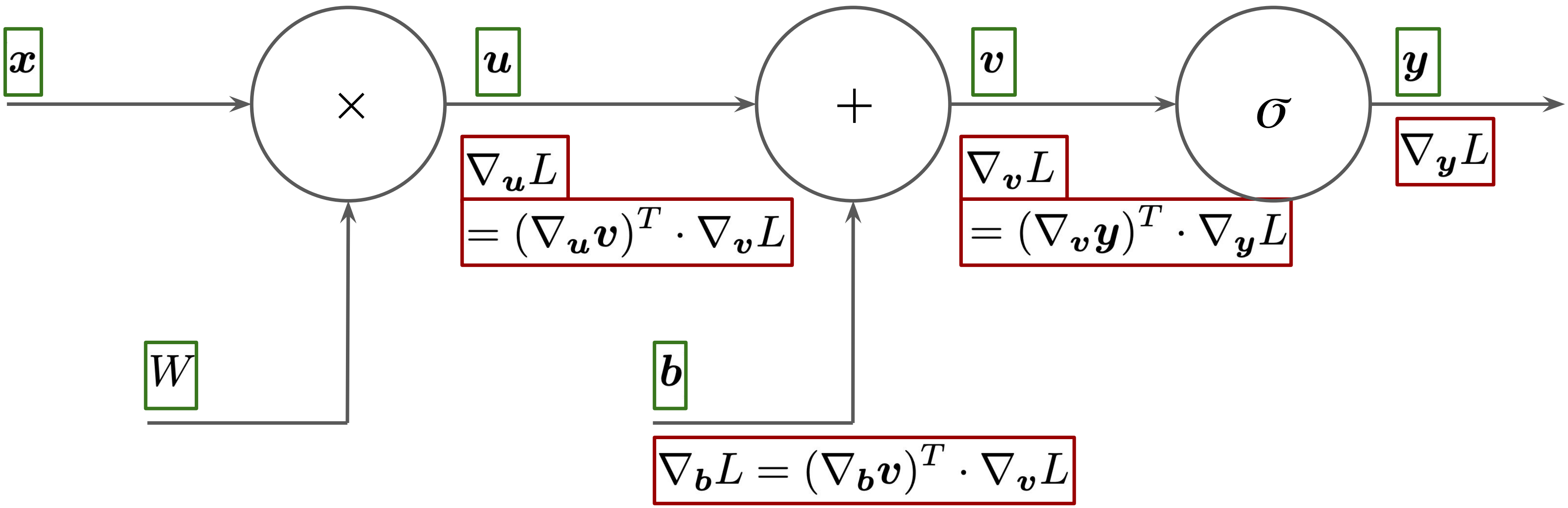
Backpropagation: vector-valued functions

A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$
Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$



Backpropagation: vector-valued functions

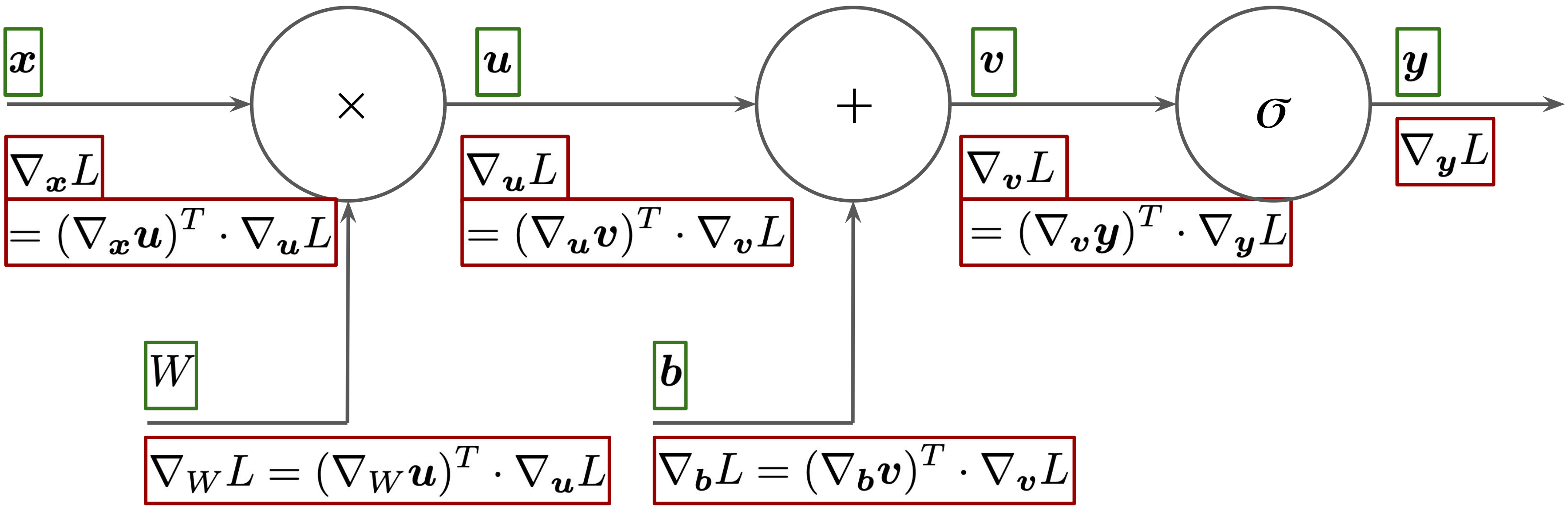
A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$
Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$



Backpropagation: vector-valued functions

A more practical example of neural networks: $\boldsymbol{y} = \sigma(W\boldsymbol{x} + \boldsymbol{b})$

Let $\boldsymbol{u} = W\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{b}$

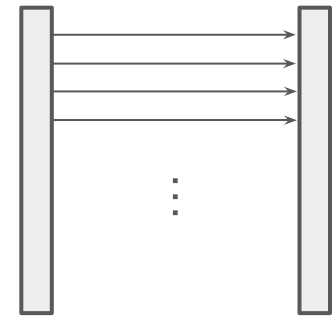


Backpropagation: vector-valued functions

Element-wise operations

Each output element depends on only one corresponding input element

Examples:



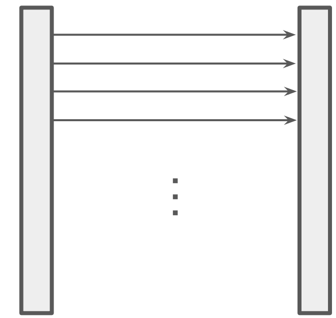
- Activation functions $\sigma()$
- Element-wise summation $\boldsymbol{x} + \boldsymbol{y}$
- Element-wise multiplication $\boldsymbol{x} \odot \boldsymbol{y}$

Gradient: also element-wise, represented by a diagonal matrix

Backpropagation: vector-valued functions

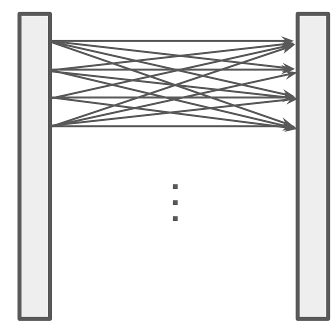
Element-wise operations
Each output element depends on only one corresponding input element
Examples:

- Activation functions $\sigma()$
- Element-wise summation $\boldsymbol{x} + \boldsymbol{y}$
- Element-wise multiplication $\boldsymbol{x} \odot \boldsymbol{y}$



Gradient: also element-wise, represented by a diagonal matrix

More general vectorized operations
Each output element depends on many input elements



Examples:

- Matrix multiplication $\boldsymbol{y} = W\boldsymbol{x}$
- Softmax $y_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$

Gradient: a dense matrix

Vanilla Gradient Descent

The loss used in vanilla version of **Gradient Descent (GD)**:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

is across the entire dataset

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left([\mathcal{L}(f_\theta(x), y)]\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)
\end{aligned}$$

Vanilla Gradient Descent

The loss used in vanilla version of **Gradient Descent (GD)**:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

is across the entire dataset

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left([\mathcal{L}(f_\theta(x), y)]\right) \\ &= \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \end{aligned}$$

Computing the gradients requires processing **all N samples** at every update step

This is computationally expensive for large dataset!

Vanilla Gradient Descent

The loss used in vanilla version of **Gradient Descent (GD)**:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

is across the entire dataset

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left([\mathcal{L}(f_\theta(x), y)]\right) \\ &= \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \end{aligned}$$

Computing the gradients requires processing **all N samples** at every update step

This is computationally expensive for large dataset!

<u>Can we make faster progress?</u>

Vanilla Gradient Descent

The loss used in vanilla version of **Gradient Descent (GD)**:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

is across the entire dataset

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left([\mathcal{L}(f_\theta(x), y)]\right) \\ &= \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \end{aligned}$$

Computing the gradients requires processing **all N samples** at every update step

This is computationally expensive for large dataset!

<u>Can we make faster progress?</u>

Make more frequent updates by using only a subset of data at each iteration!

Mini-batch Stochastic Gradient Descent (SGD)

At each iteration, sample a mini-batch (subset) $\mathcal{B} \subset \{1, 2, \ldots, N\}$ to compute loss

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell_i(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}(f_\theta(x_i), y_i)$$

Mini-batch Stochastic Gradient Descent (SGD)

At each iteration, sample a mini-batch (subset) $\mathcal{B} \subset \{1, 2, \ldots, N\}$ to compute loss

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell_i(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}(f_\theta(x_i), y_i)$$

and apply gradient descent

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta)$$

Mini-batch Stochastic Gradient Descent (SGD)

At each iteration, sample a mini-batch (subset) $\mathcal{B} \subset \{1, 2, \ldots, N\}$ to compute loss

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell_i(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}(f_\theta(x_i), y_i)$$

and apply gradient descent

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta)$$

Though the selection of mini-batch $\mathcal{B}$ is random, we want every data sample to have equal opportunity to be used in training.

This is to guarantee that the original data distribution in the dataset is unchanged!

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

Mini-batch Stochastic Gradient Descent (SGD)

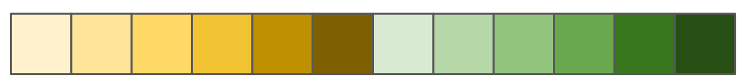
In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

Mini-batch gradient descent (GD):



Mini-batch Stochastic Gradient Descent (SGD)

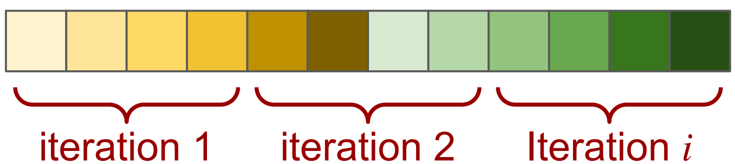
In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

Mini-batch gradient descent (GD):



Mini-batch Stochastic Gradient Descent (SGD)

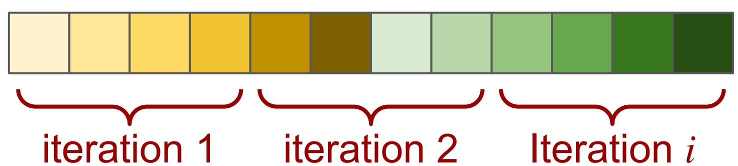
In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

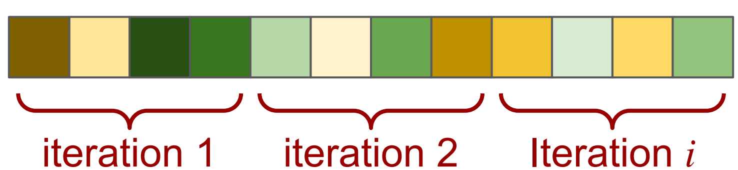
and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

Mini-batch gradient descent (GD):



Mini-batch **stochastic** gradient descent (SGD):



Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

One full sweep of the dataset is called an **epoch**

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

One full sweep of the dataset is called an **epoch**

How many iterations are there in an epoch?

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

One full sweep of the dataset is called an **epoch**

<u>How many iterations are there in an epoch?</u> $[\frac{N}{|\mathcal{B}|}]$

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

One full sweep of the dataset is called an **epoch**

How many iterations are there in an epoch? $\left[\dfrac{N}{|\mathcal{B}|}\right]$

Neural networks are usually trained on multiple epochs!

Mini-batch Stochastic Gradient Descent (SGD)

In practice, we randomly permute the dataset indices

$$(\pi_1, \pi_2, \ldots, \pi_N) \leftarrow \text{random\_permute}(1, 2, \ldots, N)$$

and sweep the dataset sequentially, where in the i-th iteration, the indices of the mini-batch using in gradient descent are

$$\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+|\mathcal{B}|-1}\}$$

One full sweep of the dataset is called an **epoch**

How many iterations are there in an epoch? $[\frac{N}{|\mathcal{B}|}]$

Neural networks are usually trained on multiple epochs!

In every epoch, the indices are randomly permuted for SGD

A short break

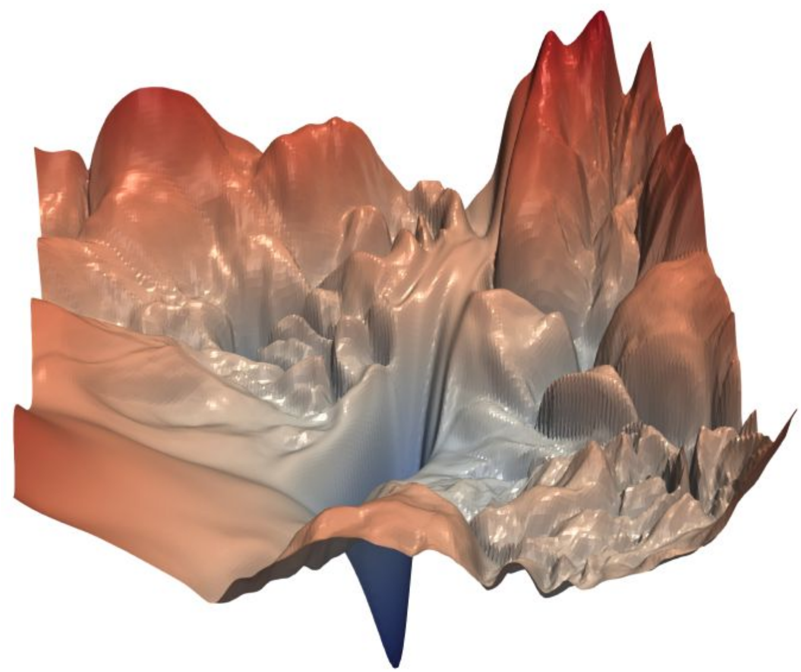
We will be back in 5 mins

Problems of Stochastic Gradient Descent (SGD):

- Loss landscape can be ill-conditioned, i.e. changes very fast in some parameter directions and very slowly in other directions.

Problems of Stochastic Gradient Descent (SGD):

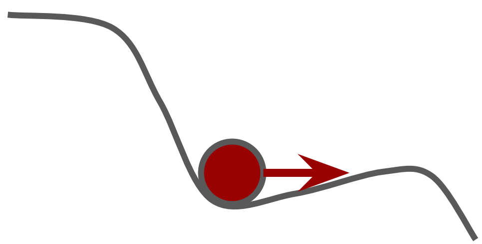
- Loss landscape can be ill-conditioned, i.e. changes very fast in some parameter directions and very slowly in other directions.
  - Large gradient in the steep direction → big steps → overshoot
  - Small gradient in flat direction → slow progress



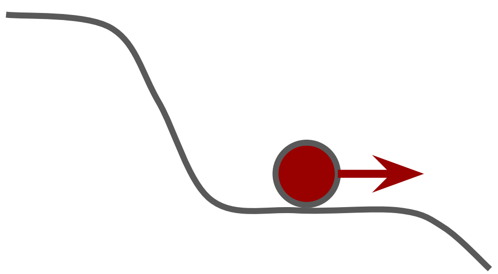
Problems of Stochastic Gradient Descent (SGD):

- Loss landscape can be ill-conditioned, i.e. changes very fast in some parameter directions and very slowly in other directions.
  - Large gradient in the steep direction → big steps → overshoot
  - Small gradient in flat direction → slow progress

- Gradients come from sampled mini-batches so they can be noisy!



Local minima



Saddle points

Problems of Stochastic Gradient Descent (SGD):

- Loss landscape can be ill-conditioned, i.e. changes very fast in some parameter directions and very slowly in other directions.
    - Large gradient in the steep direction → big steps → overshoot
    - Small gradient in flat direction → slow progress

- Gradients come from sampled mini-batches so they can be noisy!

- Parameters can have different scales
    - A single learning rate may not be optimal for all parameters

Stochastic Gradient Descent (SGD) + Momentum

Solution: SGD + Momentum:

- Build up "velocity of gradients" as a **running mean** of gradients
- Dampens oscillations in the gradients
- Decay rate ρ is usually set as 0.9 or 0.99

SGD:	**SGD + Momentum**:
while True **do**	**while** True **do**
Sample batch $\mathcal{B}$	Sample batch $\mathcal{B}$
$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$	$m_{t+1} \leftarrow \rho \cdot m_t + \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
end	$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot m_{t+1}$
	end

Stochastic Gradient Descent (SGD) + Momentum

Solution: SGD + Momentum:

- Build up "velocity of gradients" as a **running mean** of gradients
- Dampens oscillations in the gradients
- Decay rate ρ is usually set as 0.9 or 0.99

SGD:
while True **do**
Sample batch $\mathcal{B}$
$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
end

SGD + Momentum:
while True **do**
Sample batch $\mathcal{B}$
$m_{t+1} \leftarrow \rho \cdot m_t + \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot m_{t+1}$
end

SGD + Momentum (equivalent):
while True **do**
Sample batch $\mathcal{B}$
$m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
$\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

SGD + Momentum:
while True **do**

Sample batch $\mathcal{B}$

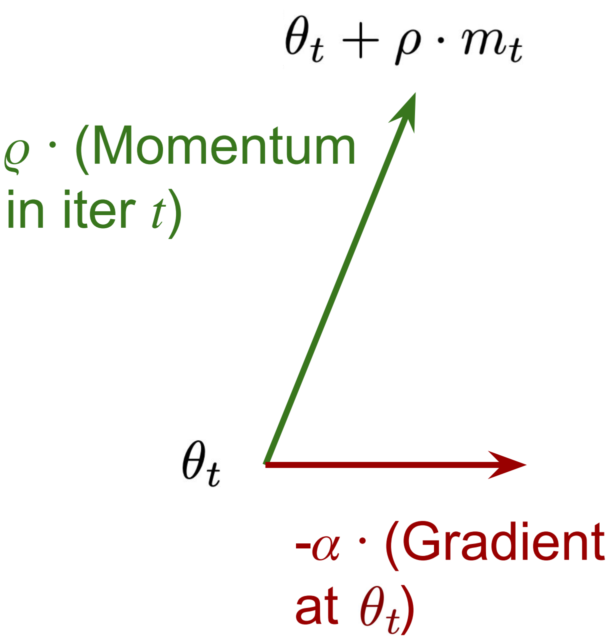
$m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$

$\theta_{t+1} \leftarrow \theta_t + m_{t+1}$

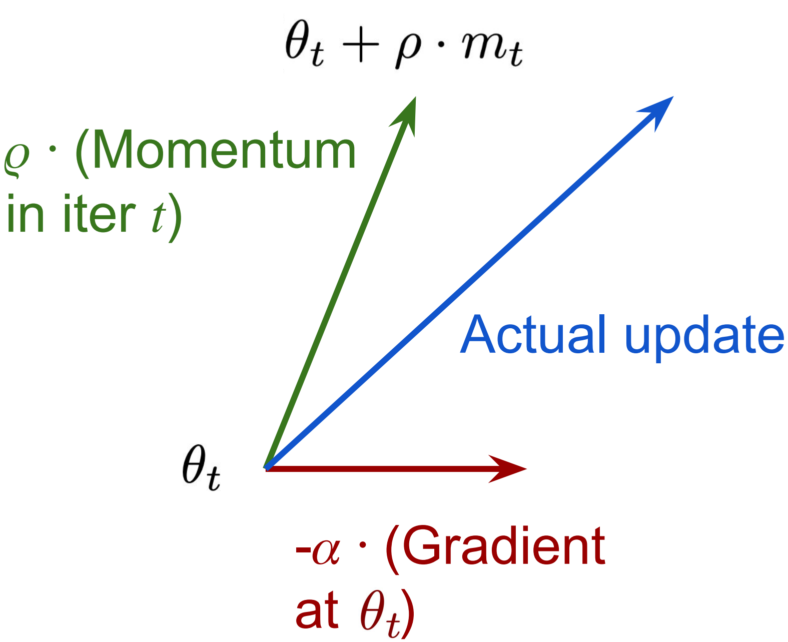
end

θ_t

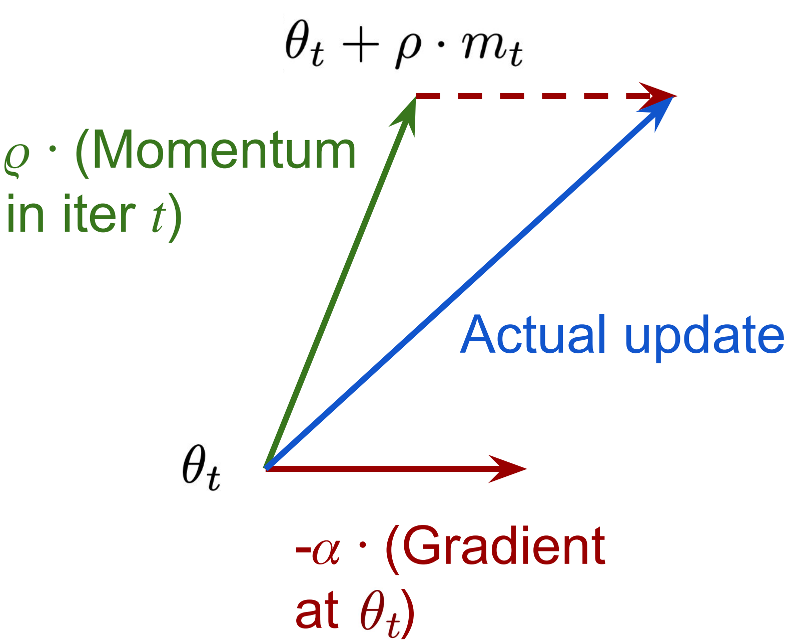
SGD + Momentum:
while True **do**
    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end



SGD + Momentum:
while True **do**
    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end



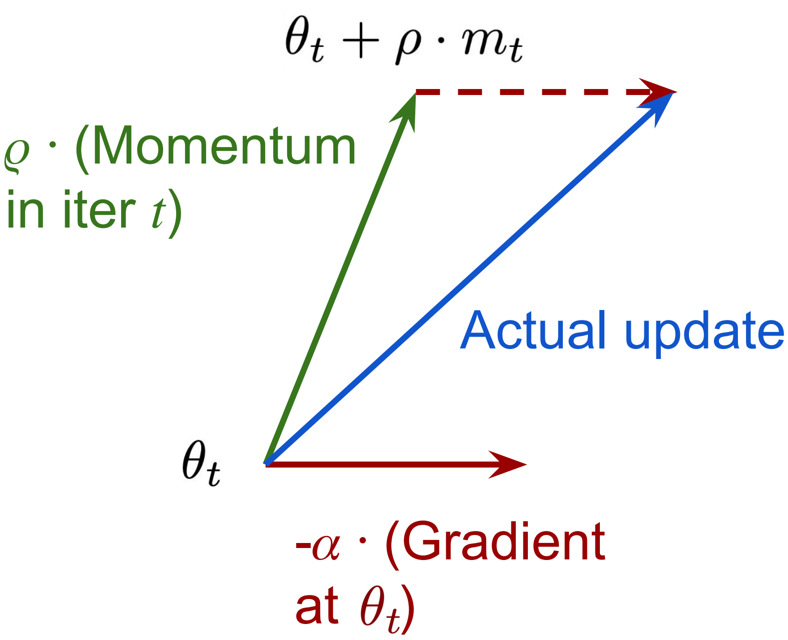
SGD + Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end



SGD + Momentum:
while True **do**

    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$

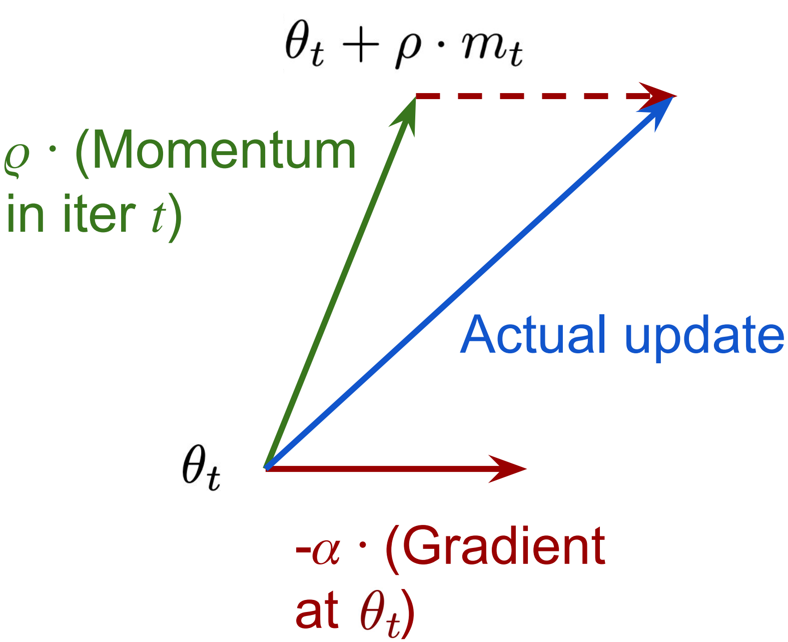
end



The actual update is essentially summation of

- ϱ · (Momentum in iter t)
- -α · (Gradient at θ_t)

SGD + Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end



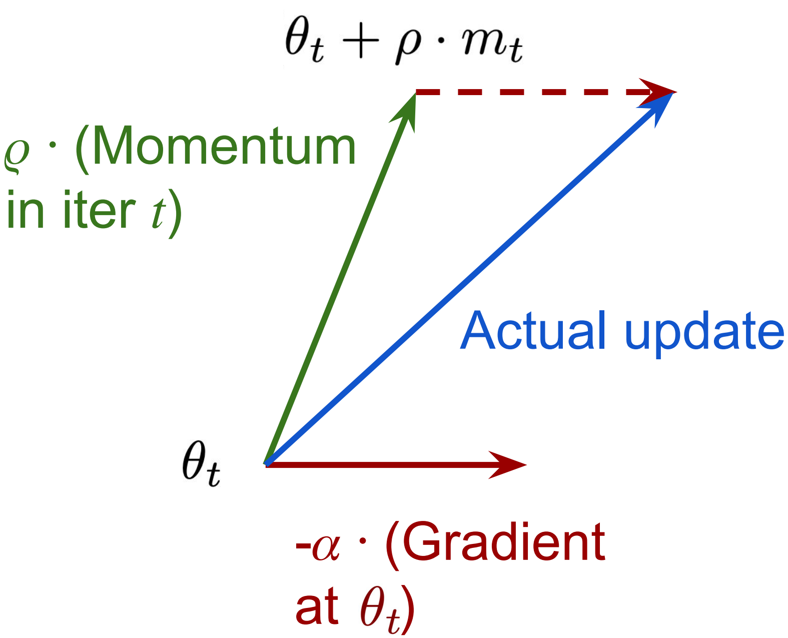
The actual update is essentially summation of

- ϱ · (Momentum in iter t)
- -α · (Gradient at θ_t)

Mismatch of where the gradient is evaluated!

Instead, where should the gradient be evaluated at?

SGD + Momentum:
while True **do**
    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end



The actual update is essentially summation of

- ϱ · (Momentum in iter t)
- -α · (Gradient at θ_t)

Mismatch of where the gradient is evaluated!

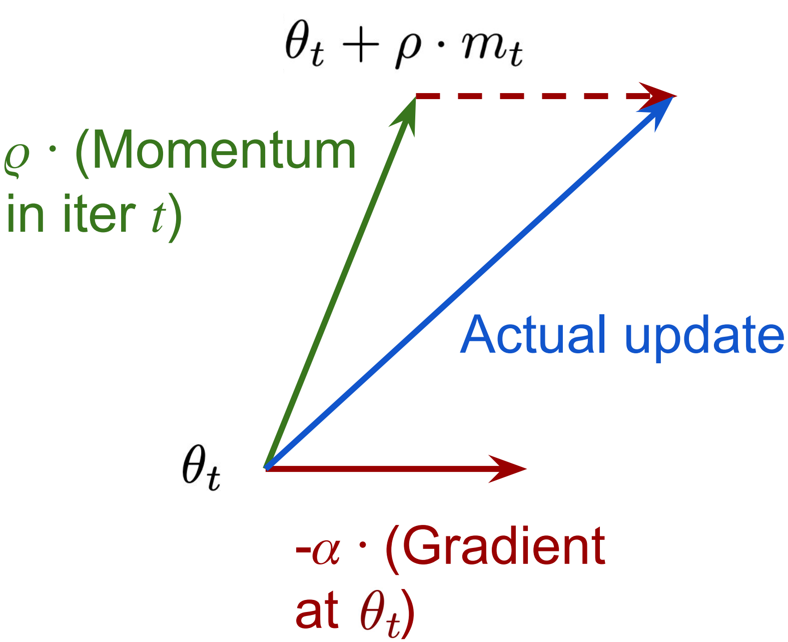
Instead, where should the gradient be evaluated at?

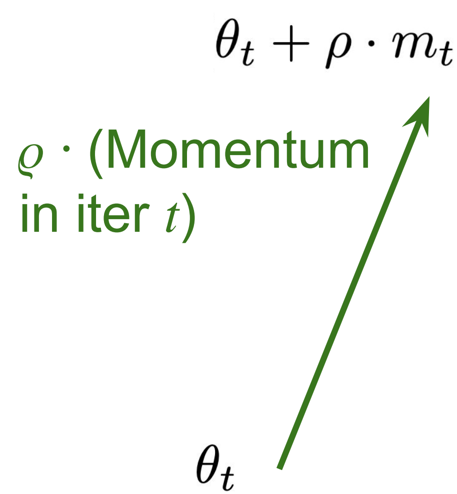
$\theta_t + \rho \cdot m_t$

Nesterov Momentum

SGD + Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

SGD + Nesterov Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot v_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t + \rho \cdot m_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

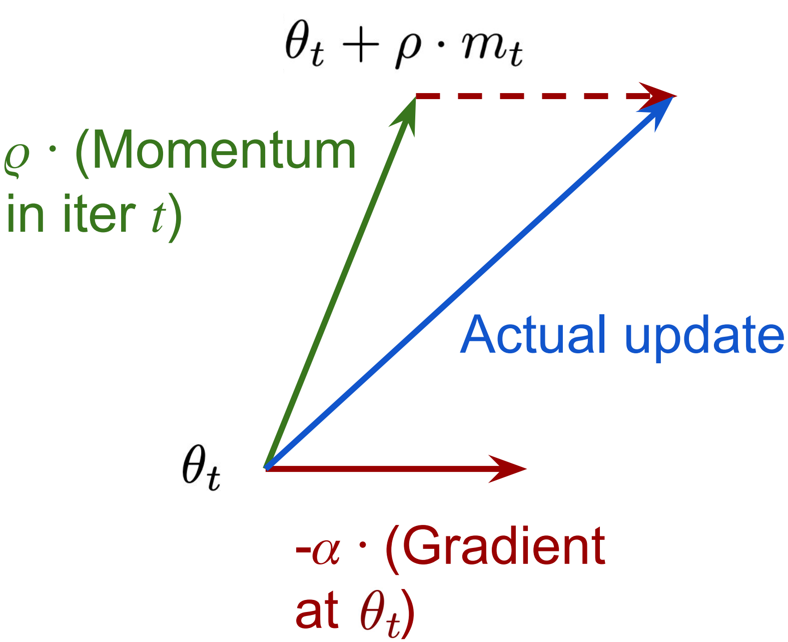


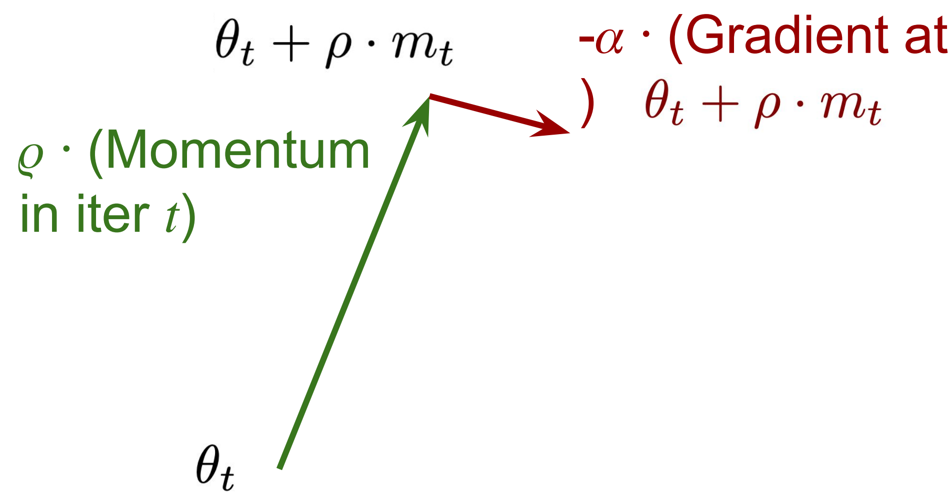


Nesterov Momentum

SGD + Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

SGD + Nesterov Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot v_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\boxed{\theta_t + \rho \cdot m_t})$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end
Lookahead point

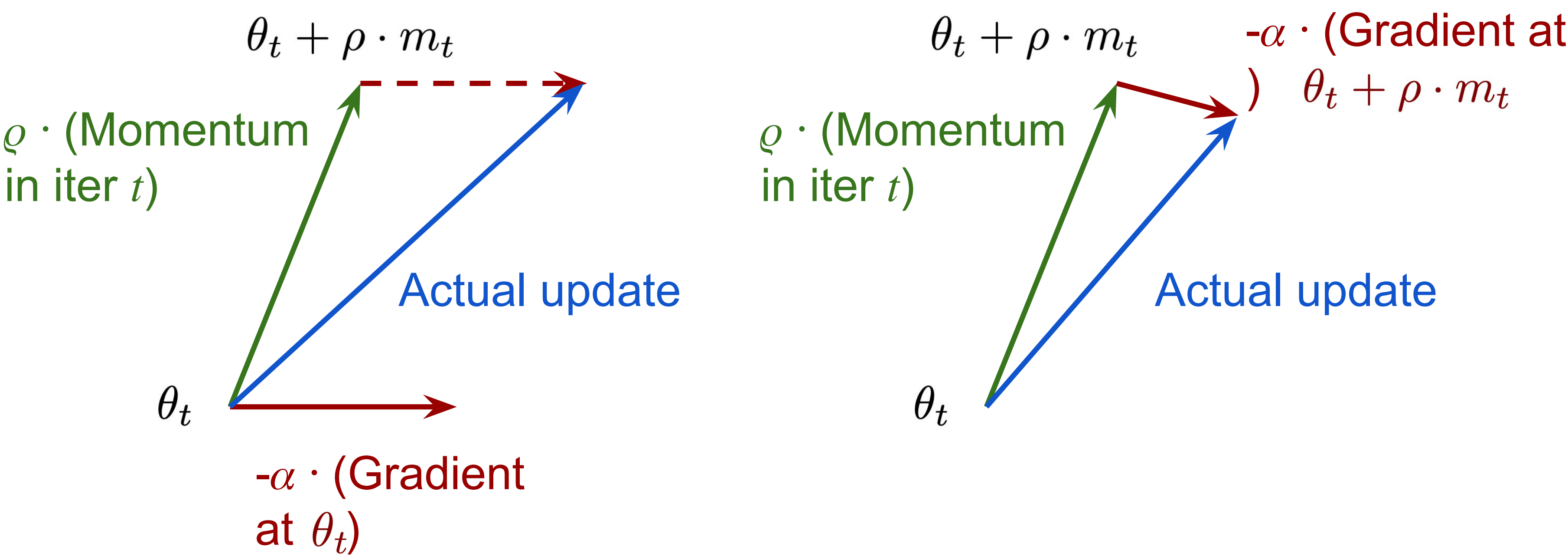




Nesterov Momentum

SGD + Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

SGD + Nesterov Momentum:
while True **do**
  Sample batch $\mathcal{B}$
  $m_{t+1} \leftarrow \rho \cdot v_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\boxed{\theta_t + \rho \cdot m_t})$
  $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
Lookahead point
end

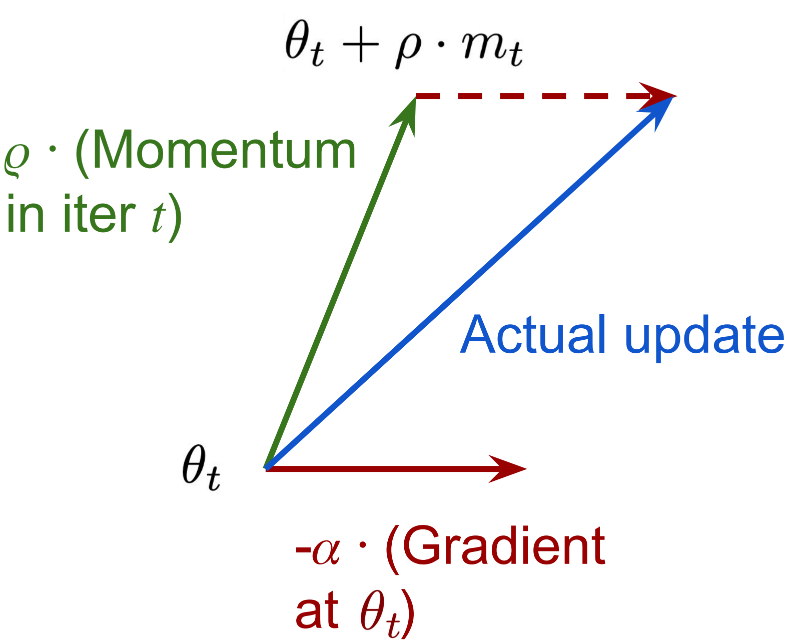


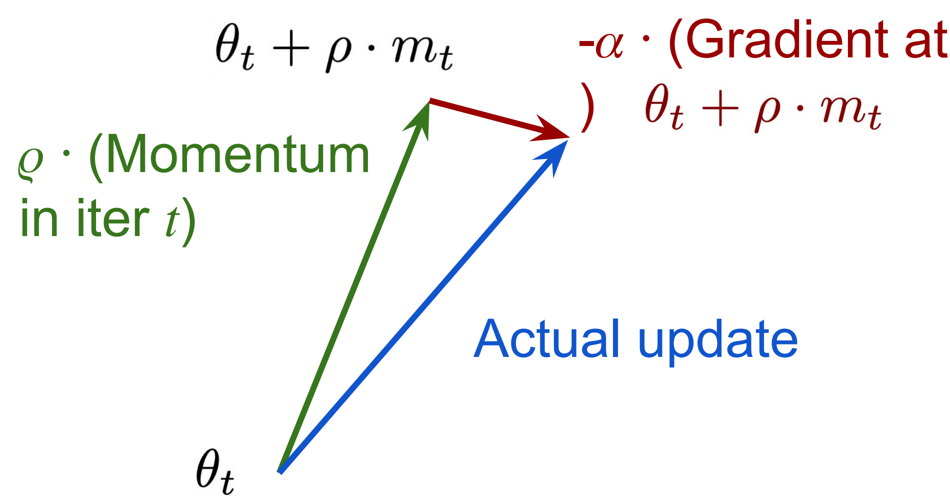
Nesterov Momentum

SGD + Momentum:
while True **do**
    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot m_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

SGD + Nesterov Momentum:
while True **do**
    Sample batch $\mathcal{B}$
    $m_{t+1} \leftarrow \rho \cdot v_t - \alpha \nabla_\theta \mathcal{L}_\mathcal{B}(\boxed{\theta_t + \rho \cdot m_t})$
    $\theta_{t+1} \leftarrow \theta_t + m_{t+1}$
end

Lookahead point





In most cases shows faster or more stable convergence than vanilla gradient

How to deal with parameters with different scales?

Adagrad

How to deal with parameters with different scales?

Initialize $v \leftarrow 0$
while True **do**
  Sample batch $\mathcal{B}$
  $v \leftarrow v + [\nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta_t)]^2$
  $$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_{\mathcal{B}}(\theta_t)}{\sqrt{v} + \epsilon}$$
end

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

Adagrad

How to deal with parameters with different scales?

Initialize $v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$\boxed{v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}$$

end

Historical element-wise sum of squares in each dimension of gradient

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

Adagrad

How to deal with parameters with different scales?

Initialize $v \leftarrow 0$
while True **do**
  Sample batch $\mathcal{B}$
  $v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$
  $\boxed{\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}}$
end

Element-wise scaling of the gradient based on the historical sum of squares in each dimension

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

Adagrad

How to deal with parameters with different scales?

Initialize $v \leftarrow 0$
while True **do**
Sample batch $\mathcal{B}$
$v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$
$$\boxed{\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}}$$
end

Element-wise scaling of the gradient based on the historical sum of squares in each dimension

Progress along historically **steep** directions is **damped**
progress along historically **flat** directions is **accelerated**

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

Adagrad

How to deal with parameters with different scales?

Initialize $v \leftarrow 0$
while True **do**
  Sample batch $\mathcal{B}$
  $v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$
  $\boxed{\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}}$
end

Element-wise scaling of the gradient based on the historical sum of squares in each dimension

Progress along historically **steep** directions is **damped**
progress along historically **flat** directions is **accelerated**

However, when $t \rightarrow +\infty, v \rightarrow +\infty$, the update will decay to zero!

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

RMSProp

How to deal with parameters with different scales?

Adagrad

Initialize $v \leftarrow 0$
while True **do**
    Sample batch $\mathcal{B}$
    $v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$
    $$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}$$
end

RMSProp

Initialize $v \leftarrow 0$
while True **do**
    Sample batch $\mathcal{B}$
    $v \leftarrow \rho \cdot v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$
    $$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}$$
end

Tieleman and Hinton, 2012

RMSProp

How to deal with parameters with different scales?

Adagrad

Initialize $v \leftarrow 0$
while True **do**
Sample batch $\mathcal{B}$
$\boxed{v \leftarrow v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$
$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}$$
end

From **accumulation** to **running mean** of element-wise square of gradient

RMSProp

Initialize $v \leftarrow 0$
while True **do**
Sample batch $\mathcal{B}$
$\boxed{v \leftarrow \rho \cdot v + [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$
$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}{\sqrt{v} + \epsilon}$$
end

Tieleman and Hinton, 2012

Adam (almost there)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Adam (almost there)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Initialize $m \leftarrow 0, v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$$m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)$$

$$v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2$$

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon}$$

end

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (almost there)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Initialize $m \leftarrow 0, v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$$\boxed{m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}$$ Momentum term

$$\boxed{v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$$ Adagrad/RMSProp term

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon}$$

end

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (almost there)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Initialize $m \leftarrow 0, v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$$\boxed{m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}$$ Momentum term

$$\boxed{v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$$ Adagrad/RMSProp term

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon}$$

end

A good starting point: $\beta_1 = 0.9, \beta_2 = 0.999$

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Initialize $m \leftarrow 0, v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$$\boxed{m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}$$

Momentum term

$$\boxed{v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$$

Adagrad/RMSProp term

$$m \leftarrow \frac{m}{1 - \beta_1^t}$$

$$v \leftarrow \frac{v}{1 - \beta_2^t}$$

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon}$$

end

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form)

Can we combine the ideas of both:

- **(Nesterov) momentum:** running mean of gradient
- **Adagrad/RMSProp:** running mean of element-wise square of gradient

Initialize $m \leftarrow 0, v \leftarrow 0$

while True **do**

Sample batch $\mathcal{B}$

$$\boxed{m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)}$$

Momentum term

$$\boxed{v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot [\nabla_\theta \mathcal{L}_\mathcal{B}(\theta_t)]^2}$$

Adagrad/RMSProp term

$$\boxed{\begin{aligned} m &\leftarrow \frac{m}{1 - \beta_1^t} \\ v &\leftarrow \frac{v}{1 - \beta_2^t} \end{aligned}}$$

Bias correction (<u>**why is it needed?**</u>)

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon}$$

end

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Adam (full form):

Where do the biases come from?
Both running means start at 0, and it needs time to warm up!

Suppose the gradient is roughly stationary with mean $\mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$
We will have

$$\mathbb{E}[m_1] = \mathbb{E}[\beta_1 \cdot m_0] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$$

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form):

Where do the biases come from?
Both running means start at 0, and it needs time to warm up!

Suppose the gradient is roughly stationary with mean $\mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$
We will have

$$\begin{aligned}\mathbb{E}[m_1] &= \mathbb{E}[\beta_1 \cdot m_0] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ &= (1-\beta_1) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]\end{aligned}$$

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form):

Where do the biases come from?
Both running means start at 0, and it needs time to warm up!

Suppose the gradient is roughly stationary with mean $\mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$
We will have

$$\begin{aligned}\mathbb{E}[m_1] &= \mathbb{E}[\beta_1 \cdot m_0] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ &= (1-\beta_1) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ \mathbb{E}[m_2] &= \mathbb{E}[\beta_1 \cdot m_1] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]\end{aligned}$$

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form):

Where do the biases come from?
Both running means start at 0, and it needs time to warm up!

Suppose the gradient is roughly stationary with mean $\mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$
We will have

$$\begin{aligned}\mathbb{E}[m_1] &= \mathbb{E}[\beta_1 \cdot m_0] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ &= (1-\beta_1) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ \mathbb{E}[m_2] &= \mathbb{E}[\beta_1 \cdot m_1] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\ &= (1-\beta_1^2) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]\end{aligned}$$

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form):

Where do the biases come from?
Both running means start at 0, and it needs time to warm up!

Suppose the gradient is roughly stationary with mean $\mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]$
We will have

$$\begin{aligned}
\mathbb{E}[m_1] &= \mathbb{E}[\beta_1 \cdot m_0] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\
&= (1-\beta_1) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\
\mathbb{E}[m_2] &= \mathbb{E}[\beta_1 \cdot m_1] + \mathbb{E}[(1-\beta_1) \cdot \nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\
&= (1-\beta_1^2) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)] \\
\cdots \\
\mathbb{E}[m_t] &= (1-\beta_1^t) \cdot \mathbb{E}[\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)]
\end{aligned}$$

So it needs to be scaled by $1/(1-\beta_1^t)$

Another way to view it: the influence of the zero initialization decays exponentially at rates β_1^t

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

Parameters:

- Determines the model's output

Hyperparameters:

- Determines how the model is trained

Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

Parameters:

- Determines the model's output
- Learned automatically during training

Hyperparameters:

- Determines how the model is trained
- Set before training

Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

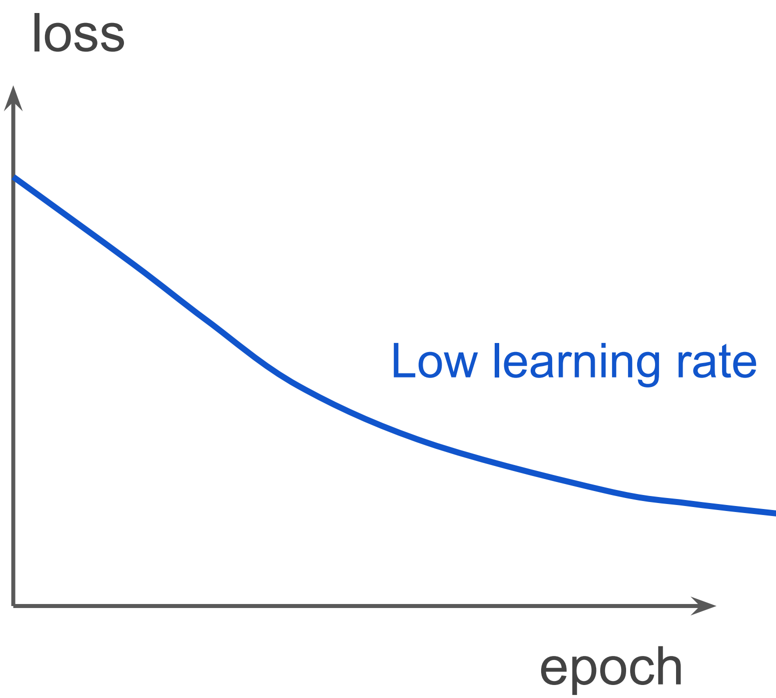
Parameters:

- Determines the model's output
- Learned automatically during training

Hyperparameters:

- Determines how the model is trained
- Set before training

What learning rate is best to use?



Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

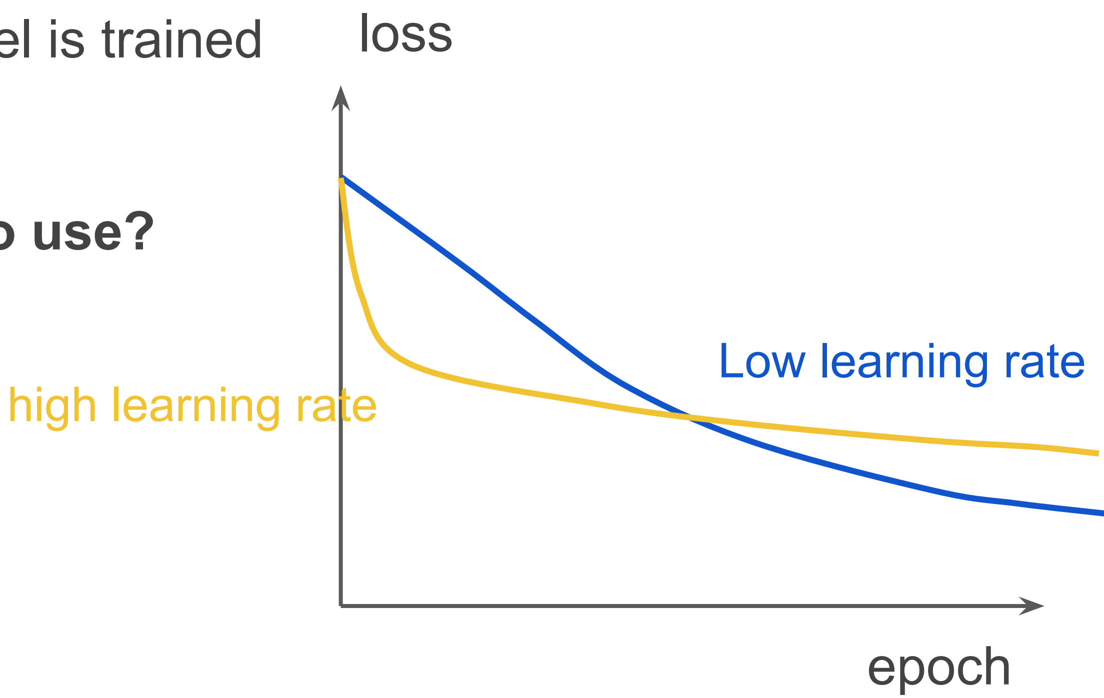
Parameters:

- Determines the model's output
- Learned automatically during training

Hyperparameters:

- Determines how the model is trained
- Set before training

What learning rate is best to use?



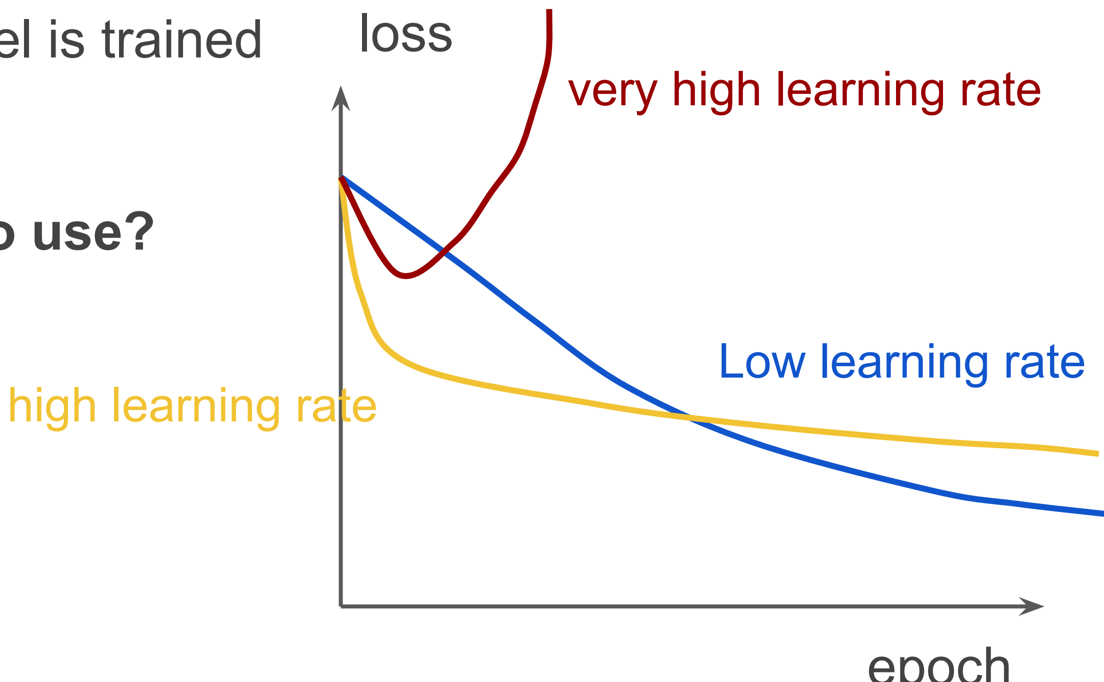
Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

Parameters:
- Determines the model's output
- Learned automatically during training

Hyperparameters:
- Determines how the model is trained
- Set before training

What learning rate is best to use?



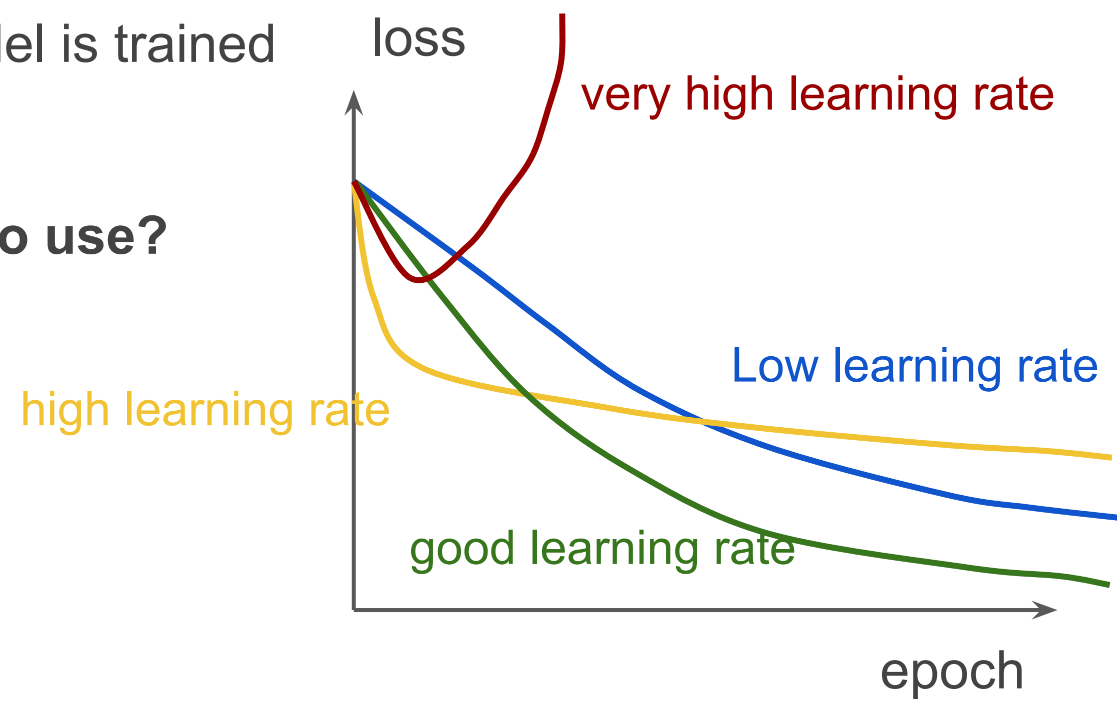
Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **<u>hyperparameter</u>**.

Parameters:
- Determines the model's output
- Learned automatically during training

Hyperparameters:
- Determines how the model is trained
- Set before training

What learning rate is best to use?



Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

A good idea:
- Large learning rate at the start of training
- Decrease / decay the learning rate as training continues

Learning rate scheduling

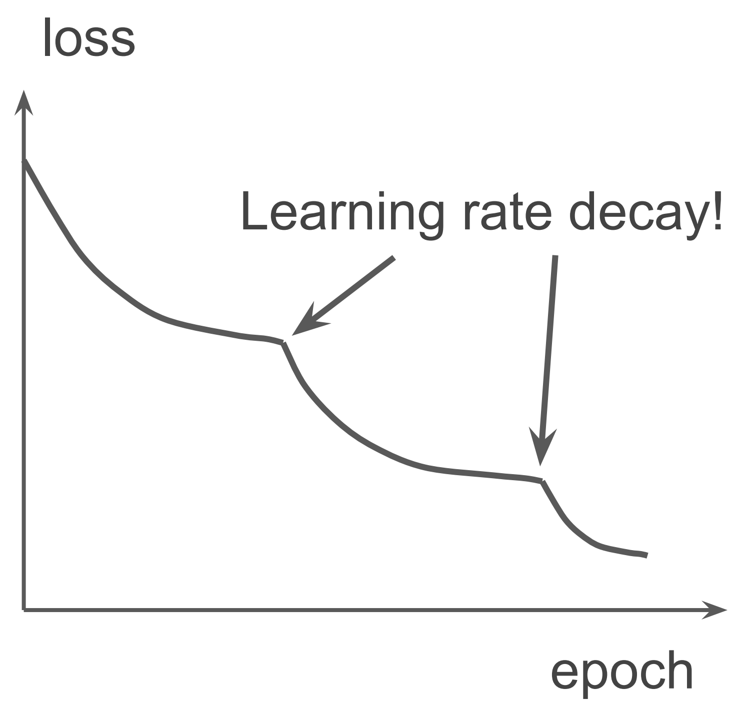
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

A good idea:

- Large learning rate at the start of training
- Decrease / decay the learning rate as training continues

Learning rate decay options:

- **Step decay**: decay the learning rate by half every few epochs



Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

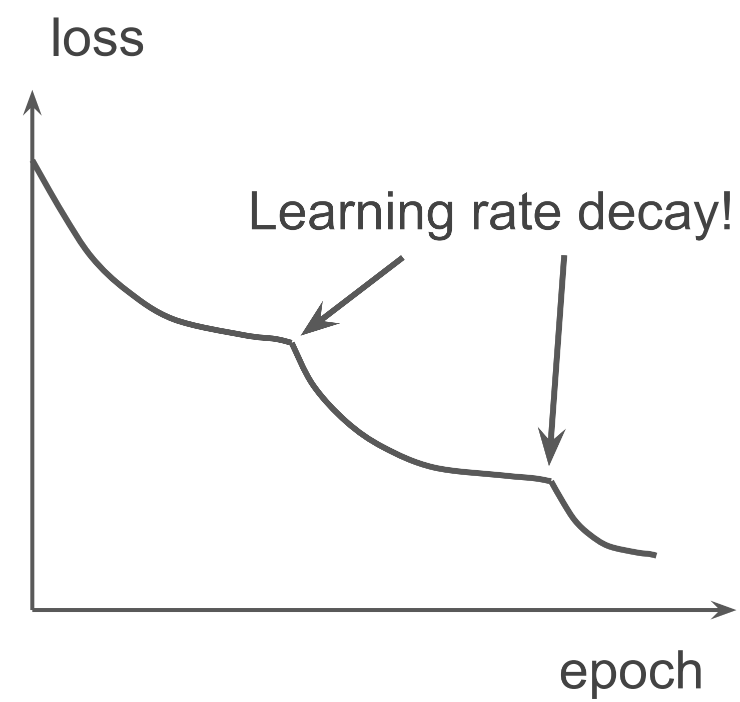
A good idea:

- Large learning rate at the start of training
- Decrease / decay the learning rate as training continues

Learning rate decay options:

- **Step decay**: decay the learning rate by half every few epochs
- **Exponential decay**:

$$\alpha = \alpha_0 e^{-kt}$$



Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **hyperparameter**.

A good idea:

- Large learning rate at the start of training
- Decrease / decay the learning rate as training continues

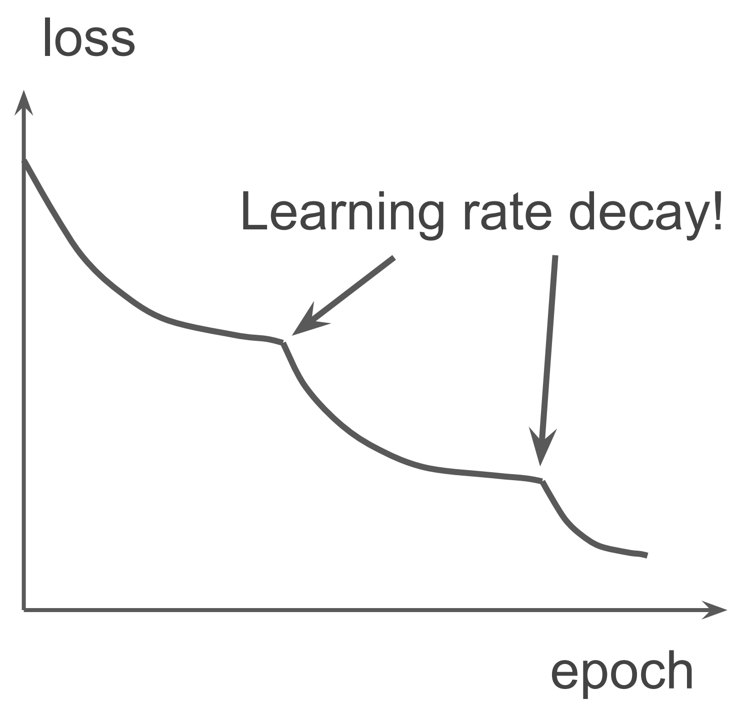
Learning rate decay options:

- **Step decay**: decay the learning rate by half every few epochs
- **Exponential decay**:

$$\alpha = \alpha_0 e^{-kt}$$

- **1/t decay**:

$$\alpha = \alpha_0 / (1 + kt)$$



Learning rate scheduling

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a **<u>hyperparameter</u>**.

A good idea:

- Large learning rate at the start of training
- Decrease / decay the learning rate as training continues

Learning rate decay options:

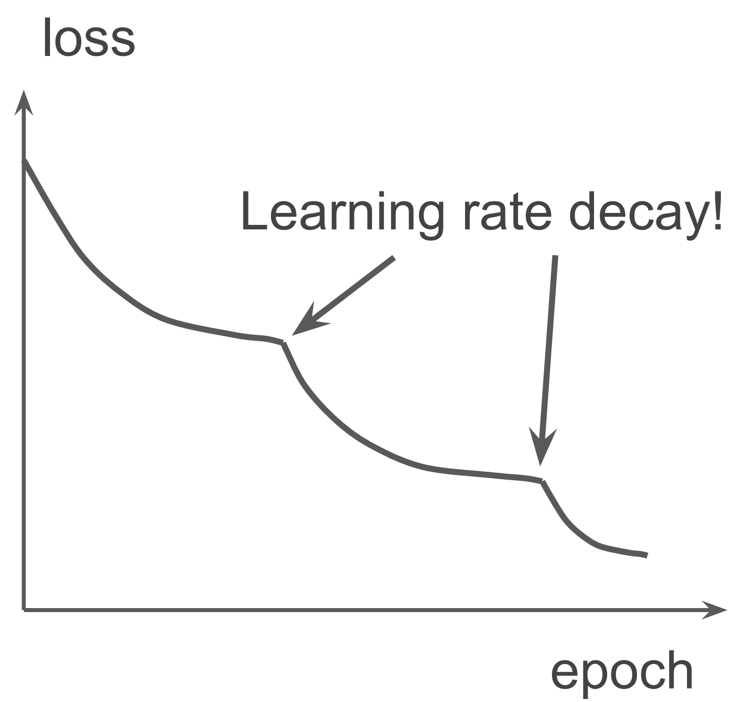
- **Step decay**: decay the learning rate by half every few epochs
- **Exponential decay**:

$$\alpha = \alpha_0 e^{-kt}$$

- **1/t decay**:

$$\alpha = \alpha_0 / (1 + kt)$$

Critical for SGD+Momentum, less common with Adam



Hyperparameter selection

Neural networks have a lot of hyperparameters, for example:

- Number of layers, number of neurons in each layer
- (Initial) learning rate α_0 and learning rate scheduling parameters
- Batch size $|\mathcal{B}|$
- Number of epochs
- Adam parameters $\beta_1, \beta_2, \epsilon$
- …

Hyperparameter selection

Neural networks have a lot of hyperparameters, for example:

- Number of layers, number of neurons in each layer
- (Initial) learning rate α_0 and learning rate scheduling parameters
- Batch size $|\mathcal{B}|$
- Number of epochs
- Adam parameters $\beta_1, \beta_2, \epsilon$
- …

How do we find the optimal set of hyperparameters to use?

Hyperparameter selection

Neural networks have a lot of hyperparameters, for example:
- Number of layers, number of neurons in each layer
- (Initial) learning rate α_0 and learning rate scheduling parameters
- Batch size $|\mathcal{B}|$
- Number of epochs
- Adam parameters $\beta_1, \beta_2, \epsilon$
- …

How do we find the optimal set of hyperparameters to use?

How do we do it given a limited amount of data?
- **Training data**: accessible during training, used to learn parameters
- **Test data**: not accessible during training, only for final evaluation

Hyperparameter selection

Idea 1: train the neural network on training data, and directly evaluate it on test data

Hyperparameter selection

Idea 1: train the neural network on training data, and directly evaluate it on test data

We will have no idea how the neural network will perform on new data

Hyperparameter selection

Idea 1: train the neural network on training data, and directly evaluate it on test data

We will have no idea how the neural network will perform on new data

Train	**Test**

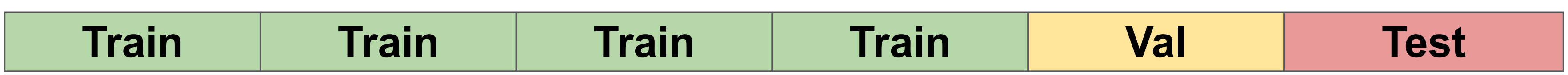
Idea 2: Split training data into train and validation sets:

Train	**Train**	**Train**	**Train**	**Val**	**Test**

Hyperparameter selection

Idea 1: train the neural network on training data, and directly evaluate it on test data
We will have no idea how the neural network will perform on new data

Idea 2: Split training data into train and validation sets:
- Train neural network on training data; choose hyperparameters that work best on validation data.
- Train neural network on both training and validation data with the best hyperparameters, and final evaluation on test data.



Hyperparameter selection

Idea 1: train the neural network on training data, and directly evaluate it on test data
We will have no idea how the neural network will perform on new data

Train	**Test**

Idea 2: Split training data into train and validation sets:

- Train neural network on training data; choose hyperparameters that work best on validation data.
- Train neural network on both training and validation data with the best hyperparameters, and final evaluation on test data.

Train	**Train**	**Train**	**Train**	**Val**	**Test**

<u>Can we use the given training data more efficiently?</u>

Hyperparameter selection

Idea 3 (cross-validation): Split training data into **folds**

- Try each fold as validation data (the rest as training data) and average the results, find the hyperparameters that work best
- Train neural network on both training and validation data with the best hyperparameters, and final evaluation on test data.

Train	**Train**	**Train**	**Train**	**Val**	**Test**
Train	**Train**	**Train**	**Val**	**Train**	**Test**
Train	**Train**	**Val**	**Train**	**Train**	**Test**
Train	**Val**	**Train**	**Train**	**Train**	**Test**
Val	**Train**	**Train**	**Train**	**Train**	**Test**

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

Typically performed on a log scale!

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

Typically performed on a log scale!

<u>For example (learning rate):</u>
Stage 1: [0.1, 0.01, 0.001, 0.0001, 0.00001];

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

Typically performed on a log scale!

For example (learning rate):
Stage 1: [0.1, 0.01, **0.001**, **0.0001**, 0.00001];

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

Typically performed on a log scale!

<u>For example (learning rate):</u>
Stage 1: [0.1, 0.01, **0.001**, **0.0001**, 0.00001];
Stage 2: [**0.001**, 0.0005, 0.0002, **0.0001**];

Hyperparameter selection

Coarse-to-fine search for *one* hyperparameter:
Stage 1 (coarse search): train only a few epochs to get rough idea of reasonable range
Stage 2 (fine search): Train longer and search more densely within that range

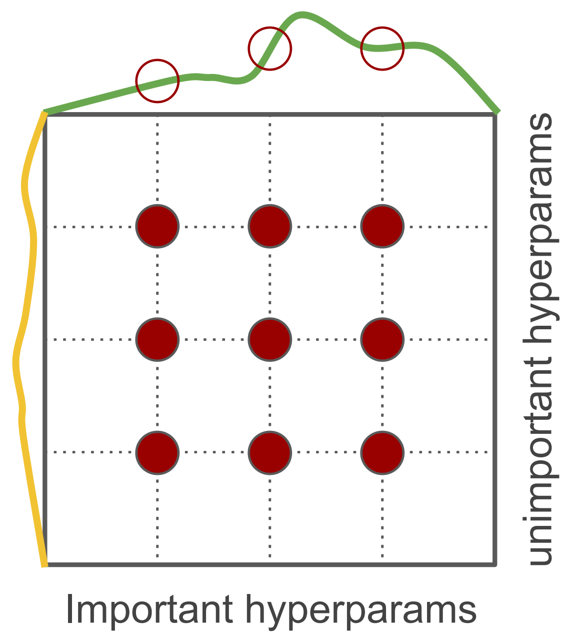
Typically performed on a log scale!

For example (learning rate):
Stage 1: [0.1, 0.01, **0.001**, **0.0001**, 0.00001];
Stage 2: [0.001, **0.0005**, **0.0002**, 0.0001];
…

Hyperparameter selection

Search for *multiple* hyperparameter:

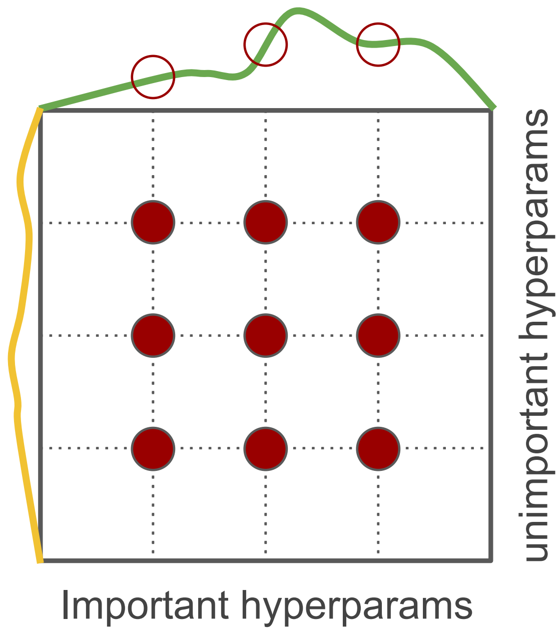
Grid search



Hyperparameter selection

Search for *multiple* hyperparameter:

Grid search

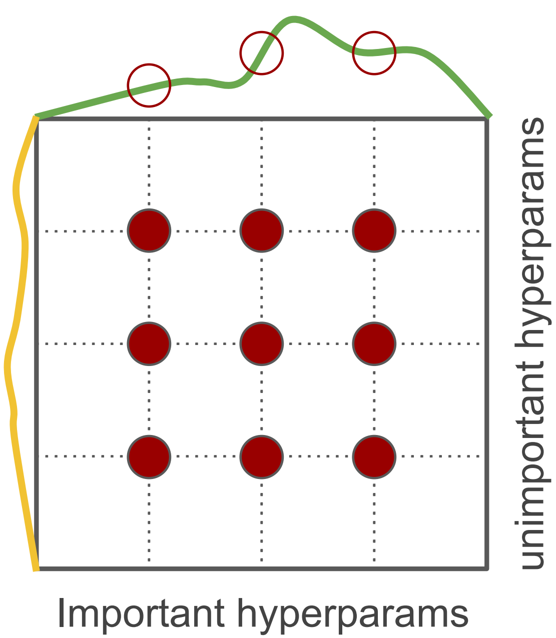


- Scales exponentially with dimension
- Wastes search on unimportant hyperparameters

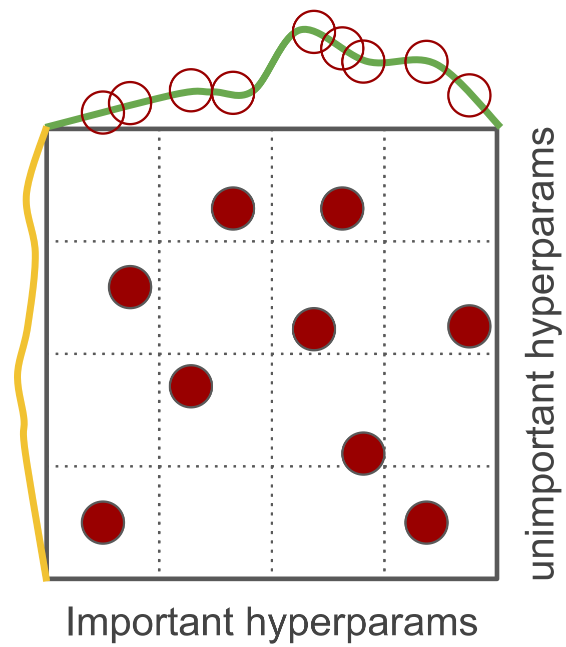
Hyperparameter selection

Search for *multiple* hyperparameter:

Grid search



Random search



- Scales exponentially with dimension
- Wastes search on unimportant hyperparameters

- **Finds good values faster when only a few hyperparameters matter**

Random Search for Hyper-Parameter Optimization, Bergstra and Bengio, 2012

Thanks