# ОСНОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ, НЕЙРОННИХ МЕРЕЖ та ГЛИБОКОГО НАВЧАННЯ

## Модуль 7. TensorFlow / KERAS

## Лекція 7.3. Обчислювальний граф

# Обчислювальний граф

**Обчислювальний граф - орієнтований граф, що складається з вершин, що відповідають операціям алгоритму, та спрямованих дуг, що відповідають передачі даних (змінні = результати операцій, шо передаються як аргументи іншим операціям) між ними.**


**!!! Не БЛОК - СХЕМА**

# Обчислювальний граф

**Обчислювальний граф — тип графу, який можна використовується для представлення та обчислення математичних виразів.**

**Обчислювальний граф — база мови, яка описує *МОДЕЛІ* у ML (DL), що забезпечує повний функціональний опис необхідних обчислень.**

**Використовується для двох різних типів обчислень:**

- **Форвардне обчислення (Forward).**
- **Зворотне обчислення (Backward, Backpropagation).**

# Обчислювальний граф

**Базова термінологія.**

**Змінна представлена вузлом на графі. Це може бути скаляр, вектор, матриця, тензор або навіть інший тип змінної.**

**Аргумент функції та залежність даних представлені ребром. Вони схожі на покажчики вузлів.**

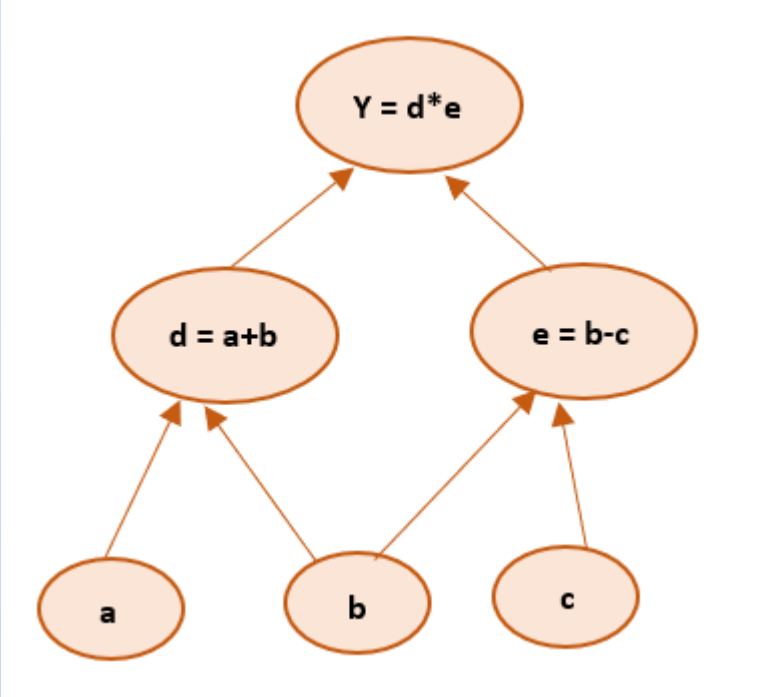
**Операція - проста функція однієї або кількох змінних. Існує набір дозволених операцій. Функції, які є більш складними, ніж ці операції в цьому наборі, можуть бути представлені шляхом поєднання кількох операцій.**

# Обчислювальний граф

**Приклад:** $Y = (a + b) * (b - c)$

$$d = (a + b); \quad e = (b - c)$$
$$Y = d * e$$

**Три операції: додавання, віднімання та множення.**
**Обчислювальний граф: три вузли, кожен з яких виконує різні операції разом із вхідними змінними. Напрямок стрілок показує напрямок передачі змінних до інших вузлів.**



# Обчислювальний граф

**Ланцюгове правило**
**(правило диференціювання складної функції)**

$$y = F(f(x))$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial f} * \frac{\partial f}{\partial x}$$

# Обчислювальний граф

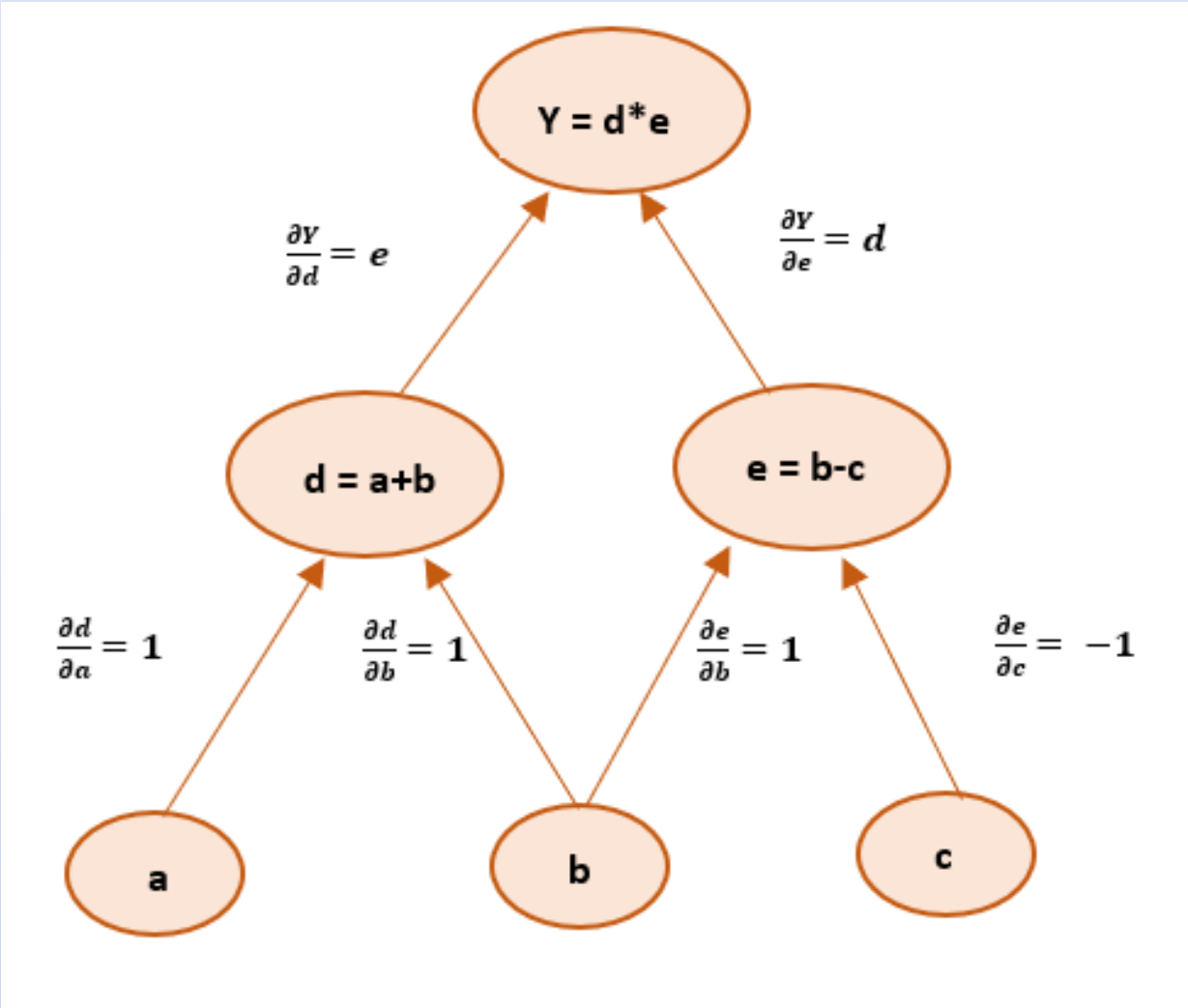
**Обчислення нейронної мережі організовано за етапом прямого розповсюдження, на якому обчислюється вихід нейронної мережі, за яким слідує крок зворотного розповсюдження, який використовується для обчислення градієнтів (похідних).**

$$Y = d * e \quad \rightarrow \quad \frac{\partial Y}{\partial d} = e; \qquad \frac{\partial Y}{\partial e} = d$$

$$d = (a + b) \rightarrow \frac{\partial d}{\partial a} = 1; \quad \frac{\partial d}{\partial b} = 1$$

$$e = (b - c) \rightarrow \frac{\partial e}{\partial b} = 1; \quad \frac{\partial e}{\partial c} = -1$$

# Обчислювальний граф



# Обчислювальний граф

**Тоді компоненти градієнту**

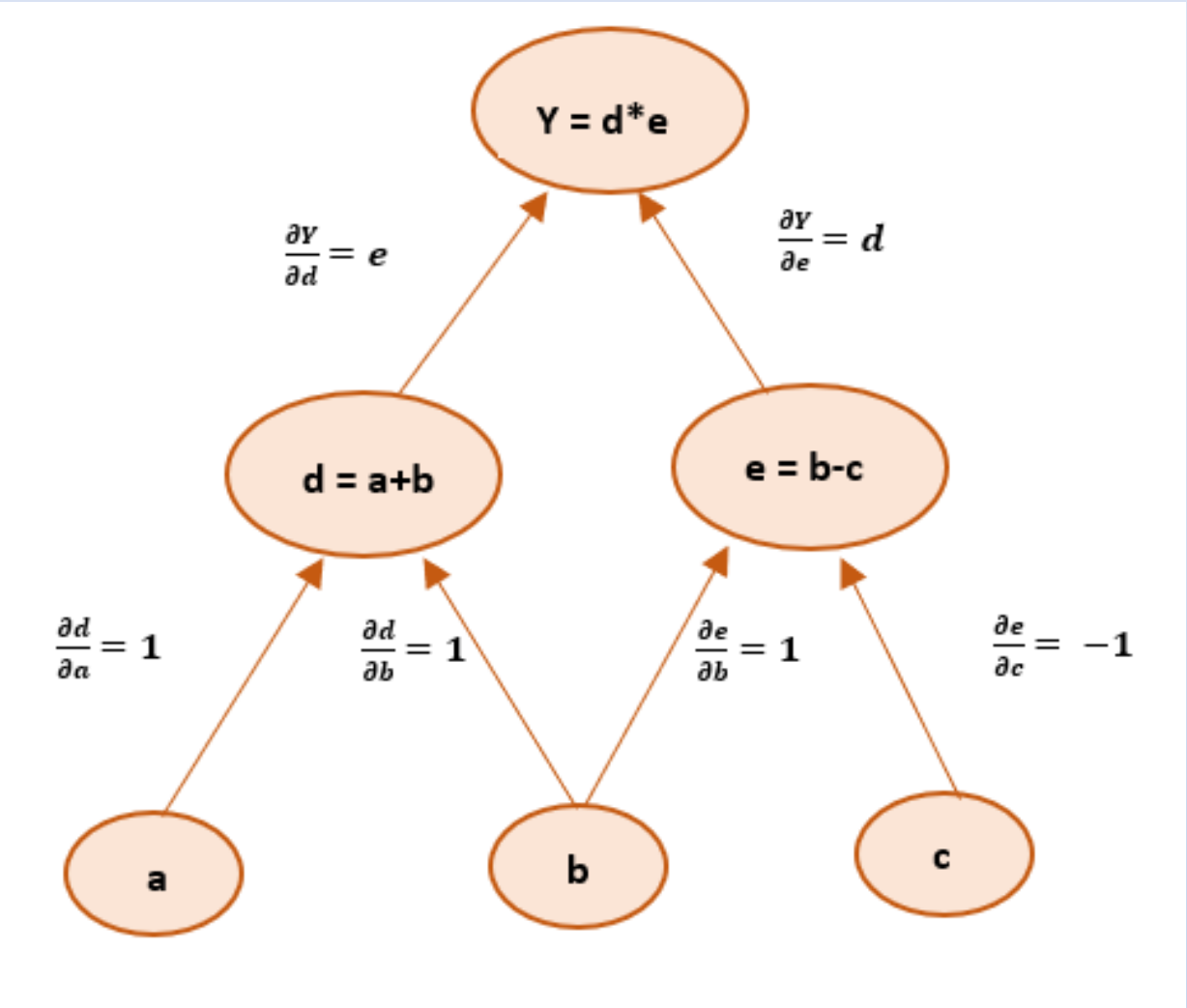
$$\frac{\partial Y}{\partial a} = \frac{\partial Y}{\partial d} * \frac{\partial d}{\partial a} = e * 1 = e$$

$$\frac{\partial Y}{\partial c} = \frac{\partial Y}{\partial e} * \frac{\partial e}{\partial c} = d * -1 = -d$$

$$\frac{\partial Y}{\partial b} = \frac{\partial d}{\partial b} * e + \frac{\partial e}{\partial b} * d = 1 * e + 1 * d = e + d$$

$$= (b - c) + (a + b) = a + 2b - c$$

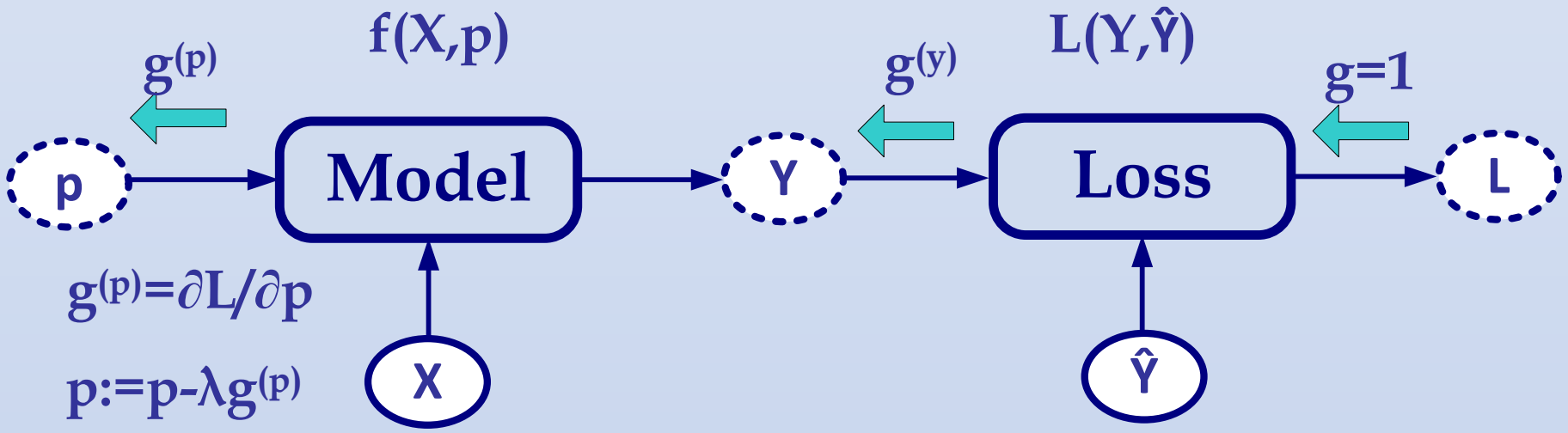
# Обчислювальний граф




$$\frac{\partial Y}{\partial a} = e \qquad \frac{\partial Y}{\partial b} = a + 2b - c \qquad \frac{\partial Y}{\partial c} = -d$$

# Обчислювальний граф Forward←→Backward

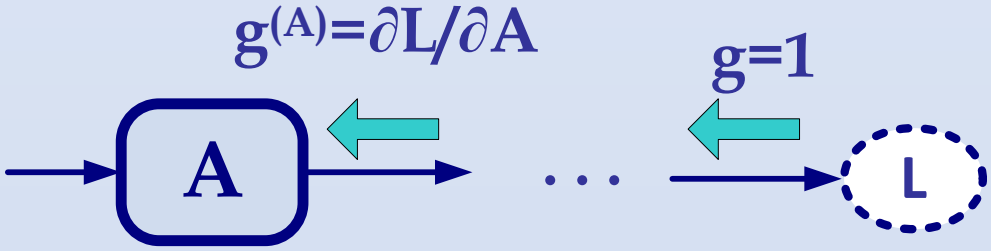
**Процес навчання → пошук параметрів, які мінімізують втрати (Loss)**



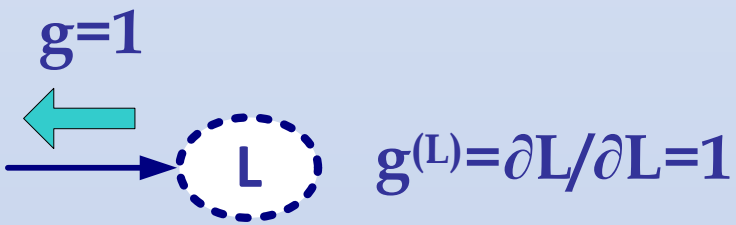

**Загальний підхід → використання методів градієнтного методу (gradient descent))**

# Backpropagation

## Зворотне поширення помилки




**Вхідний у будь-який вузол A градієнт $g^{(A)}$, має сенс похідної цільової величини (кореня графу) за значенням у цьому вузлі: $g^{(A)} = \partial L / \partial A$**

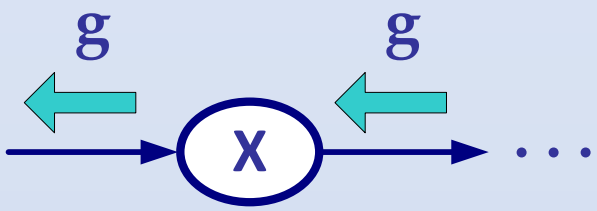



**Початковий градієнт, що входить і виходить із кореня графу L дорівнює одиниці, оскільки за визначенням це $\partial L / \partial L = 1$.**

# Backpropagation

## Зворотне поширення помилки

Через вузли змінних градієнт проходить без змін, а вузли констант "не заходить".

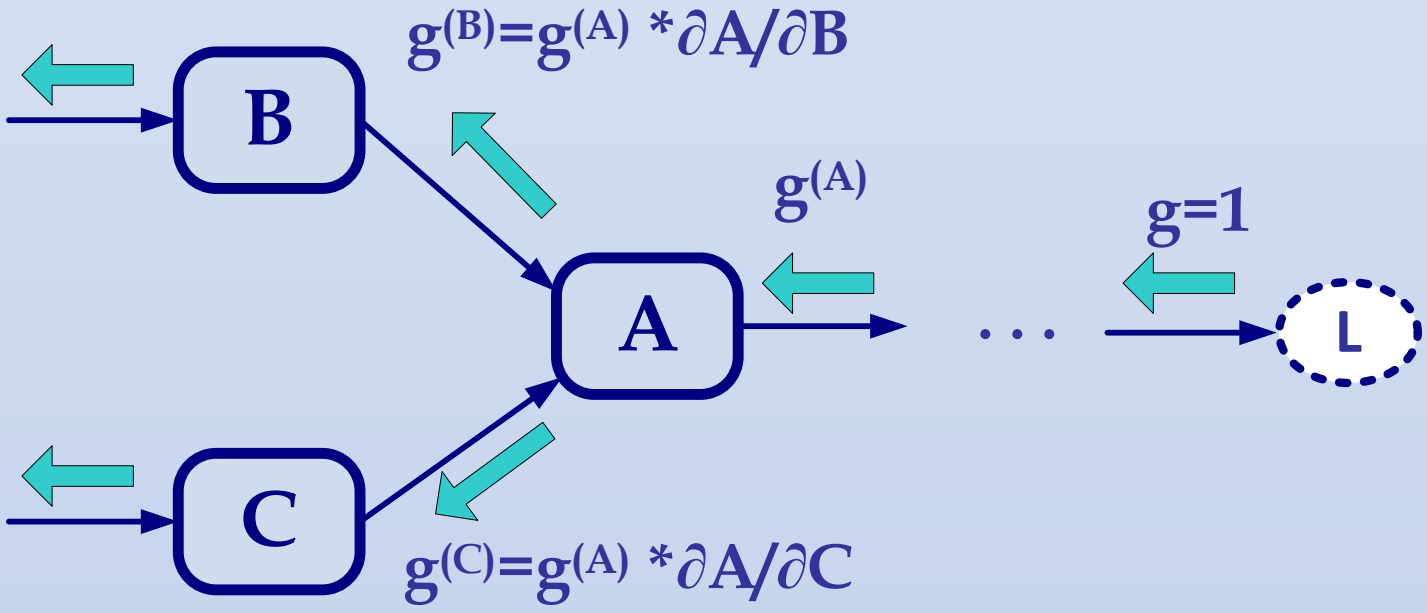



Тензорна розмірність градієнта, що входить у вузол, збігається з розмірністю тензора, одержуваного цим вузлом з його виході.

# Backpropagation

## Зворотне поширення помилки

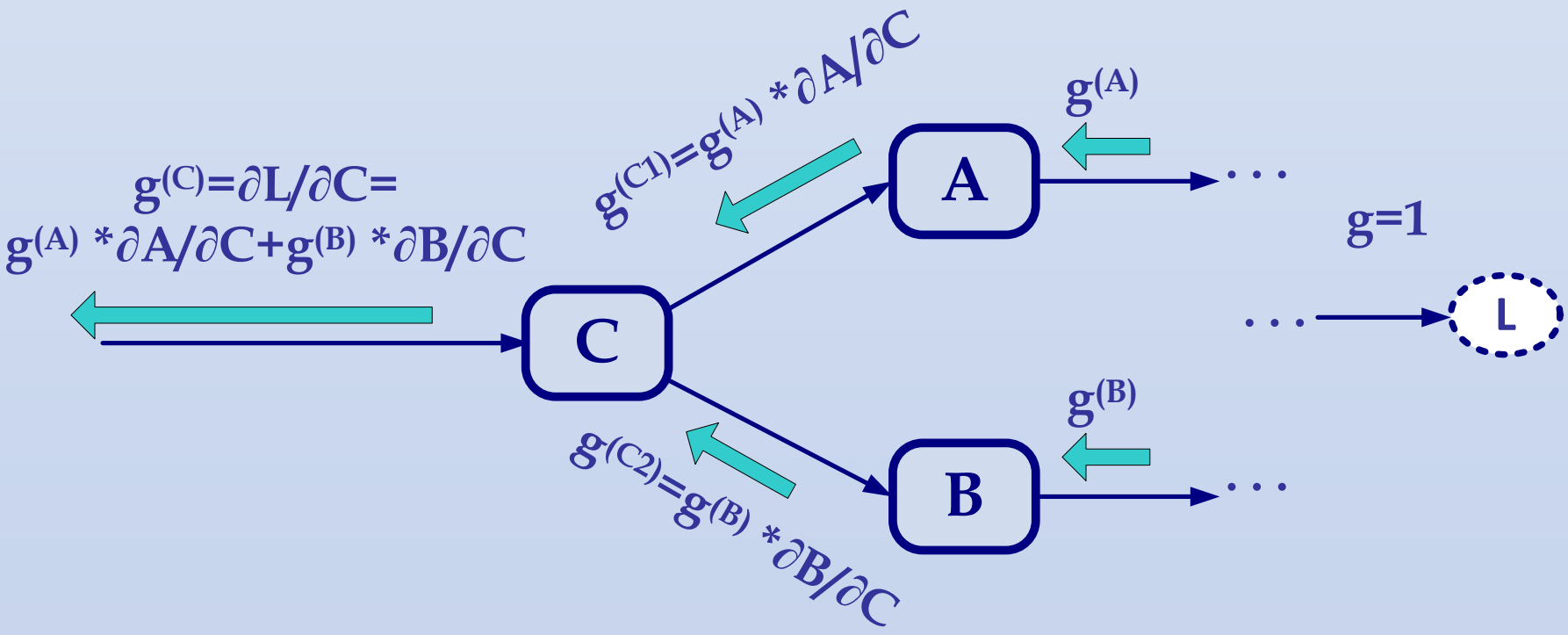
**Пройшовши через вузол, градієнт розщеплюється по всіх ребрах, що входять у вузол, і множиться на похідні між вузлами, які зв'язує ребро.**



# Backpropagation

## Зворотне поширення помилки

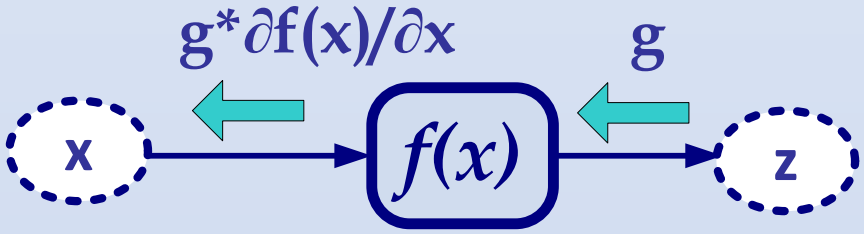
**Відповідно правила обчислення похідної від складної функції, якщо у вузол входить кілька градієнтів, то вони складаються.**



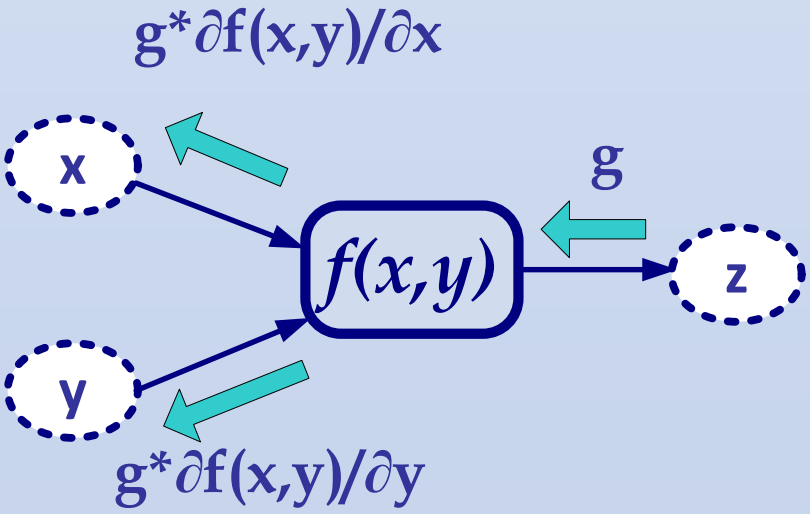
# Backpropagation

## Зворотне поширення помилки

**Похідна скалярної функції *f(x)* однієї змінної**



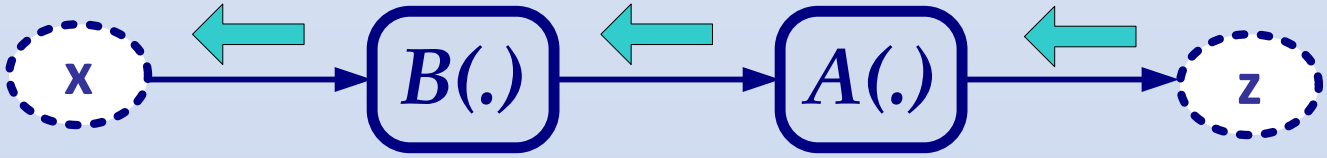

**Похідна скалярної функції *f(x,y)* двох змінних**



# Backpropagation
## Зворотне поширення помилки

**Составна функція** $z=A(B(x))$

$g^{(B)}=g^{(A)} * \partial A/\partial B$ $\quad g=1$

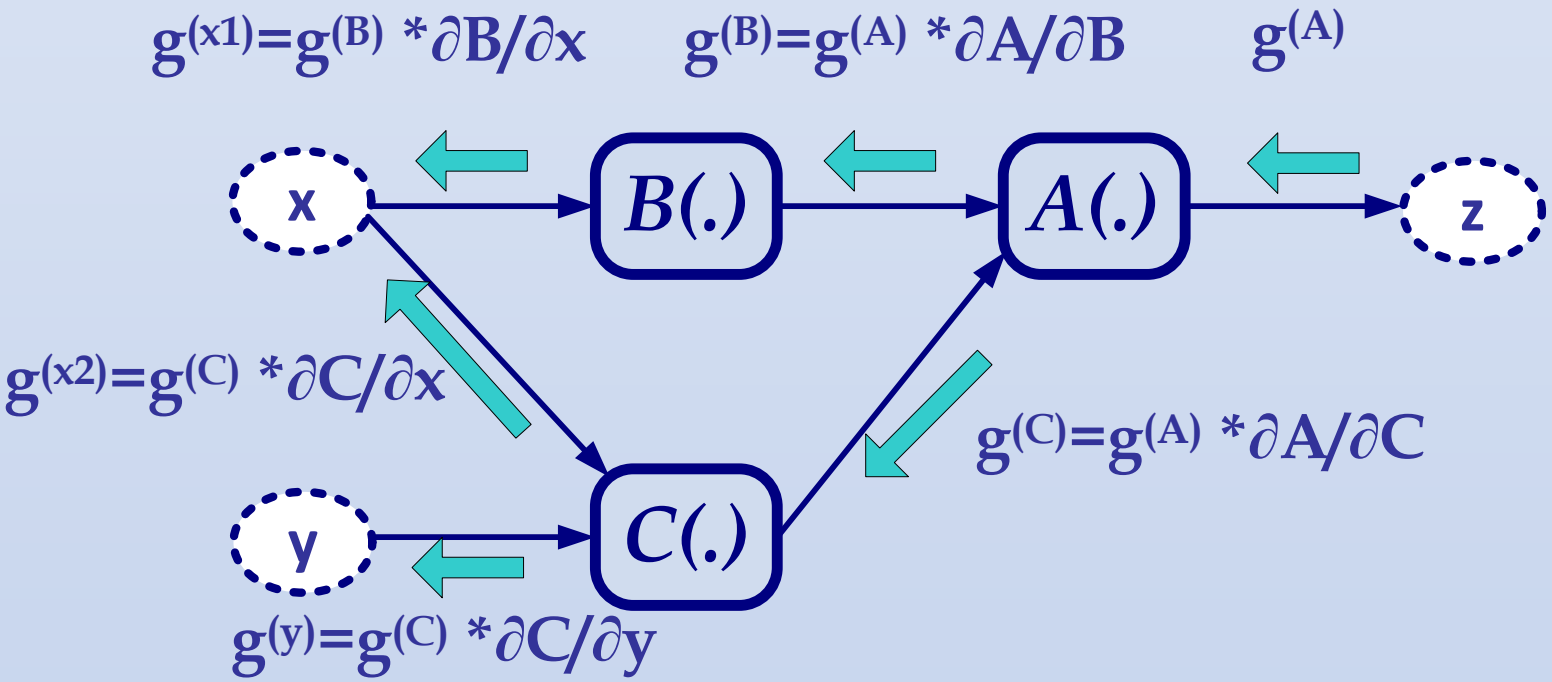



$g^{(x)}=\partial z/\partial x=\partial A/\partial x=\partial A/B*\partial B/\partial x$

# Backpropagation

## Зворотне поширення помилки

**Складна функція** $z=A(B(x),C(x,y))$




$$g^{(x)}=\partial z/\partial x=\partial A/\partial x=\partial A/B*\partial B/\partial x+\partial A/C*\partial C/\partial x$$

# Обчислювальний граф. Типи

**Тип 1: Статичні обчислювальні графіки (TensorFlow). Включає дві фази: -**
**Етап 1: Складається план архітектури моделі.**
**Етап 2: Для навчання моделі і створення прогнозів моделі передаються данні.**

**Перевага використання цього графіка полягає в тому, що він дає змогу оптимізувати та планувати потужні офлайнові графіки.**
**Як наслідок, вони мають бути швидшими, ніж динамічні графіки загалом.**
**Недоліком є те, що працювати зі структурованими даними і навіть даними змінного розміру непривабливо.**

# Обчислювальний граф. Типи

**Тип 1: Статичні обчислювальні графіки (TensorFlow).**
**Властивості вузлів і ребер: вузли представляють операції, які застосовуються безпосередньо до даних, що надходять і виходять через ребра.**

**Dynamic vs Static Computational Graphs – PyTorch and TensorFlow**

**https://www.geeksforgeeks.org/dynamic-vs-static-computational-graphs-pytorch-and-tensorflow/?ref=rp**

**Python Code**

# Обчислювальний граф. Типи

**Тип 2: Динамічні обчислювальні графи (PyTorch).**
**При виконанні пряме обчислення, граф визначається неявно (в процесі).**
**Граф має перевагу в тому, що він більш адаптивний.**
**Бібліотеки дозволяють генерувати граф та оцінювати граф поочередно (змішувати ?).**
**Налагодження динамічного графу просте. Оскільки він дозволяє виконання коду рядок за рядком, надає доступ до всіх змінних, пошук помилок у коді значно легший.**
**Недоліком використання цього графу є обмежений час для оптимізації графу, і зусилля можуть бути витрачені даремно, якщо граф не змінюється.**

# Обчислювальний граф. Типи

**Тип 2: Динамічні обчислювальні графи (PyTorch).**
**Властивості вузлів і ребер: вузли представляють дані (у формі тензорів), а ребра представляють операції, застосовані до вхідних даних.**
**Оскільки все в Pytorch створюється динамічно, нам не потрібні ніякі заповнювачі, і ми можемо визначати наші вхідні дані та операції на льоту.**
**Після визначення вхідних даних і обчислення виходу «c» ми викликаємо метод backward(), який обчислює відповідні часткові похідні відносно двох вхідних даних, доступних через специфікатор .grad.**
**Перевірити код**

# Tensor Flow 2
## Два режиму виконання

| Graph-based execution | Eager execution |
|---|---|
| Менш інтуїтивно зрозумілий. Загалом його важче налагодити ніж активне виконання | Інтуїтивно зрозумілий і простий у налагодженні |
| | Спрощує швидку розробку моделі |
| Загалом швидше, ніж активне виконання. Будує графік та компілює процес, щоб скористатися можливостями прискорення перед виконанням | Повільніше, ніж виконання на основі графів операцій. Операції виконуються одна за одною |
| Ідеально підходить для масштабних тренувань | Краще для новачків. |
| Підтримка прискорення GPU і TPU | Підтримка прискорення GPU і TPU |

# TF. Створення графу обчислень

**Функція**

**tf.gradients(ys, xs, grad_ys=None,**
**name='gradients', gate_gradients=False,**
**aggregation_method=None,**
**stop_gradients=None,**
**…)**

**Будує символічні похідні ys w.r.t. x в xs.**

**ys – тензор змінних, для яких обчислюється градієнт**

**xs – тензор змінних, відносно яких обчислюється градієнт**

tf.gradients дійсний лише в контексті графа (зокрема, це дійсне в контексті обгортки tf.function, де код виконується як граф)

https://www.tensorflow.org/api_docs/python/tf/gradients

# TF. Створення графу обчислень

**Метод**

```
tf.GradientTape(
    persistent=False, watch_accessed_variables=True
)
```

**Автоматична диференціація: є створення графу для вирахування градієнта відносно деяких вхідних даних. TensorFlow «записує» відповідні операції, що виконуються всередині контексту tf.GradientTape , на «СТРІЧКУ». Потім TensorFlow використовує цю стрічку для розрахунку градієнтів «записаного» розрахунку з використанням диференціювання в зворотному режимі.**

https://www.tensorflow.org/api_docs/python/tf/GradientTape

# TF. Створення графу обчислень

**Metod**

```
tf.GradientTape(
    persistent=False, watch_accessed_variables=True
)
```

**Persistent -** логічне значення, яке контролює, чи створюється стрічка постійного градієнта. **False** означає, що для цього об'єкта можна зробити щонайбільше один виклик методу gradient().

**Watch_accessed_variables -** логічне значення, що контролює, чи буде стрічка автоматично спостерігати за будь-якими (придатними для навчання) змінними. За замовчуванням **True.**

https://www.tensorflow.org/api_docs/python/tf/GradientTape

# Tensor Flow

## TensorFlow Official

https://www.tensorflow.org/

## TensorFlow API Documentation

https://www.tensorflow.org/api_docs/python/tf

## TensorFlow on GitHab

https://github.com/tensorflow/tensorflow

## Приклади дивись
## 2024_AI_TF_lec_03_Exmpl_1.pdf

# The END
# Модуль 7. Lec. 7.3.