



Нейронні мережі

Лекція 4: Навчання нейронних мереж

Кочура Юрій Петрович
iuriy.kochura@gmail.com
[@y_kochura](https://twitter.com/y_kochura)

Сьогодні

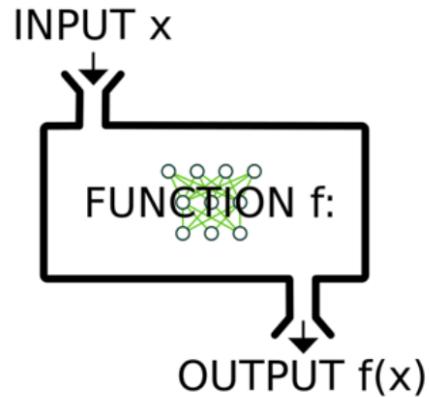
Як ефективно оптимізувати параметри?

- 🎙 Пакетний градієнтний спуск
- 🎙 Стохастичний градієнтний спуск
- 🎙 Міні-пакетний градієнтний спуск
- 🎙 Імпульс
- 🎙 Адаптивні методи:
 - AdaGrad
 - RMSProp
 - Adam

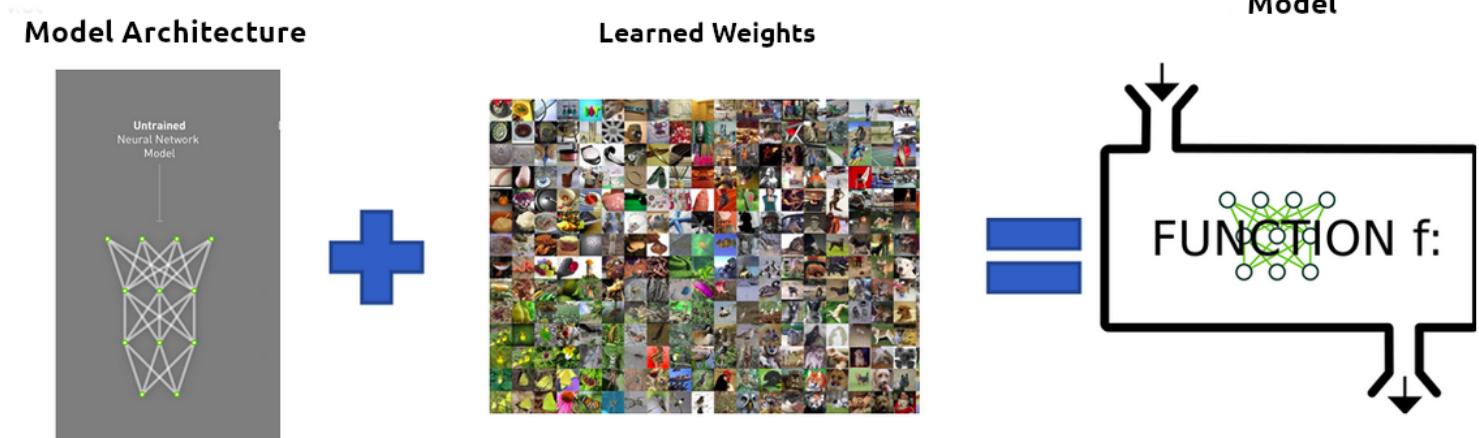
Оптимізація параметрів

Модель

Хоча те, що знаходиться всередині глибокої нейронної мережі, може бути складним, у своїй основі це просто функції. Вони приймають деякі вхідні дані та генерують деякі результати.



Компоненти моделі

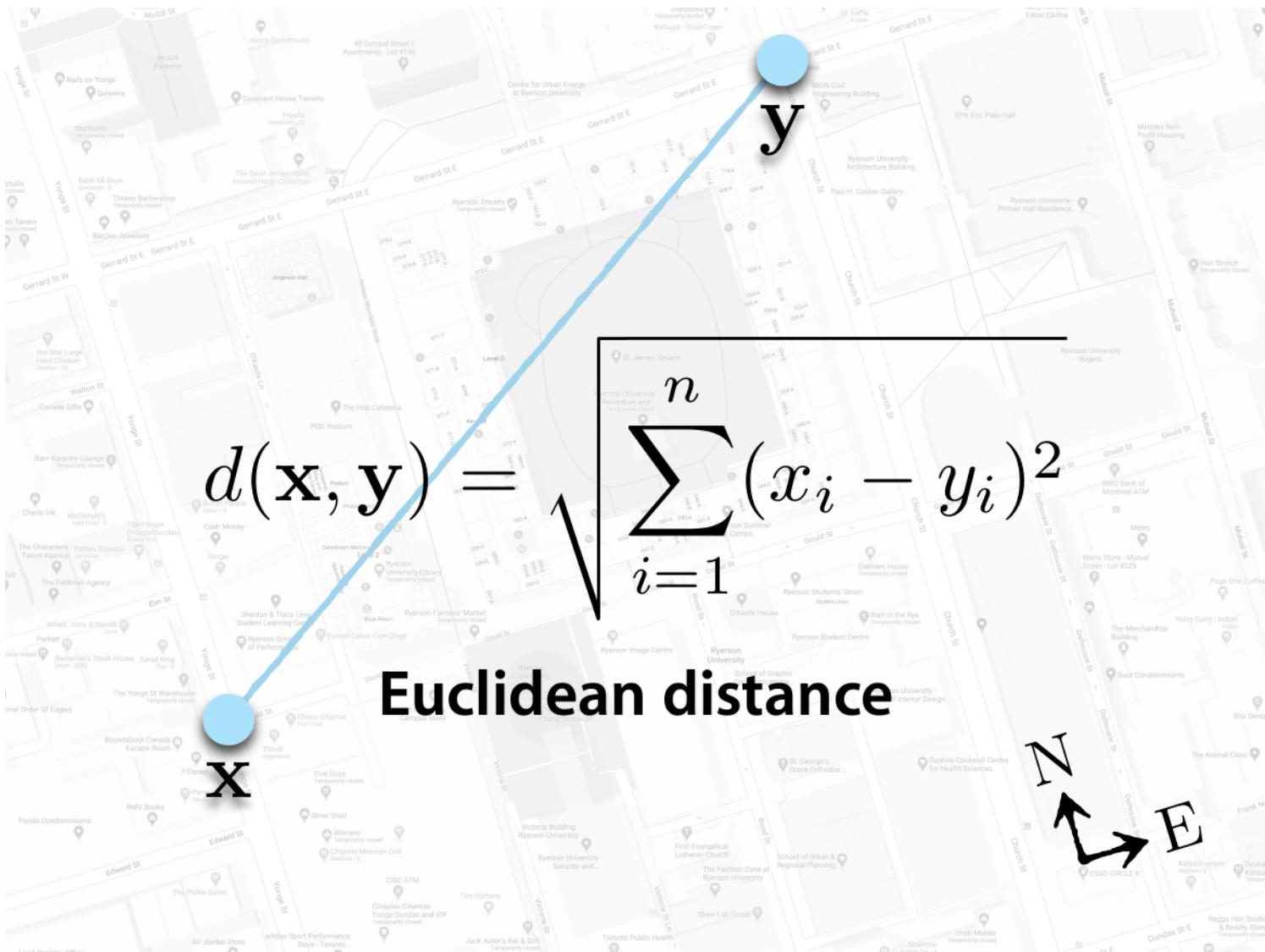


Загальний процес навчання для нейронних мереж

1. Визначте завдання + зберіть дані
2. Ініціалізуйте параметри
3. Оберіть алгоритм оптимізації
4. Повторіть ці кроки:
 - 4.1. Пряме поширення вхідних даних
 - 4.2. Обчислення цільової функцію витрат
 - 4.3. Зворотне поширення: обчисліть градієнти цільової функції втрат відносно параметрів
 - 4.4. Оновіть кожен параметр за допомогою градієнтів відповідно до алгоритму оптимізації

Загальні функції втрати

- Евклідова відстань (L2 втрати)
- Середньоквадратична похибка (MSE)
- Манхеттенська відстань (L1 втрати)
- Середня абсолютна похибка(МАЕ)
- Перехресна втрата ентропії



Евклідова відстань (L2 втрати)

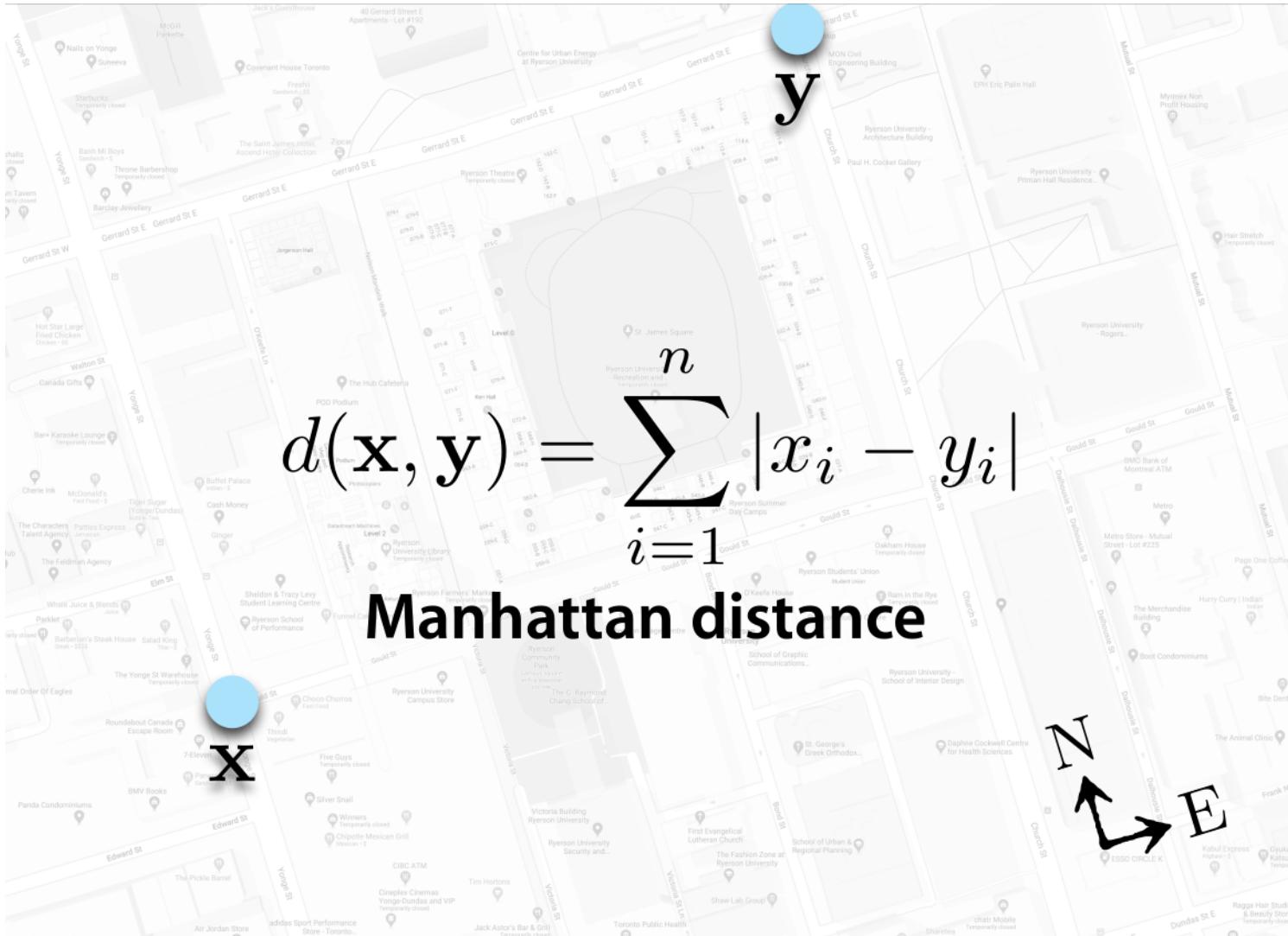
$$d(\hat{y}, y) = \sqrt{\sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2} = \|\hat{y} - y\|_2$$

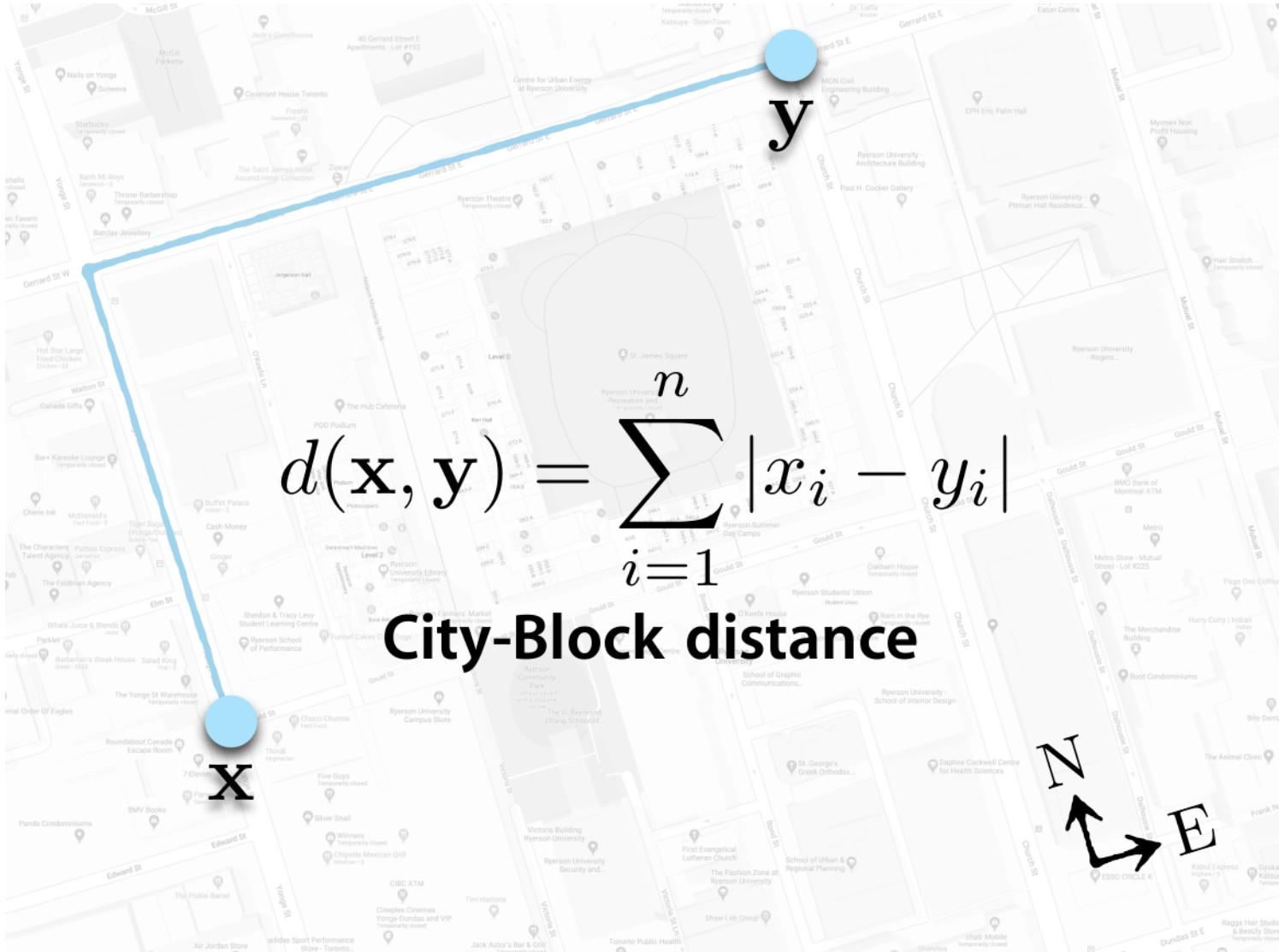
де n – загальна кількість навчальних прикладів.

Середньоквадратична похибка (MSE)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$
$$\mathcal{J}(\hat{y}, y) = \frac{1}{n} \sqrt{\sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2} = \frac{1}{n} \|\hat{y} - y\|_2$$

де n – загальна кількість навчальних прикладів.





Манхеттенська відстань (L1 втрати)

$$d(\hat{y}, y) = \sum_{i=1}^n |\hat{y}^{(i)} - y^{(i)}| = \|\hat{y} - y\|_1$$

де n – загальна кількість навчальних прикладів.

Середня абсолютна похибка(МАЕ)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = |\hat{y}^{(i)} - y^{(i)}|$$

$$\mathcal{J}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^n |\hat{y}^{(i)} - y^{(i)}| = \frac{1}{n} \|\hat{y} - y\|_1$$

де n – загальна кількість навчальних прикладів.

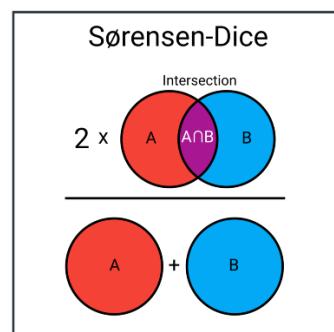
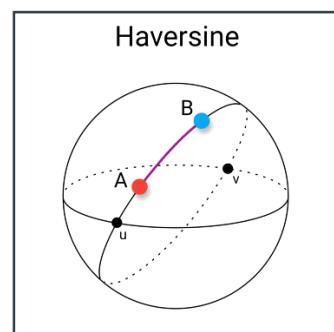
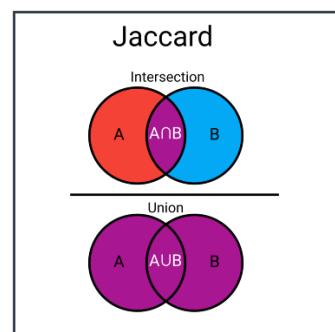
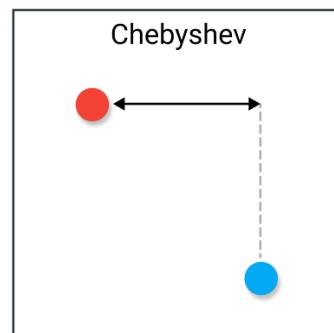
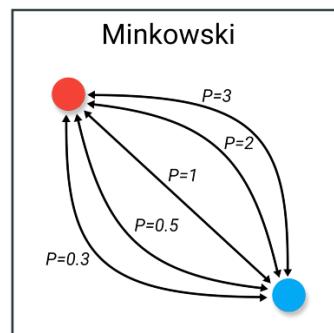
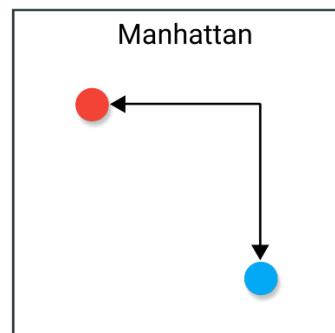
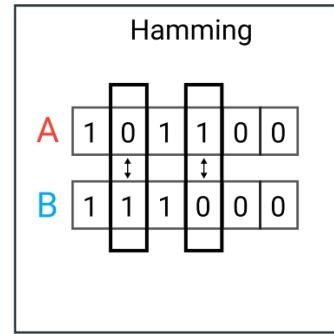
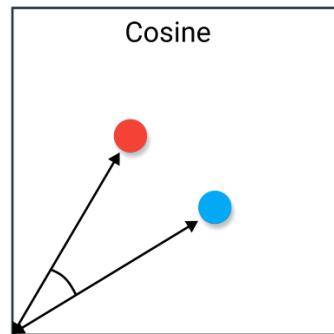
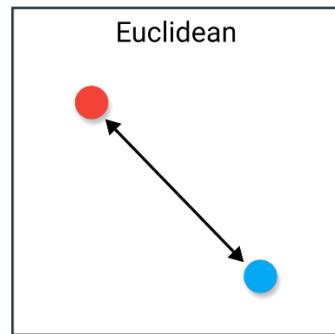
Перехресна втрата ентропії (бінарна)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

$$\mathcal{J}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^n \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

де n – загальна кількість навчальних прикладів.

9 метрик відстані в науці про дані



Optimization



Оптимізаційні алгоритми

Гіперпараметри & Параметри

- Результатом оптимізації є набір параметрів
- Гіперпараметри vs. Параметри

Гіперпараметри

- Коефіцієнт поділу датасету навчання/валідація/тестування
- Швидкість навчання в алгоритмах оптимізації
- Кількість прихованих шарів у НМ
- Кількість блоків активації в кожному шарі
- Розмір пакету

⋮

Параметри

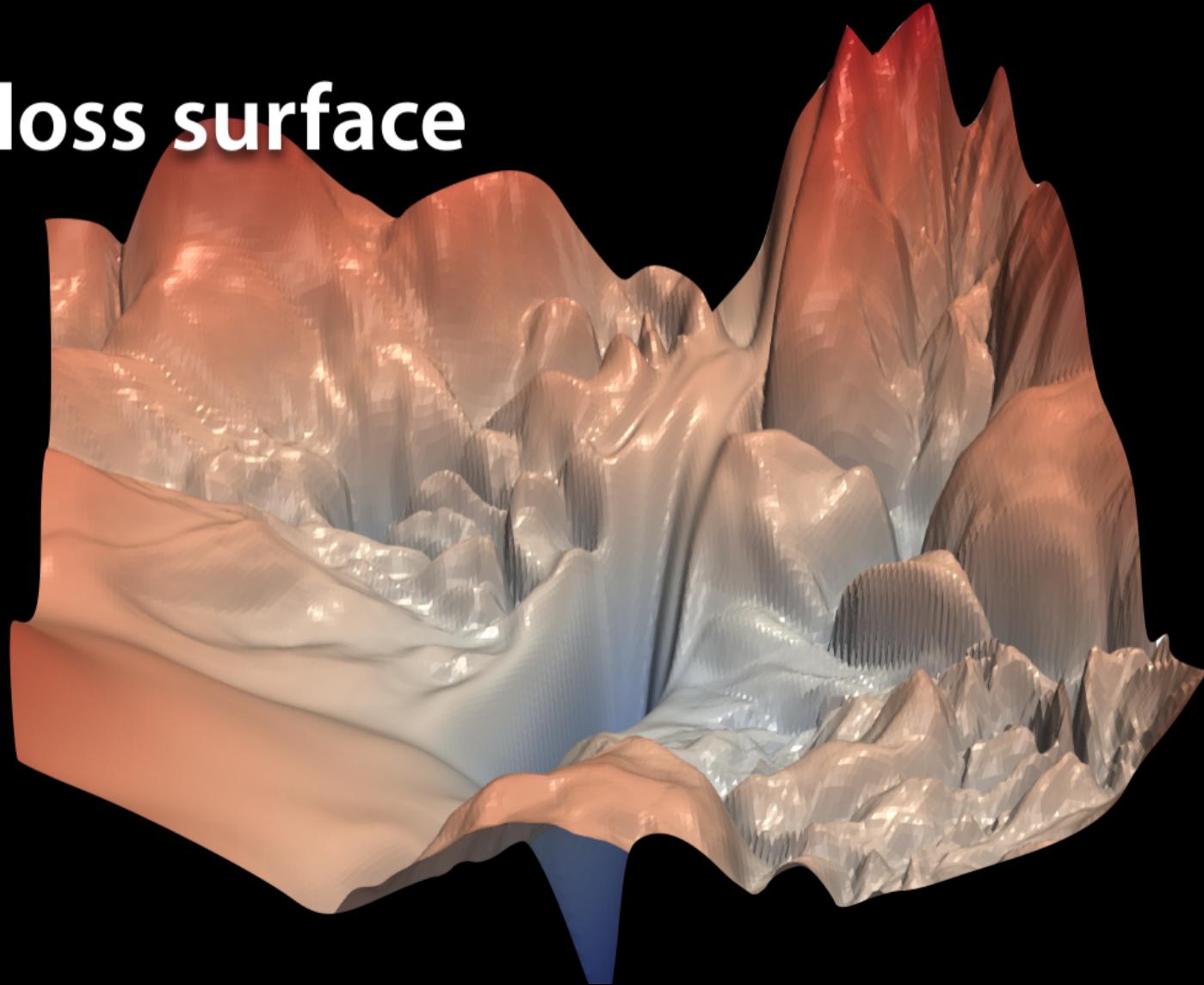
- Ваги та зсуви НМ
- Центроїди кластерів у кластеризації

Задача оптимізації

Мінімізація емпіричного ризику (втрат)

$$W_*^{\mathbf{d}} = \arg \min_W \mathcal{J}(W) = \arg \min_W \frac{1}{n} \sum_{i=1}^n \mathcal{L} \left(y^{(i)}, f(\mathbf{x}^{(i)}, W) \right)$$

loss surface



Практичні рекомендації

Навчання масивної глибокої нейронної мережі є тривалим та складним процесом.

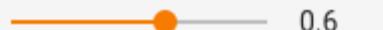
Першим кроком до розуміння, налагодження та оптимізації нейронних мереж є використання інструментів візуалізації:

- побудова графіків втрат та інших метрик продуктивності,
- візуалізація обчислювальних графіків,
- показ додаткових даних під час навчання мережі.

Show data download links Ignore outliers in chart scalingTooltip sorting
method:

default

Smoothing



Horizontal Axis

STEP

RELATIVE

WALL

Runs

Write a regex to filter runs

 train eval

TOGGLE ALL RUNS

/tmp/mnist-logs

 Filter tags (regular expressions supported)

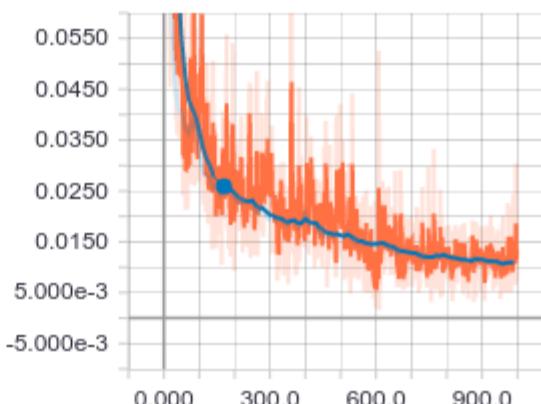
accuracy

1

cross entropy

1

cross entropy



run to download

CSV JSON

Name	Smoothed	Value	Step	Time	Relative
eval	0.02591	ep_p0.02550	ity170.0	Mon Sep 12, 15:40:41	8s
train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

mean

4

Демо

Пакетний градієнтний спуск

GD

Пакетний градієнтний спуск

Щоб мінімізувати $\mathcal{J}(W)$, **пакетний градієнтний спуск** (GD) використовує наступне правило:

$$g_t = \frac{1}{n} \sum_{i=1}^n \nabla_W \mathcal{L} \left(y^{(i)}, f(\mathbf{x}^{(i)}, W) \right) = \nabla_W \mathcal{J}(W)$$

$$W_{t+1} = W_t - \alpha g_t,$$

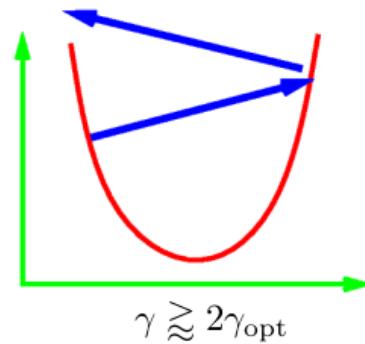
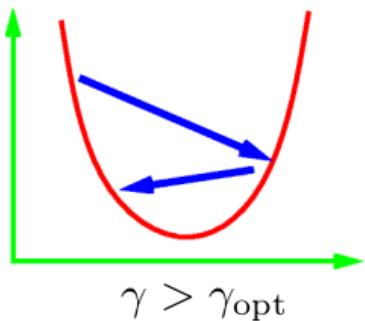
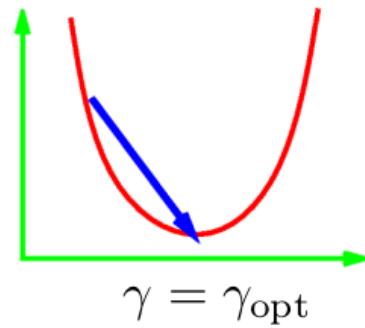
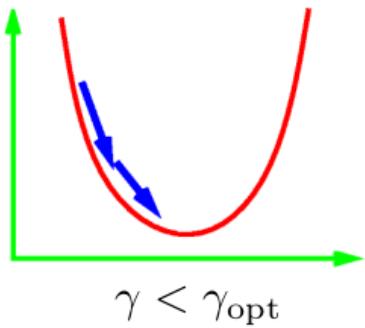
дек α – крок навчання.



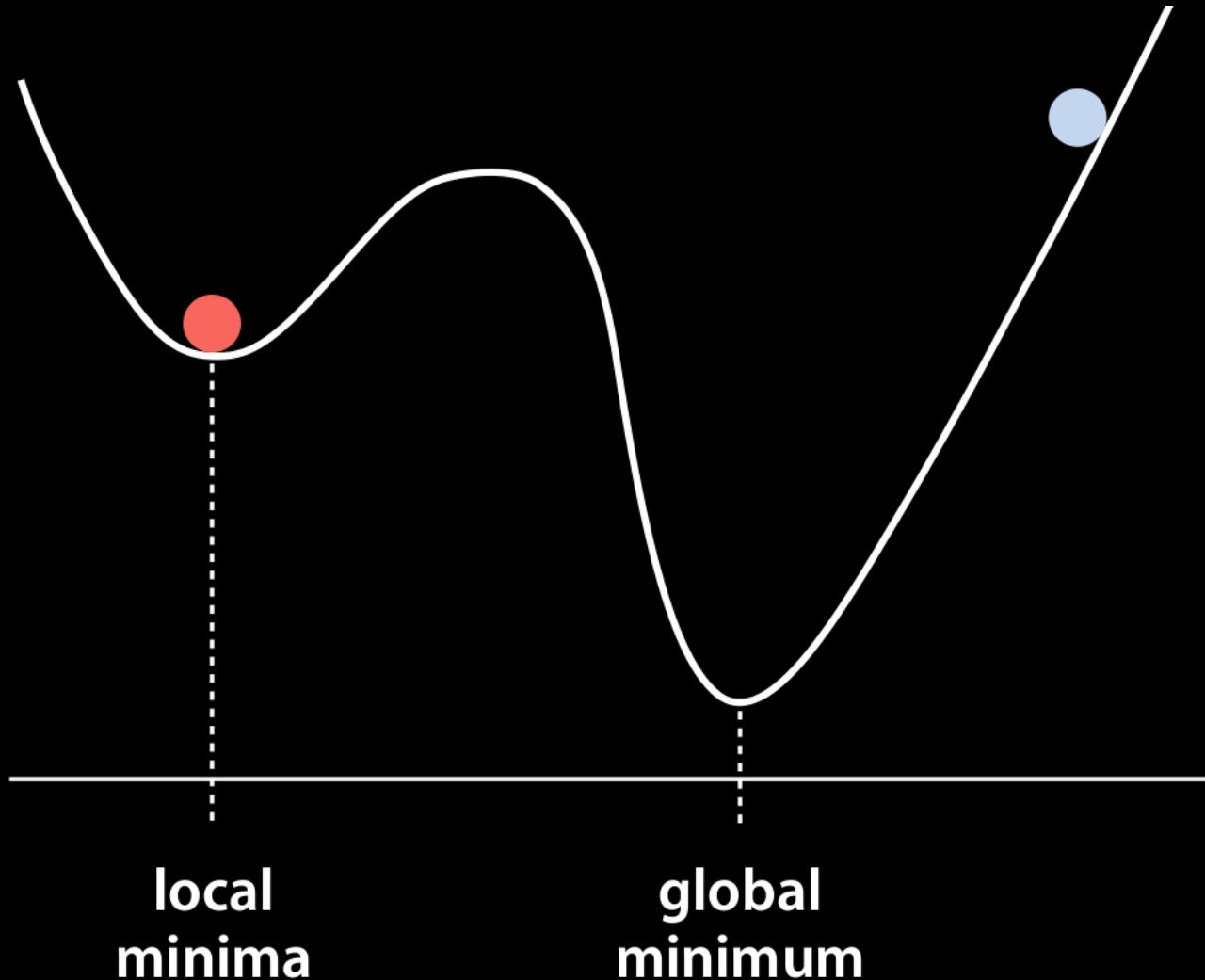
Найгірший метод оптимізації в світі

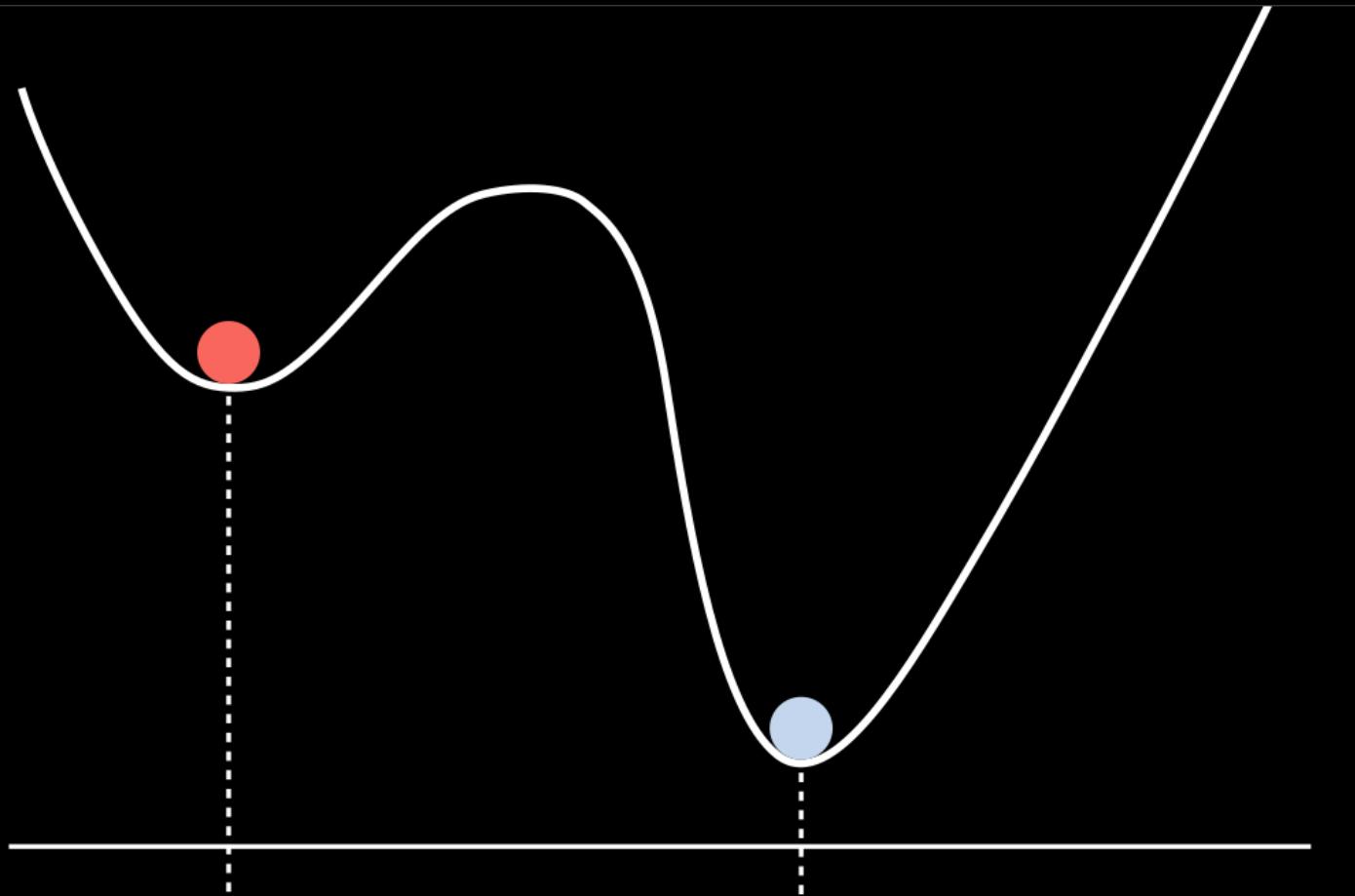
Примітка: ймовірно, Вам ніколи не слід використовувати градієнтний спуск безпосередньо, вважайте це будівельним блоком для інших методів.

Крок навчання



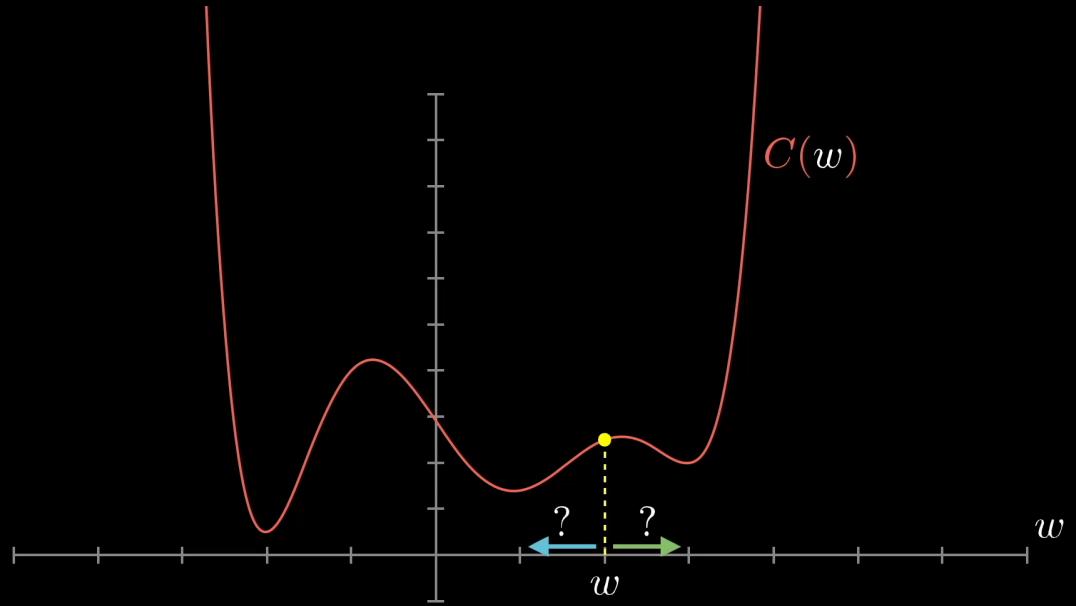
Тут $\gamma = \alpha$ – крок навчання.





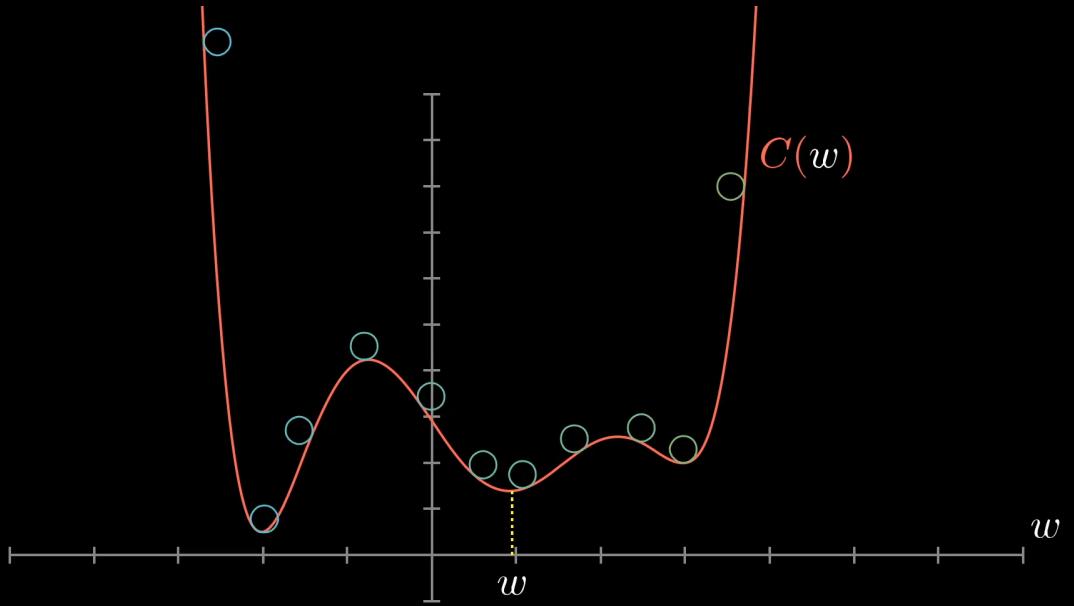
local
minima

global
minimum



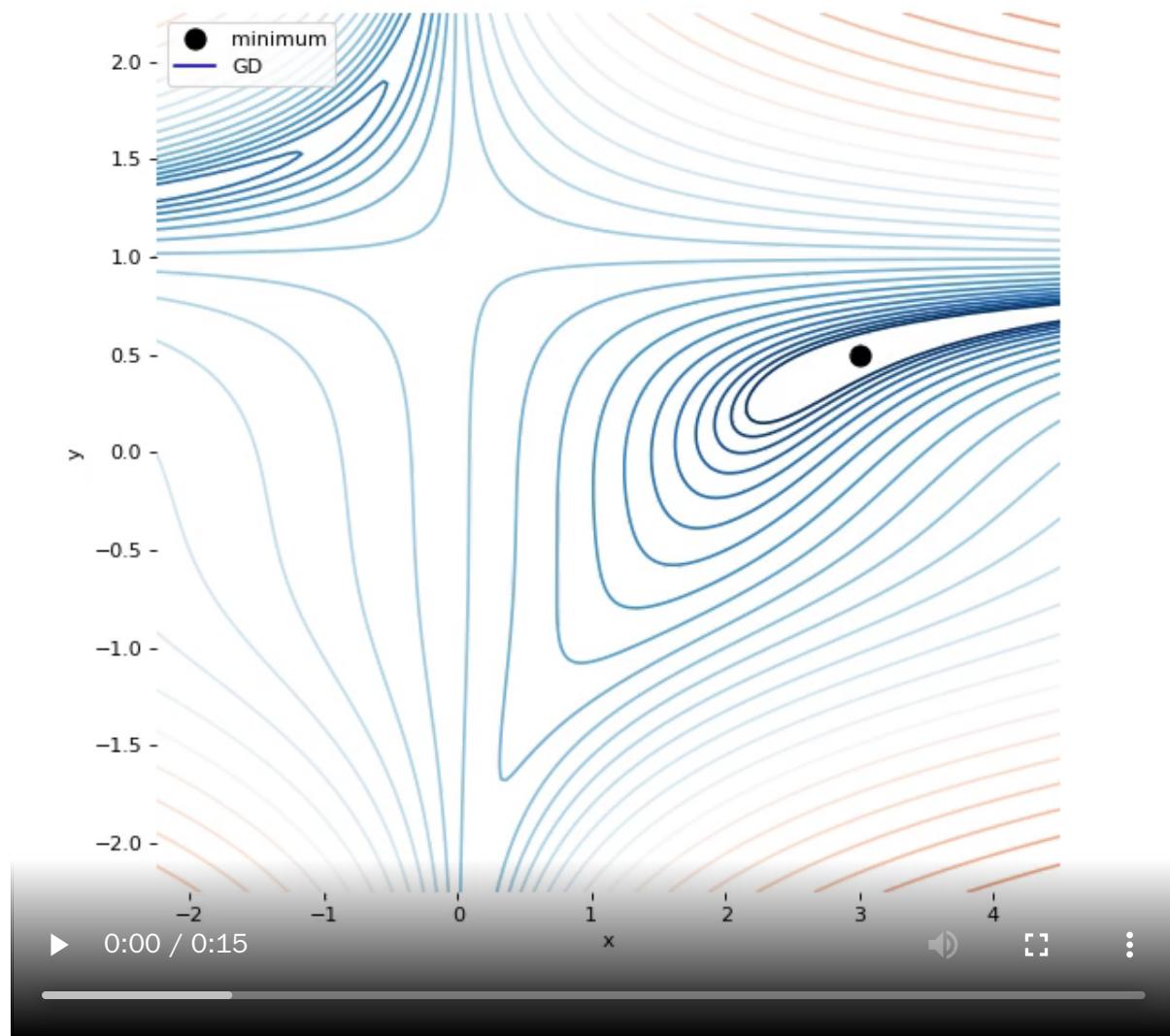
▶ 0:00 / 0:08

🔊 [] ⋮



▶ 0:00 / 0:04

🔊 [] ⋮



Стохастичний градієнтний спуск

SGD

SGD

Щоб зменшити обчислювальну складність, **стохастичний градієнтний спуск** (SGD) полягає в оновленні параметрів після кожного прикладу

$$\ell^{(i)} = \mathcal{L} \left(y^{(i)}, f(\mathbf{x}^{(i)}, W) \right)$$

$$g_t^{(i)} = \nabla_W \ell^{(i)}$$

$$W_{t+1} = W_t - \alpha g_t^{(i)}$$

Переваги SGD

- Стохастичні градієнти обчислювати значно легше (пропорційно розміру набору даних), тому Ви часто можете зробити тисячі кроків SGD за вартість одного кроку GD.
- У середньому стохастичний градієнт є хорошою оцінкою градієнта.
- Шум може перешкоджати оптимізаційному сходженню до поганих локальних мінімумів.

Міні-пакети

Міні-пакети

Обчислення втрат для міні-пакетів та оновлення параметрів

$$g_t^{(k)} = \frac{1}{B} \sum_{i=1}^B \nabla_W \mathcal{L} \left(y_k^{(i)}, f(\mathbf{x}_k^{(i)}, W) \right)$$
$$W_{t+1} = W_t - \alpha g_t^{(k)},$$

де k – індекс міні-пакета.

- Збільшення розміру пакету B зменшує дисперсію оцінок градієнта та забезпечує прискорення пакетної обробки.
- Взаємозв'язок між B і α все ще незрозумілий.

Импульс

Імпульс

SGD + Імпульс = Стохастичний метод важкої кулі

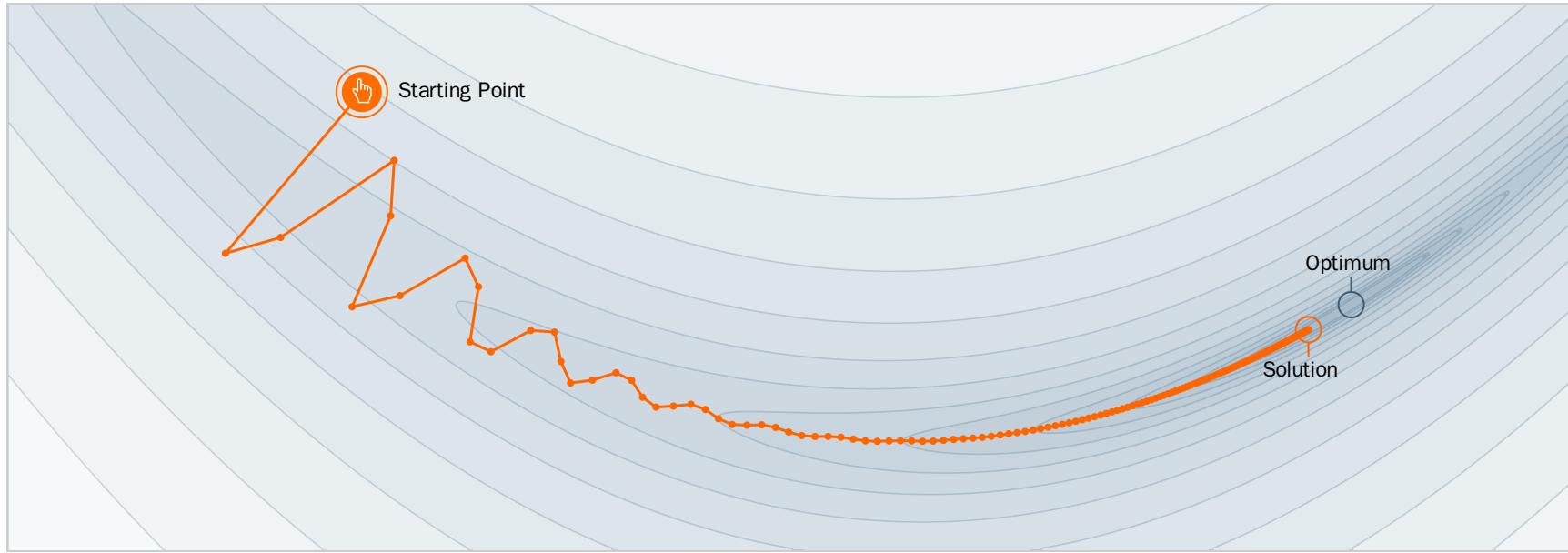
$$p_{t+1} = \beta_t p_t + \nabla \ell^{(i)}(W_t)$$

$$W_{t+1} = W_t - \alpha_t p_{t+1}$$

Правило оновлення:

$$W_{t+1} = W_t - \alpha_t \nabla \ell^{(i)}(W_t) + \beta_t (W_t - W_{t-1})$$

Ключова ідея: Наступний крок стає комбінацією напрямку попереднього кроку та нового негативного градієнта.



Step-size $\alpha = 0.02$

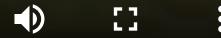


Momentum $\beta = 0.99$

We often think of Momentum as a means of dampening oscillations, speeding up the iterations, leading to faster convergence. But it creates its own oscillations. What is going on?

SGD with Momentum

▶ 0:00 / 0:46



Переваги

SGD з імпульсом має **три** хороші властивості:

- він може пройти через локальні бар'єри
- прискорюється, якщо градієнт не сильно змінюється
- гасить коливання у вузьких долинах

Практичні аспекти імпульсу

По суті, це «**безкоштовний обід**», майже в усіх ситуаціях **SGD + імпульс** краще, ніж SGD, і дуже рідко гірше!

Рекомендовані параметри:

$\beta = 0.9$ or 0.99 майже завжди працють добре. Іноді можна отримати невеликі переваги, налаштувавши його.

Параметр розміру кроку (α) зазвичай потрібно зменшувати, коли параметр імпульсу збільшується, щоб підтримувати збіжність.

Демо

Адаптивні методи

Адаптивні методи

Величина градієнтів часто сильно відрізняється між шарами, тому глобальна швидкість навчання може не працювати належним чином.

Загальна ідея: Замість того, щоб використовувати однакову швидкість навчання для кожної ваги в нашій мережі, **підтримуйте оцінку кращої швидкості окремо для кожної ваги.**

Точний спосіб адаптації до швидкості навчання залежить від алгоритму, але більшість методів або **пристосовуються до дисперсії ваг**, або до **локальної кривизни** проблеми.

AdaGrad

Зменшення масштабу для кожного параметра на квадратний корінь із суми квадратів усіх його історичних значень.

$$r_t = r_{t-1} + g_t \odot g_t$$
$$W_{t+1} = W_t - \frac{\alpha}{\varepsilon + \sqrt{r_t}} \odot g_t$$

- AdaGrad позбавляє від необхідності вручну налаштовувати швидкість навчання. Більшість реалізацій використовують $\alpha = 0.01$ за замовчуванням.
- Добре, коли цільова функція опукла.
- r_t необмежено зростає під час навчання, що може спричинити зменшення розміру кроку та зрештою стати нескінченно малим.
- ε адитивна константа, яка гарантує, що ми не ділимо на 0.

RMSProp

Те саме, що AdaGrad, але накопичує експоненціально спадне середнє значення градієнта.

Ключова ідея: нормалізувати за середньоквадратичним значенням градієнта

$$r_t = \rho r_{t-1} + (1 - \rho)g_t \odot g_t$$
$$W_{t+1} = W_t - \frac{\alpha}{\varepsilon + \sqrt{r_t}} \odot g_t$$

- Ефективніший, коли цільова функція не є опуклою.

Adam: RMSprop з імпульсом

“Adaptive Moment Estimation”

Подібно до RMSProp з імпульсом, але з умовами корекції зсуву для першого та другого моментів.

$$\begin{aligned} p_t &= \rho_1 p_{t-1} + (1 - \rho_1) g_t \\ \hat{p}_t &= \frac{p_t}{1 - \rho_1^t} \\ r_t &= \rho_2 r_{t-1} + (1 - \rho_2) g_t \odot g_t \\ \hat{r}_t &= \frac{r_t}{1 - \rho_2^t} \\ W_{t+1} &= W_t - \alpha \frac{\hat{p}_t}{\varepsilon + \sqrt{\hat{r}_t}} \end{aligned}$$

- Хороші значення за замовчуванням $\rho_1 = 0.9$ і $\rho_2 = 0.999$.
- Подібно до того, як імпульс покращує SGD, він також покращує RMSProp.

Практична сторона

Для погано обумовлених задач Adam часто набагато кращий, ніж SGD.

Використовуйте Adam замість RMSprop завдяки очевидним перевагам імпульсу.

Але, **Adam** погано вивчений теоретично та має відомі недоліки::

- Зовсім не сходиться на деяких простих прикладах задач!
- Дає гіршу помилку узагальнення для багатьох задач комп'ютерного зору (наприклад, ImageNet)
- Потрібно більше пам'яті, ніж SGD
- Має 2 параметри моментів, тому може знадобитися деяке налаштування

Демо

demo - losslandscape

