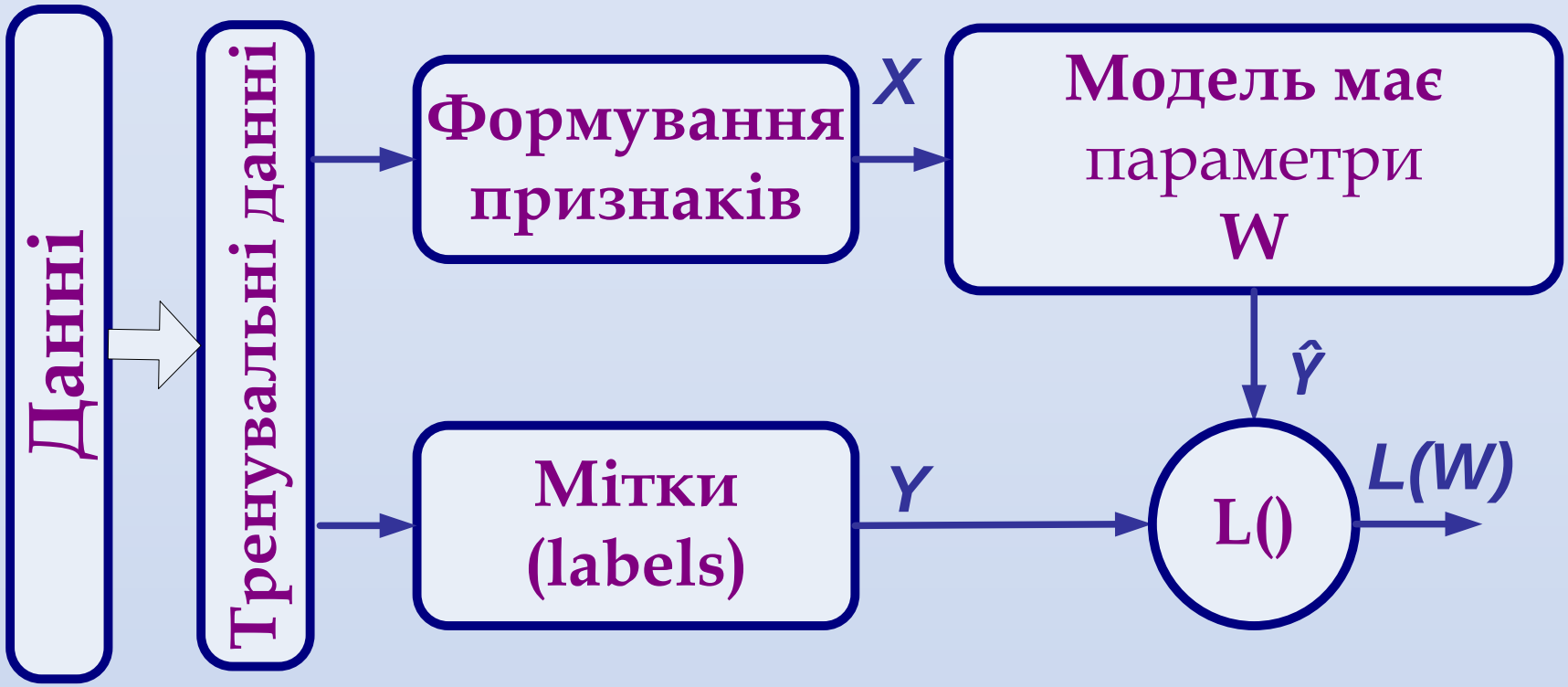
ОСНОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ, НЕЙРОННИХ МЕРЕЖ та ГЛИБОКОГО НАВЧАННЯ

Модуль 7. TensorFlow / KERAS

Лекція 7.4. TensorFlow. Градієнтний спуск / Gradient Descent Оптимізатори спуску

Навчання з вчителем



Навчання: знайти W , що мінімізують похибку (втрати) моделі

Задача оптимізації

Стандартна постановка:

Задано:

- Допустиме безліч незалежних
- змінних $\mathbb{X} = \{\vec{x} | g_i(\vec{x}) \leq 0, i = 0,1, \dots, m\} \in \mathbb{R}^n$
- Цільова функція – відображення $f: \mathbb{X} \to \mathbb{R}$
- Обмеження …
- Критерій пошуку (***min*** або ***max*** цільової функції)

Необхідно: знайти таке $\vec{x}^* \in \mathbb{X}$, що

$$f(\vec{x}^*) = \min_{\vec{x} \in \mathbb{X}} f(\vec{x})$$

Взагалі вирішенням таких задач займається ***теорія математичного програмування.***

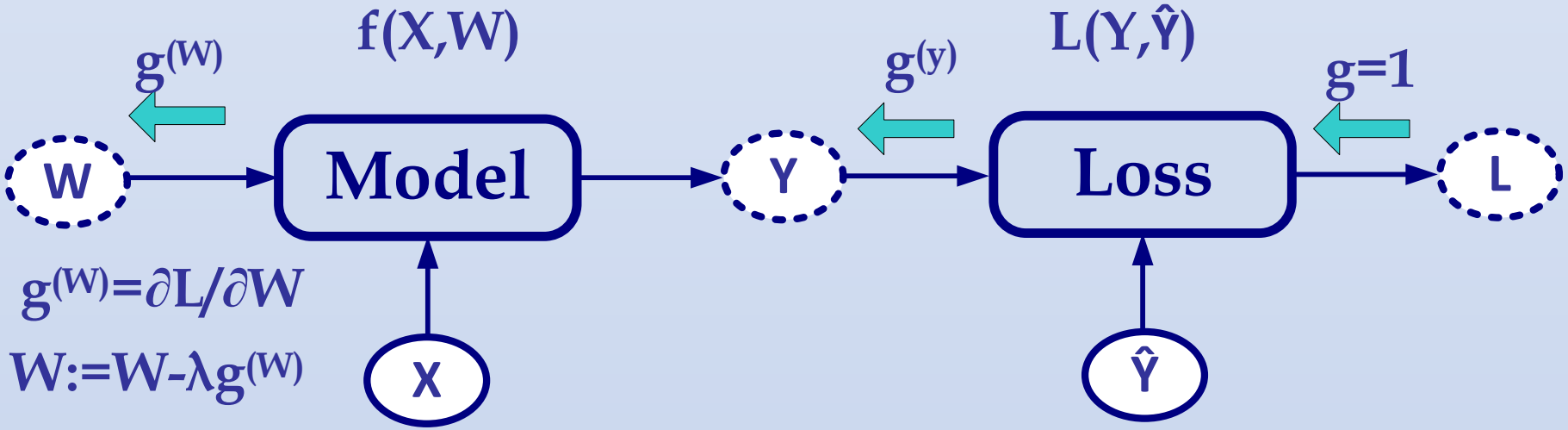
Градієнтний спуск

Градієнтний спуск (gradient descent) — ітераційний алгоритм оптимізації, в якому для знаходження локального мінімуму функції здійснюються кроки, пропорційні протилежному значенню градієнту (або наближеного градієнту) функції в поточній точці.

Градієнтний спуск відомий також як найшвидший спуск (steepest descent), або метод найшвидшого спуску (method of steepest descent).

←Backward

Процес навчання → пошук параметрів $\overline{W}$, які мінімізують втрати (Loss)



Загальний підхід → використання методів градієнтного методу (gradient descent)

Градієнтний спуск

Маєм тренувальний набір

$X = \{x_i | i = 0, 1, \ldots < N - 1\}$ **- множина векторів ознак**

$Y = \{y_i | i = 0, 1, \ldots < N - 1\}$ **- множина міток**

Деяким чином визначені початкові значення ваг моделі W

Визначена функція похибки (втрат, Loss)

$$L(W) = F(W, X, Y)$$

Важливо: $L(W)$ **залежить тільки від** W

Градієнтний спуск

Визначена функція $L(W)$

Необхідно знайти таке $\overline{W}$, що

$$L\left(\overline{W}\right) = min\,(L(W))$$

$\overline{W}$ - ваги, для якої функція похибки досягає свого мінімального значення.

Узагальнено ітераційний процес пошуку $\overline{W}$:

$$W^{(t+1)} = W^{(t)} - \Delta W^{(t)}; \Delta W^{(t)} = \lambda \nabla L\left(W^{(t)}\right);$$

$$\nabla L\left(W^{(t)}\right) = \frac{\partial L(W)}{\partial W}$$

t – ітерація, t=1,2, …

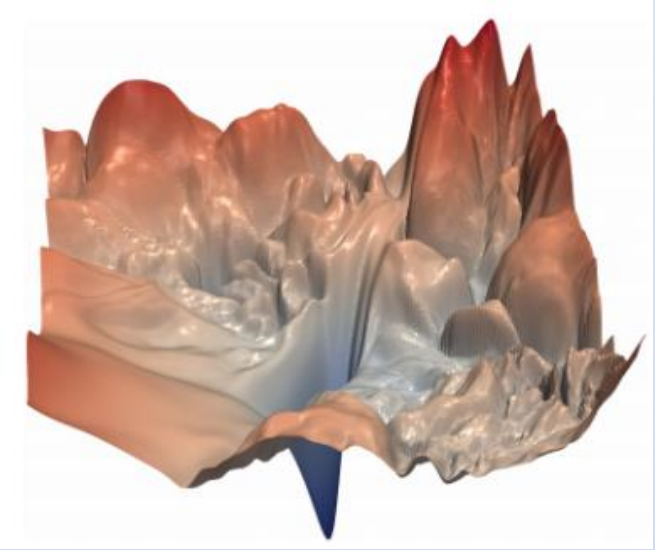
$\Delta W^{(t)}$- крок оптимізації ваг W

$\nabla L\left(W^{(t)}\right)$– градієнт функції похибки в точці $W^{(t)}$

λ– швидкість навчання (розмір кроку навчання - learning rate)

Проблеми градієнтного спуску

Багатовимірна (!!! Багато) функція $L(W)$



Проблеми:

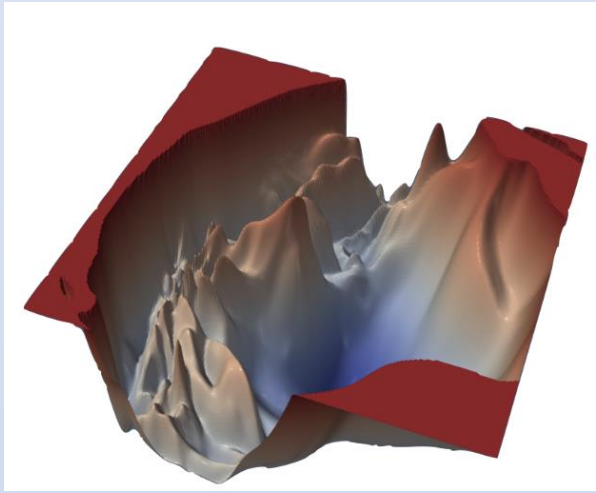
- **Локальні мінімуми. Алгоритм просто застряє у локальному мінімумі, так і не потрапивши на глобальний мінімум.**
- **Сідлові точки. Дуже малі значення компонент градієнту.**
- **Яри, перетин ярів.**

Яр – це протяжна вузька долина, що має крутий ухил в одному напрямку (тобто по сторонах долини) і плавний ухил в іншому (тобто вздовж долини). Приклад – функція Розенброка.

Проблеми градієнтного спуску

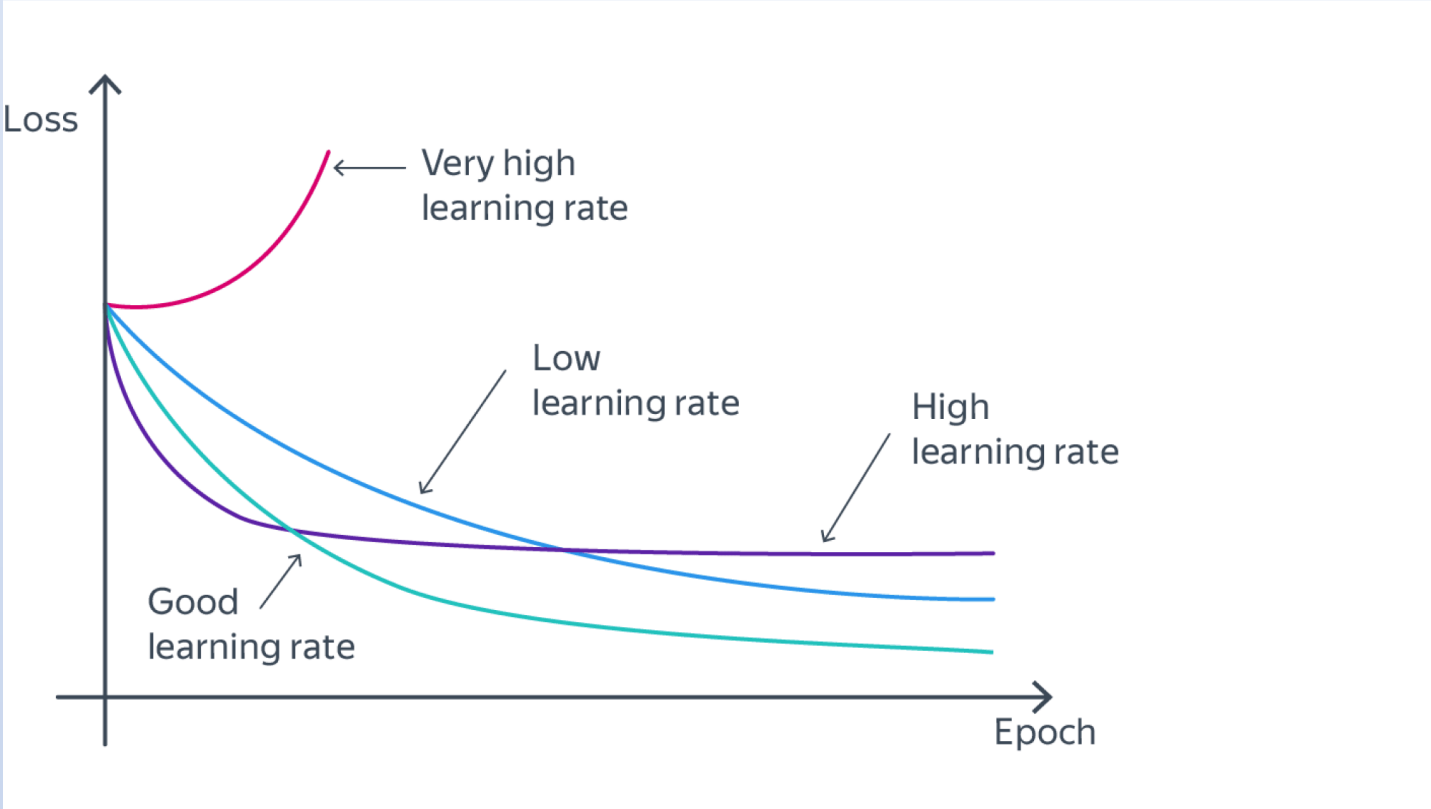
Багатовимірна (!!! Багато) функція $L(W)$

- Для невдало обумовлених опуклих задач градієнтний спуск «зигзагує» все більше, коли градієнт вказує майже ортогонально до найкоротшого напряму до точки мінімуму.



Проблеми градієнтного спуску

Як обирати швидкість навчання λ(learning rate)



Learning rate : потрібно вибирати вкрай акуратно - алгоритм може передчасно вийти на плато, або зовсім розійтися.

Методи оптимізації

Momentum (метод моментів). Проблема з SGD – якщо функція потрапляє у "яр", тобто по одному з напрямків маємо швидкий спуск, а по-іншому повільний, то SGD призводить до осциляції і вкрай повільної збіжності до мінімуму.

Зміна параметрів розраховується як зважена сума зсуву на попередньому кроці та нового на основі градієнта.

$$\Delta_{t+1} = \gamma \Delta_t + \lambda * \nabla L \left(w_t \right)$$

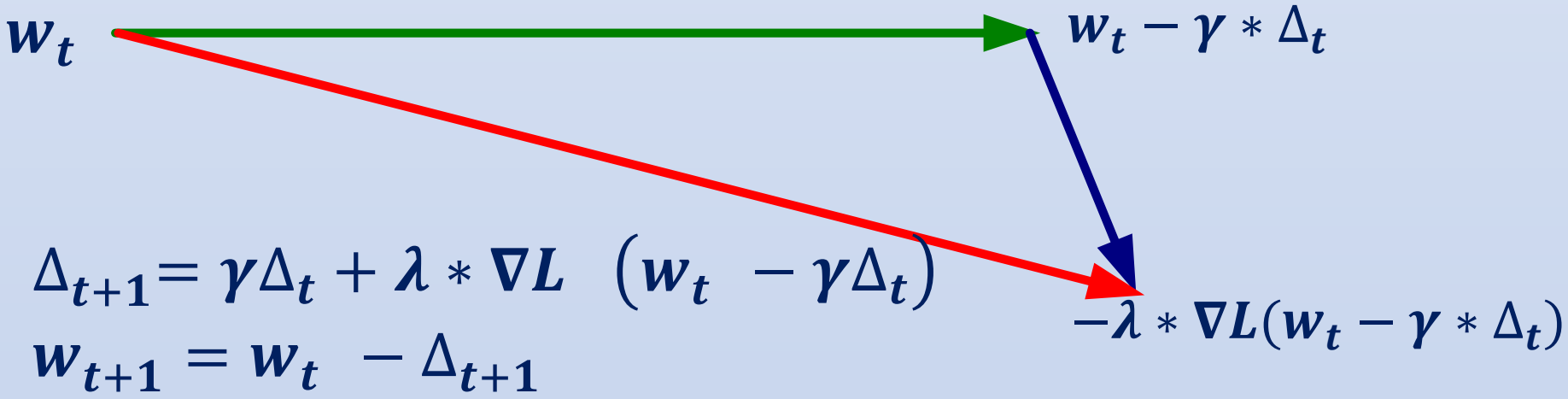
$$w_{t+1} = w_t - \Delta_{t+1}$$

Швидкість руху в напрямку мінімуму збільшується (бо цей напрямок присутній у всіх градієнтах), а осциляція гаситься. Ваговий параметр γ зазвичай вибирається рівним 0.9 чи близько до того.

Методи оптимізації

Прискорені градієнти Нестерова (Nesterov accelerated gradient)

Замість того, щоб обчислювати градієнт у поточній точці, використовується градієнт у точці "передбаченої" на підставі зсуву, розрахованого на попередньому кроці.



$$\Delta_{t+1} = \gamma\Delta_t + \lambda * \nabla L \quad (w_t - \gamma\Delta_t)$$

$$w_{t+1} = w_t - \Delta_{t+1}$$

Основний внесок в вектор зсуву дає перша складова, а складова із градієнтом лише «уточнює». Тому градієнт обчислюється в окресті нової точки, а не в поточної .

Методи оптимізації

AdaGrad (адаптивний градієнт). Загальна ідея – змінювати швидкість навчання λ для кожного параметра окремо, в залежності від того, як сильно змінюється параметр. Замість скаляру λ на кожній t ітерації використовується вектор

$$\lambda_t = (\lambda_t^{(1)}, \lambda_t^{(2)}, \dots, \lambda_t^{(d)})$$

Для $t = 1$ (перша епоха)

$\lambda_1^i = \lambda, i = 0, 1, \dots, d$, d – кількість параметрів (ваг).

Для t-ї епохи маємо:

$w_t = \left(w_t^{(1)}, w_t^{(2)}, \dots, w_t^{(d)}\right)$ - ваги.

$\lambda_t = (\lambda_t^{(1)}, \lambda_t^{(2)}, \dots, \lambda_t^{(d)})$ - швидкості навчання.

$\nabla L(w_t) = \left(g_t^{(1)}, g_t^{(2)}, \dots, g_t^{(d)}\right)$ - вектор градієнтів.

Методи оптимізації

AdaGrad.

Визначається додатковий вектор

$$G_t = \left(G_t^{(1)}, G_t^{(2)}, \dots, G_t^{(d)}\right),$$

де кожен компонент є сума квадратів часткових похідних функції помилки за відповідним параметром, тобто

$$G_t^{(i)} = \sum_{j=1}^{t} (g_j^{(i)})^2; i = 1, 2, \dots, d \ .$$

Кожен елемент вектору швидкості визначається як

$$\lambda_t^{(i)} = \lambda \Big/ \sqrt{G_t^{(i)} + \epsilon}.$$

Тут $\epsilon \approx 10^{-8}$ мала, запобіжник від ділення на нуль.

Методи оптимізації

AdaGrad.

На останнє

$$w_{t+1} = w_t - \lambda_t \odot \nabla L(w_t),$$

Операція $\odot$ - покомпонентне множення вектору на вектор, або

$$w_{t+1}^{(i)} = w_t^{(i)} - \lambda_t^{(i)} g_t^{(i)}, i = 1, 2, \dots, d$$

Оптимізація градієнтного спуску

Оптимізатор	Рік	Швидкість навчання	Градієнт
Momentum	1964		Yes
AdaGrad	2011	Yes	
AdaDelta	2012	Yes	
Nesterov	2013		Yes
Adam	2014	Yes	Yes
AdaMax	2015	Yes	Yes
Nadam	2015	Yes	Yes
AMSGrad	2018	Yes	Yes

TF2 Оптимізатори

Оптимізатор	Посилання
SGD	tf.keras.optimizers.experimental.SGD
AdaGrad	Tf.keras,optimizers,experimental.Adagrad
AdaDelta	tf.keras.optimizers.experimental.Adadelta
Nesterov	tf.keras.optimizers.experimental.SGD (Nesterov)
Adam	tf.keras.optimizers.Adam
AdaMax	tf.keras.optimizers.experimental.Adamax
Nadam	tf.keras.optimizers.experimental.Nadam
AMSGrad	

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

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_04_Exmpl_1.pdf

The END
Модуль 7. Лекція 7.4