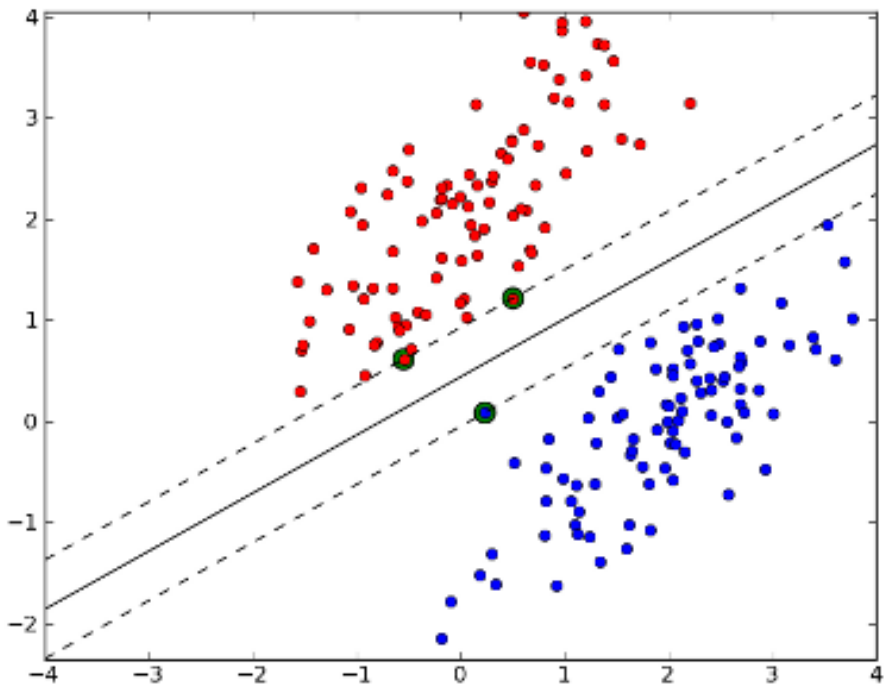
Тема 4 «Кластеризація та зменшення вимірності. Навчання з підкріпленням»

Метод опорних векторів (Support Vector Machines)

Розглянемо такий метод вирішення задачі класифікації на два класи, як **метод опорних векторів (support vector machine).**

Якщо у нас є певна множина прикладів, де кожен приклад належить до одного з двох класів, ми відповідно маємо певну множину точок, яку можна розділити на дві частини певною прямою або гіперплощиною (якщо ми маємо справу з багатовимірними даними), яка буде знаходитись посередині між класами і відстань до найближчих точок буде максимальною.

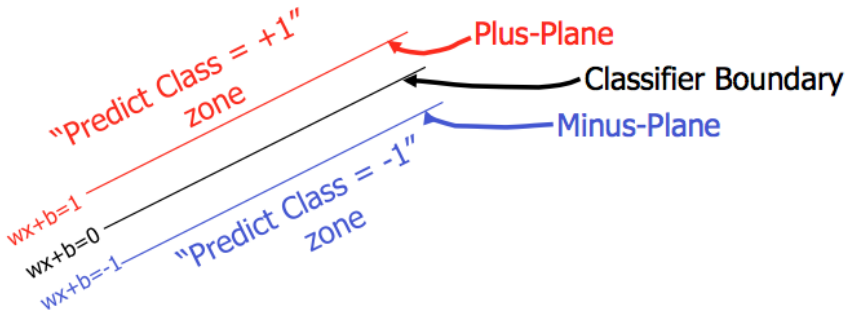


Метод опорних векторів вирішує задачу знаходження такої гіперплощини, відстань від якої до найближчих прикладів з кожного боку площини буде максимальною.

Методом опорних векторів ми шукаємо таку функцію, яка для кожного із прикладів першого класу буде більше 0, а із другого класу – менше 0. Мітки

класів ми можемо розглядати відповідно як +1 та -1. В результаті нам потрібно знайти таку функцію, щоб відстань до найближчих векторів з кожного боку буда максимальною. Саме такі вектори і називаються **опорними**, бо мають максимальний вплив на те, якою буде пряма, що розділяє два класи відповідає на питання, до якого класу належить точка.

Процес вирішення задачі методом опорних векторів зводиться до переформулювання задачі в систему рівнянь, в якій ми шукаємо такий набір вагових коефіцієнтів.



- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} . \boldsymbol{x} + b = -1 \}$

Classify as.. +1 if $\boldsymbol{w} . \boldsymbol{x} + b >= 1$

-1 if $\boldsymbol{w} . \boldsymbol{x} + b <= -1$

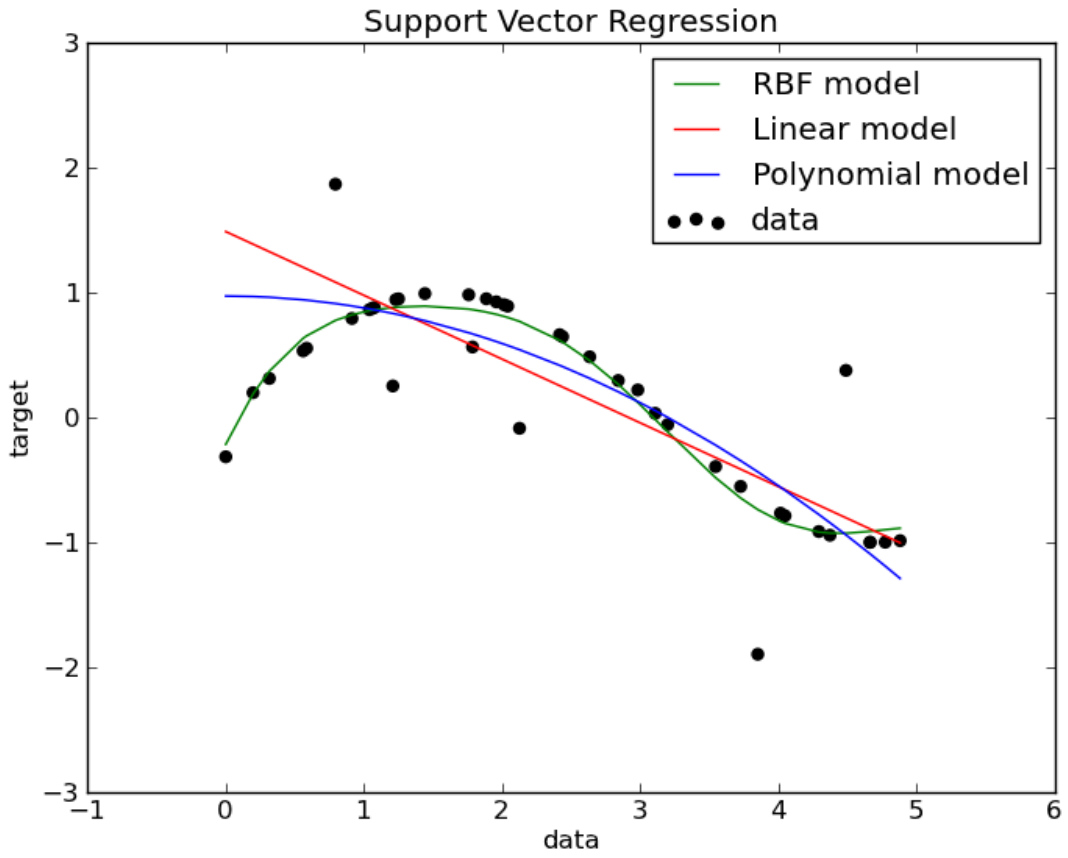
Universe explodes if $-1 < \boldsymbol{w} . \boldsymbol{x} + b < 1$

Звідки беруться вагові коефіцієнти? Нагадаємо, що пряма на площині задається рівнянням y=kx+b, де k – відповідає за нахил прямої, а b – за зміщення прямої вздовж осі. У випадку багатовимірної площини замість k ми маємо певну матрицю коефіцієнтів. Якщо замість x ми підставимо конкретний приклад і помножимо на транспоновану матрицю вагових коефіцієнтів, на виході отримаємо додатне чи від'ємне число. Таким чином ми можемо стверджувати, до якого з класів належить точка.

В реальності досить часто виникає ситуація, коли точки не можуть бути розділені лінійно. Буває, що класи мають невеликий перетин або не є ідеально лінійно роздільними. Таким чином ми намагаємось знайти таку

пряму чи гіперплощину, в яких окрім того, що найближчі вектори з кожного боку будуть максимально віддаленими, сума різниць векторів з протилежного боку мають бути мінімальними. Це так зване «епсілон», яке ми додаємо в нашу систему рівнянь.

В такому випадку наша пряма чи гіперплощина не буде ідеально розділяти два класи, але це буде оптимальна пряма чи гіперплощина, яка на більшій кількості прикладів видасть нам необхідний розподіл.



Метод опорних векторів використовується досить широко через його неймовірну простоту та доступність реалізації. У випадку даних, які можна розділити лінійно і ми точно впевнені, що можна провести одну пряму чи гіперплощину і отримати точний результат належності точки до одного з класів, в процесі знаходження такої прямої чи гіперплощини ми шукаємо певний набір вагових коефіцієнтів. Підставивши їх в наше рівняння, ми вважаємо, що наша пряма чи гіперплощина повинна проходити чітко

посередині між найближчими прикладами з кожного класу. Такі найближчі приклади будуть лежати на паралельних прямих чи гіперплощинах і називаються **опорними векторами**, оскільки вони впливають на те, під яким кутом знаходиться пряма чи гіперплощина.

Попри те, що після етапу формулювання системи рівнянь методи вирішення цієї системи не такі очевидні і описання математичного апарату такого рішення потребує значної кількості сил та часу, для того, щоб використовувати цей метод для вирішення прикладних задач із використанням доступних бібліотек машинного навчання, достатньо лише розуміти, що застосування методу опорних векторів – це пошук прямої, яка розділяє максимально ефективно два класи.

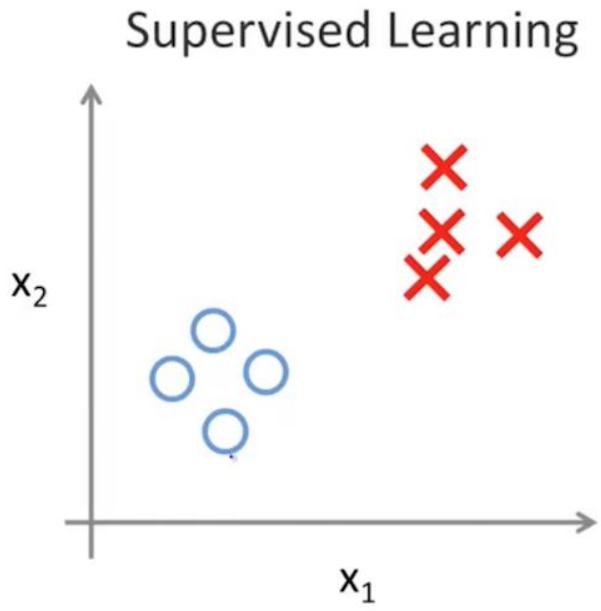
Кластеризація та зменшення вимірності. Навчання з підкріпленням

В попередніх матеріалах, що стосувалися методів навчання нейронних мереж – оптимізації логістичної регресії в задачах класифікації, кластеризації тощо – процес навчання виглядав наступним чином. Є певний набір об’єктів з набором ознак, які кодуються певним чином і даються на вхід системі. На виході ми знали правильні відповіді. Тобто, якщо ми хотіли натренувати систему ефективно розділяти зображення на два класи, отже ми заздалегідь знали, які зображення з тренувальної вибірки до яких класів належать.

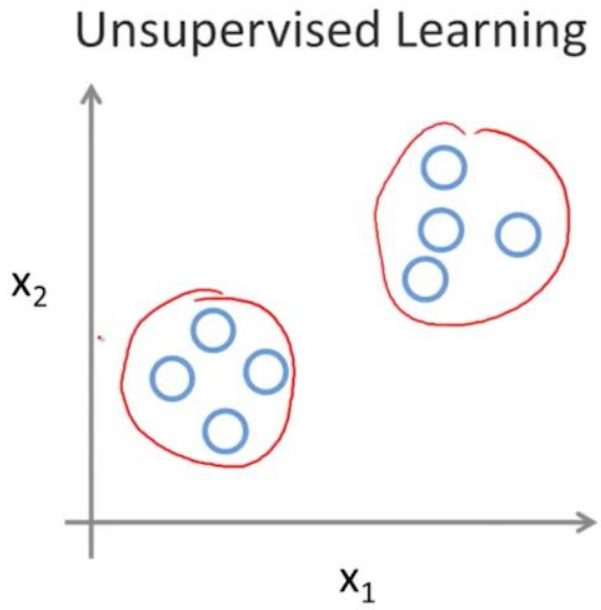
Але існують ситуації, коли ми маємо дані, набір об’єктів, їх певне ознакове описання, але ми не маємо інформації, до яких класів вони належать. Більше того, є необхідність дістати з ознакового описання цих об’єктів якісь дані, які б ми могли використати для класифікації чи кластеризації об’єктів.

Отже, перша задача, коли ми маємо мітки на виході, і друга задача – коли ми маємо тільки ознакове описання, відрізняються між собою методами

навчання. Для першої тип машинного навчання – **навчання з вчителем** (ми виступаємо в ролі вчителя, бо ми надаємо приклади правильних відповідей), для другої – **навчання без вчителя**, коли система, маючи інформацію про розподіл даних, повинна екстрагувати з цього розподілу корисну для навчання інформацію.

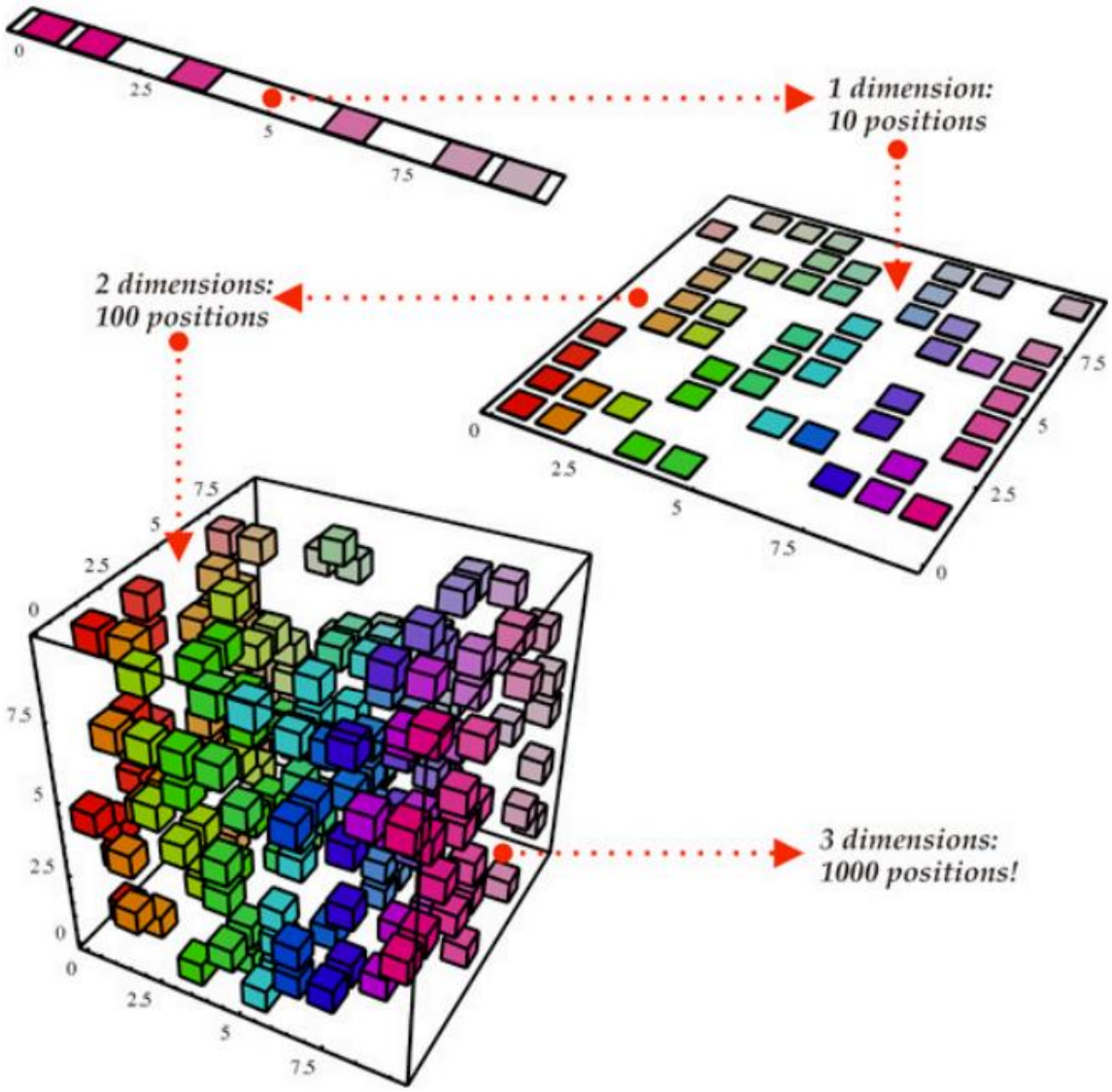


Є декілька типових задач в навчанні без вчителя. Одна з найпопулярніших – задача кластеризації: коли є велика кількість прикладів, які потрібно розбити на певну кількість класів, ми заздалегідь не знаємо, який із прикладів до якого класу належить. Алгоритм кластеризації має вивчити закономірності і видати номери класів і які елементи до яких класів належать.



Один із прикладів алгоритму кластеризації ми розглядали – **k-means**. Він є базисом багатьох інших більш комплексних алгоритмів. В задачах навчання без вчителя, як правило, наші дані представлені або у вигляді ознакового описання, або у вигляді матриці відстаней.

Припустимо, ми маємо сто тисяч точок у стовимірному просторі, і кожній точці відповідає стовимірний вектор, і нам потрібно знати, як ці точки розподілені – які знаходяться поряд чи далеко, чи є закономірності. Один із методів вирішення даної задачі – взяти попарні відстані між усіма точками, отримаємо матрицю відстаней. В цій матриці ми ігноруємо конкретну інформацію про ознакове описання значення кожного конкретного вектора, оскільки для нас цінність несе тільки позиція кожної точки відносно всіх інших. Тут є нюанс: людина досить погано сприймає інформацію в більш ніж тривимірних проекціях і може нормально розібратися із структурою даних у дво- чи тривимірній репрезентації. Але в задачах машинного навчання ми маємо справу із багатовимірними векторами.



Отже, часто для того, щоб зрозуміти наскільки ефективно певний алгоритм класифікації чи кластеризації перетворює дані в багатьох вимірах, нам потрібно зменшити вимірність.

Задача зменшення вимірності – типова задача навчання без вчителя. Ми можемо дістати з наших даних матрицю відстаней і на її основі сконструювати меншвимірну репрезентацію точок, розподіл яких буде максимально відображати розподіл цих точок в оригінальній багатовимірній розмірності векторів.

Як правило, нас цікавить інформація про те, які точки знаходяться поряд, а які – далеко. Не завжди потрібно використовувати всю матрицю відстаней і

не завжди необхідно враховувати відстань до дуже віддалених точок, особливо у випадку багатовимірних репрезентацій.

Часто використовуються відстані до найближчих сусідів: ми беремо набір точок і до кожної точки дивимось, припустимо, 100 найближчих сусідів, зберігаємо їхні індекси і відстані до них. Інші точки ми ігноруємо.

Як показує практика, якщо ми маємо справу із реально великою кількістю точок, 100 найближчих сусідів достатньо для того, щоб викрити локальну і нелокальну структуру в розподілі даних. Такий метод вибору найближчих сусідів часто використовується в таких алгоритмах зменшення вимірності, як, наприклад, t-SNE.

Ідея в тому, що не потрібно знати, на скільки Ви знайомі із кожним із 7 мільярдів людей на Планеті, достатньо знати список друзів однієї конкретної людини. І, якщо ми візьмемо весь граф списку друзів кожної людини, то ми можемо в сукупності відтворити структуру загального графу соціальних зв’язків людей на всій Землі.

Кластеризація та зменшення вимірності. Автокодувальники.

Сама задача зменшення вимірності для свого вирішення часто потребує знаходження певних прихованих закономірностей в розподілі даних.

Процес знаходження прихованих закономірностей в розподілі даних є невід'ємним в процесі вивчення чогось нового і генералізації.

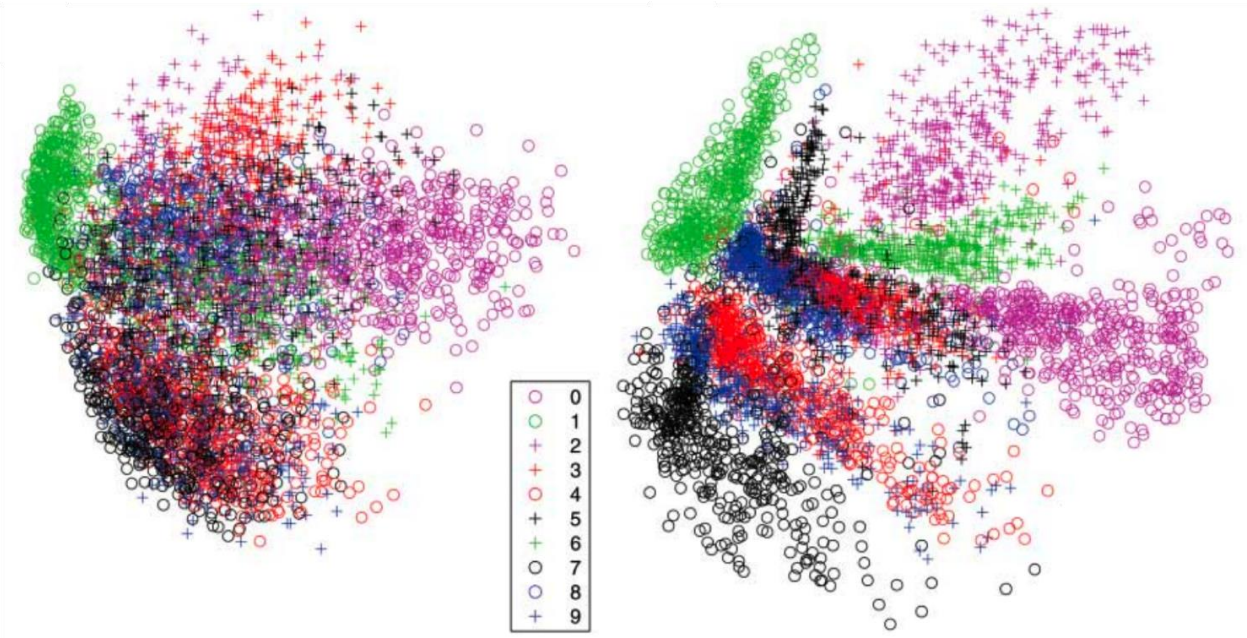
Ця ідея лягла в основу такого методу отримання нового з даних, як **автоенкодери**, або **автокодувальники**. Ідея полягає в наступному. Припустимо, ми маємо певний набір об'єктів з ознаковим описанням – набір картинок з рукописними символами цифр від 0 до 9.

Картинки мають розмір 10 на 10 і вони є чорно-білими. Вхідна картинка – це набір ознакового описання, де ознакою є один піксель із певним рівнем яскравості І ми знаємо, що символів є усього 10. Ми **не знаємо**, який конкретно символ намальований на кожній картинці.

Як ми можемо вирішити задачу навчання машини розпізнавати символи без початкового навчання машини, що на одній картинці намальовано цифру 1, на іншій – 2 і т.д.?

Коли ми дивимось на символи, ми бачимо, що на картинках є певні приховані закономірності. Звісно, всі цифри 1, написані різними людьми від руки різними шрифтами – відрізняються. Але у всіх цифр 1 між собою є набір спільних ознак. Так само у всіх цифр 5, 8 і т.д.

Відповідно, було б чудово сконструювати систему, яка б викрила ці закономірності і, використовуючи їх, кластеризувала всі приклади на 10 кластерів.



Завдяки цьому, для розпізнавання системою символів нам не потрібно розмічати кожен із сотень тисяч прикладів (для більш складних задач), а достатньо подивитися на кластери і помітити, що в конкретному кластері система зібрала до купи всі картинки, на яких зображена, наприклад, цифра 4. Відповідно, даний кластер можна віднести до цієї цифри.

Тобто ми отримуємо систему, яка розпізнаватиме символи без фактичного вказування системі, що означає цифра на кожному із прикладів.

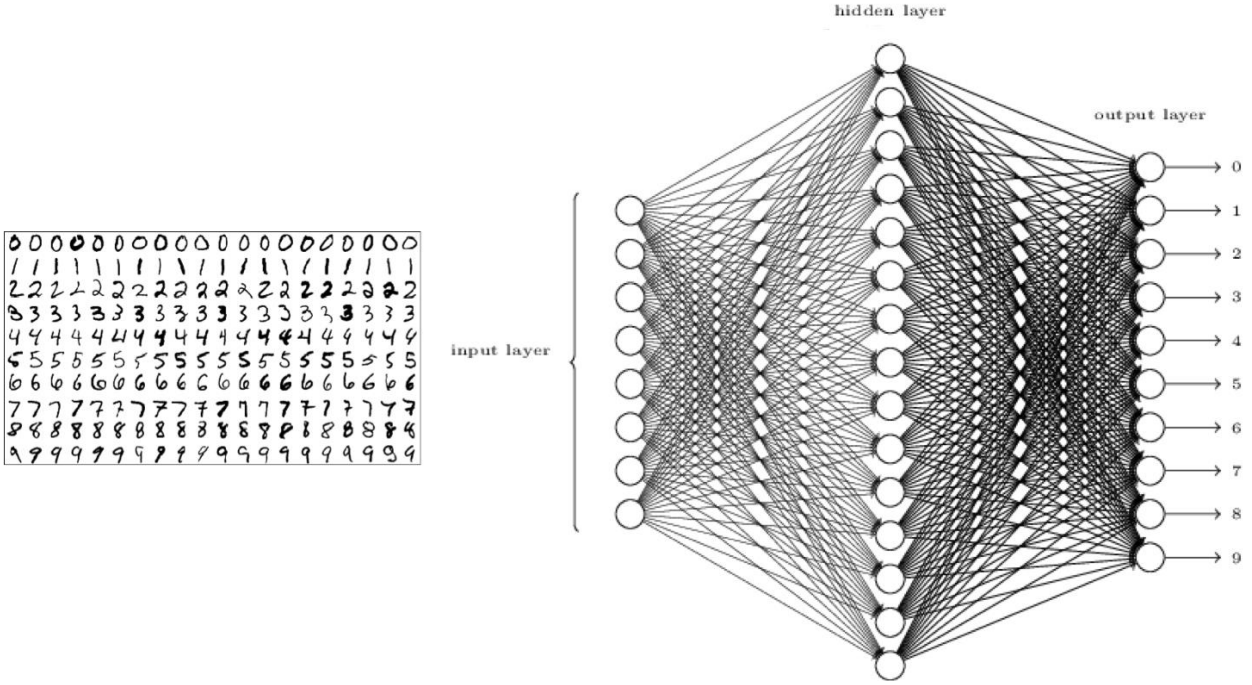
За рахунок чого стають можливими подібні речі? Припустимо, ми маємо картинку 10 на 10 пікселів, тобто вхідні дані мають розмірність 100. Тобто це буде стовимірний вектор, з яскравістю точок від 0 до 1. Ми не знаємо, якій цифрі відповідає конкретний вектор, але ми можемо сконструювати систему

із наступною задачею: за наявним прикладом **реконструювати** на виході таку саму картинку.

Це не є просте копіювання. Ідея полягає в наступному.

Припустимо, що вхідна розмірність – 100 пікселів, а на виході ми маємо отримати цю ж картинку із 100 пікселів з мінімальною втратою якості.

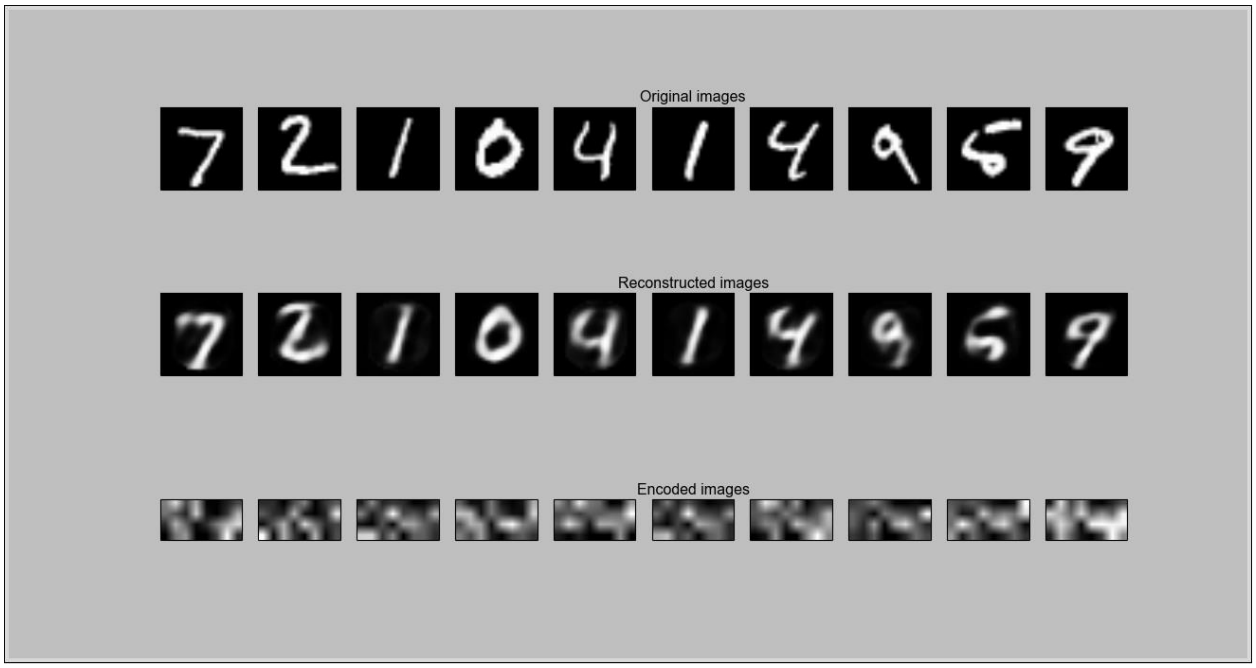
Сконструюємо одношарову нейронну мережу, у якої перший шар – вхід розмірністю 100, другий прихований шар – 500, а вихідний – 100.



Після того, як ми надамо такій НМ декілька тисяч картинок, ми помітимо, що вона реконструює кожну картинку майже піксель в піксель з невеликими похибками. Як працює така нейромережа? Спочатку на вході вектор розмірністю 100 трансформується у вектор розмірністю 500 з цією ж закодованою інформацією, а потім він назад стискається у вектор розмірністю 100. Тому процес знаходження прихованих закономірностей в розподілі даних є невід'ємним в процесі вивчення нового і генералізації І якщо у прихованому шарі розмірність більша в 5 разів за розмірність картинки, то проблеми в репрезентації патернів, які є в картинці, немає.

Але у випадку, коли середній шар буде не 500 нейронів, а, припустимо, 10 нейронів, ситуація кардинально зміниться. Архітектура нашої нейромережі

буде найвужча посередині: вхідний шар розмірністю 100 – прихований 10 – вихідний 100. У даному випадку задача навчання зводиться до того, щоб пропустити інформацію, яка зберігається на 100 пікселях, через 10 нейронів так, щоб можна було ефективно реконструювати оригінальну картинку з мінімальною втратою якості. Ітерація за ітерацією, навчаючи нейромережу методом зворотного поширення помилки, ми помітимо, що реконструкція символу стає більш подібною на оригінал.



Цифра 1 схожа на 1, 5 на 5 і т.д. Яким чином нам вдалося ефективно стиснути інформацію розмірності 100 до розмірності 10. Єдиний спосіб цього досягти – знайти певні приховані **закономірності** в цій картинці. Наприклад, вивчити кути, краї, прямі, частини картинки.

Зрозуміло, що в практичній реалізації архітектура буде складнішою за тришарову: між вхідним шаром і найвужчим шаром (**bottleneck-шаром**) буде декілька конволютивних шарів (ми вивчимо фільтри, щоб визначити з чого складається наше зображення, щоб описати його більш високоабстрактною мовою).

В результаті, коли система навчилася описувати інформацію на картинці не мовою пікселів, а мовою країв, комплексних градієнтів, складних частин,

довжина цього описання зменшиться в рази.

Якщо ми перекодуємо кожну з картинок в стислу десятивимірну репрезентацію, а потім застосуємо алгоритм кластеризації десятивимірних векторів, ми помітимо, що в десятивимірному просторі різні картинки скупчуються в десятьох зонах. Звісно, будуть приклади, які знаходитимуться майже посередині (існуватимуть приклади написання символів 1 та 7, що фактично знаходитимуться посередині між кластерами 1 та 7). Але центроїди кластерів будуть очевидні.

Відповідно, не маючи конкретних правильних відповідей, дивлячись на розподіл прикладів, ми можемо постфактум надати мітки кластерів, і або вручну перевірити, чи проектується дана картинка в якийсь кластер, або закодувати картинку в стиснену високорівневу репрезентацію і подивитися, як далеко до кожного із кластерів знаходиться наш вектор.

Кластеризація та зменшення вимірності. Векторна репрезентація мови.

Це приклад застосування **алгоритму навчання без вчителя – алгоритм автоенкодер**. Автоенкодер буває різної структури, але основна ідея в них однакова: половина мережі звужує репрезентацію перед bottleneck – т.зв. енкодер, а інше розширює репрезентацію – декодер (декодує інформацію із стисненої репрезентації).

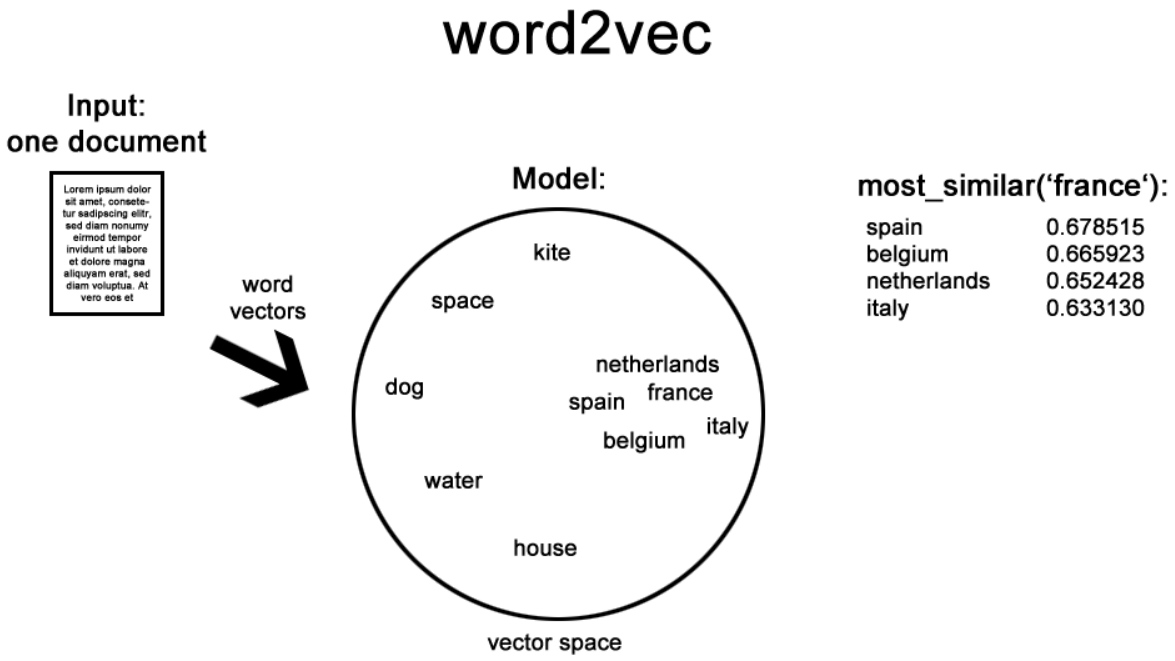
Подібний метод (хоча це не зовсім автоенкодер) використовується для створення векторної моделі мови.

В результаті роботи автоенкодера,завдяки його половині - енкодеру, ми маємо систему, яка отримує на вхід картинку, а на виході видає певний

вектор, з яким працювати простіше, оскільки він кодує в собі високоабстрактну інформацію, а не просто колір пікселів.

Аналогічна проблема існує з обробкою тексту. Символи в слові мало говорять про сенс цього слова. Відповідно потрібно перекодувати слова в таку репрезентацію, в якій вектор, що відповідає слову, міститиме більше інформації про сенс цього слова і його зв'язок з іншими словами.

Відомий алгоритм для цього – **word2vec**, який дозволяє навчити комп’ютер працювати зі словами на більш високоабстрактному рівні.



Припустимо, ми маємо весь текст української Вікіпедії. Після фільтрації від усіх непотрібних символів та частин її розмір складе порядку 8-9 Гб тексту. Відповідно ми маємо основу для навчання без вчителя – велика кількість даних із купою прихованих закономірностей. Ми хочемо навчити комп’ютер читати слова на більш високоабстрактному рівні, ніж просто набір символів.

Що ми робимо? Ми робимо словник, де записуємо не тільки всі слова, що вживалися у Вікіпедії, але й скільки разів вживалося кожне слово. Візьмемо, наприклад, всі слова, які зустрічались принаймні 10 разів. В результаті отримуємо словник, припустимо, на 100 тисяч слів. Відповідно весь наш

гігантський дата сет на 9 Гб ми можемо перекодувати і замінити кожне слово на номер в словнику. Відповідно вся Вікіпедія виглядатиме, як величезна послідовність індексів слів.

List

index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

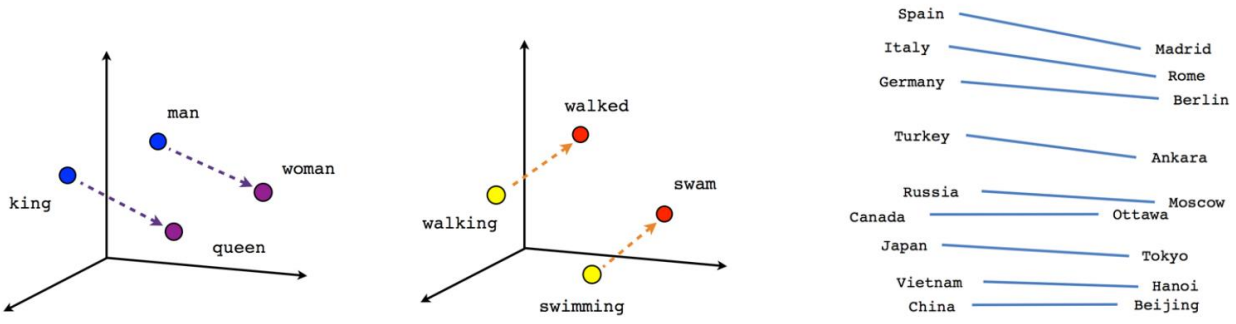
Створимо нейронну мережу з двох шарів, яка на вхід вона отримує набір індексів слів з вікном на 5 слів (нейромережа проходить вікном в 5 слів і дивиться на індекси кожного слова, що потрапляє в поле зору). Як правило ці слова будуть тим чи іншим чином пов’язані між собою. Сформулюємо для неї задачу наступним чином: маючи певну послідовність слів з вікна у 5 слів, отримуючи на вхід всі індекси, що були у цьому вікні, окрім центрального слова, та маючи інформацію про індекси сусідніх слів, спрогнозувати, яке слово знаходиться посередині. Або можемо сформулювати задачу інакше: маючи індекс певного слова, спрогнозувати, які слова його будуть оточувати.

Ці дані у нас наявні, відповідно ми автоматично отримуємо мітки із сусідніх слів або центрального слова. Тренуємо мережу, яка має на вході кількість нейронів відповідно до розмірів нашого словника, а на виході вона прогнозує очікування по кожному із тисяч слів словника. Прихований шар покликаний поставити у відповідність кожному слову певний вектор розмірністю у 100 нейронів. Відповідно наша нейромережа матиме наступну архітектуру: 100 000 нейронів вхідних – 100 нейронів прихований шар – 100 000 вихідний шар. На вхід ми активуємо індекси сусідніх слів, вони активують прихований шар, а використовуючи дані в прихованому шарі ми хочемо отримати

розподіл очікувань, які слова будуть поряд чи яке слово було центральним.

Повторюючи даний процес мільярди разів, рухаючись віконечком по Вікіпедії, ваги, що відповідають кожному із 100 тисяч слів, будуть коригуватись таким чином, що асоціативно наближені слова будуть мати дуже подібні векторні репрезентації.

Стовимірні вектори, що відповідають словам «автомобіль» і «машина», будуть досить подібні і набагато ближчі, ніж, наприклад, слово «гуляти». В результаті навчання системи, ми можемо пройтись по всьому нашому словнику, дістати стовимірну векторну репрезентацію кожного слова і отримати словник стовимірної карти: сто тисяч точок (слів), яким відповідатимуть асоціативно пов'язані слова, які зустрічаються в подібних контекстах. Більше того, в даних векторних просторах ми отримуємо багато переваг у порівнянні із звичайною роботою із текстами. Окрім того, що сусідні за значенням слова проектуватимуться у сусідні точки, досить часто паралельним векторам відповідатимуть аналогії. Наприклад, якщо взяти вектор від точки «чоловік» і точки «король», і додати його до точки «жінка», то ми помітимо, що він вкаже на точку «королева». Відповідно паралельним перенесенням відповідатимуть аналогії.



Male-Female Verb tense Country-Capital

Якщо конструювати таку модель тексту з декількох мов, то паралельне перенесення відповідатиме перекладам з однієї мови на іншу. Звісно, точність методу залежить від різноманітності даних, кількості тренувань тощо.

Але сама ідея векторної репрезентації слів для обробки і аналізу тексту принципово змінила підхід до всіх задач natural language processing – обробки природньої мови. Тепер ми можемо думати про слова, як певні точки в гіперпросторах, додавати, віднімати, брати середнє значення від декількох слів і екстрагувати точки, які відповідають певним неявним сенсам. Ми можемо спостерігати закономірності в формулюваннях речень, яким відповідають специфічні траєкторії в гіперпросторі (кожне слово – точка в багатовимірному просторі, речення – траєкторія руху по цим точкам в цьому просторі). Аналізуючи траєкторії, закономірності, відхилення, ми можемо класифікувати сенс, який міститься у реченнях, а не просто аналізувати його на рівні правил заміни, символів, префіксів тощо.

Більшість даних, особливо в Інтернеті, доступні у вигляді нерозмічених даних. Вікіпедія – гігантський об’єм доступних даних, але до кожного слова ми не маємо окремої мітки. Підбираючи вірні алгоритми машинного навчання, ми можемо дістати нову цінну інформацію з доступних ресурсів. І тільки від нас залежить, який ефективний алгоритм машинного навчання без вчителя ми підберемо для того, щоб на основі них дістати нові корисні дані і використовувати їх, як базу для машинного навчання з вчителем або для вирішення конкретних прикладних задач.

Навчання з підкріпленням. Функція підкріплення.

Ми ознайомилися з декількома типами вирішення задач регресії, класифікації, кластеризації. І, як правило, дані задачі вирішувалися **методом навчання з вчителем**. Також ми розглядали, що таке **навчання без вчителя** і в чому полягає ключова відмінність.

Але окрім типового навчання з вчителем, коли ми маємо відомий набір прикладів і відомий набір очікуваних міток до цих прикладів, та окрім

типового навчання без вчителя, коли ми маємо набір прикладів і нам потрібно їх, наприклад, кластеризувати, існує метод навчання, коли від системи потрібно отримати пропозиції ефективних дій в певній ситуації. Досить часто, коли подібні задачі виникають, у нас відсутня наперед заготовлена тренувальна вибірка, якій ми можемо спочатку підготувати тренувальний датасет, в якому є приклади і мітки, і далі навчати систему проектувати простір прикладів, простір міток.

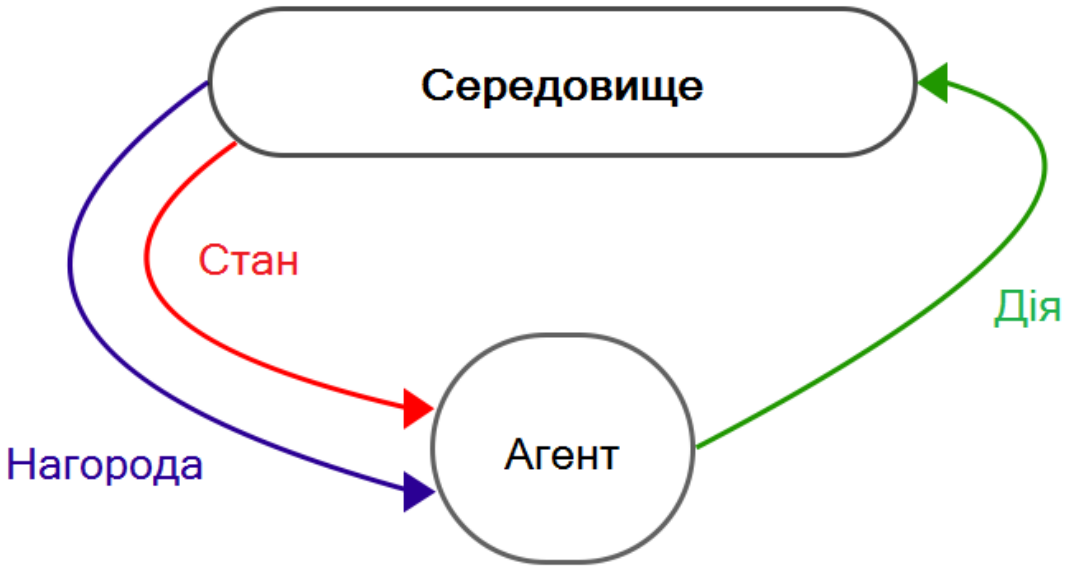
Типовою задачею, яка не підпадає ні під класичне навчання з вчителем, ні під навчання без вчителя, є задача керування автомобілем. Припустимо, віртуальною моделлю автомобіля у грі. В чому полягають базові відмінності?

Як правило, існують задачі, в яких ми не можемо заздалегідь підготувати дата сет, оскільки дії, які повинна вибрати система, залежать від поведінки і реакції середовища.

Задачі знаходження ефективних правил поведінки в певному середовищі

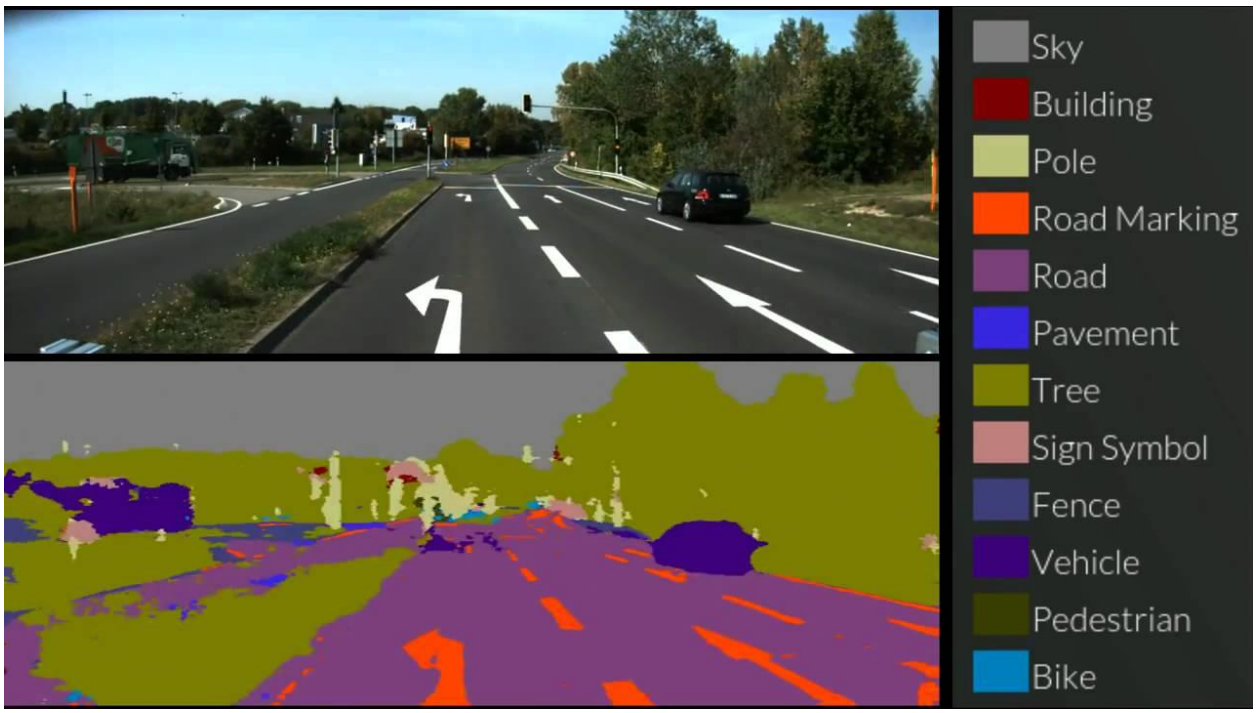
вирішуються методом **навчання з підкріпленням.**

Є певний агент, який може приймати певні дії, і є середовище, в якому він знаходиться. Агент приймає певну із можливих дій і середовище на неї реагує тим чи іншим чином.



Задача агента – розробити стратегію вибору дій так, щоб максимізувати позитивний або мінімізувати негативний відклик середовища.

Припустимо, у нас є гра, в якій моделюється віртуальний автомобіль на трасі. І ми хочемо натренувати систему так, щоб вона могла керувати автомобілем так, щоб він не з'їжджав з дороги, вписувався в повороти і не стояв на місці. Фактично, ми маємо середовище, в якому з різним рівнем деталізації моделюються певні фізичні закономірності, є агент – автомобіль, є набір сенсорів цього автомобіля (тобто ми певним чином можемо надати системі інформацію про поточний стан агента – позиція, зображення з віртуальної камери спостереження і т.п.). Автомобіль може робити певні дії – пришвидшуватись, вповільнюватись, повертати тощо.



В нас є певний словник можливих дій.

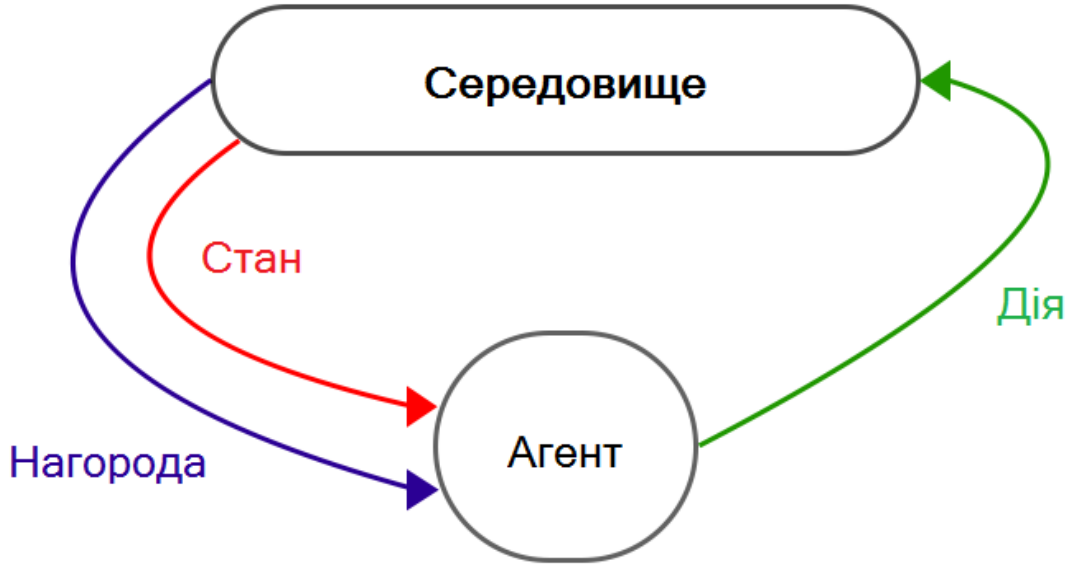
І ітерація за ітерацією ми спостерігаємо поточний стан автомобіля – де знаходиться, яка відстань до країв дороги, яка швидкість. Ми описуємо всі ці ознаки у вигляді певного вхідного вектора в систему. Система, отримавши вектор, має видати список дій – одну або декілька – яку має зробити автомобіль на наступній ітерації. Наприклад, система видала – пришвидшитись вперед – машина пришвидшилась, середовище відреагувало відповідним чином – припустимо, машина виїхала за межі дороги. Відповідно, як відклик середовища, ми попередньо розробляємо певну функцію – **функцію підкріплення.** В даному випадку, якщо ми хочемо уникнути виїзду автомобіля за межі дороги, коли відстань до межі дороги стала від’ємною, ми можемо, помноживши на певний коефіцієнт цю відстань, надати системі негативне підкріплення.

Навчання з підкріпленням. Етапи та основні проблеми.

Далі методи оптимізації істотно відрізняються, але в загальній структурі метод навчання з підкріпленням розподіляється на декілька базових етапів:

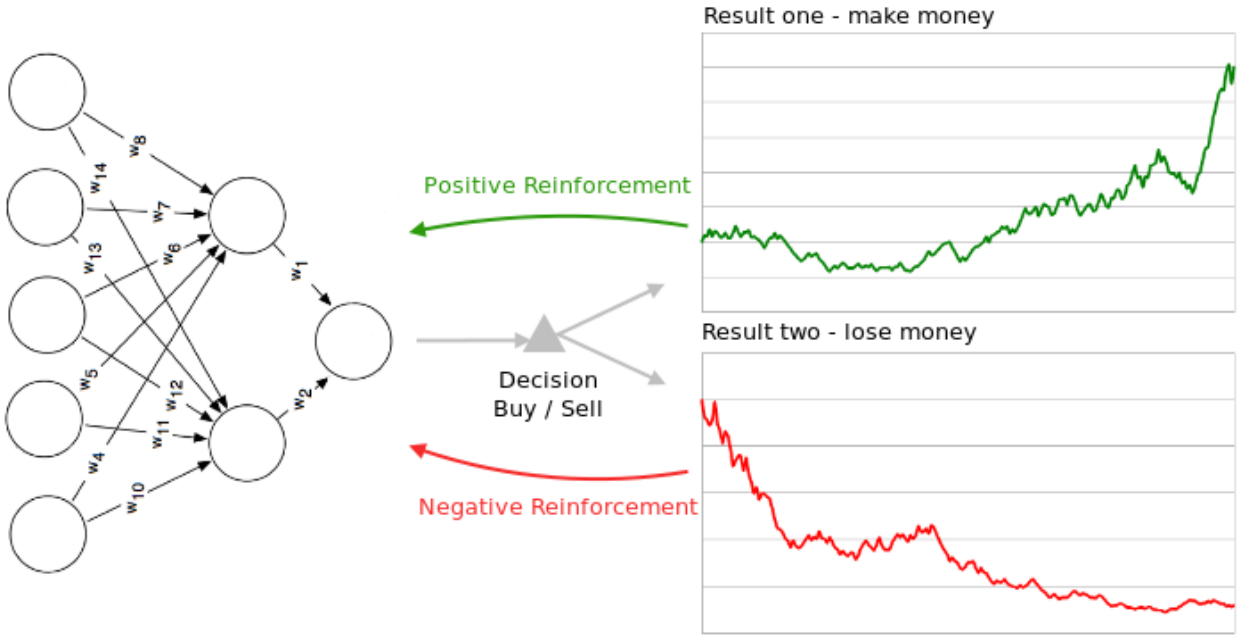
- Етап отримання поточного стану агента в середовищі;
- Етап аналізу цього стану і видачі стратегії дії;
- Етап дії;
- Етап відклику середовища.

В результаті, ми отримуємо зациклену систему, в якій середовище і агент постійно взаємодіють таким чином, щоб агент адаптувався до середовища.



Задачі машинного навчання, які використовуються в навчанні з підкріпленням, відрізняються між собою досить сильно в тому, як реалізується фактично кожен із етапів навчання, оскільки це може бути ігрове, змодельоване або реальне середовище. Наприклад, ми можемо тренувати агента, задачею якого є максимізація прибутку на біржі. Відповідно середовищем є біржа і всі приховані закономірності, що їй притаманні. В агента є доступні дії – купити/продати на певну суму акції і

максимізувати прибуток за певний період.



Навчання з підкріпленням досить широко використовується в робототехніці, оскільки зазвичай стратегія поведінка робота в конкретній ситуації є невідомою, але ми точно впевнені, яку його поведінку можна обмежувати, а яку – підкріплювати.

Є декілька ключових проблем, які вирішуються в машинному навчанні з підкріпленням.

Для того, щоб агент обрав ефективну стратегію поведінки, йому потрібно вивчити відклики середовища на кожну із можливих дій. Часто буває так, що можливих дій, які агент може прийняти, досить велика кількість. Навіть, якщо словник відкликів невеликий, як правило в навчанні з підкріпленням враховуються попередні стани агента та його попередні дії. Наприклад, у випадку з автомобілем на дорозі, в залежності від того, на якій частині дороги авто знаходиться, відклики на її дії можуть відрізнятися досить сильно.

Якщо сформулювати задачу, як просто максимізація позитивного відклику середовища і знайти таку дію, яка дає такий відклик, вірогідна ситуація, коли система просто із перших можливих дій обере ту, що надає максимальний

відклик і буде її повторювати. Тому існує велика імовірність потрапити в локальний мінімум і не вивчити загалом середовище.

Тому основною проблемою, яку потрібно вирішити є **максимізація дослідження середовища**. Для того, щоб агент був ефективним, він має максимально дослідити простір можливих дій. У випадку тренування віртуального автомобіля, ми маємо сконструювати середовище і процес отримання відклику таким чином, щоб агент мав потрапити у максимальну кількість можливих ситуацій. Від цього залежить, яку кількість можливих сценаріїв поведінки агент зможе генералізувати і на базі них обрати стратегію. Але, з іншого боку, якщо ситуації будуть сильно відрізнятися (в реальному світі можливих ситуацій є незліченна кількість), нам потрібно певним чином обмежити допустимі дії.

Це як правило вирішується так: припустимо, є абстрактне середовище, 10 доступних дій, які може зробити агент, і є певний вектор відклику середовища, якщо ми будемо приймати максимально ефективну дію в поточний момент часу, агент не дослідить усе середовище. З іншого боку, якщо ми будемо просто перебирати всі доступні дії, система може не досягнути конвергенції і все одно не зібрати достатню кількість статистики, через те, що можливих відкликів середовищ може бути незліченна кількість. Отож, як правило вводиться певний відсоток випадкових дій. Припустимо, 90% дій агент робить на базі стратегії, яку він намагається оптимізувати на кожній ітерації, а 10% - випадково, просто для того, щоб дослідити, які ще можливі дії ми можемо зробити, щоб не застрягнути у локальному мінімумі.

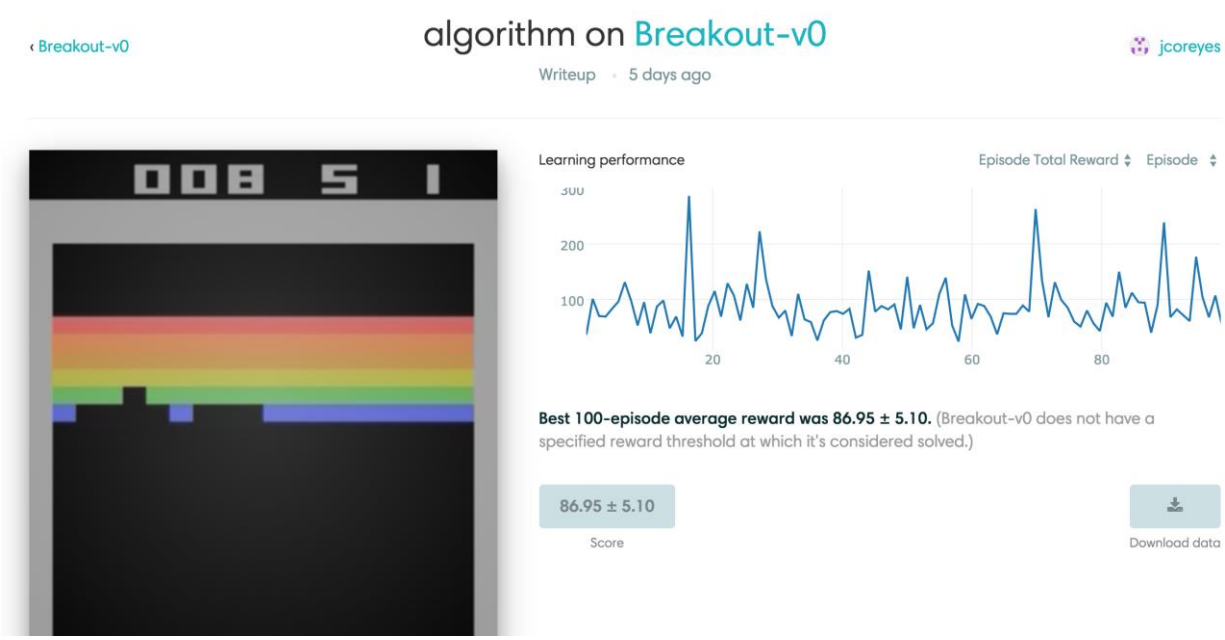
Вірогідність того, що ми віднайдемо стратегію, яка надасть нам найкращі результати – збільшується.

Навчання з підкріпленням. Середовища та ресурси.

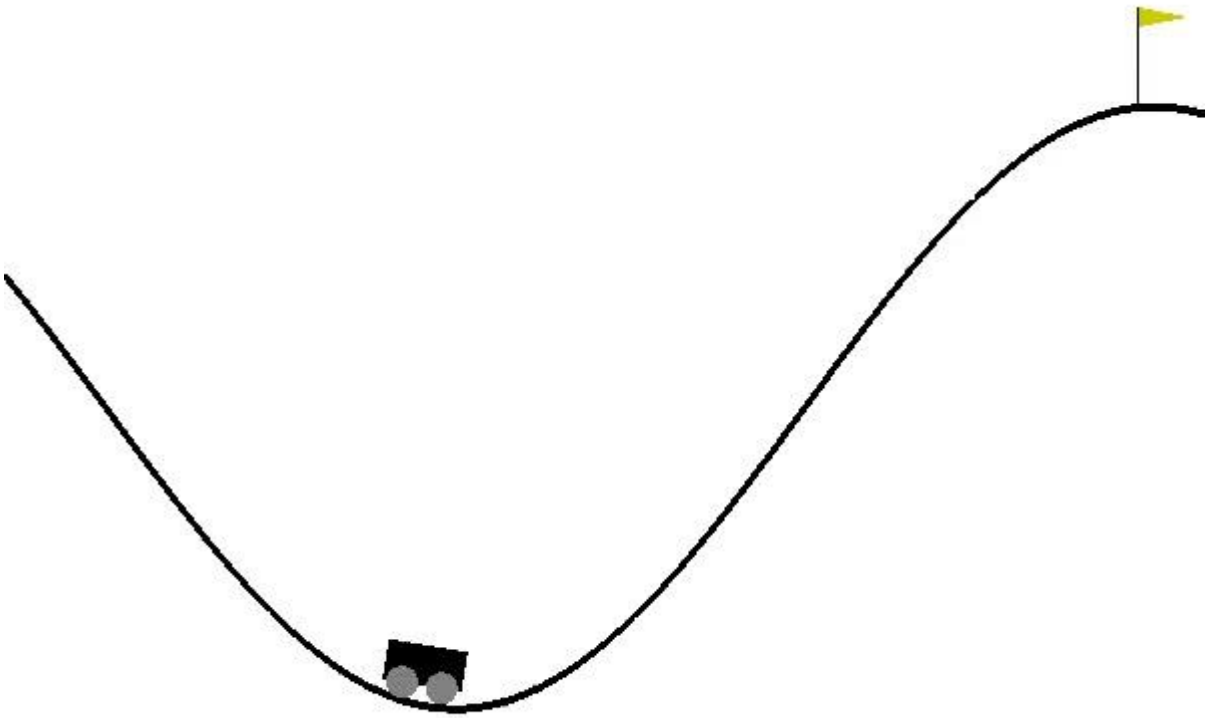
В задачах машинного навчання з підкріпленням досить імовірна ситуація, коли система взагалі не вирішить задачу. Це свідчить про те, що ми або не повністю зрозуміли усі приховані нюанси нашої задачі, або складність системи, яка приймає рішення – недостатня. Припустимо, якщо ми задачу вибору необхідних рішень сформалізуємо просто як задачу класифікації, де ми маємо на вході певні вектори і нам потрібно їх класифікувати по максимально ефективній дії, то даний метод підійде не під кожну задачу.

У зв'язку із розвитком машинного навчання загалом, та навчання з підкріпленням зокрема, на даний момент вже розроблено велику кількість середовищ, в яких можна тренувати агентів. Досить часто у вирішенні задач машинного навчання з підкріпленням ключовою проблемою є так розроблення логіки роботи самого агента, як грамотне конструювання середовища, симуляції, в якій агент прийматиме рішення.

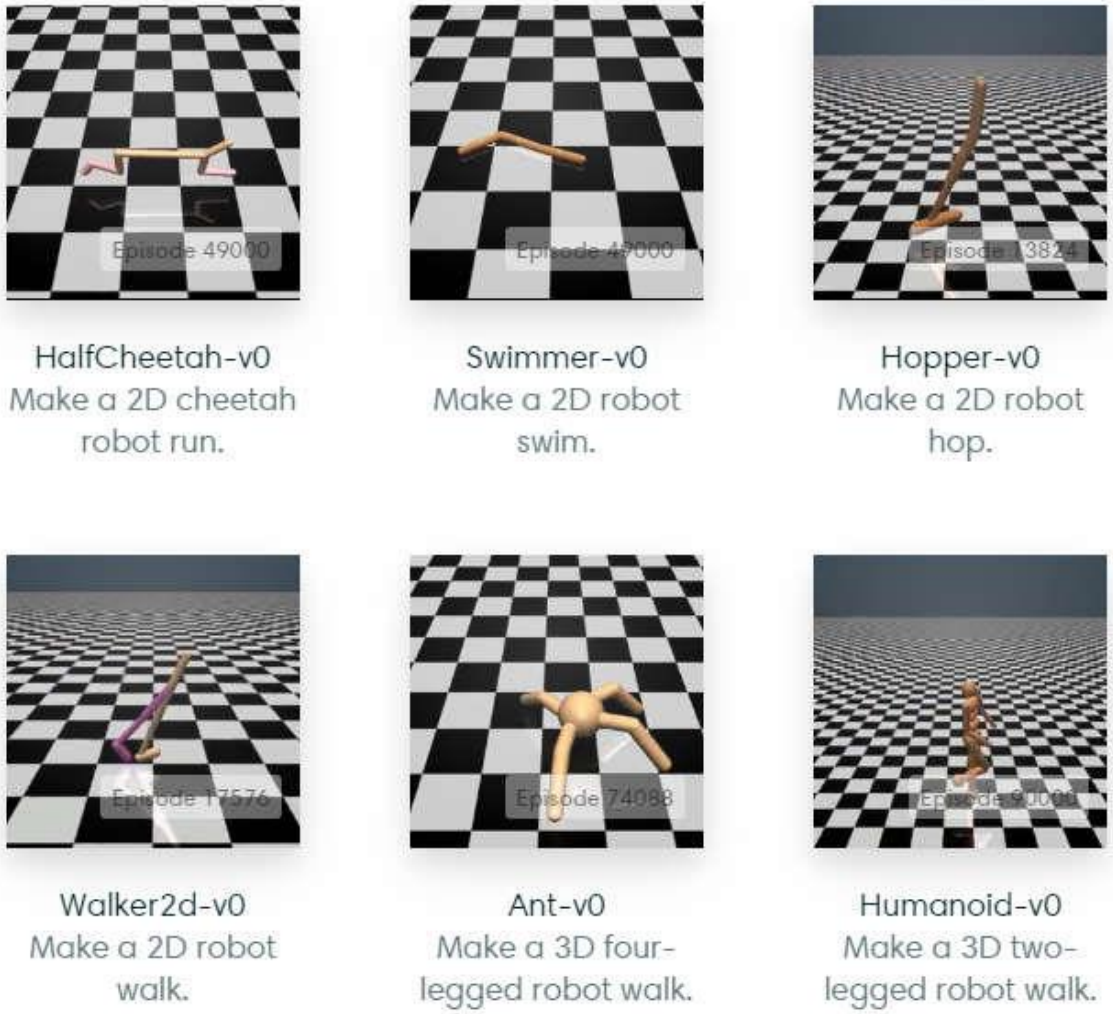
Існують такі ресурси **OpenAI Gym**, на якому вже надаються середовища, в яких можна тренувати різноманітні моделі поведінки агентів – текстові середовища, логічні задачі, двовимірні ігри Atari games, на яких можна потестувати велику кількість алгоритмів,



box 2D середовище, в яких моделюються певні фізичні закономірності реакцій об’єктів із примітивною, але достатньою фізикою,



і тривимірні середовища, в яких ми можемо моделювати агентів, які тренуються керувати певними тривимірними віртуальними роботами.



Якщо наш підхід показав нам в такому середовищі хороший результат, ми можемо використовувати дану методику до більш практичних сфер застосування машинного навчання.

Багато задач машинного навчання з підкріпленням ефективно вирішуються нейронними мережами. Спочатку конструюємо нейронну мережу в залежності від того, яке в нас середовище. Припустимо – це OpenAI Gym з середовищем моделювання руху в двовимірному просторі з перепонами і подоланням перепон (таких середовищ в OpenAI Gym декілька). Нейронна мережа конструюється залежно від того, в якому вигляді надаються дані середовищем і який є допустимий словник дій. А далі, як модуль інтегруємо нашу нейромережу в середовище навчання, де ітерація за ітерацією в паралельному режимі (паралельно моделюється поведінка великої кількості агентів) або в ітеративному послідовному режимі (маємо одного агента, який

раз за разом з різними базами ініціалізації намагається підібрати певну стратегію, щоб, припустимо, виграти в гру).

Машинне навчання з підкріпленням може практично ефективно застосовуватись в задачах вибору стратегії показу реклами, наприклад, залежно від часу доби і наявної інформації про користувачів. Середовищем виступає система-агрегатор і відклик користувачів – конверсія кліків по банеру. Агент намагається сконфігурувати таку стратегію видачі, щоб максимізувати конверсію, бо поверхнева аналітика часто не дає очевидну стратегію. Інше застосування – підбір стратегій торгівлі на біржі.

Ресурси, які дозволяють слідкувати за прогресом в сфері машинного навчання.

В продовж курсу ми ознайомились, які основні методи машинного навчання існують, які типові задачі вирішуються, що таке лінійна та логістична регресія, чим класифікація відрізняється від кластеризації, в чому суть навчання з вчителем та без, що таке навчання з підкріпленням, ознайомилися із можливостями бібліотеки Keras, розглянули, як конструювати нейронні мережі зі згорткою та рекурентні нейронні мережі, навчилися конструювати векторні моделі мови. Це далеко не весь список задач і методів, що доступні в машинному навчанні. Їх список поповнюється щодня. І на ресурсі arxiv.org можна кожного дня побачити декілька десятків нових публікацій, в яких пропонуються нові методи вирішення різноманітних задач.

Сфера машинного навчання розвивається неймовірними темпами. Лише за останній рік ефективність вирішення типових задач зросла в десятки разів.

Keras – не єдина бібліотека для конструювання нейромереж. Також

ефективними є TensorFlow, Caffe, MXNet та ін.

Спостерігати за публікаціями алгоритмів зручно на ресурсі GitHub, де можна вивчати реалізації від людей, які надали їх у відкритий доступ.

При тому, що корисно розуміти, як працюють алгоритми та конструювати їх «з нуля», не завжди потрібно «винаходити велосипед». Більш ефективно спочатку пошукати вирішення Вашої задачі на цих ресурсах.

Також слід звернути увагу на ресурс kaggle.com. На ньому публікуються задачі, які очікують вирішення методами машинного навчання, інколи навіть за значну винагороду. До задач часом додаються дуже якісні тренувальні вибірки. Тут можна позмагатися з іншими людьми в ефективності алгоритмів.

Машинне навчання ще немає великої кількості канонів. Не бійтесь експериментувати та розвивати і вдосконалювати цю корисну сферу.