**ЛЕКЦІЯ № 7**
**НЕЙРОМЕРЕЖІ І ГЛИБОКЕ НАВЧАННЯ**

ПЛАН

1. Глибокі мережі: переваги та труднощі
2. Поняття глибокого (глибинного) навчання
3. Згорткові Нейромережі (CNN)
4. Автоенкодери
5. Рекурентні Нейромережі (RNN)

ЛІТЕРАТУРА

Сайти http://www.mmf.lnu.edu.ua/ar/1739

https://datascience.org.ua/vvedenie-v-reinforcement-learning-ili-obuchenie-s-podkrepleniem/

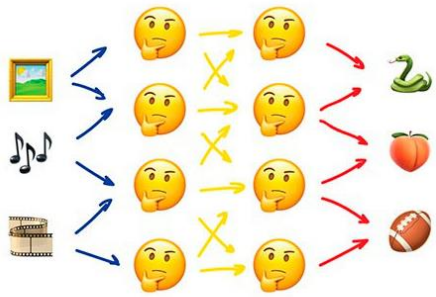
Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. — СПб.: Питер, 2018. — 480 с.: ил. — (Серия «Библиотека программиста»).

## ВСТУП

На попередній лекції ми вивчили що таке машинне навчання, його складові, провели класифікацію методів машинного навчання та розглянули методи класичного навчання.

**Мета машинного навчання** - передбачити результат за вхідними даними. Чим різноманітніші вхідні дані, тим простіше машині знайти закономірності і тим точніший результат.

Отже, якщо ми хочемо навчити машину, нам потрібні три речі: **дані, ознаки, алгоритми.**



Neural Networks «У нас є мережа з тисячі шарів, десятки відеокарт, але ми все ще не придумали де це може бути корисним. Нехай малює котиків! »

Сьогодні використовують для:

Замість всіх перелічених вище алгоритмів

Визначення об'єктів на фото і відео

Розпізнавання і синтез мови

Обробка зображень, перенесення стилю

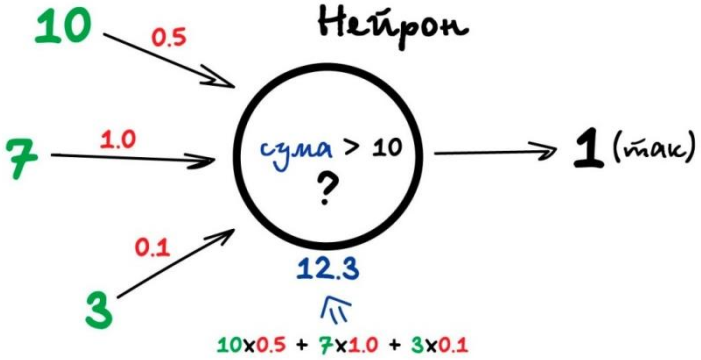
Машинний переклад

Популярні архітектури: Перцептрон, Згорткові Мережі (CNN), Рекурентні Мережі(RNN), Автоенкодери

Якщо вам хоча б раз не намагалися пояснити нейромережі на прикладі

Будь-яка нейромережа - це набір нейронів і зв'язків між ними. Нейрон найкраще уявляти собі просто як функцію з купою входів і одним виходом. Завдання нейрона - взяти цифри зі своїх входів, виконати над ними функцію і віддати результат на вихід. Простий приклад корисного нейрона: знайти суму всіх цифр зі входів, і якщо їх сума більше N - видати на вихід одиницю, інакше - нуль.

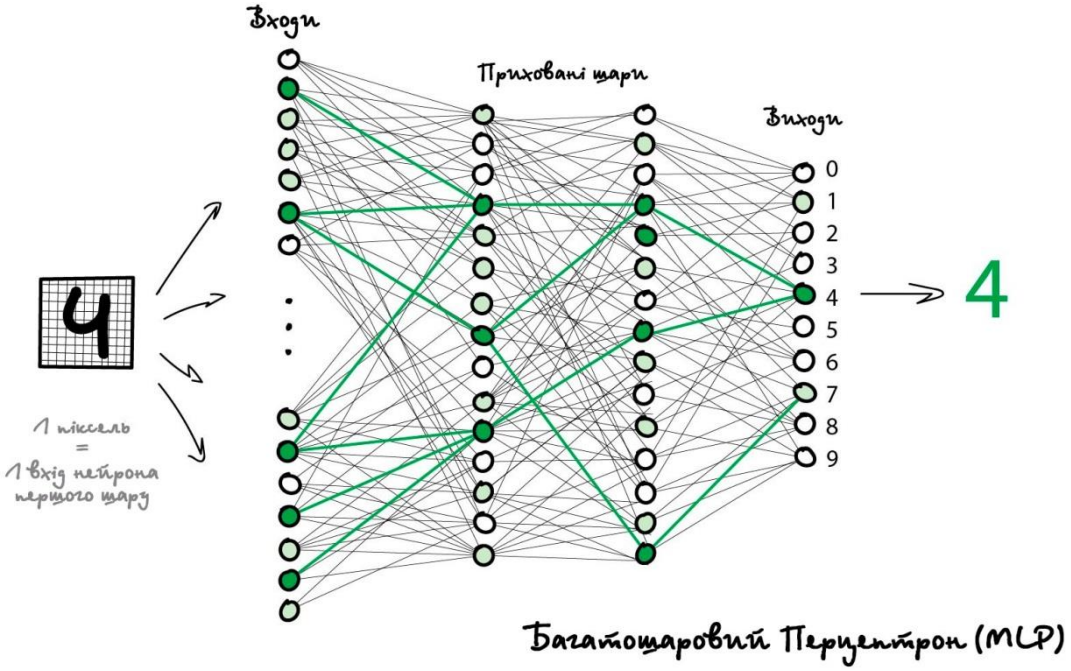
Зв'язки - це канали, через які нейрони шлють один одному цифри. Кожен зв'язок має свою вагу - її єдиний параметр, який можна умовно уявити як міцність зв'язку. Коли через зв'язок з вагою 0,5 проходить число 10, воно перетворюється в 5. Сам нейрон не розбирається, що до нього прийшло і сумує все підряд - ось ваги й потрібні, щоб керувати на які входи нейрон повинен реагувати, а на які - ні.




Щоб мережа не перетворилася в анархію, нейрони вирішили пов'язувати не як захочеться, а по шарах. Всередині одного шару нейрони ніяк не

пов'язані, але з'єднані з нейронами наступного і попереднього шару. Дані в такій мережі йдуть строго в одному напрямку - від входів першого шару до виходів останнього.

Якщо створити достатню кількість шарів і правильно розставити ваги в такій мережі, виходить наступне - подавши на вхід, скажімо, зображення написаної від руки цифри 4, чорні пікселі активують пов'язані з ними нейрони, ті активують наступні шари, і так далі і далі, поки в результаті не висвітлиться той самий вихід, який відповідає за четвірку. Результат досягнуто.




У реальному програмуванні, очевидно, жодних нейронів і зв'язків не пишуть, все зображають матрицями, тому що потрібна швидкість.

Така мережа, де є декілька шарів і між ними пов'язані всі нейрони, називається перцептроном (MLP) і вважається найпростішою архітектурою для новачків. У бойових задачах особисто я ніколи її не зустрічав.

Коли ми побудували мережу, наша задача так правильно розставити ваги, щоб нейрони реагували на потрібні сигнали. Тут потрібно згадати, що у нас же є дані - приклади «входів» і правильних «виходів». Будемо показувати нейромережі малюнок тієї ж цифри 4 і говорити «побудуй свої ваги так, щоб при такому вході на твоєму виході завжди спалахувала четвірка».

Спочатку всі ваги просто розставлені випадково, ми показуємо мережі цифру, вона видає якусь випадкову відповідь (ваг же немає), а ми порівнюємо, наскільки результат відрізняється від потрібного нам. Потім йдемо по мережі в зворотному напрямку, від виходів до входів, і говоримо кожному нейрону - так, ти ось тут чогось активувався, через тебе все пішло не так, давай ти будеш трохи менше реагувати на ось цей зв'язок і трохи більше на отой, ок?

Через тисяч сто таких циклів «прогнали-перевірили-покарали» є надія, що ваги в мережі відкоригуються так, як би ми хотіли. Науково цей підхід називається Backpropagation або «Метод зворотного поширення помилки». Цікаво те, що щоб відкрити цей метод знадобилося двадцять років. До нього нейромережі навчали як могли.

Добре навчена нейромережа могла прикидатися будь-яким алгоритмом з вивчених нами, а часто працювати навіть точніше. Така універсальність зробила нейромережі дуже популярними. Нарешті у нас є архітектура людського мозку, говорили вони, потрібно просто зібрати багато шарів і навчити їх на будь-яких даних, сподівалися вони. Потім почалася перша Зима ШІ, потім відлига, потім друга хвиля розчарування.

Виявилося, що для навчання мережі з великою кількістю шарів були потрібні неможливі на ті часи потужності. Зараз будь-яке ігрове відро з жифорсами перевищує потужність тодішнього датацентру. Тоді навіть надії на це не було, і в нейромережах всі дуже сильно розчарувалися.

У середині 2000-х років, у машинному навчанні розпочалася революція. У 2005-2006 роках групи дослідників під керівництвом Джеффрі Хінтона (Geoffrey Hinton) в університеті Торонто та Йошуа Бенджі (Yoshua Bengio) в університеті Монреаля навчилися навчати глибокі нейронні мережі. І це перевернуло весь світ машинного навчання! Тепер у найрізноманітніших предметних областях найкращі результати виходять за допомогою глибоких нейронних мереж. Одним із перших гучних промислових успіхів стало розпізнавання мови: розроблені групою Хінтона глибокі мережі дуже швидко радикально покращили результати розпізнавання в порівнянні з класичними підходами, що відточувалися десятиліттями, і сьогодні будь-який розпізнавач, включаючи голосові помічники на кшталт Apple Siri і Google Now, працює виключно на глибоких нейронних мережах. А зараз, люди навчилися навчати різні архітектури глибоких нейронних мереж, і ті вирішують абсолютно різні завдання: від розпізнавання облич до водіння автомобілів та гри в Го.

Але ідеї більшості таких моделей з'явилися ще у 80-90-х роках XX століття, а то й раніше. Штучні нейронні мережі стали предметом досліджень дуже давно; вони були однією з перших добре оформлених ідей штучного інтелекту, коли й слів таких — «штучний інтелект» — ще ніхто не чув. Але з початку 90-х років XX століття до середини нульових цього нейронні мережі були, м'яко кажучи, не в моді. Відомий дослідник нейронних мереж Джон Денкер (John Denker) в 1994 році сказав: «Нейронні мережі - це другий найкращий спосіб зробити практично що завгодно». І справді, на той момент вже було відомо, що нейронна мережа теоретично може наблизити будь-яку функцію і навчитися вирішувати будь-яке завдання, а глибока нейронна мережа здатна ще й ефективніше вирішити набагато більше задач... Але навчати глибокі мережі нікому не вдавалося, інші методи на конкретних прикладах працювали краще.

Рішення, запропоноване групою Хінтона в середині 2000-х років, прийшло *у вигляді навчання без вчителя, коли мережа спочатку навчається на великому наборі даних без розмітки, а потім вже донавчається на розмічених даних, використовуючи це наближення*. Наприклад, якщо ми хочемо розпізнавати людські обличчя, то давайте спочатку понавчаємо нейронну мережу на фотографіях з людьми взагалі, без розмітки (таких фотографій можна легко набрати скільки завгодно багато), а вже потім, коли мережа «надивиться» на нерозмічені фотографії, донавчимо її на наявному розміченому наборі даних. Виявилося, що при цьому кінцевий результат стає набагато кращим, а гарні та цікаві ознаки мережа починає виділяти ще на етапі навчання без вчителя. Звичайно, тут поки що зовсім незрозуміло, що мережа, власне, повинна робити з цими нерозміченими фотографіями, і в цьому і *полягав прорив середини нульових років*. Ми трохи поговоримо про ці методи проте без подробиць, тому що сьогодні ці методи практично не використовуються.

Чому? Тому що все добре працює і без них! Виявилося, що зараз ми можемо успішно навчати нейронні мережі, у тому числі глибокі, фактично тими самими методами, якими раніше це зробити ніяк не вдавалося. Методи навчання без вчителя виявилися лише «спусковим гачком» для революції глибокого навчання. Другою найважливішою причиною став, власне, прогрес у обчислювальній техніці та у розмірах доступних для навчання наборів даних. З розвитком Інтернету даних ставало все більше: наприклад, класичний набір даних MNIST, на якому добрий десяток років тестувалися моделі комп'ютерного зору — це 70 тисяч зображень рукописних цифр розміром 28 x 28 пікселів, сумарно близько 10 Мбайт даних; а сучасний стандартний датасет для моделей комп'ютерного зору ImageNet містить близько 1,2 Тбайт зображень.

Крім розвитку техніки, розвивалися і загальні методи навчання нейронних мереж, і класичні архітектури нейронних мереж, рекурентні мережі. З'являлися і абсолютно нові архітектури: породжувальні змагальні мережі зуміли перетворити нейронні мережі на породжувальні моделі, нейронні мережі в навчанні з підкріпленням призвели до небачених раніше проривів, нейробайесівські методи поєднали нейронні мережі та класичний висновок за допомогою варіаційних наближень, а потреби конкретних додатків привели до розробки таких нових архітектур, як мережі з увагою та мережі з пам'яттю, які вже знаходять й інші застосування.


## 1. Глибокі мережі: переваги та труднощі

А зараз перейдемо до питання, так би мовити, телеологічного характеру: навіщо взагалі потрібні глибокі мережі? Класична теорема Хорника, заснована на ранніх роботах Колмогорова, стверджує, що будь-яку безперервну функцію можна як завгодно точно наблизити нейронною мережею з одним прихованим рівнем. Здавалося б, цього має бути достатньо. Навіщо плодити зайву складність на рівному місці?

*Справа в тому, що глибокі архітектури часто дозволяють висловити те саме, наблизити ті ж функції набагато ефективніше, ніж неглибокі*.

З рівнями нейронної мережі виникає той самий ефект: ту саму функцію часто можна набагато краще наблизити глибшою мережею, ніж дрібною, навіть якщо загальна кількість нейронів у мережі залишити постійною.

*А інший бік сили глибоких мереж* — те, що глибока нейронна мережа створює непросто глибоке, а й розподілене уявлення. Тут йдеться про те, що кожен рівень глибокої мережі складається не з одного нейрона, а відразу з багатьох, і комбінації значень цих нейронів справляють справжнісінький експоненційний вибух у просторі входів!

Проілюструємо це прикладом на рис. 1. Уявімо, що ми можемо розділяти точки на площині за допомогою трьох можливих ознак, кожна з яких - це лінійна функція: $f_1, f_2, f_3$. На рис. 3.8 а зображена розділяюча поверхня по одному з них, тобто пряма на площині, і розмічені частини, на які пряма ділить площину. Частин цих, зрозуміло, дві.

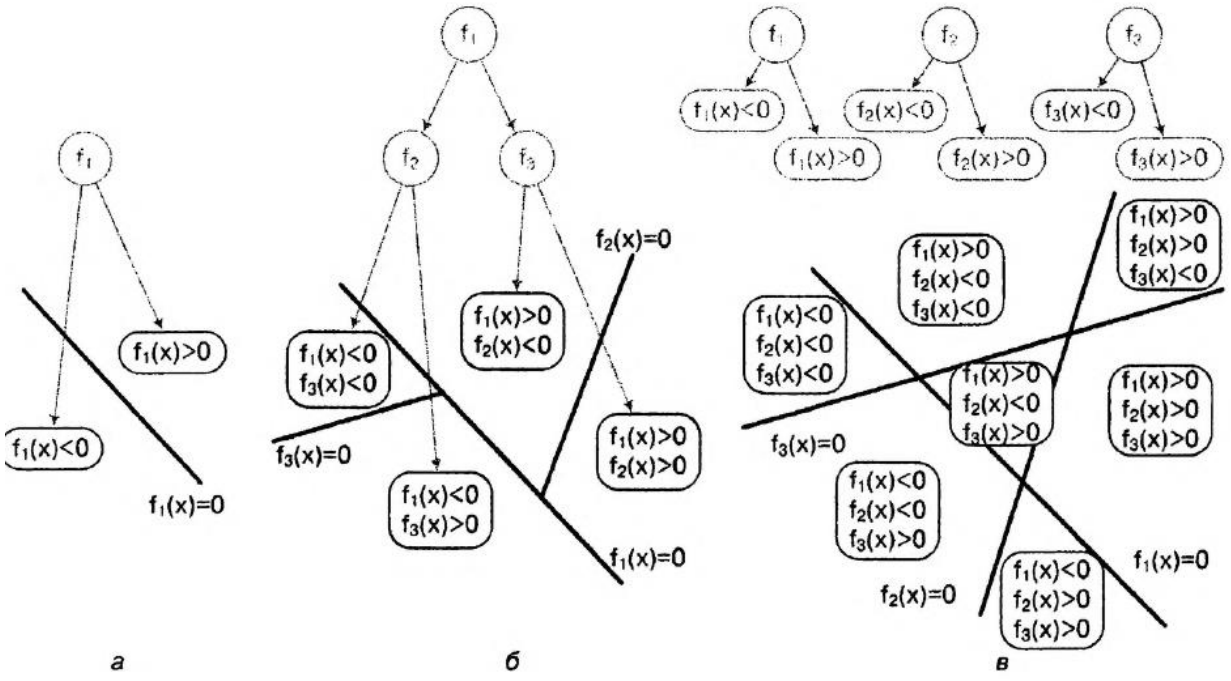



Рис. 1. Сила розподіленого подання: а — поверхня, що розділяє, однієї ознаки; б - розділяюча поверхня дерева глибини 2 на трьох ознаках;
в - розділяюча поверхня трьох ознак відразу

Як слід поєднувати ці ознаки у нашому класифікаторі? По-перше, можна спробувати додати глибини та зробити дерево прийняття рішень; приклад такого дерева і відповідна поверхня, що розділяє показані на рис. 1б. Зверніть увагу, що дерево ділить площину на стільки ж частин, скільки в нього листя: щоб розібрати чотири різні можливі випадки, нам потрібно було побудувати дерево з чотирма листками. А на рис. 1в, зображено розподілене уявлення: давайте уявімо, що наш другий шар може складатися з трьох нейронів відразу. Тоді всілякі комбінації цих трьох нейронів можуть розпізнати одночасно $2^3 = 8$ різних випадків (на малюнку їх реалізується 7), і це число різних варіантів зростає експоненційно при зростанні числа нейронів.

А тепер уявіть (намалювати це було б вже складно — занадто заплутана поверхня вийшла б), що за другим рівнем є ще третій, тобто ми можемо складати різні комбінації з таких поверхонь, що розділяють, по три прямі в кожній...

Ну добре. Припустимо, ми вас переконали, що глибокі нейронні мережі дійсно потрібні. Але тоді постає друге питання: а в чому, власне, проблема? *Чому глибокі мережі, у яких немає нічого теоретично складного і які, як ми вже бачили, з'явилися майже одночасно з алгоритмом зворотного розповсюдження помилки, що так довго викликали такі складності?* Чому революція глибокого навчання відбулася в середині 2000-х, а не 1980-х?

На це питання є дві відповіді. *Перша – математична*. Справа в тому, що глибокі нейронні мережі навчати, звичайно, можна тим же алгоритмом градієнтного спуску, але в базовому варіанті, без додаткових хитрощів працювати це не буде. Уявіть, що ви почали навчати глибоку мережу

алгоритмом зворотного поширення помилки. Останній, найближчий до виходів, рівень навчиться досить швидко і дуже добре. Але що буде далі? Далі виявиться, що більшість нейронів останнього рівня на всіх тестових прикладах вже «визначилися» зі своїм значенням, тобто їх вихід близький або до нуля, або до одиниці. Якщо у них класична сигмоїдальна функція активації, то це означає, що похідна у цієї функції активації близька до нуля з обох боків... але ж на цю похідну ми повинні помножити всі градієнти алгоритму зворотного поширення!

Таким чином, виходить, що останній шар нейронів, що навчився, «блокує» поширення градієнтів далі назад за графом обчислень, і більш ранні рівні глибокої мережі в результаті навчаються дуже повільно, фактично стоять на місці. Цей ефект називається проблемою загасаючих градієнтів (vanishing gradients).

А з рекурентними мережами, які фактично за визначенням є дуже глибокими, виникає ще й *обернена проблема: іноді градієнти можуть почати "вибухати", експоненційно збільшуватися в міру "розгортання" нейронної мережі (exploding gradients).* Обидві проблеми виникають часто, і досить довго дослідники не могли задовільно їх вирішити.

На питання про те, чому глибокі нейронні мережі було складно навчати, є й інша відповідь. Вона дуже проста і може вас розчарувати: Справа в тому, що раніше комп'ютери були повільнішими, а доступних для навчання даних було набагато менше, ніж зараз.

У середині 2000-х років все зійшлося воєдино: технічно комп'ютери стали досить потужними, щоб навчати великі нейронні мережі (більше того, обчислення в нейронних мережах невдовзі навчилися делегувати відеокартам, що прискорило процес навчання ще цілий порядок), набори даних стали досить великими, щоб навчання великих мереж мало сенс, а математиці нейронних мереж відбулося чергове просування. І почалася та сама революція глибокого навчання.

## 2. Поняття глибокого (глибинного) навчання

**Глибинне навчання** (також відоме як глибинне структурне навчання, ієрархічне навчання, глибинне машинне навчання, англ. deep learning, deep structured learning, hierarchical learning, deep machine learning) — це галузь машинного навчання, що ґрунтується на наборі алгоритмів, які намагаються моделювати високорівневі абстракції в даних, застосовуючи глибинний граф із декількома обробними шарами, що побудовано з кількох лінійних або нелінійних перетворень.

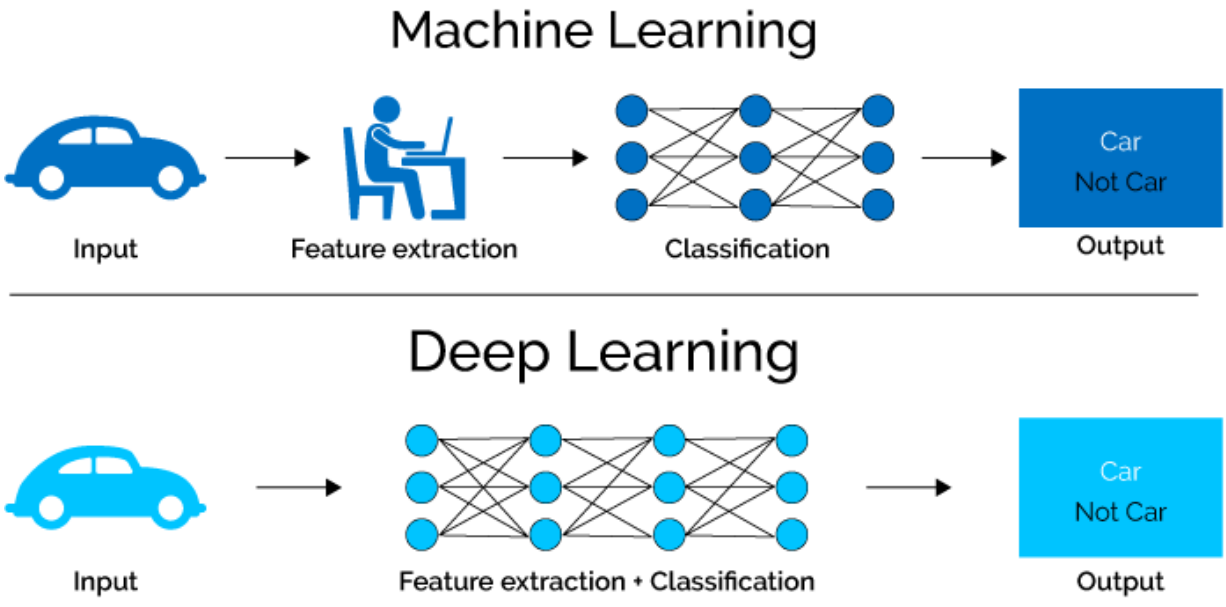
Дослідження в цій області намагаються зробити кращі представлення та створити моделі для навчання цих представлень з великомасштабних нерозмічених даних. Деякі з цих представлень було зроблено під натхненням досягнень в нейронауці та з мотивів схем обробки та передавання інформації в нервовій системі, таких як нервове кодування, що намагається визначити

зв'язок між різноманітними стимулами та пов'язаними нейронними реакціями в мозку.

Різні архітектури глибинного навчання, такі як глибинні нейронні мережі, згорткові глибинні нейронні мережі, глибинні мережі переконань та рекурентні нейронні мережі застосовувалися в таких областях, як комп'ютерне бачення, автоматичне розпізнавання мовлення, обробка природної мови, розпізнавання звуків та біоінформатика, де вони, як було показано, представляють передові результати в різноманітних задачах.

Концепція глибокого навчання (Deep Learning - DL) вперше з'явилася в 2006 році як нова галузь досліджень у машинному навчанні. Спочатку воно було відоме як ієрархічне навчання [2], і як правило воно включало в себе безліч областей досліджень, пов'язаних з розпізнаванням образів. Глибоке навчання в основному бере до уваги два ключові фактори: нелінійна обробка в декількох шарах або стадіях і навчання під наглядом або без нього [4]. Нелінійна обробка в кількох шарах відноситься до алгоритму, в якому поточний шар приймає як вхідні дані вихідні дані попереднього шару. Ієрархія встановлюється між шарами, щоб упорядкувати важливість даних, корисність яких слід встановити. З іншого боку, контрольоване та неконтрольоване навчання пов'язане з міткою класів цілей: її присутність має на увазі контрольовану систему, а відсутність – неконтрольовану.

**Відмінність глибокого навчання від класичних нейромереж полягала в нових методах навчання, які справлялися з великими розмірами мереж**. Однак сьогодні лише теоретики розділяють, яке навчання можна вважати глибоким, а яке не надто. Ми ж, як практики, використовуємо популярні «глибокі» бібліотеки типу Keras , TensorFlow і PyTorch навіть коли нам треба зібрати міні-сітку на п'ять шарів. Просто тому що вони зручніші того, що було раніше. Ми називаємо це просто нейромережами.



**Найбільш популярні глибокі мейромережі: згорткова, енкоери, рекурентні.**

## 3. Згорткові Нейромережі (CNN)

Давайте спочатку дамо відповідь на питання: Але що ж, власне, таке в даному випадку *згортка* і як вона стосується нейронних мереж? Щоб відповісти на це питання, спочатку відступимо на крок назад. *Наріжним каменем всіх нейронних мереж є* ***афінні*** *перетворення.* (Афінне перетворення (лат. affinis, «пов'язаний з») — відображення площини або простору в собі, при якому паралельні прямі переходять у паралельні прямі, пересічні — в пересічні, мимобіжні — в мимобіжні) У кожному шарі повнозв'язкової мережі повторюється та сама операція: на вхід подається вектор, який множиться на матрицю ваг, а результату додається вектор вільних членів; Тільки після цього до результату застосовується певна нелінійна функція активації. І у всіх мережах, які ми досі будували, такий підхід використовувався постійно, незалежно від структури чи походження даних. Чи то зображення, текст чи музика, ми знову і знову застосовуємо афінне перетворення в кожному шарі нашої мережі, попередньо привівши дані до векторної форми. Однак багато типів даних мають свою власну внутрішню структуру, яка добре відома нам заздалегідь. Головний приклад такої структури - зображення, яке зазвичай представляють як масив векторів чисел: якщо зображення чорно-біле, то це просто масив інтенсивностей, а якщо кольорове, то масив векторів із трьох чисел, що позначають інтенсивність трьох основних кольорів (червоного, зеленого і синього у стандартному RGB). Якщо ж узагальнити таку внутрішню структуру то опис вийде такий:

1) вхідні дані є багатовимірним масивом («тензор»);

2) серед розмірностей цього масиву є одна або більше осей, порядок вздовж яких відіграє важливу роль; наприклад, це може бути розташування пікселів у зображенні, часова шкала для музичного твору, порядок слів чи символів у тексті;

3) інші осі позначають «*канали*», що описують властивості кожного елемента за попередньою підмножиною осей; наприклад, три компоненти для зображень, два компоненти (правий і лівий) для стереозвуку і т.д.

Основна ідея згорткової мережі полягає в тому, що обробка ділянки зображення дуже часто повинна відбуватися незалежно від конкретного розташування цієї ділянки. Грубо кажучи, якщо ви хочете дізнатися на фотографії свого друга Васю, зовсім не важливо, на 100 або на 200 пікселів вухо Васі від лівого краю фотографії. Дізнатися про Васю можна було б і на сильно обрізаній фотографії, де немає нічого, крім його обличчя; це локальне завдання, яке можна вирішувати локальними засобами. Звичайно, взаємне розташування об'єктів відіграє важливу роль, але спочатку їх потрібно в будь-якому випадку розпізнати, і це розпізнавання локально і незалежно від конкретного положення ділянки з об'єктом усередині великої картинки. Тому

мережа сітки просто робить це припущення в явному вигляді: давайте покриємо вхід невеликими вікнами (скажімо, 5x5 пікселів) і виділятимемо ознаки в кожному такому вікні невеликою нейронною мережею. Причому — і тут ключове міркування — ***ознаки виділятимемо у кожному вікні одні й самі***, тобто Невелика нейронна мережа буде лише одна, входів в неї буде всього 5 x 5 = 25, а з кожної картинки неї може вийти дуже багато різних входів.

Потім результати цієї нейронної мережі знову можна буде уявити у вигляді «картинки», замінюючи вікна 5 x 5 на їх центральні пікселі, і на ній можна буде застосувати другий шар згортання, з вже іншою маленькою нейронною мережею, і т. д. Незабаром ми побачимо, що в кожному згортуваному шарі буде зовсім небагато вільних параметрів, особливо в порівнянні з повнозв'язковими аналогами.

Перш ніж переходити безпосередньо до формальних визначень операції згортки, розберемося з поняттям *каналу* в зображенні. Зазвичай кольорові картинки, що подаються на вхід нейронної мережі, представлені у вигляді декількох прямокутних матриць, кожна з яких задає рівень одного з каналів кольорів в кожному пікселі зображення. Картинка розміром 200 x 200 пікселів - це насправді 120 000 чисел, три матриці інтенсивностей розміром 200 x 200 кожна. Якщо зображення чорно-біле, як, наприклад, MNIST, то така матриця буде одна. А якщо це не проста картинка, а, скажімо, результат зображає мас-спектрометрії, коли в кожному пікселі знаходиться цілий спектр, то матриць може бути дуже багато. *Але в будь-якому випадку ми припускатимемо, що в кожному пікселі вхідного зображення стоїть певний тензор (зазвичай одномірний, тобто вектор чисел), і його компоненти називаються каналами (channels).*

Такі ж матриці будуть виходити і після згорткового шару: у них, як і раніше, буде просторова структура, що відповідає вихідній картинці (але не в точності така сама — незабаром про це поговоримо), проте каналів тепер може побільшати. Значення кожної ознаки, які ми виділили з вікон у вихідному зображенні, тепер будуть цілою матрицею. Кожна така матриця називається *картою ознак (feature map)*. У принципі, канали вихідного зображення також можна називати картами ознак; аналогічно ми часто називатимемо карти ознак чергового шару каналами.

Тепер залишилося тільки формально визначити, що ж таке згортка і як влаштовані шари мережі згортки. Згортка — це лише лінійне перетворення вхідних даних особливого виду. Якщо $x^l$ — карта ознак у шарі під номером ι, то результат двовимірної згортки з ядром розміру $2d+1$ і матрицею ваг W розміру $(2d+1) \times (2d+1)$ на наступному шарі буде таким:

$$y^l_{i,j} = \sum_{-d<=a,b<=d} W_{a,b} x^l_{i+a,j+b},$$

де $y_{i,j}^{l}$ - результат згортки на рівні $l$, а $x_{i,j}^{l}$ - її вхід, тобто вихід всього попереднього шару. Інакше кажучи, щоб отримати компонент $(i,j)$ наступного рівня, ми застосовуємо лінійне перетворення до квадратного вікна попереднього рівня, тобто скалярно множимо пікселі із вікна на вектор згортки. Це показано на рис. 2: ми застосовуємо пакунок із матрицею ваг W розміру 3 x 3 до матриці X розміру 5x5. Зверніть увагу, що множення підматриці вихідної матриці X f відповідає вікну, і матриці ваг W - це не множення матриць, а просто скалярний добуток відповідних векторів. А всього вікно вміщується в матриці X дев'ять разів, і виходить

$$\begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 0 \\ 0 & 4 & 3 & 2 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 5 & 4 \\ 8 & 8 & 10 \\ 8 & 15 & 12 \end{pmatrix}$$

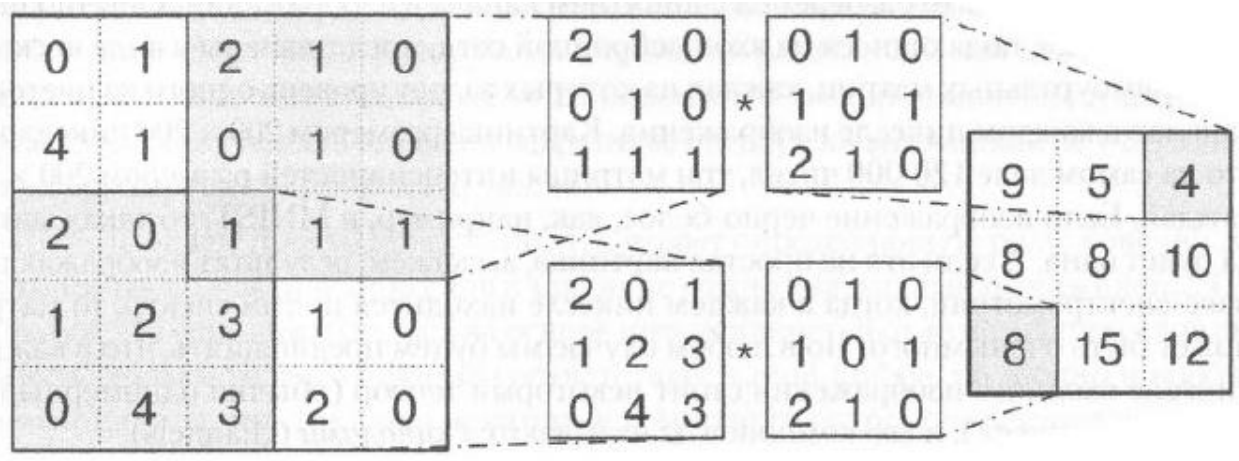



Рис. 2. Приклад підрахунку результату згортки: два приклади підматриці та загальний результат

Тут ми позначили згортку карти ознак X за допомогою матриці ваг W через X * W, як прийнято позначати згортку в математиці.

Якщо розмір згортки буде виражатися парним числом, то одному з випадків у межах підсумовування нерівність стане суворою; тут більше немає «природного» вибору для центру вікна, і його вибір із чотирьох варіантів може залежати від реалізації у конкретній бібліотеці.

Це перетворення має саме ті властивості, про які ми говорили вище:

• згортка зберігає структуру входу (порядок в одновимірному випадку, взаємне розташування пікселів у двовимірному тощо), оскільки застосовується до кожної ділянки вхідних даних окремо;

• операція згортки має властивість розрідженості, оскільки значення кожного нейрона чергового шару залежить тільки від невеликої частки

вхідних нейронів (а, наприклад, у повнозв'язковій нейронній мережі кожен нейрон залежав б від усіх нейронів попереднього шару);

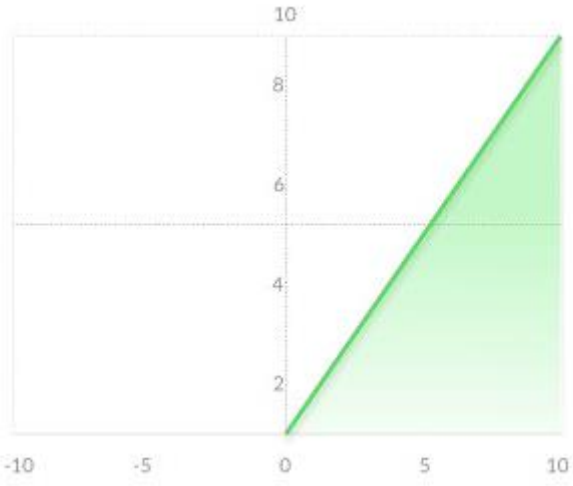
• згортка багаторазово перевикористовує ті самі ваги, оскільки вони повторно застосовуються до різних ділянок входу.

Майже завжди після згортки в нейронній мережі слідує нелінійність, яку ми можемо записати так:

$$z_{i,j}^{l} = h(y_{i,j}^{l}).$$

Як функцію *h* часто використовують ReLU, особливо в дуже глибоких мережах, але й класичні σ і *tanh* теж зустрічаються. Докладно зупинятися на типах нелінійностей, що використовуються в мережах згортання, ми зараз не будемо; наше першочергове завдання - сформувати інтуїції з приводу згорткових мереж.

Функція ReLU




Rectified Linear Unit — це функція активації, що найчастіше використовується при глибокому навчанні. Ця функція повертає 0, якщо приймає негативний аргумент, у разі позитивного аргументу, функція повертає саме число. Тобто, вона може бути записана як f(z)=max(0,z). На перший погляд може здатися, що вона лінійна і має ті ж проблеми, що і лінійна функція, але це не так і її можна використовувати в нейронних мережах з безліччю шарів. Функція ReLU має кілька переваг перед сигмоїдою та гіперболічним тангенсом:

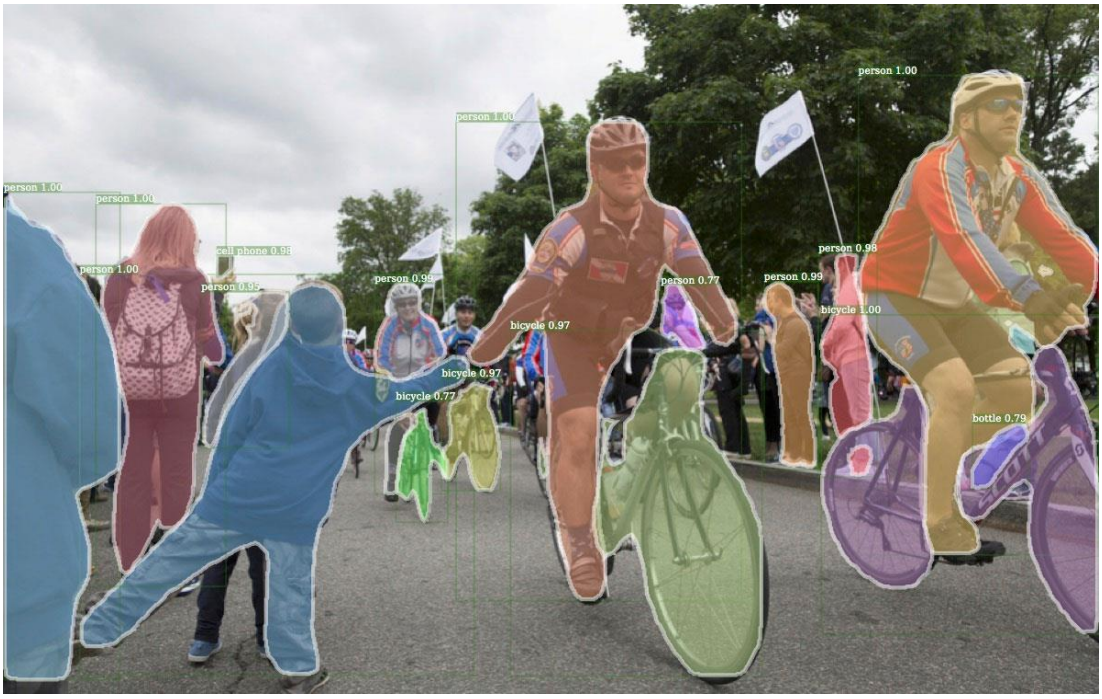
- Дуже швидко і просто розраховується похідна. Для негативних значень – 0, для позитивних – 1.
- Розрідженість активації. У мережах з дуже великою кількістю нейронів використання сигмоїдної функції або гіперболічного тангенсу як активаційної функції тягне за собою активацію майже всіх нейронів, що може позначитися на продуктивності навчання моделі. Якщо ж використовувати ReLU, то кількість нейронів, що

включаються, стане меншою, в силу характеристик функції, і сама мережа стане легше.

Ця функція має один недолік, що називається проблемою вмираючого ReLU[2]. Так як частина похідної функції дорівнює нулю, то градієнт для неї буде нульовим, а це означає, що ваги не будуть змінюватися під час спуску і нейронна мережа перестане навчатися.

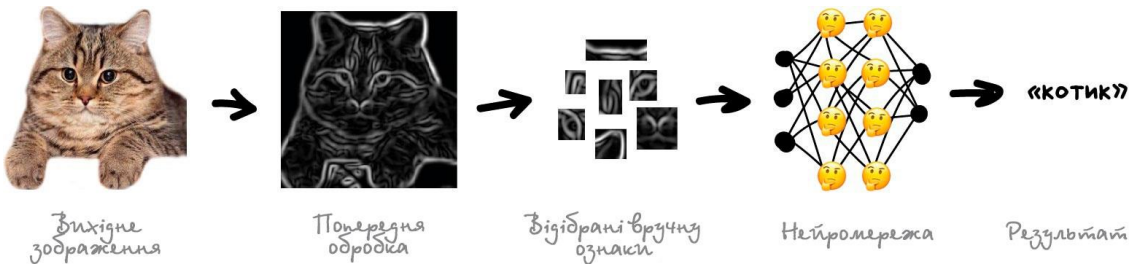
Функцію активації ReLU слід використовувати, якщо немає особливих вимог для вихідного значення нейрона, на зразок необмеженої області визначення. Але якщо після навчання моделі результати вийшли не оптимальні, варто перейти до інших функцій, які можуть дати кращий результат.

Згорткові мережі зараз на піку популярності. Вони використовуються для пошуку об'єктів на фото і відео, розпізнавання осіб, перенесення стилю, генерації і домальовування зображень, створення ефектів типу слоу-мо і поліпшення якості фотографій. Сьогодні CNN застосовують всюди, де є картинки або відео. Навіть у вашому айфоні кілька таких мереж дивляться на ваші фотографії, щоб розпізнати об'єкти на них.




Картинка вище - результат роботи бібліотеки Detectron , яку Facebook недавно заопенсорсив

Проблеми з зображеннями завжди були в тому, що незрозуміло, як виділяти на них ознаки. Текст можна розбити за пропозиціями, взяти властивості слів зі словників. Картинки ж доводилося мітити руками, пояснюючи машині, де у котика на фотографії вушка, а де хвіст. Такий підхід навіть назвали «*handcrafting ознак*» й раніше все так і робили.




*Проблем у ручного крафтингу багато.*

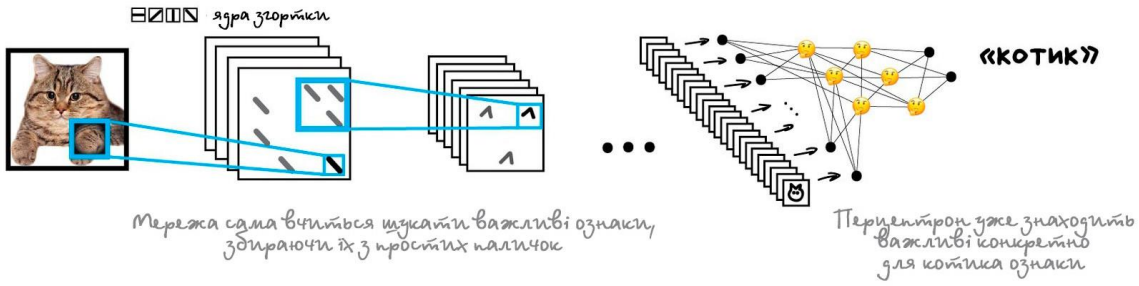
По-перше, якщо котик на фотографії притиснув вушка або відвернувся - все, нейромережа нічого не побачить.

По-друге, спробуйте самі зараз назвати хоча б десять характерних ознак, що відрізняють котиків від інших тварин. Я от не зміг. Однак коли вночі повз мене пробігає чорна пляма, навіть краєм ока я можу сказати котик це чи щур. Тому що людина не дивиться тільки на форму вух і кількість лап - вона оцінює об'єкт за купою різних ознак, про які навіть сама не замислюється. А отже, не розуміє і не може пояснити машині.

Як наслідок, складається враження, що машині треба самій вчитися шукати ці ознаки, складаючи з якихось базових ліній. Будемо робити так: для початку розділимо зображення на блоки 8x8 пікселів і виберемо яка лінія домінує в кожному - горизонтальна [-], вертикальна [|] або одна з діагональних [/]. Можуть і дві, і три, так теж буває, ми не завжди точно впевнені.

На виході ми отримаємо кілька масивів паличок, які, по-суті, є простими ознаками наявності обрисів об'єктів на зображенні. По-суті, це теж картинки, просто з паличок. Значить ми можемо знову вибрати блок 8x8 і подивитися вже, як ці палички поєднуються одна з одною. А потім ще і ще, і ще...

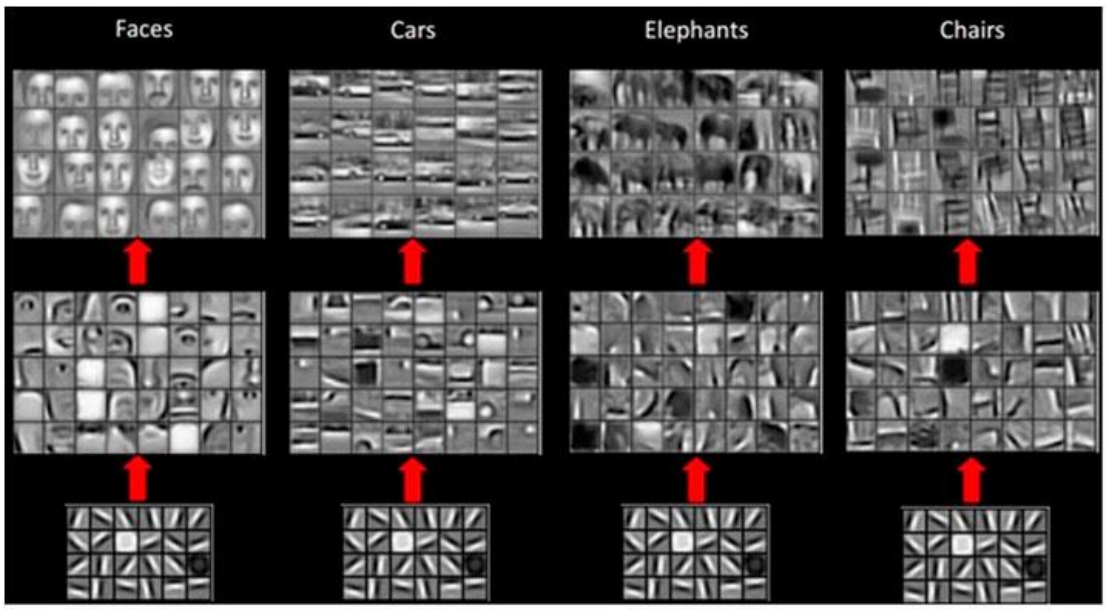
Така операція називається згорткою, звідки і пішла назва методу. Згортку можна уявити як шар нейромережі, адже нейрон - абсолютно будь-яка функція.




**Згорткова Нейромережа (CNN)**

Коли ми проганяємо через нашу нейромережу купу фотографій котів, вона автоматично розставляє великі ваги тим сполученням з паличок, які побачила найчастіше. Причому неважливо, це пряма лінія спини або складний геометричний об'єкт типу мордочки - щось обов'язково буде яскраво активуватися.

На виході ж ми поставимо простий перцептрон, який буде дивитися які поєднання активувалися і говорити кому вони більше характерні - кішці або собаці.

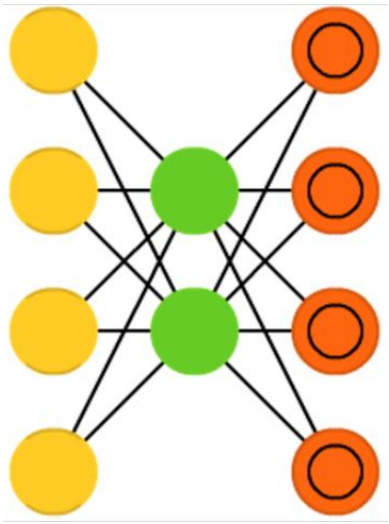



Краса ідеї в тому, що у нас вийшла нейромережа, яка сама знаходить характерні ознаки об'єктів. Нам більше не треба відбирати їх руками. Ми можемо скільки завгодно годувати її зображеннями будь-яких об'єктів, просто наґуґліть мільйон картинок з ними - мережа сама складе карти ознак з паличок і навчиться визначати що завгодно.

Щодо цього існує навіть несмішний жарт:

Дай нейромережі рибу - вона зможе визначати рибу до кінця життя. Дай нейромережі вудку - вона зможе визначати і вудку до кінця життя ...

## АВТОЕНКОДЕРИ



**Автоенкодери ( автокодувальники, англ. Autoencoders, AE) —** щось подібне FFNN, це швидше інший спосіб використання FFNN, чим принципово нова архітектура. Основна ідея автоенкодеров — автоматичне кодування (як при стискуванні, а не при шифруванні) інформації, звідси і назва. Нагадує за формою пісочний годинник, так як скритий шар менше, чим вхідний та вихідний; до того ж вона симметрична відносно середніх шарів (одного або двох, в залежності від парності/непарності загальної кількості шарів). Самий маленький шар майже завжди середній, в ньому максимально сжато все, що розміщено до середини — кодуюча частина, вище середини — декодуюча, а в середині (ви не повірите) — код. AE навчають методом зворотного поширення помилок, подають вхідні дані і задають помилку рівною різницею між входом і виходом. AE можна побудувати симметричними і з точки зору ваг, викладаючи кодуючі вагові коефіцієнти рівними декодуючими.

Основна ідея автокодувальників настільки ж проста, як і геніальна: давайте перетворимо завдання навчання без вчителя на завдання навчання з учителем, самі собі придумаємо тестові приклади з відомими правильними відповідями. І зробимо ми це так: попросимо модель навчитися видавати на виході той самий приклад, який подавали їй на вхід! При цьому вона навчатиметься спочатку створювати якесь внутрішнє уявлення, кодувати вхід якимись ознаками, а потім декодувати їх назад, щоб відновити вихідний вектор входів. Ми зобразили загальну схему автокодувальника на рис. 3. Говорячи трохи формальніше, ми хочемо навчити функцію $f(\boldsymbol{x};\boldsymbol{\theta}) \approx \boldsymbol{x}$, где $\boldsymbol{\theta}$ - це параметри нейронної мережі.


Рис.3. Як працює автокодувальник

Що не відбувалося звичайного копіювання необхідно накладання додаткових обмежень на подання. Мистецтво побудови автокодувальників, полягає в тому, щоб придумати, які обмеження та яким чином накласти на

нейронну мережу, щоб ознаки, що виходять на прихованих шарах, дійсно виявилися «цікавими».

У класичному автокодувальнику додаткове обмеження дуже просте: щоб на прихованому шарі не можна було просто скопіювати вхід, зменшимо його розмірність порівняно з розмірністю входу.

За такого підходу виходить, що автокодировщик фактично вирішує завдання зниження розмірності (dimensionality reduction): як відобразити великий простір зі складними взаємозв'язками в простір нижчої розмірності, де, сподіватимемося, складні взаємозв'язки перейдуть у простішу залежність.

Є безліч класичних методів зниження розмірності: аналіз основних компонент (principal component analysis, PC A), сингулярне розкладання матриць (singular value decomposition, SVD), аналіз незалежних компонент (independent component analysis, ICA) та багато інших.

*Важлива відмінність того, що ми робимо зараз, від класичних методів зниження розмірності в тому, що нейронні мережі можуть висловити більше різних складних різноманітностей, їм не потрібні настільки сильні припущення про структуру даних, як класичним моделям.* Гнучкість моделей, що задаються нейронними мережами, часто дозволяє виділити дуже «цікаві» ознаки, і хоча моделі стають більш складними та крихкими, очевидно, що саме це і потрібно у реальному житті: спробуйте уявити собі, як виглядає різноманіття осмислених текстів українською мовою у просторі послідовностей букв.

На практиці автокодувальники найчастіше використовують для отримання таких ознак з даних, які в результаті дозволяють зменшити помилку при подальшому навчанні з учителем. І виявляється, що в цьому випадку автокодувальники, в прихованому шарі яких нейронів більше, ніж у вхідному (їх називають *overcomplete autoencoders*, а ті що реально знижують розмірність - *undercomplete*), найчастіше виявляються вкрай корисними.

Звичайно, автокодувальники за двадцять років свого існування отримали одразу кілька серйозних апгрейдів. «Звичайні» автокодувальники, які ми розглядали вище, намагаються відновити вхід по ньому самому, тобто намагаються навчити тотожну функцію $f(\boldsymbol{x}) = \boldsymbol{x}$ або за допомогою засобів, недостатніх, щоб це висловити, або з додатковими обмеженнями та цілями, які заважають просто взяти та скопіювати вхід у вихід. Тим не менш, у міру того як розмірність прихованих шарів і виразність моделі будуть зростати - адже ми хочемо, щоб вони росли, - ми будемо все точніше відновлювати вхід по ньому ж самому, і справлятися з потенційним оверфіттингом буде все складніше і складніше.

***Шумопригнічуючий автокодувальник (denoising autoencoder)*** - це оригінальний і дуже простий спосіб майже повністю позбавитися проблем оверфіттингу. Давайте відновлюватимемо не вхід $\boldsymbol{x}$ по ньому самому, а вхід $\tilde{x}$ за деяким його зашумленим варіантом ж. Наприклад, давайте виберемо 10% пікселів зображення та замінимо їх нулями (чорними пікселами) або

випадковими значеннями інтенсивностей, але відновити при цьому попросимо не спотворений варіант картинки, а вихідний, у якому всі пікселі стоять на своїх місцях. Таким чином, автокодувальник повинен буде не просто стиснути отриманий приклад, але ще й частково відновити втрачені в процесі зашумлення дані, навчити не тотожну функцію $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{x},$ як ми робили в цьому розділі раніше, а досить складну функцію $f(\tilde{\boldsymbol{x}};\boldsymbol{\theta}) = \boldsymbol{x},$ яка вже неминуче описуватиме багато цікавих властивостей поданих на вхід даних.

Інший важливий варіант – ***розріджені автокодувальники***.

Виявляється, що розрідженість (sparsity) активацій - це бажана властивість не тільки для мозку, для якого це було життєво важливим, але і для штучних моделей. Іноді ми можемо інтерпретувати значення активації нейрона саме як його активність, і тоді значення, близькі до одиниці, означають, що нейрон посилає імпульси, і навпаки, за близьких до нуля значень нейрон можна вважати «погашеним», неактивним. У деяких завданнях ми хотіли б отримати нейронну мережу, більшість нейронів якої неактивні у кожний момент часу. З одного боку, це дозволяє сподіватися, що за різні «корисні» властивості, витягнуті з даних, відповідають окремі, поодинокі нейрони; на відміну від ситуації, коли одразу велика група нейронів відповідає за деяку властивість, нам дуже важко інтерпретувати їх значення, і модель виходить занадто крихкою.

Але як зробити нейронну мережу розрідженою на практиці? Для збільшення розрідженості прихованого шару нам хотілося б, щоб ймовірність активації кожного окремо взятого нейрона була низькою. Нехай нейрон є випадковою величиною з розподілом Бернуллі; простіше кажучи, нейрон - це звичайна монета, і середнє значення активації нейрона прихованого шару відповідає ймовірності отримання одиниці (наприклад, решки) у випробуванні Бернуллі. Нехай тепер бажана ймовірність активації нейрона дорівнює $\rho$, а отримане з даних емпіричне середнє дорівнює $\hat{\rho}$. Відстань Кульбака — Лейблера між модельним розподілом та розподілом даних дорівнює

$$\mathrm{KL}(\rho \| \hat{\rho}) = \rho \log \frac{\rho}{\hat{\rho}}.$$

І тепер цю відстань, тобто міру несхожості між розподілами, можна просто додати цільову функцію як регуляризатор.

Отже, в 2 та 3 питанні ми познайомилися і з основними концепціями мереж згортання, і з автокодувальниками. Згорткові мережі - це одна з найважливіших архітектур нейронних мереж, яка використовує згорткові фільтри із загальними вагами, застосовуючи тим самим одні й ті самі перетворення до різних частин входу. Така екстремальна регуляризація дозволяє згортковим мережам навчатися дуже ефективно і бути глибокими навіть «з коробки», а такі сучасніші ідеї, як залишкові зв'язки, дозволяють

навчати нейронні мережі практично необмеженої глибини. Тепер нам потрібно поговорити про інший стовп сучасних нейронних мереж — мереж рекурентних.

## 5. Рекурентні Нейромережі (RNN)

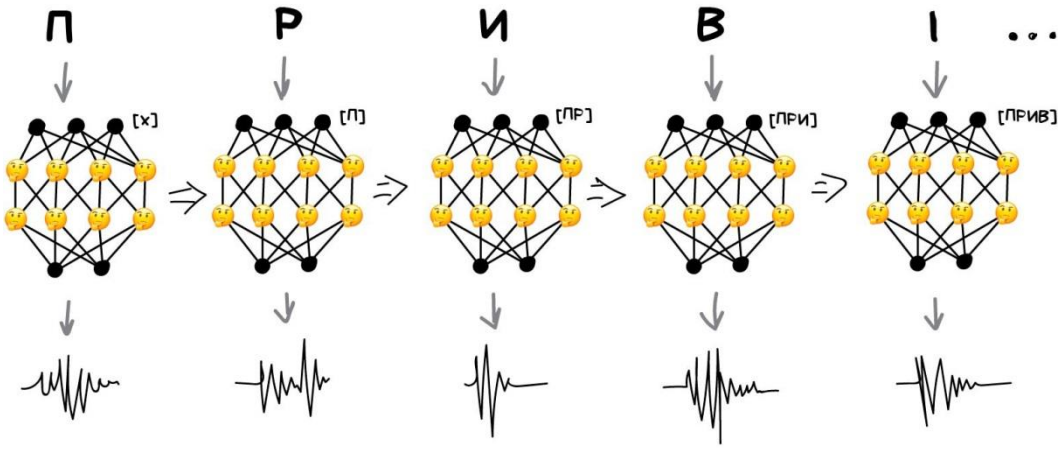
Друга за популярністю архітектура на сьогоднішній день. Завдяки рекурентним мережам у нас є такі корисні речі, як машинний переклад текстів і комп'ютерний синтез мови. На них розв'язують усі завдання, пов'язані з послідовностями - голосові, текстові або музичні.

Пам'ятаєте олдскульні голосові синтезатори типу Microsoft Sam з Windows XP, який смішно вимовляв слова по буквах, намагаючись якось склеїти їх між собою? А тепер подивіться на Amazon Alexa або Алісу від Яндекса - вони сьогодні не просто вимовляють слова без помилок, вони навіть розставляють акценти в реченні!

Це відбувається тому, що сучасні голосові помічники навчають говорити не буквами, а фразами. Але одразу змусити нейромережу повністю видавати фрази не вийде, адже тоді їй треба буде запам'ятати всі фрази в мові і їх розмір буде велетенським. Тут на допомогу приходить той факт, що текст, мова або музика - це послідовності. Кожне слово або звук - це, так би мовити, самостійна одиниця, але яка залежить від попередніх. Коли цей зв'язок втрачається - виходить дабстеп.

Можна достатньо легко навчити мережу вимовляти окремі слова або букви. Беремо купу розмічених на слова аудіофайлів і навчаємо по вхідному слову видавати нам послідовність сигналів, схожих на його вимову. Порівнюємо з оригіналом від диктора і намагаємося максимально наблизитися до ідеалу. Для такого підійде навіть перцептрон.

Ось тільки з послідовністю знову біда, адже перцептрон не запам'ятовує те, що він генерував раніше. Для нього кожен запуск як перший раз. З'явилася ідея додати до кожного нейрона пам'ять. Так були придумані рекурентні мережі, в яких кожен нейрон запам'ятовував всі свої попередні відповіді і при наступному запуску використовував їх як додатковий вхід. Тобто нейрон міг сказати самому собі в майбутньому - агов, чувак, наступний звук повинен звучати вище, у нас тут голосна була (дуже спрощений приклад).




Рекурентна Нейромережа (RNN)

Була лише одна проблема - коли кожен нейрон запам'ятовував всі минулі результати, в мережі виникала така дика кількість входів, що навчити таку кількість зв'язків ставало нереально.

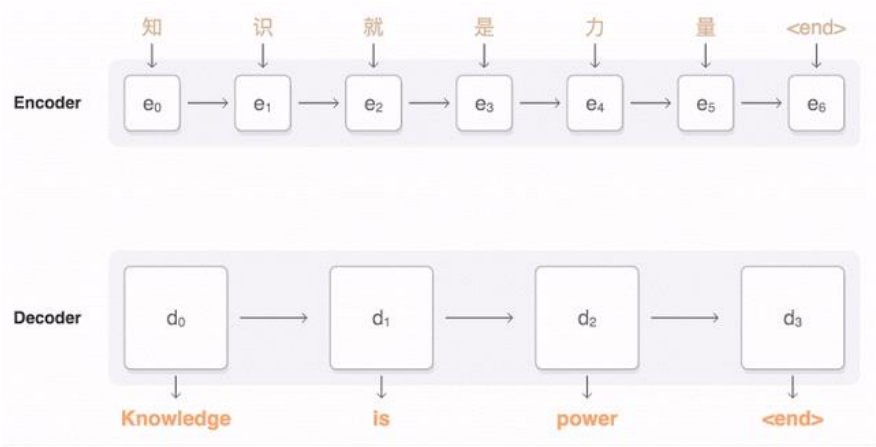
***Коли нейромережа не вміє забувати - її не можна навчити (у людей та ж фігня).***

Спочатку проблему вирішили в лоб - обрубали кожному нейрону пам'ять. Але потім придумали в ролі цієї «пам'яті» використовувати спеціальні комірки, схожі на пам'ять комп'ютера або регістри процесора. Кожна комірка дозволяла записати в себе цифру, прочитати або скинути - їх назвали комірки (осередки) довгої і короткострокової пам'яті (LSTM).

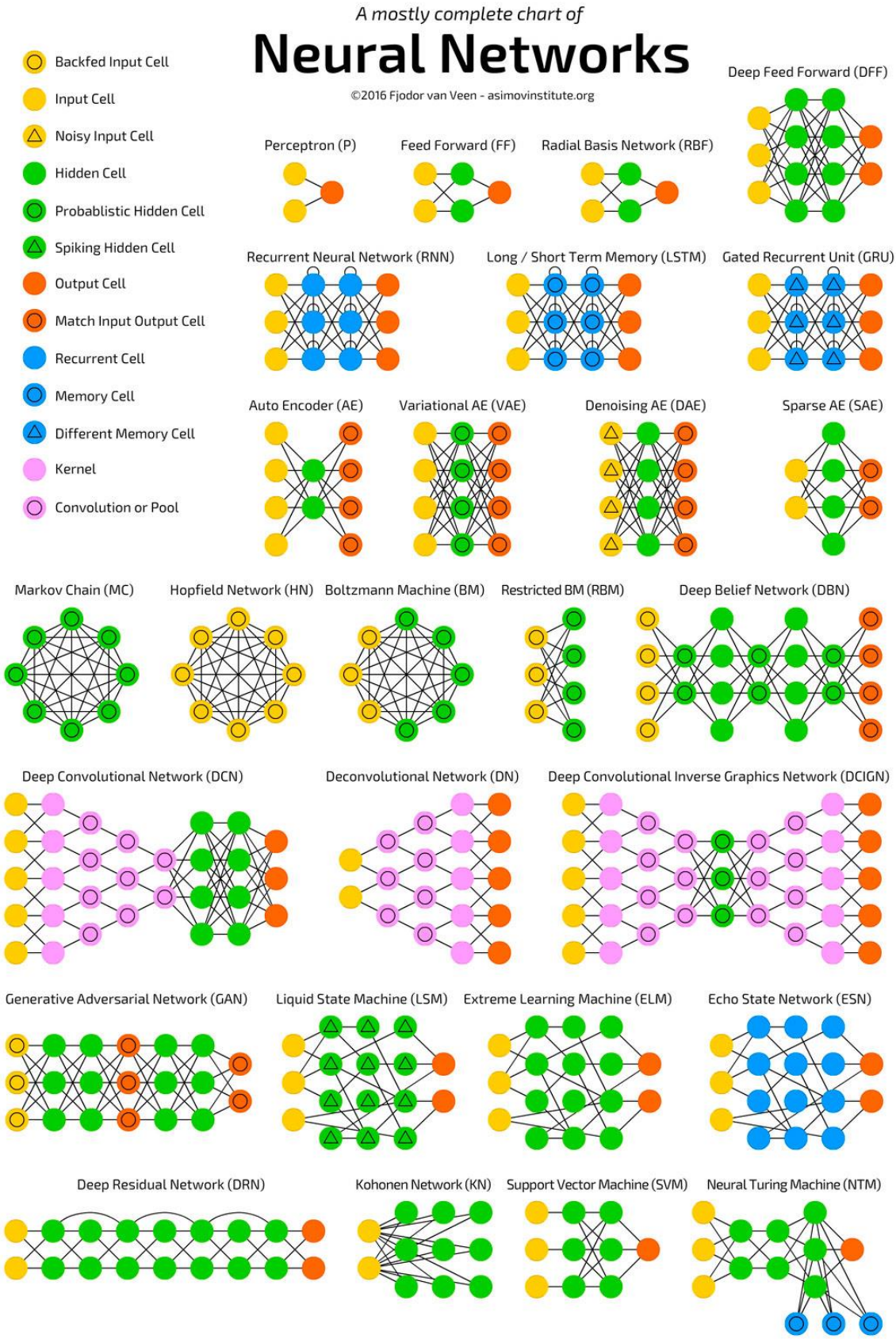
Коли нейрону було потрібно поставити собі нагадування на майбутнє - він писав це в комірку, коли навпаки вся історія ставала непотрібною (речення, наприклад, закінчилося) - комірки видалялися, залишаючи тільки «довгострокові» зв'язки, як в класичному перцептроні. Іншими словами, мережа навчалася не тільки встановлювати поточні зв'язки, а й ставити нагадування.

Просто, але працює!

Озвучені тексти для навчання почали брати звідки завгодно. Навіть базфід зміг вивантажити відеозаписи виступів Обами і вельми непогано навчити нейромережу розмовляти його голосом. На цьому прикладі видно, що імітувати голос - досить просте завдання для сьогоднішніх машин. З відео складніше, але це поки що.




Про архітектури нейромереж можна говорити нескінченно довго. Допитливих відправляю дивитися схему і читати статтю Neural Network Zoo, де зібрані всі типи нейронних мереж (є й росвересія).


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

## Висновки

Глибоке навчання - застосування машинного навчання, що дійсно швидко зростає. Численні додатки, описані вище, доводять його стрімкий розвиток лише за кілька років. Використання цих алгоритмів у різних галузях показує його універсальність. Аналіз публікацій, виконаний у цьому дослідженні, ясно демонструє актуальність цієї технології та дає чітку ілюстрацію зростання глибокого навчання та тенденцій щодо майбутніх досліджень у цій галузі.

Ієрархія рівнів та контроль у навчанні є ключовими факторами для розробки успішного застосування щодо глибокого навчання. Ієрархія важлива відповідної класифікації даних, тоді як контроль враховує важливість самої бази даних як частини процесу. Основна цінність глибокого навчання полягає в оптимізації існуючих програм у машинному навчанні завдяки інноваційності ієрархічної обробки. Глибоке навчання може забезпечити ефективні результати під час цифрової обробки зображень та розпізнавання мови. Зниження відсотка помилок (від 10 до 20%) явно підтверджує покращення порівняно з існуючими та перевіреними методами.

У нинішню епоху та в майбутньому глибоке навчання може стати корисним інструментом безпеки завдяки поєднанню розпізнавання облич та мови. Крім цього, цифрова обробка зображень є областю досліджень, яка може застосовуватися в багатьох інших областях. З цієї причини і довівши справжню оптимізацію, глибоке навчання є сучасним та цікавим предметом розвитку штучного інтелекту.