Тема 2 «Нейронні мережі»

Дані в машинному навчанні. Вступ до нейронних мереж, як найпоширенішого методу машинного навчання.

В попередніх лекціях ми ознайомились із базовими поняттями **машинного навчання** (МН), його видами, лінійною і логістичною регресією.

Поговоримо про те, як ми представляємо дані не тільки в ознаковому описанні об'єктів, а й про те, як варто уявляти собі дані, коли ми маємо справу із машинним навчанням.
Якщо ви візьмете якусь базу даних, то побачите величезну таблицю з великою кількістю колонок, рядків і т.д. Коли ми говоримо про дані в машинному навчанні, то конкретні приклади, випадки, спостереження – це **точки**.
Якщо приймати дані точками, що розкидані в певному просторі, а процес навчання – способом викривляти цей простір таким чином, щоб внаслідок його деформації всі точки, залежно від нашої задачі, розклалися на класи, змістилися чи розтягнулися уздовж певної прямої, тоді буде набагато простіше зрозуміти, що власне роблять системи машинного навчання.

Наприклад, візьмемо нейронні мережі.
Все, що робить **нейронна мережа** - ітерація за ітерацією деформує вхідний простір прикладів таким чином, щоб внаслідок деформації точки, які ми дали на вході, потрапили в зони, де ми очікуємо їх побачити не виході.
Кожен із параметрів нейронної мережі відповідає за певний вид деформації простору. Це може бути лінійна деформація, стискання, розтискання, більш складні деформації.
Будь-яка тренувальна вибірка – це набір точок, які знаходяться в певному просторі, незалежно від того, чи це картинки, чи це відео, чи це набір результатів соціологічного дослідження, чи характеристики веб-сторінки, яку ми аналізуємо. Є набір ознак, кожна ознака задає певний вимір і, відповідно, наші дані – це величезна кількість точок в багатовимірному просторі.

Ми працюємо з багатовимірними даними і намагаємося деформувати ці виміри таким чином, щоб точки прямували до певних атракторів (певних зон). У такому випадку одні точки будуть туди прямувати, а інші – ні.
Власне, весь процес навчання – це **процес деформування багатовимірних просторів**. Якщо просто уявити, що ми, фактично, просто розтягуємо певний об'єм, стискаємо його, перекручуємо його в певних місцях і більше

нічого не робимо, то розуміння того, що відбувається всередині нейронних мереж стає набагато простішим.

Методи, якими можна деформувати простір прикладів так, щоб, вкинувши нові точки із тестувальної вибірки, простір деформувався задля отримання вірних результатів, в сукупності і є **машинним навчанням.**
Яким чином ми будемо деформувати простір, залежить від того, яка архітектура системи.

Одним із найпоширеніших методів машинного навчання є **нейронні мережі** (НМ).
Нейронні мережі, як модель, з'явилися в середині XX ст. Передумовами для їх виникнення були дослідження процесів функціонування нервової системи людини. Стало відомо, що за нервову діяльність відповідають спеціальні клітини – **нейрони.**

Нейрони мають відростки – дендрити і аксон. Дендрити і аксони між собою мають контакти – синапси.

Через синапси відбувається передача інформації методом передачі нейротрансмітерів і електромагнітних імпульсів. Різні синапси мають різну пропускну здатність.

Кожен нейрон має певний набір сусідів і має контакти із ними. Сила контактів може варіюватися. Залежно від того, чи збуджуються нейрони одночасно і синхронно, чи розсинхронізовуються в своїй активації, сила зв'язків може посилюватись чи послаблюватись. Цей ефект називається **нейропластичністю.**

Спостерігаючи за передачею імпульсів між нейронами в біологічній системі, люди прагли сконструювати математичну модель, що могла б відтворити загальну динаміку поведінки нейронів без деталізації на молекулярному рівні.

Штучні НМ – це модель роботи нейронів на базі того, що було відомо в середині XX століття: модель синаптичної пластичності і того, що дозволяє біологічним нейронам адаптуватися та навчатися.

Як виглядає штучний нейрон? Це, фактично, набір зв'язків із іншими нейронами.

Уявімо собі найпростішу мережу – так званий **одношаровий перцептрон.** Він складається з одного шару нейронів.

Його архітектура виглядає так: є певна кількість сенсорних (вхідних) нейронів, кожен з яких може активуватися в певному діапазоні залежно від архітектури (0, 1 або -1, 1 або -1 тощо). Нейрон має зв'язки з іншими нейронами на попередньому шарі. У випадку одношарового перцептрона – це зв'язок із даними, які ми подаємо на вхід.

Коли нейрон отримав вхідний імпульс, він має зв'язки із іншими нейронами і, залежно від того, яка сила цих зв'язків, він впливає на активацію інших нейронів.

Вони, в свою чергу, отримуючи імпульси з попереднього шару, активуються на певний рівень, залежно від сили зв'язків, і передають естафету наступному шару.

Один нейрон окремо – це проста система, яка множить на певний набір коефіцієнтів вхідні імпульси, сумує і застосовує певну функцію активації (лінійну, сигмоїдальну, обмежену в певних діапазонів).

Функція активації – це спосіб трансформації нашого імпульсу в іншу форму. Наприклад, якщо нам потрібно зробити, щоб нейрон завжди активувався в межах від 0 до 1 незалежно від сили імпульсу від попередніх нейронів, ми застосовуємо сигмоїдальну функцію активації (Ви про неї чули у розділі логістичної регресії).

Отже, один окремий нейрон є досить примітивною і простою математичною функцією.

Але ще в середині 20 століття стало зрозуміло, що системи з великою кількістю нейронів можуть апроксимувати (наближати і описувати) досить непрості закономірності, знаходити закономірності між вхідними і вихідними даними і будувати моделі-узагальнення. Тобто НМ чудово підходять для задач моделювання певних даних і прогнозування поведінки цих даних на тестовій вибірці, а не тільки на тренувальній вибірці.

НМ – це математична модель, яка складається із нейронів – маленьких апроксиматорів – які мають зв'язки із попереднім шаром.

Іншими словами**, нейронна мережа** – це певна функція, яка має величезний набір параметрів, які ми даємо на вхід (т.зв. вектор ознак). Ця функція перемножує велику кількість разів параметри на певні навчені коефіцієнти

таким чином, що на виході ми отримуємо і аналізуємо інший вектор необхідної розмірності.
В нейронних мережах завжди на вхід подається певний вектор і на виході теж отримується вектор.
Далі цей вектор можна проаналізувати. Система машинного навчання має справу тільки з векторами в тому чи іншому виді і деформує їх в багатовимірному просторі.

Останніми роками інтерес до напрямку нейронних мереж знову зріс, оскільки з'явилася достатня кількість нових алгоритмів, які можуть пришвидшити процес навчання, вирішити задачі, які раніше складно вирішувалися. Було розв’язано декілька ключових проблем в так званих глибоких мережах, коли ми маємо більше, ніж один шар нейронів.
Як вже було сказано, нейронні мережі, виникли ще в середині двадцятого століття як підрозділ математики, але практичного інструментарію для експериментів із ними не було, хоча певні симуляції робилися у п'ятдесятих-шістдесятих роках. Потім інтерес до них зріс у вісімдесятих роках, бо збільшилась доступність комп'ютерів і можливість науковців експериментувати з різними архітектурами.
Останні 5-6 років ми спостерігаємо новий бум зацікавленості в нейронних мережах, оскільки виникла достатня кількість технічного забезпечення, яке може дозволити проводити величезну кількість обчислень на відеокартах в паралельному режимі.

Чому відеокарти так сильно впливають на швидкість роботи нейронних мереж? Все, що роблять нейронні мережі, це перемножують багатовимірні матриці між собою. Процесор виконує операцію за операцією послідовно: одну операцію виконав – перейшов до наступної. Якщо ми маємо справу із багатовимірними матрицями розмірами 100000 на 100000 і нам потрібно їх між собою перемножити, уявіть собі, яку кількість послідовних операцій необхідно виконати процесору.
Відеокарта працює інакшим чином. Відеокарта – це, фактично, набір великої кількості маленьких процесорів, кожен з яких отримує свою маленьку команду, виконує її, але не послідовно, а паралельно. Тобто ми одну задачу передаємо сотням або тисячам маленьких процесорів.

Процес навчання нейронної мережі методом зворотного поширення помилки. Оцінка помилки.

Минулого разу ми ознайомились із тим, що таке нейронні мережі (НМ).

Нагадаємо, що **лінійна модель нейронних мереж** працює таким чином: у нас є певний набір вхідних даних, які представляються у виді багатовимірних векторів (набір ознак), є очікувана відповідь нейронної мережі, і процес **навчання** нейронної мережі – це знаходження такого набору вагів, (коефіцієнтів зв'язків між нейронами), щоб вихід нашої НМ відповідав нашим очікуванням відносно вхідних прикладів.

Яким чином поширюються сигнали уздовж НМ? Уявімо, що ми маємо структуру із нейронів (вершини нашого графа) і зв'язків між нейронами (ребра). Лінійна нейронна мережа має, як правило, шарову структуру, тобто має вхідних шар, вихідний шар та приховані шари. Кожен шар має зв'язки із попереднім і з наступним (окрім вхідного і вихідного). Імпульси рухаються вглиб мережі від входу до виходу, при цьому змінюються коефіцієнти відповідно до того, як натренована наша мережа.

Яким чином відбувається **навчання**, і що таке навчання саме в розрізі нейронних мереж, особливо в розрізі лінійної моделі нейронних мереж?

В процесі навчання НМ одним із найпоширеніших методів навчання є **метод зворотного поширення помилки.**

Це метод проходження по нейронній мережі, тільки в зворотному напрямку. І, комбінуючи проходження по НМ вперед і назад, НМ –нейронні мережі вчаться.

Як це виглядає?

Припустимо, у нас є набір прикладів, де у нас є вхідний і вихідний вектори. Припустимо, на виході маємо 2 нейрони, а на вході 10. Кожен із них має певний рівень активації.

Ми беремо вхідний приклад, активуємо кожен із вхідних нейронів, отримуємо рівень активації входу. Далі, знаючи, які нейрони вхідного шару з якими нейронами першого рівня прихованого шару з'єднані, ми множимо на відповідні коефіцієнти вагів кожен із елементів вхідного вектора. В результаті отримуємо рівень активації прихованого шару. Якщо у нас там 10 нейронів – отримаємо 10 значень. Якщо НМ має глибшу структуру, аналогічним чином вираховуємо другий прихований шар і т.д. Наприкінці

вираховуємо рівень активації вихідного шару і отримуємо в результаті певні значення.

Ці значення, як правило, не будуть відповідати нашим очікуванням і будуть мати певну величину помилки. Що ми робимо в такому випадку?

По кожному із елементів, маючи приклад правильної класифікації (нашого очікування) і поточної класифікації роботи нашої НМ, ми віднімаємо від кожного із елементів його еталонне значення і отримуємо **величину помилки.**

Зворотне поширення помилки. Посилення зв'язків.

Тепер ми знаємо, яку помилку зробив кожен із вихідних нейронів.

Але, враховуючи те, що кожен із вихідних нейронів активувався до певного рівня внаслідок того, що нейрони з попереднього шару теж активувались певним чином, відповідно кожен із нейронів попереднього шару робить внесок у помилку, яку робить даний нейрон.

Якщо у нас НМ глибша за одношарову, значить і нейрони ще глибшого попереднього шару теж роблять певний внесок у помилку нейронів другого шару і т.д. до самого початку.

Якщо ми хочемо, щоб наша НМ робила мінімальну помилку, нам потрібно скорегувати ваги між нейронами так, щоб структура нашого графу НМ відповідала даним, на які ми її тренували. Тобто треба віднайти такий набір параметрів, який буде максимально наближати дану функцію до реального розподілу даних.

Що для цього необхідно зробити?

Ми отримали значення помилки на вихідних нейронах. На наступній ітерації нам потрібно визначити, який внесок зробив кожен із нейронів попереднього шару у об'єм помилки. Це потрібно для того, щоб визначити, на яку величину треба скорегувати силу зв'язку на кожному із ребер між нейронами.

Якщо у нас 2 шари по 10 нейронів, це означає, що у нас є 10×10 ребер між елементами графу і нам потрібно 10×10 елементів скорегувати. Відповідно до кожного з цих елементів нам потрібно знайти певне відхилення, на яке на даній ітерації роботи алгоритму нам потрібно збільшити або зменшити силу зв'язку на даному ребрі.

Отже, для того, щоб знайти величину, на яку ми повинні скорегувати кожне із ребер нашого графа, тобто **посилити чи послабити зв'язок** між двома нейронами, нам потрібно спочатку визначити, який об’єм помилки належить саме цьому ребру, тобто який внесок в поточну помилку робить сила зв’язку між цими двома нейронами.

Щоб це зробити, для останнього шару ми вираховуємо різницю між нашими очікуваними даними і тим, що видала мережа по кожному із параметрів і змінюємо силу зв’язку пропорційно до об’єму помилки, помноживши її на певний коефіцієнт, який ми задаємо перед процесом навчання. Він буде «згладжувати» процес навчання, щоб приклади, які занадто відрізняються від середнього розподілу прикладів, не сильно змінювали зв’язки в мережі для прикладів, що вибиваються зі статистики.

Отримавши величини, на які ми повинні скорегувати ваги передостаннього шару, ми йдемо вглиб і, знаючи величини помилок на останньому шарі, аналогічно вираховуємо, на скільки потрібно змінити силу зв’язків між елементами передостаннього шару.

Врешті-решт, коли ми вирахували об’єм вкладу в помилку кожного із ребер, ми корегуємо зв’язки пропорційно до цього об’єму в протилежному напрямку до градієнту помилки (для того, щоб система рухалася в напрямку, де помилка зменшується, а не збільшується).

Загалом, процес зворотного поширення помилки ми повторюємо після кожного з прикладів, які отримує наша система. І сила зв’язків між елементами ітерація за ітерацією наближається до правильного набору значень.

Зворотне поширення помилки. Зміна величини кроку. Локальні мінімуми.

В процесі навчання НМ методом зворотного поширення помилки є декілька «наріжних» каменів, на які треба звернути увагу.

Першим критичним параметром є **величина кроку**, на яку ми змінюємо ваги. Якщо б ми корегували кожен раз силу зв’язків між нейронами так, щоб мінімізувати помилку до 0, наша НМ мало б чого навчилася, оскільки б вона «зависла» в середньостатистичному локальному мінімумі і не могла б знайти оптимальну точку, в якій максимально точно прогнозує вихідні дані.

Відповідно процес навчання в нас аналогічний до того, що відбувається в процесі **градієнтного спуску**. Нагадаємо, що це метод знаходження мінімуму помилки. У нас є певна функція помилки і на певних даних їй відповідає певна гіперплощина. У певних точках площини величина помилки або дуже велика, або дуже низька. Ми хочемо підібрати такі параметри функції, щоб застосувавши до неї нашу функцію помилки, ми знаходилися в ідеалі в найнижчій точці площини помилки.

Алгоритм зворотного поширення помилки фактично робить ту ж річ і так само йому характерні недоліки даного алгоритму, а саме наявність **локальних мінімумів**:

На певній гіперповерхні можуть бути заглибини, в яких з усіх сторін точки знаходитимуться вище, тобто всюди навколо помилка є вищою. Але дана точка не є найглибшою точкою на поверхні помилки. Відповідно, якщо система не зможе робити певний крок більшим, ніж певне відхилення, вона не зможе вийти з локального мінімуму.

Коли ми маємо справу із функцією із мільйоном параметрів, а НМ є саме такими функціями, то поверхня нашої функції помилки є досить неоднорідною і має велику кількість заглиблень, жолобів, і «застрягти» в певному локальному мінімумі – досить легко. Якщо крок зміни системи буде досить маленький, то ми не можемо зробити достатній ривок, щоб вистрибнути з мінімуму. Якщо крок буде занадто великим, система стрибатиме постійно між досить віддаленими точками на поверхні помилки і ймовірність того, що вона потрапить в глобальний чи достатній локальний мінімум – дуже низька.

Які є методи оптимізації процесу знаходження локальних мінімумів?

Це **поступове зменшення величини кроку**, з якою ми рухаємось. Так можна знайти досить масштабні заглибини. Далі ітерація за ітерацією в конкретній локації можна знайти менші заглибини. Але і тут ми не застраховані від маленького локального мінімуму, хоча поряд є більші мінімуми.

Тому є варіанти навчання, коли величина кроку, з яким ми рухаємось (величина коефіцієнту, на який домножується різниця між очікуваним і фактичним значенням на вході і виході на кожному із ребер, від чого залежить, на скільки сильно в даному напрямку зміститься мережа) змінюється **адаптивно**.

Зворотне поширення помилки. Адаптивна зміна величини кроку.

Ми спостерігаємо, на скільки в процесі навчання зменшилася наша помилка. Якщо крок за кроком, збільшуючи величину кроку, наша НМ перестає зменшувати помилку, це означає, що далі крок необхідно зменшувати. Ми починаємо зменшувати розміру кроку, тримаючи темп зменшення помилки, щоб він коливався в певному невеликому діапазоні і не «стрибав» вгору або вниз. Таким чином ми поступово зменшуємо і збільшуємо крок, з яким рухаємось, і, таким чином, збільшуємо вірогідність встановлення навіть глобального мінімуму.

Метод зворотного поширення помилки зараз став фактично канонічним методом в навчанні лінійних НМ і від розуміння принципу, з яким ми корегуємо наші ваги між нейронами, залежить те, наскільки буде зрозумілим подальший матеріал, про який ми говоритимемо: **регуляризація** та **методи ініціалізації нейронних мереж**.

Підсумовуючи: **Метод зворотного поширення помилки** – спосіб корегування вагів в НМ таким чином, щоб внаслідок корегування наша НМ видавала меншу помилку на даному прикладі. Ітерація за ітерацією, надаючи нейронній мережі приклад за прикладом і корегуючи відповідним чином ваги в НМ пропорційно до величини помилки, ми наближаємо вектор зв’язків в НМ до такого вигляду, в якому НМ буде видавати максимально репрезентативні дані, які відповідатимуть нашим очікуванням.

Одними із ключових параметрів, які впливають на процес навчання – це величина кроку, методи зміни кроку, а також метод ініціалізації початкових значень вагів в НМ.

Нейронні мережі зі згорткою (конволютивні)

Ми вже дізналися, що таке лінійні моделі нейронних мереж (НМ), а також дізналися, якими методами їх можна вивчати, зокрема методом зворотного поширення помилки.

Зараз зупинимося на тому, чим відрізняються **повнозв’язні моделі лінійних НМ** і **моделі лінійних НМ зі згорткою**.

Популярність НМ, особливо в останній час, завдячується великому успіху в сфері розпізнавання образів – статичних та в відеопотоці. Ці задачі були нерозв’язними в певних масштабах 5-6 років тому назад. Зараз просто і

ефективно для вирішення цих проблем використовуються моделі НМ, які може зібрати в себе на комп'ютері людина з досить посереднім рівнем уявлення про те, як працюють НМ.

Завдяки чому стався такий прорив?

Одним із ключових недоліків НМ є те, що це функції з неймовірною кількістю параметрів (інколи сотні мільйонів). Це потребує досить великих обчислювальних потужностей, високого рівня паралелізму (паралельні обчислювальні системи - відеокарти).

Причиною прориву є не тільки збільшення обчислювальних потужностей, а й **поширення конволютивних нейронних мереж зі згорткою**.

Оскільки велика кількість параметрів вимагає значного часу на тренування (дні та тижні в залежності від доступних ресурсів), навчити систему – дуже складно.

Уявімо, наприклад, що системі необхідно розпізнати образи на картинці розміром 256 на 256 пікселів, яка є кольоровою (RGB). Оскільки маємо три кольори – три канали, то фактично на вхід до НМ ми подаємо 3 картинки, кожна розміром 256 на 256 пікселів. Кількість вхідних ознак ми можемо підсумувати і вона вийде вражаючою – кожен піксель кожного каналу буде ознакою. Тобто кількість ознак дорівнюватиме 256×256×3.

Задачею нашої НМ, якщо ми хочемо, наприклад, навчити її розпізнавати котиків на зображенні, є – із даної кількості ознак знайти певні закономірності, які дозволять вирішити, наприклад, бінарну задачу – є котик на зображенні чи ні.

Нейронні мережі зі згорткою. Приклад роботи.

Ми можемо натренувати дану НМ, що займе певний час.

Але в ситуації, якщо наша тренувальна вибірка з 10000 картинок котиків містила більшість зображень, де котики розміщуються в центральній частині зображення або в лівому нижньому куті, і не було жодного котика в правому верхньому куті, ми потрапимо в складне становище.

Наша НМ зможе непогано відрізняти зображення з котиками від зображень, де їх немає, але лише там, де котик знаходиться у центральній частині рисунку. Але якщо котик буде у правій верхній частині – система із великою імовірністю класифікує картинку неправильно.

Це відбувається тому, що на процес класифікації мали вплив конкретні нейрони, які отримують вхідний сигнал,з координатами, наприклад, 250 на 250 (в правому нижньому куті зображення). В певній конфігурації з іншими нейронами це буде призводити до того, що рівень очікувань, що на даній картинці є котик, буде збільшуватись. Натомість, нейрон з аналогічним віддаленням, але в правому верхньому куті буде мати інший вплив. І чутливість до кожного з цих нейронів навчалася окремо.

Що таке нейромережі зі згорткою і як вони допоможуть вирішити цю задачу?

Нейромережі зі згорткою вирішують відразу 2 задачі:

1. *Вивчення нелокальних закономірностей* – тобто здатність знаходити певні закономірності, певні патерни не тільки з прив’язкою до їхнього локального значення, а, якщо це випадок із картинкою, по всій площі картинки.
2. *Неймовірне зменшення кількості параметрів*, які ми вивчаємо.

Розглянемо повнозв’язну НМ, яка на вхід отримує картинку, є одношаровою, шар має 1000 нейронів і вихідний шар є класифікатором. Відповідно кожен із 1000 нейронів має зв'язок із усіма нейронами, які були на вході: 256×256×255×3×1000.

Що пропонує НМ зі згорткою? Припустимо, ми робимо систему з такою ж кількістю нейронів на другому шарі – 1000. Але, на відміну від повнозв’язної НМ, де кожен нейрон має зв'язок з усіма нейронами на попередньому шарі, в НМ зі згорткою ми не маємо безпосередніх зв’язків з конкретними нейронами попереднього шару. Натомість ми розглядаємо картинку і дивимось на неї певним «віконечком» заданого розміру.

Як виглядає процес прямоточного проходження сигналу через конволютивну НМ?

Ми беремо нашу картинку 256×256×3 і на першому рівні вивчаємо, припустимо, 128 фільтрів. Розмір фільтру 3×3 пікселі. Наша НМ замість того, щоб дивитися на всю картинку загалом, ітерація за ітерацією, по кожному із фільтрів буде проходитися маленьким віконечком 3×3 пікселі і бачитиме в конкретний момент часу лише маленький блок з 27 значень: 3 множимо на 3 і на 3.

В результаті НМ отримує рівень активації кожного із фільтрів.

Кожен із фільтрів в процесі навчання зворотного поширення помилки – це певний паттерн, певний фільтр, який вивчається. І, відповідно, все зображення розглядається даним фільтром, як відхилення від даного фільтру. Тобто, якщо фрагмент, який ми бачимо, дуже подібний до конкретного фільтру (припустимо, фільтр – це градієнт від темного до світлого згори вниз, і конкретний фрагмент є досить подібним до даного фільтру), на який ми зараз дивимося, відповідно даний фільтр в конкретній зоні активується дуже сильно. Якщо він не подібний, він активується на низьке значення, а, можливо, буде нульовим.

В результаті, ми 128-ма фільтрами продивилися все зображення з віконечком 3×3 пікселі. В кожен момент спостереження отримали певний рівень активації.

Картинку розміром 256×256×3 ми перетворили в 128 картинок аналогічного розміру, але тепер на кожній із цих картинок на відміну від RGB-каналу активність в певних зонах має певне значення. Якщо ми візьмемо фільтр, який вивчив вертикальні лінії, то ми помітимо, що всі зони на картинці, де були наявні вертикальні лінії, на даній карті активації цього фільтру – досить яскраво світяться. Натомість, всі зони, де лінії перпендикулярні – досить низько активовані.

В результаті, кожен із фільтрів вивчає свої досить специфічні ознаки. І ці ознаки, як правило, в процесі навчання будуть підбиратися таким чином, щоб вивчити фільтри, які максимально критично впливають на якість класифікації зображень.

Маючи такий блок 128×256×256 рівнів активації фільтрів, ми повторюємо цей же алгоритм, тобто конструюємо шар активації, по якому тепер проходимося аналогічними фільтрами – 3×3 чи 5×5 і т.п. Це само по собі збільшує кількість параметрів, які ми вивчаємо. Припустимо, маючи вже даний блок, ми проходимося знову вікном 3×3, але в глибину вже на 128 рівнів активації і вивчаємо фільтри 2-го порядку, які на вхід отримують не RGB-картинку, а рівні активації фільтрів в певній зоні і вивчають ознаки комбінації фільтрів.

Якщо ми вивчали 256 можливих фільтрів 2-го порядку, ці фільтри будуть вивчати ознаки більш високоабстрактні, оскільки вони оперують не ознаками кольору, а ознаками структурності країв, градієнтів, комбінацій цих градієнтів.

І на виході ми теж отримуємо також блок активації 256 фільтрів.

Max-пулінг

Ще один із методів обробки цих фільтрів в конволютивних НМ – це так званий **max-пулінг**.

Ми отримали в результаті картинку розміром 256×256×256 – великий блок рівнів активації різних фільтрів. Але, якщо ми хочемо збільшувати рівень абстрактності, нам потрібно збільшити зону, за яку відповідає один рівень активації фільтрів.

Тому нам потрібно зменшити саме зображення.

Зменшення самого зображення, яке ми роздивляємося, призводить до зменшення кількості обчислень і збільшує репрезентативну здатність наших фільтрів.

Ми проходимося по нашій картинці 256×256 наших фільтрів активації з віконечком 2×2 і залишаємо для наступного шару тільки найбільше значення в даному віконечку. Тобто, якщо в певному фрагменті ми бачимо значення 0,1; 0,2; 0,2; 0,3, то на наступний рівень переходить просто максимальне значення.

В результаті, ми зменшуємо розмір картинки в 4 рази, оскільки лише кожне 4 максимальне значення фільтру переходить на наступний шар, і тепер кожному із цих значень відповідає зона зображення, яка в 4 рази більше, ніж цей піксель.

На наступному шарі ми можемо аналогічним чином тренувати набір фільтрів, які на вхід отримують лише відфільтровані максимальні рівні активації попереднього рівня.

Фільтри, які натренуються на вищому ієрархічному рівні, матимуть ще більш високоабстрактні ознаки.

Тобто, якщо на першому рівні ми вивчали певні комбінації кольорів і градієнти кольорів, на другому – комбінації країв, градієнтів, то на третьому – спостерігатимемо фільтри, які активуються, коли в місцях зображення, наприклад, є білий кут чи текстура асфальту тощо.

Зі збільшенням глибини мережі, з додаванням рівнів фільтрації, рівень абстрактності зростатиме. На певних рівнях ми навіть можемо отримати фільтри, які активуються в місцях, де є людське обличчя.

В результаті, чергуючи конволютивні шари (тобто **процес згортки** – спостереження не цілісної картинки, а спостереження за активацією наборів віконечок відповідно до певного фільтру), які збільшують абстрактність репрезентації даних, і шари макс-пулінгу, які зменшують розмірність, на певному рівні ми дійдемо до досить високоабстрактних репрезентацій нашої картинки.

На відміну від сирих даних, якби ми давали на вхід активацію пікселів, тепер кожне із значень чітко корелює із високоабстрактними ознаками – лапа або вухо кота тощо.

Подавши на вхід нейронній мережі прямого поширення блок активації фільтрів, а не сирі дані, наша НМ буде в результаті тренування вивчати не локальні закономірності: незалежно від того, в якій зоні зображення буде кіт, НМ його знайде.

Крім того, кількість параметрів, які ми оптимізуємо – зменшилась на порядки. Звісно, у нас на кожному із рівнів може бути по декілька сотень фільтрів, але це все набагато менше, ніж коли ми з'єднуємо повнозв'язний шар із вхідними піксельними значеннями.

Спроектувавши рівні активації на прихований шар нейронів (512 або 1024 нейрони), на які проектуються наші фільтри, і після нього шар класифікації – є кіт чи немає – 1 або 0, відповідно ми можемо навчати цю систему методом зворотного поширення помилки, оскільки на кожному із етапів ми можемо визначити, який внесок в помилку був у кожного із елементів.

В процесі навчання фільтри підберуться таким чином, щоб набір фільтрів відповідав тим ознакам, які максимально критично відповідають ідентифікації наявності котика.

Тобто наші фільтри вивчатимуть певні ознаки, які характерні саме котам.

І при їх наявності будуть збільшувати ймовірність класифікації до 1 класу, а не нульового.