Когнітивні науки в інженерії програмного забезпечення

6 кредитів

Іспит

Чим займається когнітивна наука

Когнітивістика, або інакше Cognitive Science - цей науковий напрям, який займається вивченням пізнавальних і розумових процесів і моделюванням принципів, по яких працюють природні і штучні системи.

Когнітивістика об'єднує багато напрямів і теорії : когнітивну психологію, нейрофізіологію, теорію штучного інтелекту і пізнання, а також лінгвістику.

У чому сенс когнітивістики

Розвиток когнітивістики може допомогти в майбутньому придбати нові знання про розум і отримати повну картину процесів, що відбуваються в людському мозку, які відповідають за вищу нервову систему.

Ці знання наближають людство до створення штучного інтелекту,

Когнітивістика і інженерія програмного забезпечення

Зверніть увагу на складові цієї науки.

Цей напрям точно для вас, якщо вам цікаві:

- розвиток штучного інтелекту
- робота мозку і пізнавальні процеси психіки, що відбуваються
- механізми, які лежать в основі розумної поведінки
- робота, розвиток і функціонування нервової системи
- комп'ютерні технології і обчислювальна техніка
- філософія, співвідношення свідомості і фізичної реальності

Магістратура в Нідерландах

Програма Brain and Cognitive Science в Амстердамському університеті включає три основні напрями: поведінкова нейронаука, когнітивна нейронаука і когнітивна наука.

У рамках програми ви можете змінювати напрями, поєднуючи курси так, щоб вони відповідали вашим індивідуальним перевагам. Вимоги до минулої освіти кандидатів залежать від вибраних ними напрямів.

Для іноземних студентів пропонуються декілька стипендій, серед яких Amsterdam Excellence Scholarship, Amsterdam Merit Scholarship і Holland Scholarship.

Штучні нейромережі

Штучні нейромережі є моделями нейронної структури мозку, який головним чином навчається з досвіду.

Природній аналог доводить, що множина проблем, які поки що не підвладні розв'язуванню наявними комп'ютерами, можуть бути ефективно вирішені блоками нейромереж.

Тривалий період еволюції додав мозку людини багато якостей, що відсутні в сучасних комп'ютерах з архітектурою фон Неймана. До них відносяться:

- розподілене представлення інформації і паралельні обчислення;
- здатність до навчання і здатність до узагальнення;
- адаптивність;
- терпимість до помилок
- низьке енергоспоживання.

- Здатність нейронної мережі до навчання вперше досліджена Дж. Маккалоком і У. Піттом. У 1943 році вийшла їх робота "Логічне числення ідей, що відносяться до нервової діяльності", в якій була побудована модель нейрона, і сформульовані принципи побудови штучних нейронних мереж.
- В 1959 р. Бернард Відроу (*Bernard Widrow*) та Марсіан Хофф (*Marcian Hoff*) розробили моделі *ADALINE* та *MADALINE* (Множинні Адаптивні Лінійні Елементи (*Multiple ADAptive LINear Elements*)). *MADALINE* діяла, як адаптивний фільтр, що усував відлуння на телефонних лініях. Ця нейромережа й досі в комерційному використанні.
- Значний поштовх розвитку нейрокібернетики дав американський нейрофізіолог Френк Розенблат, який запропонував в 1962 році свою модель нейронної мережі - персептрон. Незважаючи на це в 70-ті роки було запропоновано багато цікавих розробок, таких, наприклад, як когнітрон, здатний добре розпізнавати досить складні образи незалежно від повороту і зміни масштабу зобр

Здатність нейронної мережі до навчання вперше досліджена Дж. Маккалоком і У. Піттом. У 1943 році вийшла їх робота "Логічне числення ідей, що відносяться до нервової діяльності", в якій була побудована модель нейрона, і сформульовані принципи побудови штучних нейронних мереж.

В 1959 р. Бернард Відроу (*Bernard Widrow*) та Марсіан Хофф (*Marcian Hoff*) розробили моделі *ADALINE* та *MADALINE* (Множинні Адаптивні Лінійні Елементи (*Multiple ADAptive LINear Elements*)). *MADALINE* діяла, як адаптивний фільтр, що усував відлуння на телефонних лініях. Ця нейромережа й досі в комерційному використанні.

Backpropagation

Метод зворотного поширення помилки (англ. Backpropagation) - метод навчання багатошарового перцептрона. Вперше метод був описаний в 1986 р Девідом І. Румельхартом, Дж. Е. Хінтоном і Рональдом Дж. Вільямсом .

Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи багатошарового перцептрона і отримання бажаного виходa.

Метод є модифікацією класичного методу градієнтного спуску.

Джеффрі Хінтон

Джеффрі Хінтон (англ. Geoffrey Hinton; рід. 6 грудня 1947 року) - британський інформатик, відомий своїми роботами над штучними нейромережами.

Хінтон закінчив Кембриджський університет в 1970 році, де отримав ступінь бакалавра мистецтв в області експериментальної психології, а в 1978 році отримав докторський ступінь в Единбурзькому університеті в області штучного інтелекту.

На даний момент є професором департаменту комп'ютерних наук в Торонтського університету. У березні 2013 року він почав працювати з Google після того як його компанія DNNresearch Inc. увійшла до її складу.

Був одним з дослідників, які запропонували використовувати метод зворотного поширення помилки для тренування багатошарової нейронної мережі. Разом з Террі Сейновскі винайшов машину Больцмана.

Його пра-прадідом був **Джордж Буль**, чиї роботи в подальшому лягли в основу сучасної комп'ютерної науки.

Значний поштовх розвитку нейрокібернетики дав американський нейрофізіолог Френк Розенблат, який запропонував в 1962 році свою модель нейронної мережі - персептрон. Незважаючи на це в 70-ті роки було запропоновано багато цікавих розробок, таких, наприклад, як когнітрон, здатний добре розпізнавати досить складні образи незалежно від повороту і зміни масштабу зображення.

У 1982 році американський біофізик Дж. Хопфілд запропонував оригінальну модель нейронної мережі, названу його ім'ям. У наступні кілька років було знайдено безліч ефективних алгоритмів: мережа зустрічного потоку, двонаправлена асоціативна пам'ять і ін.

Застосування нейронних мереж

Класифікація образів розпізнавання образів. Завдання полягає у визначенні приналежності вхідного образу, представленого вектором ознак, одному чи декільком попередньо визначеним класам. До відомих застосувань відносяться розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограми, класифікація кліток крові.

Кластеризація / категоризація. При рішенні задачі кластеризації, що відома також як класифікація образів "без вчителя", навчальна множина з визначеними класами відсутня. Алгоритм кластеризації заснований на подобі образів і розміщує близькі образи в один кластер. Відомі випадки застосування кластеризації для видобутку знань, стиснення даних і дослідження властивостей даних.

Апроксимація функцій. Припустимо, що є навчальна вибірка $((x_1,y_1), (x_2,y_2)..., (x_n,y_n))$ (пари даних вхід-вихід), яка генерується невідомою функцією F, спотвореної шумом. Завдання апроксимації полягає в знаходженні невідомої функції F. Апроксимація функцій необхідна при рішенні численних інженерних і наукових задач моделювання.

Застосування нейронних мереж

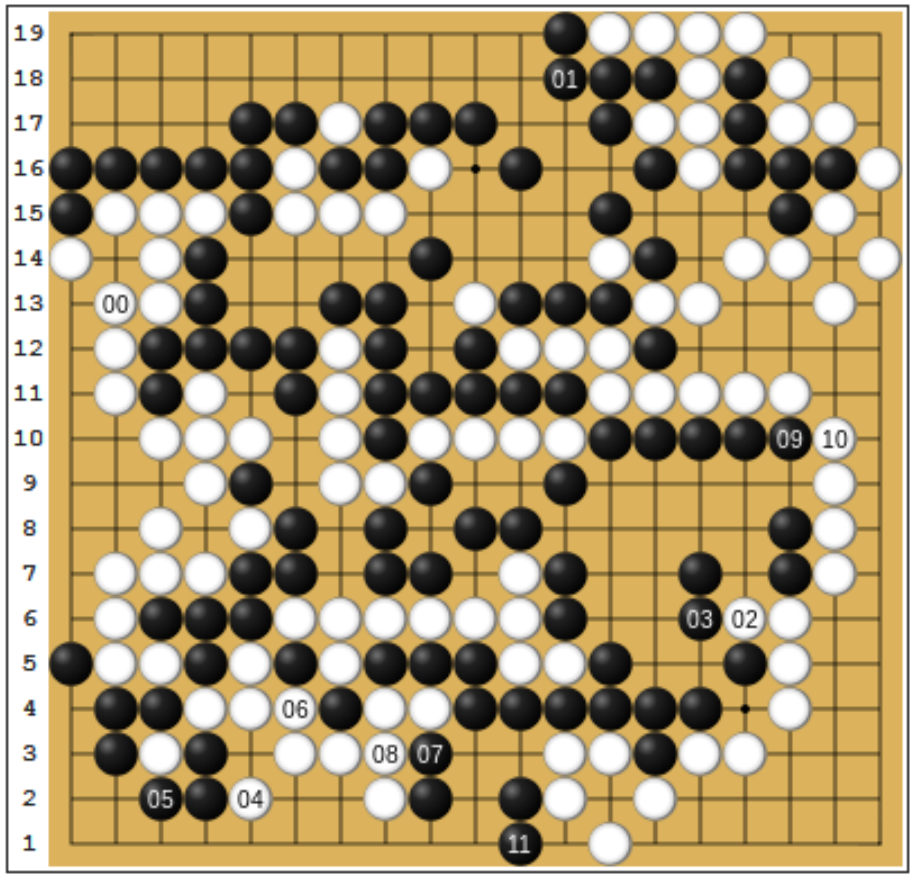
Передбачення/прогноз. Нехай задані n дискретних відліків $\{y(t_1), y(t_2), ..., y(t_n)\}$ у послідовні моменти часу $t_1, t_2,..., t_n$. Завдання полягає в передбаченні значення $y(t_n+1)$ у деякий майбутній момент часу t_n+1. Передбачення/прогноз мають значний вплив на прийняття рішень у бізнесі, науці і техніці (передбачення цін на фондовій біржі, прогноз погоди).

Оптимізація. Численні проблеми в математиці, статистиці, техніці, науці, медицині й економіці можуть розглядатися як проблеми оптимізації. Задачею алгоритму оптимізації є знаходження такого рішення, що задовольняє системі обмежень і максимізує чи мінімізує цільову функцію.

Застосування нейронних мереж

Асоціативна пам'ять. В традиційних комп'ютерах звертання до пам'яті доступно тільки за допомогою адреси, що не залежить від змісту пам'яті Асоціативна пам'ять, чи пам'ять, що адресується за змістом, доступна за вказівкою заданого змісту. Вміст пам'яті може бути викликано навіть по частковому входу чи спотвореному змісту. Асоціативна пам'ять надзвичайно бажана при створенні мультимедійних інформаційних баз даних.

Керування. Розглянемо динамічну систему, задану сукупністю $\{u(t), y(t)\}$, де $u(t)$ є вхідним керуючим впливом, а $y(t)$ - виходом системи в момент часу t. В системах керування з еталонною моделлю метою керування є розрахунок такого вхідного впливу $u(t)$, при якому система діє по бажаній траєкторії, заданою еталонною моделлю. Прикладом є оптимальне керування двигуном.


19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
00
01
02
03
04
05
06
07
08
09
10
11

ходы 200—211

AlphaGo

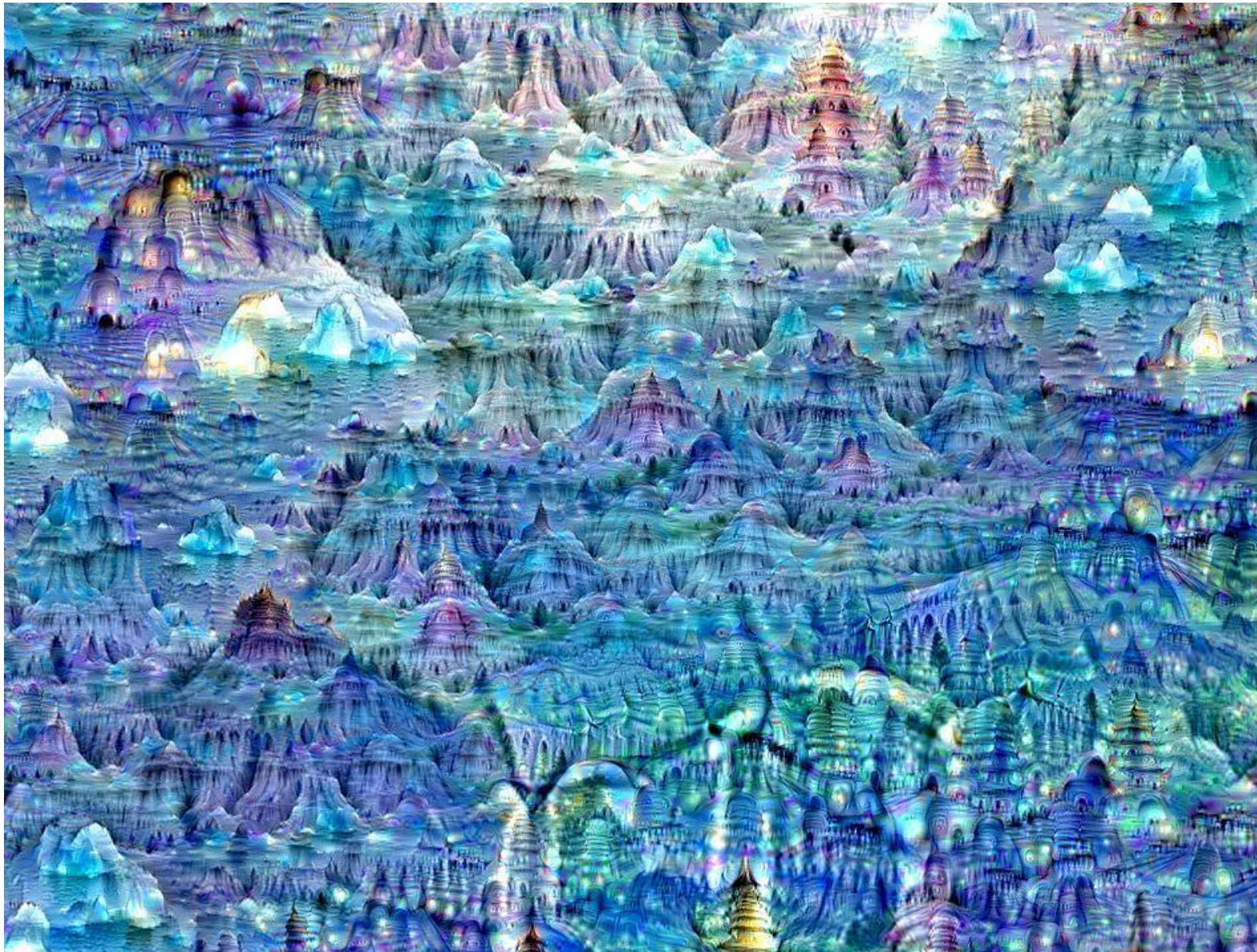
Седол Лі, чемпіон світу з Го з Південної Кореї, програв матч в серії ігор з системою штучного інтелекту AlphaGo, яку програмісти з компанії Google представили світу в січні цього року, повідомляє Американська асоціація Го.

Система II AlphaGo була розроблена Девідом Сильвером і його колегами з підрозділу DeepMind компанії Google, що базується в Лондоні (Великобританія) в кінці минулого року,

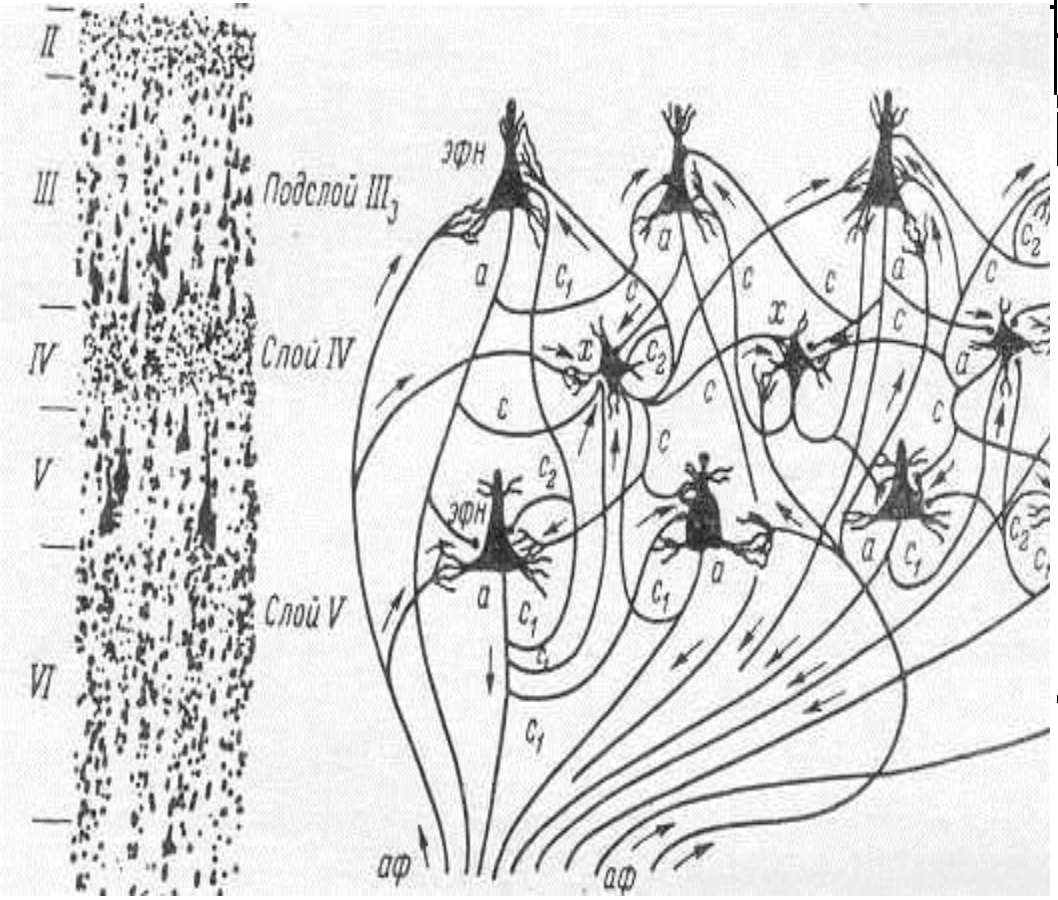
Для роботи AlphaGo використовувалися 1920 процесорів і 280 графічних процесорів, що працюють в розподіленої мережі. Ігри транслювалися в прямому ефірі на YouTube . Матч завершився перемогою AlphaGo з рахунком 4: 1

AlphaZero

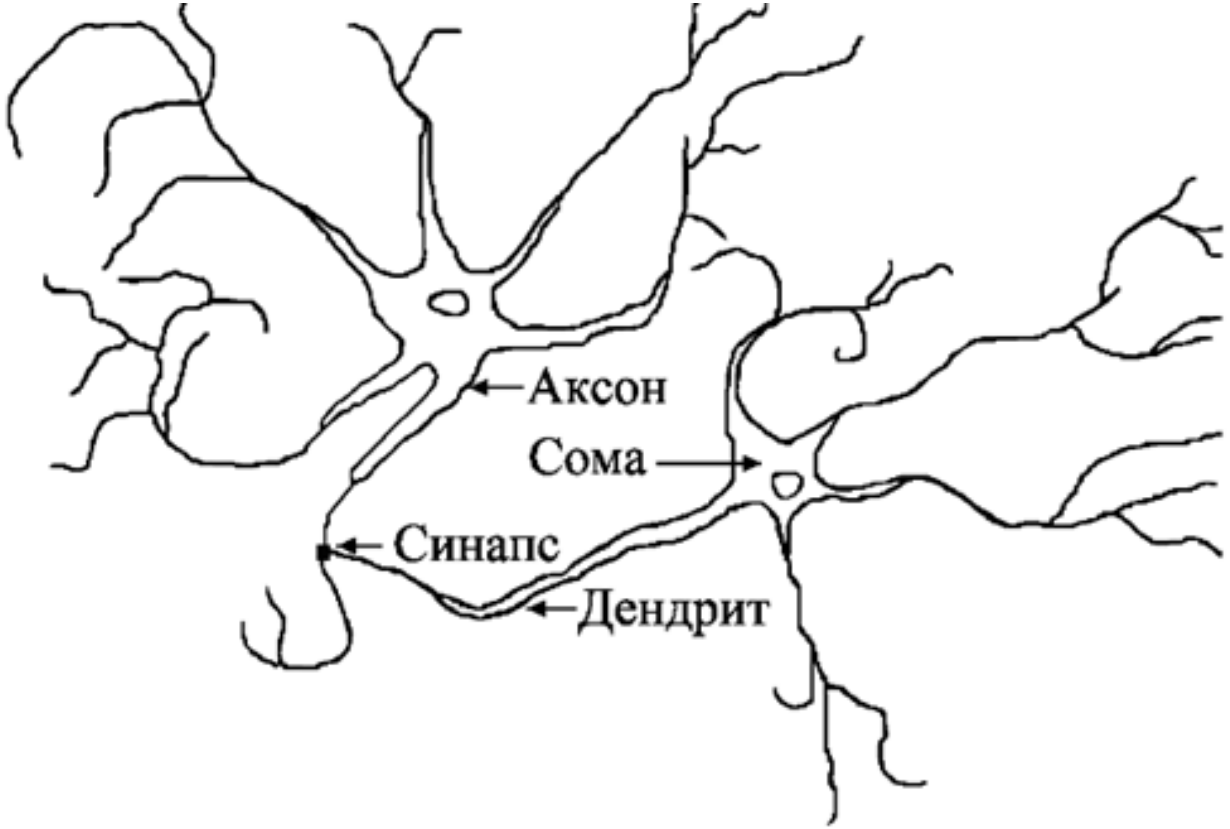
AlphaZero — це комп'ютерна програма, розроблена компанією DeepMind, яка використовує узагальнений підхід AlphaGo Zero. 5 грудня 2017 року колектив DeepMind випустив препринтне введення AlphaZero, яке впродовж 24 годин досягнуло надлюдського рівня гри в шахи, Сьоґі, і ґо, перемігши чемпіонів світу серед програм, Stockfish, Elmo. **Alpha**Zero в кожному випадку, використовуючи краще комп'ютерне обладнання відносно своїх опонентів. AlphaZero переміг Stockfish через 4 години самостійної гри, без доступу до дебютних баз та ендшпільних таблиць, але граючи з кращим комп'ютерним обладнанням виділеним для AlphaZero



Фрагмент нервової тканини кори мозку



Біологічний прототип



Теорема Колмогорова-Арнольда

Теорема Колмогорова-Арнольда (про яку часто не підозрюють практики) служить математичною основою нейронних мереж.

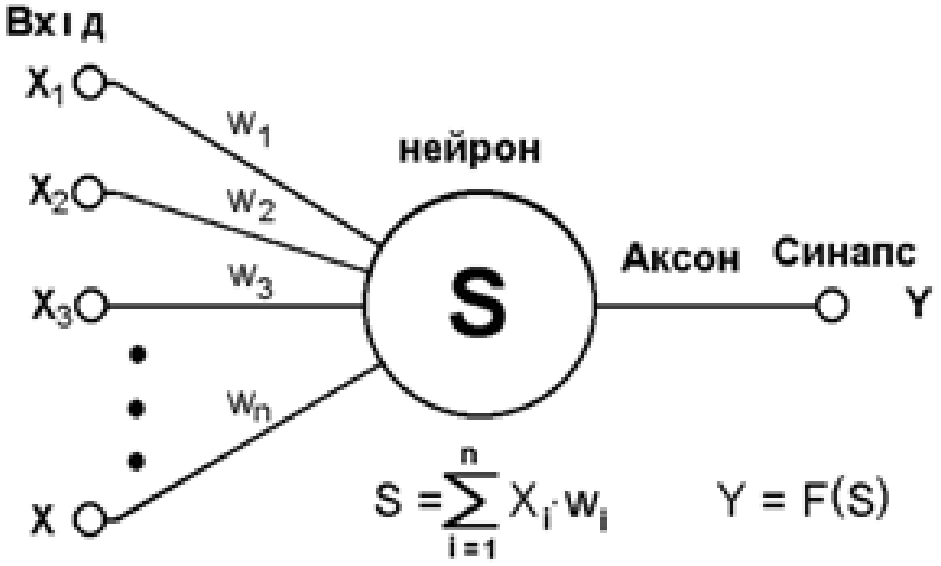
Теорема Колмогорова стверджує, що будь-яка безперервна функція *f,* визначена на n - вимірному одиничному кубі, може бути представлена у вигляді суми 2n+1 суперпозицій безперервних і монотонних відображень одиничних відрізків :

$$f(x_1,\ldots,x_n) = \sum_{q=1}^{2n+1} g_q\left(\sum_{p=1}^{n} \phi_{pq}(x_p)\right)$$

x = (x1, .., xn), 0 ≤ xi ≤ 1

Ліворуч в цій формулі стоїть довільна неперервна функція, визначена на багатовимірному кубі, справа функції визначені на відрізках [0, 1].

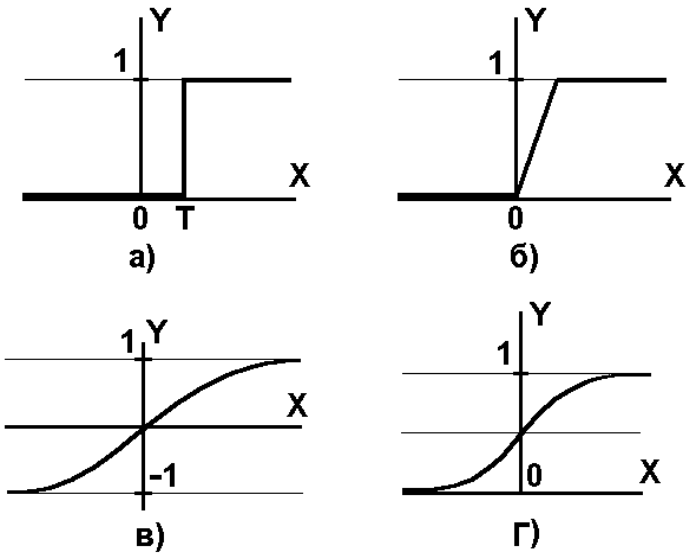
Штучний нейрон



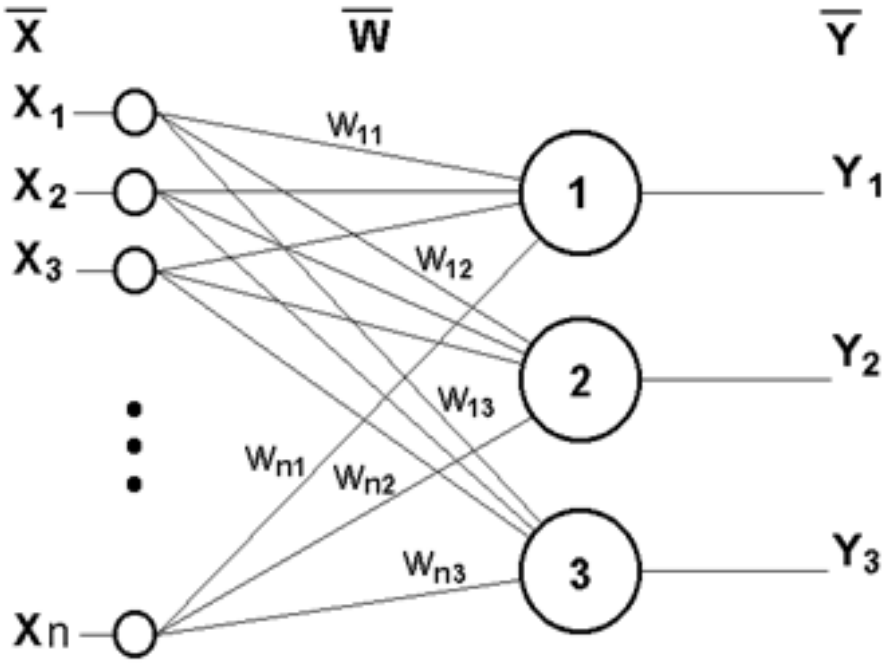
Штучний нейрон

$$S_p = \sum_{i=0}^{n} W_i \overline{X}_{pi}$$

$$F(S) = \frac{2}{1+e^{-aS}} - 1$$



Одношаровий персептрон



$$y_j = F\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right]$$

Лабораторна робота №1

Завдання: Побудувати бінарний класифікатор з допомогою моделі штучного нейрону.

Алгоритм лабораторної роботи №1

1.Обираємо задачу класифікації образів за неявними ознаками $\mathbf{A_j}$ та готуємо базу прикладів для навчання (навчальну вибірку) у вигляді таблиці:

Xpi	**A1**	**A2**	**...**	**An**	**dp**
P1	X_{11}	X_{12}		X_{1n}	$\mathbf{d_1}$
P2	X_{21}	X_{22}		X_{2n}	$\mathbf{d_2}$
...					
$\mathbf{P_M}$	X_{M1}	X_{M2}		X_{Mn}	$\mathbf{d_M}$

Де X_{pi} - числові значення ознаки **Aj** для прикладу $\mathbf{P_p}$,

d_p - ознака класу, $d_p = +1$ – I клас, $d_p = -1$ – II клас.

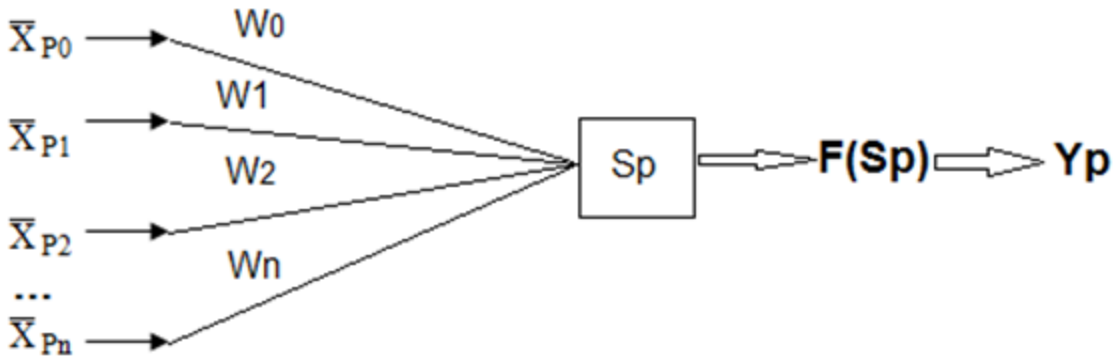
Число ознак повинно бути близько 5, а число прикладів більше 10, контрольна вибірка близько 5.

Лабораторна робота №1

2. Виконуємо природну нормалізацію ознак:

$$\overline{X_{pi}} = \frac{X_{pi} - X_{\min i}}{X_{\max i} - X_{\min i}}$$

3. Ініціалізуємо нейрон:



Лабораторна робота №1

Початкові значення вагових параметрів –невеликі, випадкові:

Wi = 0.01*rand(-1,+1).

Параметр ітерації: t=0.

Функцію активації приймаємо у вигляді:

$$F(S) = \frac{2}{1 + e^{-aS}} - 1$$

, де *a=2.*

Задаємо параметр швидкості навчання *η =0.9 і* точність навчання

ε =0.01.

p=1

Лабораторна робота №1

4. Основний цикл навчання за методом Відроу-Хебба:

a) Обираємо випадково новий приклад з навчальної вибірки з ознаками і обчислюємо зважену суму, вихідне значення і помилку:

$$S_p = \sum_{i=0}^{n} W_i \overline{X}_{pi} \qquad Yp=F(S_p),$$

$$\Delta p=(Y_p - d_p)$$

b) Обчислюємо вагові параметри для ітерації (t+1):

$$W_i(t+1) = W_i(t) - \eta \overline{X}_{pi} \Delta_p$$

c) Повторити b) для i=1,2,...,n.

d) Перейти до a) до вичерпання всіх прикладів навчальної вибірки.

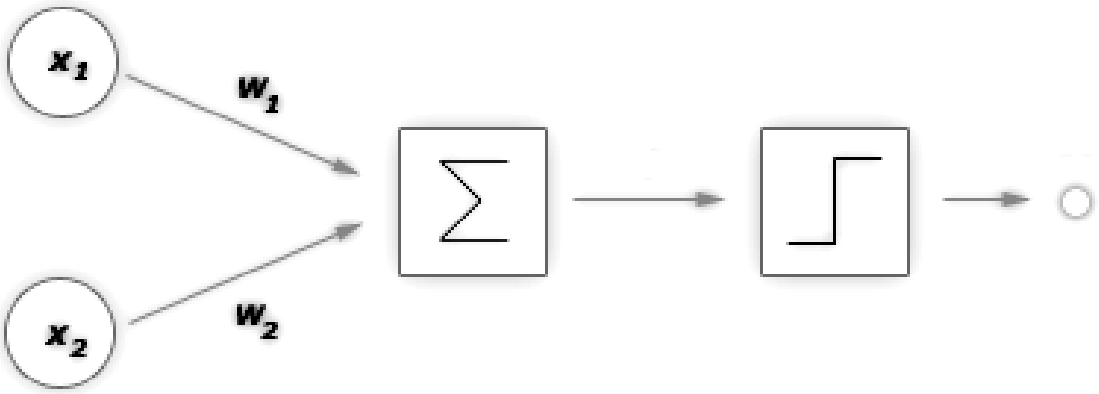
e) Обчислюємо сумарну помилку: $E = \frac{1}{2} \sum_{p=1}^{M} \Delta_p$

f) При умові E > ε^2 продовжуємо ітерації *t=t+1*, переходимо до a), при умові E <= ε^2 - вихід.

Проблема лінійного розподілу в НМ

Часто, для того, щоб продемонструвати обмежені можливості одношарових персептронів при рішенні завдань удаються до розгляду так званої проблеми XOR - що виключає АБО.

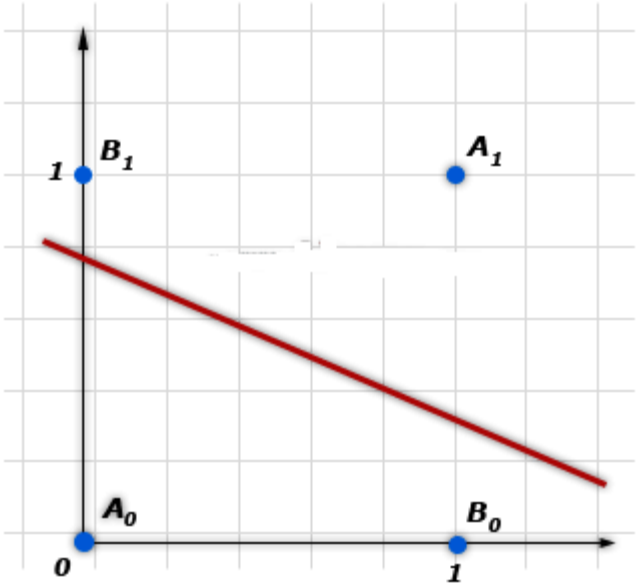
Суть завдання полягають в наступному. Дана логічна функція XOR - що виключає АБО. Це функція від двох аргументів, кожен з яких може бути нулем або одиницею. Вона набуває значення, коли один з аргументів дорівнює одиниці, але не обоє, інакше . Проблему можна проілюструвати за допомогою одношарової одно нейронної системи з двома входами, показаної на малюнку нижче.


x_1
w_1
x_2
w_2

Позначимо один вхід через X1, а інший через X2, тоді усі їх можливі комбінації складатимуться з чотирьох точок на площині.

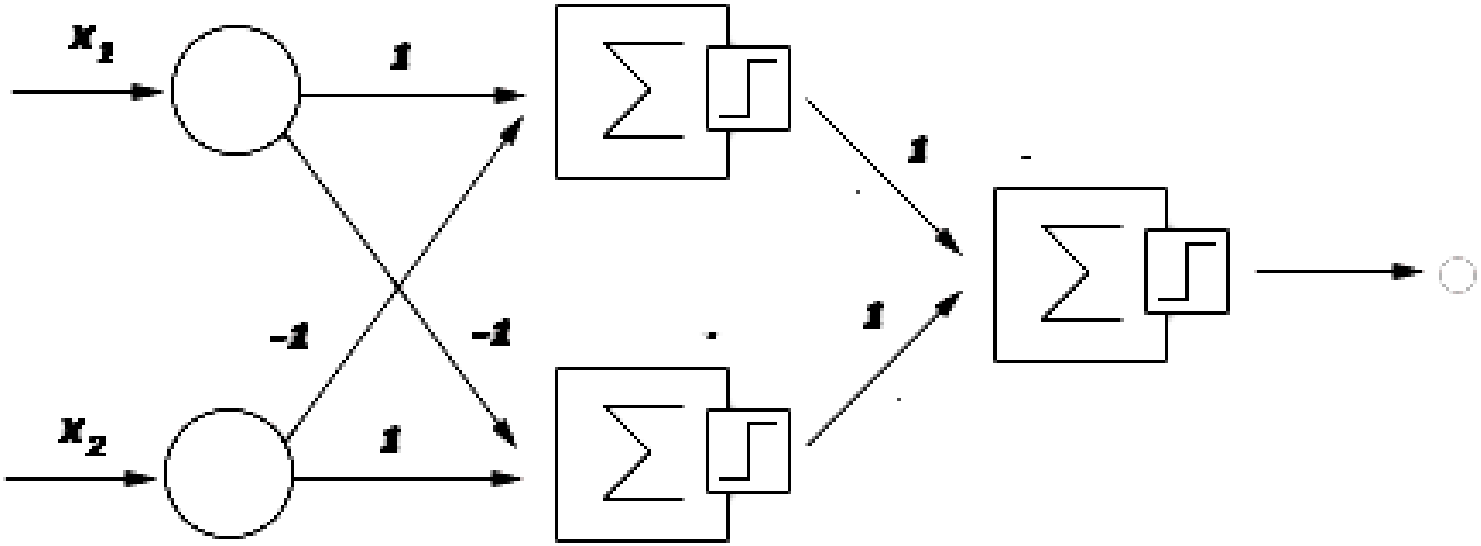
Один нейрон з двома входами може сформувати вирішальну поверхню у вигляді довільної прямої:

$$X1*W1+X2*W2 = T$$

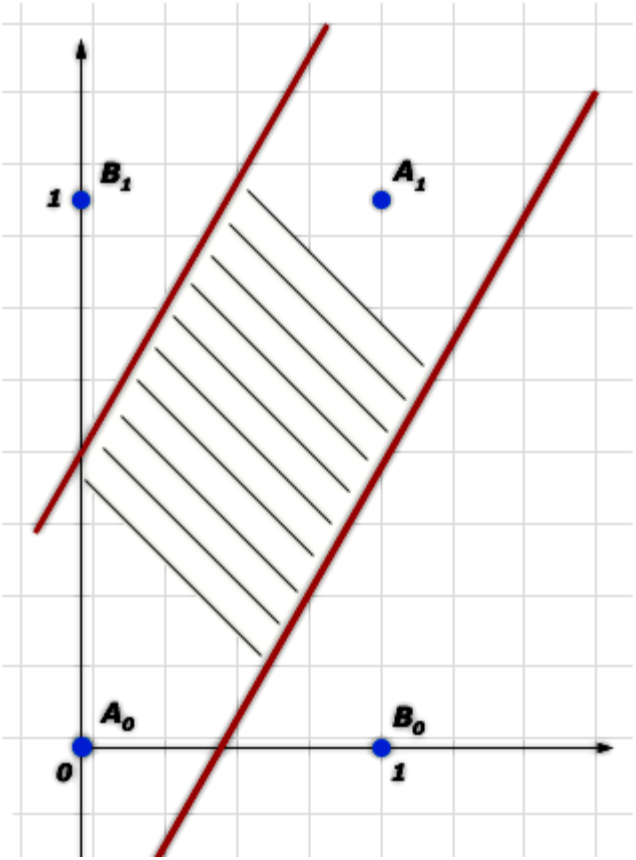

1
B_1
A_1
A_0
B_0
0
1

- Це означає, що які б значення не приписувалися вагам і порогу, одношарова нейронна мережа нездатна відтворити співвідношення між входом і виходом, потрібне для представлення функції XOR.

Проте функція XOR легко формується вже двошаровою мережею, причому багатьма способами. Розглянемо один з таких способів. Модернізуємо мережу, додавши ще один прихований шар нейронів


x_1
x_2
1
1
-1
-1
1
1

- Кожен з двох нейрон першого шару формує вирішальну поверхню у вигляді довільної прямої (ділить площину на дві напівплощини), а нейрон вихідного шару об'єднує ці два рішення, утворюючи вирішальну поверхню у вигляді смуги, утвореної паралельними прямими нейронів першого шару,


B1
A1
1
A0
B0
0
1

Навчання НМ без вчителя

Навчання людського мозку, на перший погляд, відбувається без вчителя: на зорові, слухові, тактильні та інші рецептори поступає інформація ззовні, і усередині нервової системи відбувається якась самоорганізація.

Проте, не можна заперечувати і того, що в житті людини не мало вчителів - і в буквальному, і у переносному розумінні, - які координують зовнішні дії.

Разом в тим, чим би не закінчилася суперечка прихильників цих двох концепцій навчання, вони обидві мають право на існування. Головна межа, що робить навчання без вчителя привабливим, - це його "самостійність".

Очевидно, що підстроювання синапсів може проводитися тільки на підставі інформації, доступної в нейроні, тобто його стану і вже наявних вагових коефіцієнтів. Виходячи з цього міркування і, що важливіше, по аналогії з відомими принципами самоорганізації нервових клітин, побудовані алгоритми навчання Хебба.

Сигнальний метод навчання Хебба

полягає в зміні ваг за наступним правилом:

$$w_{ij}(t)=w_{ij}(t-1)+\eta * y_i^{(n-1)} * y_j^{(n)} \qquad (1)$$

де $y_i^{(n-1)}$ - вхідне значення нейрону і шару; $y_i^{(0)} = x_i$;

$y_j^{(n)}$ - вихідне значення нейрона j шару n;

$w_{ij}(t)$, $w_{ij}(t-1)$ - ваговий коефіцієнт синапсу, що сполучає ці нейрони, на ітераціях t і $t-1$ відповідно;

η- коефіцієнт швидкості навчання.

Тут і далі, для спільності, під n мається на увазі довільний шар мережі. При навчанні по даному методу посилюються зв'язки між збудженими нейронами.

Існує також і диференціальний метод навчання Хебба

$$w_{ij}(t)=w_{ij}(t-1)+\eta * [y_i^{(n-1)}(t) - y_i^{(n-1)}(t-1)] * [y_i^{(n)}(t) - y_i^{(n)}(t-1)] \qquad (2)$$

Повний алгоритм Хебба навчання без учителя

1. На стадії ініціалізації всім ваговим коефіцієнтам привласнюються невеликі випадкові значення.
2. На вході мережі подається вхідний образ, і сигнали збудження розповсюджуються по всіх шарах згідно принципам класичних прямоточних (FeedForward) мереж тобто для кожного нейрона розраховується зважена сума його входів, до якої потім застосовується активаційна функція нейрона, внаслідок чого виходить його вихідне значення ***yi(n)***, i=0...Mi, де Mi - число нейронів в шарі i; n=1...N, а N - число шарів в мережі.
3. На підставі набутих вихідних значень нейронів по формулах проводиться зміна вагових коефіцієнтів.
4. Цикл повторюється з кроку 2, поки вихідні значення мережі не стабілізуються із заданою точністю. Застосування цього нового способу визначення завершення навчання, відмінного від зворотного розповсюдження, обумовлене тим, що підстроювані значення синапсів фактично не обмежені.

На другому кроці циклу поперемінно пред'являються всі образи з вхідного набору. Слід зазначити, що вид відгуків на кожен клас вхідних образів невідомий заздалегідь і буде довільним поєднанням станів нейронів вихідного шару, обумовлене випадковим розподілом ваг на стадії ініціалізації.

Разом з тим, мережа здатна узагальнювати схожі образи, відносячи їх до одного класу.

Алгоритм навчання без учителя - алгоритм Кохонена

- передбачає підстроювання синапсів на підставі їх значень від попередньої ітерації

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \eta[y_i^{(n-1)} - w_{ij}^{(n)}(t-1)] \quad (3)$$

З наведеної вище формули видно, що навчання зводиться до мінімізації різниці між вхідними сигналами нейрона, що поступають з виходів нейронів попереднього шару $y_i^{(n-1)}$ і ваговими коефіцієнтами його синапсів $w_{ij}^{(n)}$.

Повний алгоритм навчання має приблизно таку ж структуру, як в методах Хебба, але на кроці 3 з усього шару вибирається нейрон, значення синапсів якого максимально схожі на вхідний образ, і підстроювання ваг по формулі (3) проводиться тільки для нього.

Алгоритм навчання без учителя - алгоритм Кохонена

- Повний алгоритм навчання має приблизно таку ж структуру, як в методах Хебба, але на кроці 3 з усього шару вибирається нейрон, значення синапсів якого максимально схожі на вхідний образ, і підстроювання ваг по формулі (3) проводиться тільки для нього.
- Ця, так звана, акредитація може супроводжуватися гальмуванням усіх інших нейронів шару і введенням вибраного нейрона в насичення.
- Вибір такого нейрона може здійснюватися, наприклад, розрахунком скалярного добутку вектору вагових коефіцієнтів на вектор вхідних значень. Максимальне значення дає нейрон, що виграв.

Алгоритм навчання без учителя - алгоритм Кохонена

Інший варіант - розрахунок відстані між цими векторами в p - мірному просторі, де p - розмірність векторів

$$D_j = \sqrt{\sum_{i=0}^{p-1} (y_i^{(n-1)} - w_{ij})^2} \qquad (4)$$

де j - індекс нейрона в шарі n, i - індекс підсумовування по нейронах шару (n - 1),

w_{ij} - вага синапсу, що сполучає нейрони; виходи нейронів шару (n - 1) є вхідними значеннями для шару n.

Корінь у формулі (4) брати не обов'язково, оскільки важлива лише відносна оцінка різних Dj. В даному випадку, "перемагає" нейрон з найменшою відстанню.

Іноді занадто часто нейрони, що одержують акредитацію примусово виключаються з розгляду, щоб "зрівняти права" усіх нейронів шару. Простий варіант такого алгоритму полягає в гальмуванні нейрона, що тільки що виграв.

Алгоритм навчання без учителя - алгоритм Кохонена

При використанні навчання по алгоритму Кохонена існує практика нормалізації вхідних образів, а так само - на стадії ініціалізації - і нормалізації початкових значень вагових коефіцієнтів

$$x_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n-1} x_j^2}} \qquad (5)$$

де *xi* - і -а компонента вектору вхідного образу або вектору вагових коефіцієнтів, а n - його розмірність.

Це дозволяє скоротити тривалість процесу навчання. Ініціалізація вагових коефіцієнтів випадковими значеннями може привести до того, що різні класи, яким відповідають щільно розподілені вхідні образи, зіллються або, навпаки, роздрібняться на додаткові підкласи у разі близьких образів одного і того ж класу.

Алгоритм навчання без учителя - алгоритм Кохонена

Для уникнення такої ситуації використовується метод опуклої комбінації[3]. Суть його зводиться до того, що вхідні нормалізовані образи піддаються перетворенню :

$$x_i = \alpha x_i + \frac{1 - \alpha}{\sqrt{n}} \qquad (6)$$

де xi - і -а компонента вхідного образу, n - загальне число його компонент, α(t) - коефіцієнт, що змінюється в процесі навчання від нуля до одиниці, внаслідок чого спочатку на входи мережі подаються практично однакові образи, а з часом вони все більше сходяться до початкових. Вагові коефіцієнти встановлюються на кроці ініціалізації рівними величині

$$w_i = \frac{1}{\sqrt{n}} \qquad (7)$$

де n - розмірність вектору ваг для нейронів ініціалізованого шару.

Алгоритм навчання без учителя - алгоритм Кохонена

На основі розглянутого вище методу будуються нейронні мережі особливого типу - так звані структури, що самоорганізуються, - **self - organizing feature maps** (цей сталий переклад з англійського, на мій погляд, не дуже вдалий, оскільки, йдеться не про зміну структури мережі, а тільки про підстроювання синапсів). Для них після вибору з шару n нейрона j з мінімальною відстанню Dj (4) навчається по формулі (3) не лише цей нейрон, але і його сусіди, розташовані в околі R.

Величина R на перших ітераціях дуже велика, так що навчаються усі нейрони, але з часом вона зменшується до нуля. Таким чином, чим ближче кінець навчання, тим точніше визначається група нейронів, що відповідають кожному класу образів.

Лабораторна робота № 2

З допомогою одношарової мережі з двома нейронами на виході класифікувати навчальну вибірку із лабораторної роботи №1, використовуючи навчання без учителя сигнальним методом Хебба.

Алгоритм лабораторної роботи № 2

1. На стадії ініціалізації усім ваговим коефіцієнтам привласнюються невеликі випадкові значення.
2. На входи мережі подається вхідний образ, і сигнали збудження поширюються по усіх шарах згідно з принципами класичних прямопоточних (feedforward) мереж, тобто для кожного нейрона розраховується зважена сума його входів, до якої потім застосовується активаційна (передавальна) функція нейрона,

$$Sp = \sum_{i=0}^{n} Wi \cdot \bar{X}pi \qquad Yp=F(Sp),$$

внаслідок чого виходить його вихідне значення

$$Yj,\ j=1..M,$$

де Mi - число нейронів в шарі i; n=1..N, а N - число шарів в мережі.

3. На підставі набутих вихідних значень нейронів по формулах

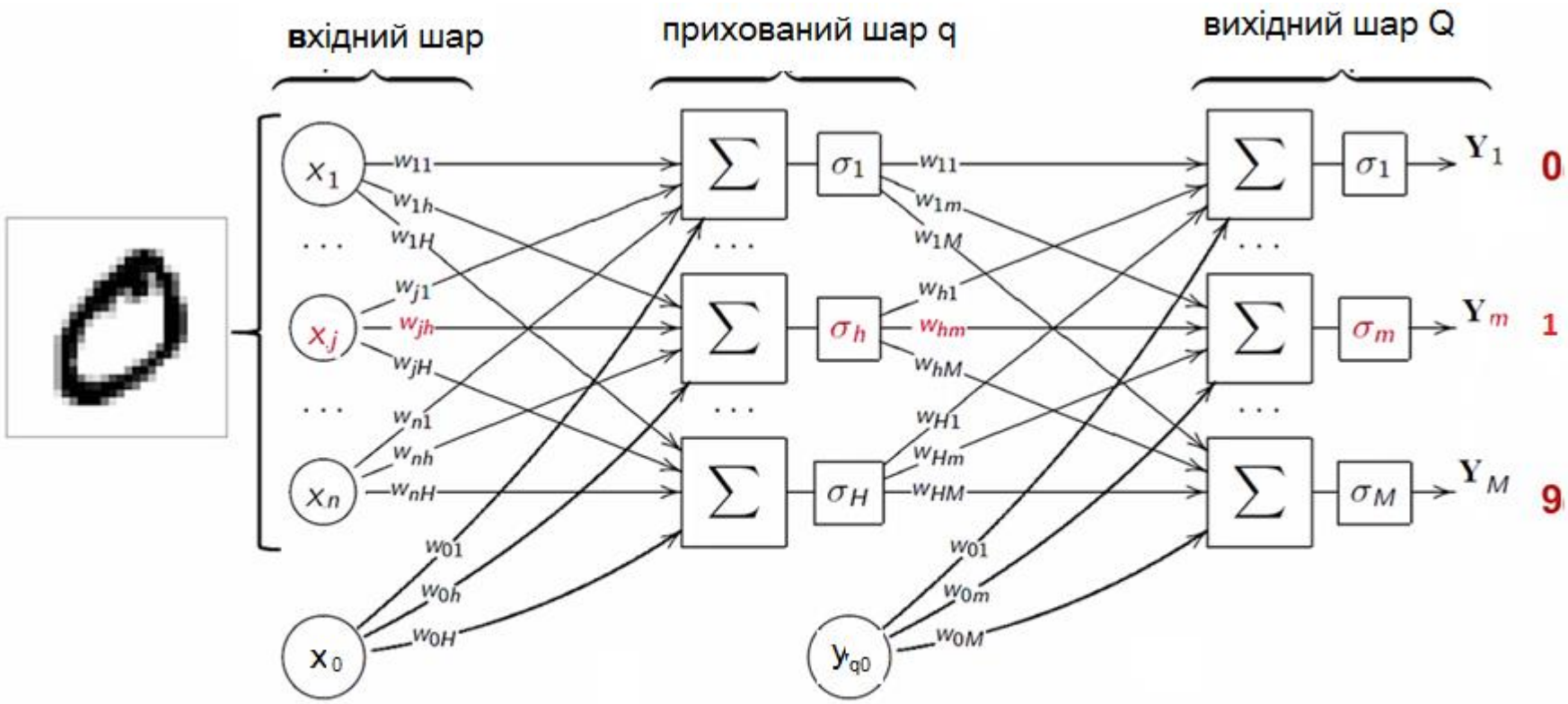
$$wij(t)=wij(t-1)+\eta * Xi * Yj$$

виконується зміна вагових коефіцієнтів.

4. Цикл з кроку 2, поки вихідні значення мережі не стабілізуються із заданою точністю. Застосування цього нового способу визначення завершення навчання, відмінного від зворотного поширення, що використалося для мережі, обумовлене тим, що налаштовувані значення синапсів фактично не обмежені.

На другому кроці циклу поперемінно пред'являються усі образи з вхідного набору.

Багатошарова НМ



Градієнтний спуск

Цільова функція помилки

$$E(w_{ij}^{(q)}) = \frac{1}{2}\sum_{p}\sum_{j=1}^{M} (y_{j,p}^{(Q)} - d_{j,p})^2 \to \min$$

Мінімізуємо функцію помилки для кожного прикладу по черзі

$$E_p(w_{ij}^{(q)}) = \frac{1}{2}\sum_{j=1}^{M} (y_{j,p}^{(Q)} - d_{j,p})^2 = \frac{1}{2}\sum_{j=1}^{M} \Delta_{j,p}^2 \to \min$$

Тоді

$$E(w_{ij}^{(q)}) = \sum_{p=1}^{P} E_p(w_{ij}^{(q)})$$

Градієнтний спуск

Згідно з методом градієнтного спуску η - крок

$$w_{ij}^{(q)}(t+1) = w_{ij}^{(q)}(t) + \Delta w_{ij}^{(q)} = w_{ij}^{(q)}(t) - \eta \cdot \frac{\partial E_p(w_{ij}^{(q)}(t))}{\partial w_{ij}^{(q)}}$$

Так як $\Delta_{j,p} = (y_{j,p}^{(Q)} - d_{j,p})$

То при *q*=*Q* $\frac{\partial E_p}{\partial w_{ij}^{(Q)}} = \frac{\partial E_p}{\partial \Delta_{j,p}} \cdot \frac{\partial \Delta_{j,p}}{\partial y_{j,p}^{(Q)}} \cdot \frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}}$

$$\frac{\partial E_p}{\partial \Delta_{j,p}} = \Delta_{j,p}; \frac{\partial \Delta_{j,p}}{\partial y_{j,p}^{(Q)}} = 1;$$

Оскільки $y_{j,p}^{(Q)} = F(\sum_{i=0}^{n_{Q-1}} w_{ij}^{(Q)} y_{i,p}^{(Q-1)}) = F(s_{j,p}^{(Q)})$

Градієнтний спуск

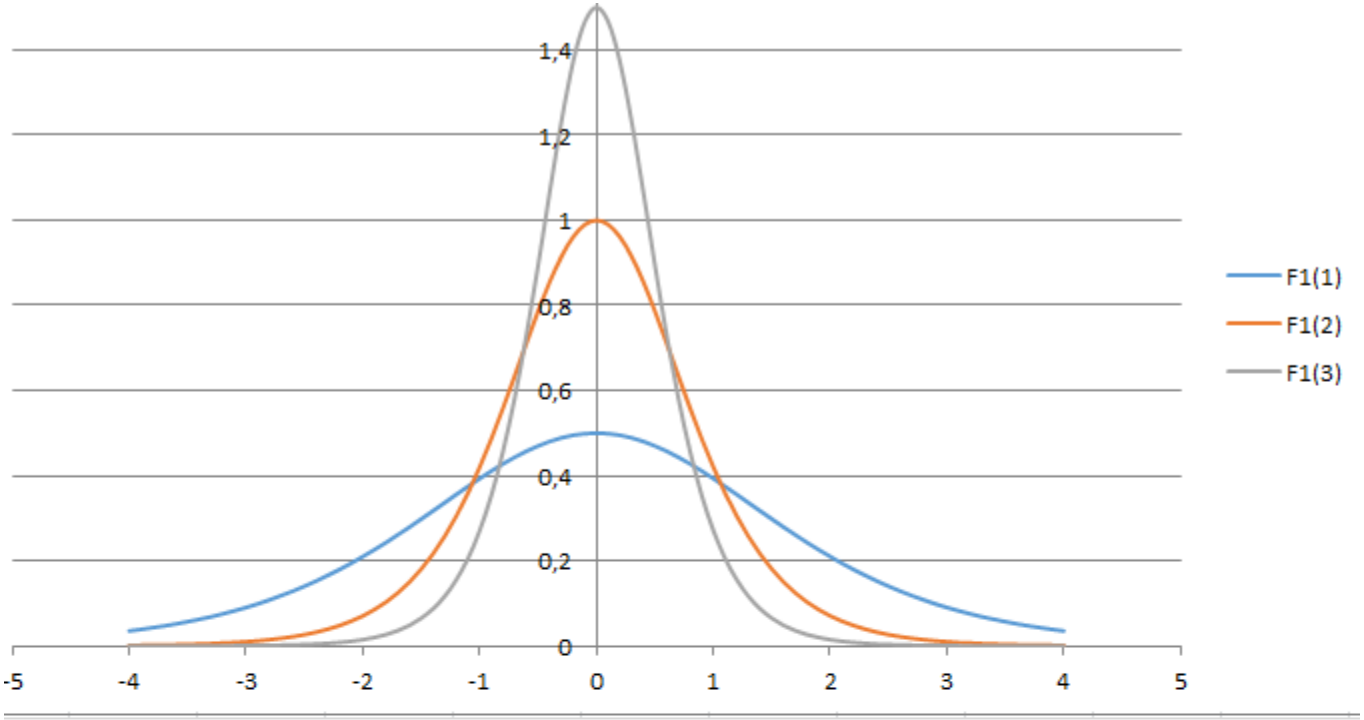
то

$$\frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}} = F'(s_{j,p}^{(Q)}) \cdot \frac{\partial s_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}} = F'(s_{j,p}^{(Q)}) \cdot y_{i,p}^{(Q-1)}$$

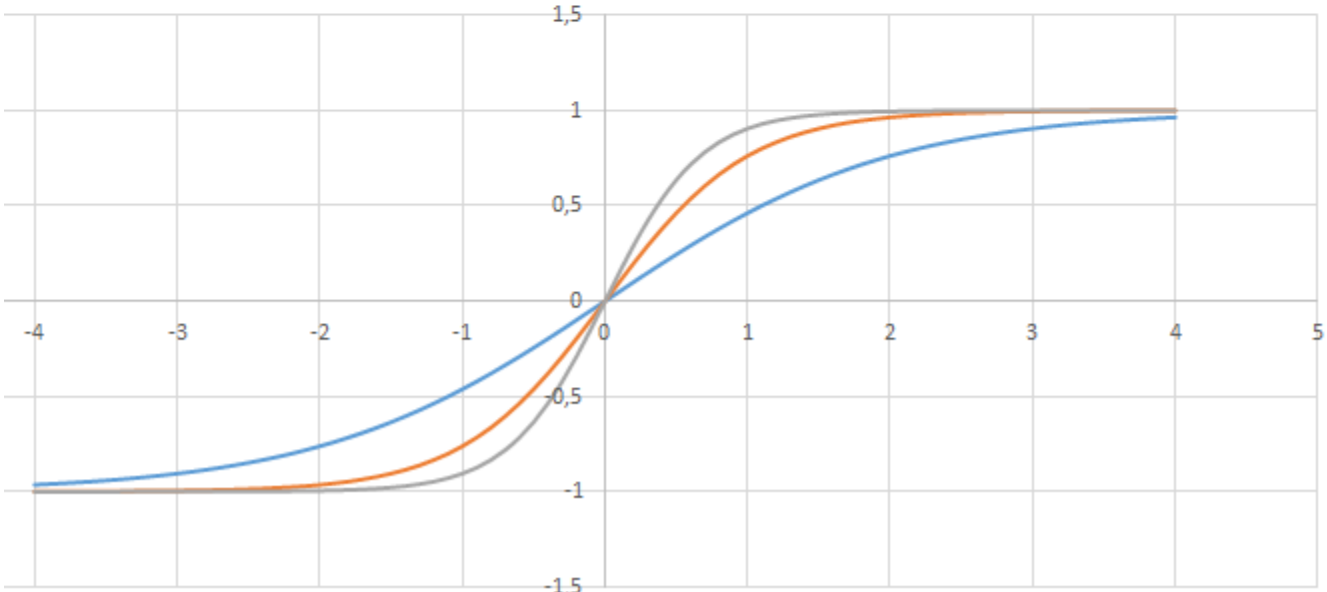
Оскільки множник $\frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}}$ є похідною функції по її аргументу, з цього виходить, що похідна активаційної функція має бути визначена на усій осі абсцис, тому застосовуються такі функції, як гіперболічний тангенс або класичний сигмоїд з експонентою.

$$F(s) = \frac{2}{1+e^{-a \cdot s}} - 1 = th(\frac{a \cdot s}{2})$$

$$F'(s) = \frac{a}{2 \cdot ch^2(\frac{a \cdot s}{2})} = \frac{a}{2}(1 - th^2(\frac{a \cdot s}{2})) = \frac{a}{2}(1 - F^2(s))$$


1,4
1,2
1
0,8
0,6
0,4
0,2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
F1(1)
F1(2)
F1(3)

Сігмоїд


1,5
1
0,5
0
-0,5
-1
-1,5
-4
-3
-2
-1
0
1
2
3
4
5

Градієнтний спуск

При одношаровій мережі Q=1, $y^{(0)}=x_i$

$$\frac{\partial E_p}{\partial w_{ij}^{(Q)}} = \Delta_{j,p} \cdot F'(s_{j,p}^{(Q)}) \cdot x_{i,p}$$

Формула для уточнення $w_{ij}^{(Q)}$ матиме вигляд

$$w_{ij}^{(Q)}(t+1) = w_{ij}^{(Q)}(t) - \eta \cdot (\Delta_{j,p} \cdot F'(s_{j,p}^{(Q)})) \cdot x_{i,p}$$

І процес навчання буде повторювати процедуру Відроу-Хебба, тобто уточнення

Для багатошарових НМ невідомі помилки у проміжних шарах

Алгоритм Back propagation error

Запропонований Дэвидом И. Румельхартом, Дж. Е. Хинтоном и Рональдом Дж. Вильямсом в работе

***Rumelhart D.E., Hinton G.E., Williams R.J.*, Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing, vol. 1, pp. 318—362. Cambridge, MA, MIT Press. 1986.**

Згідно з методом найменших квадратів, цільовою функцією помилки НС, що мінімізується, є величина:

$$E(w) = \frac{1}{2}\sum_{j,p}(y_{j,p}^{(Q)} - d_{j,p})^2 \qquad (2.1)$$

де $y_{j,p}^{(N)}$ - реальний вихідний стан нейрона j вихідного шару N нейронної мережі при подачі на її входи p -го образу;

d_{jp} - ідеальний (бажане) вихідний стан цього нейрона.

Підсумовування ведеться по усіх нейронах вихідного шару і по усіх оброблюваних мережею образах.

Алгоритм Back propagation error

Мінімізація ведеться методом градієнтного спуску, що означає підстроювання вагових коефіцієнтів таким чином:

$$w_{ij}^{(q)}(t+1) = w_{ij}^{(q)}(t) + \Delta w_{ij}^{(q)} = w_{ij}^{(q)}(t) - \eta \cdot \frac{\partial E_p}{\partial w_{ij}^{(q)}} \qquad (2)$$

Тут w_{ij} - ваговий коефіцієнт синаптичного зв'язку, що сполучає і –й нейрон шару n – 1 з j –м нейроном шару n,

η - коефіцієнт швидкості навчання, 0< η <1.

Очевидно що,

$$\frac{\partial E}{\partial w_j^{(q)}} = \frac{\partial E}{\partial y_j^{(q)}} \cdot \frac{dy_j^{(q)}}{ds_j^{(q)}} \cdot \frac{\partial s_j^{(q)}}{\partial w_j^{(q)}} \qquad (3)$$

Тут під y_j, як і раніше, мається на увазі вихід нейрона j, а під s_j – зважена сума його вхідних сигналів, тобто аргумент активаційної функції.

$$\frac{dy}{ds} = F'(s) = \frac{a}{2}(1 - y^2) \qquad (4)$$

Алгоритм Back propagation error

Третій множник ds_j/dw_{ij}), очевидно, дорівнює виходу нейрона попереднього шару: $y_i^{(q-1)}$.

$$s_j^{(q)} = \sum_{i=0}^{n_q} y_i^{(q-1)} \cdot w_{ij}^{(q)} \qquad s_j^{(q+1)} = \sum_{i=0}^{n_q} y_i^{(q)} \cdot w_{ij}^{(q+1)}$$

Що стосується першого множника в (3), він легко розкладається таким чином:

$$\frac{\partial E}{\partial y_j^{(q)}} = \sum_{i=0}^{n_{q+1}} \frac{\partial E}{\partial y_i^{(q+1)}} \cdot \frac{dy_i^{(q+1)}}{ds_i^{(q+1)}} \cdot \frac{\partial s_i^{(q+1)}}{\partial y_i^{(q+1)}} = \sum_{i=0}^{n_{q+1}} \left(\frac{\partial E}{\partial y_i^{(q+1)}} \cdot \frac{dy_i^{(q+1)}}{ds_i^{(q+1)}}\right) \cdot w_{ji}^{(q+1)} \tag{5}$$

Тут підсумовування по і виконується серед нейронів шару *n+1*.

Ввівши нову змінну

$$\delta_j^{(q)} = \frac{\partial E}{\partial y_j^{(q)}} \cdot \frac{dy_j^{(q)}}{ds_j^{(q)}} \tag{6}$$

отримаємо рекурсивну формулу для розрахунків величин (j(n) шару n з величин (k(n+1) більше старшого шару n+1.

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(q+1)}\right] \cdot F'(s_j^{(q)}) \tag{7}$$

Для вихідного ж шару

$$\delta_j^{(Q)} = (y_j^{(Q)} - d_j) \cdot F'(s_j^{(Q)})$$

Алгоритм Back propagation error

(8)

Тепер ми можемо записати (2) в розкритому виді:

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(n+1)} \right] \cdot F'(s_j^{(q)}) \tag{9}$$

Іноді для надання процесу корекції ваг деякої інерційності, що згладжує різкі скачки при переміщенні по поверхні цільової функції, (9) доповнюється значенням зміни ваги на попередній ітерації

$$\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}$$

$$\Delta w_{ij}^{(q)}(t+1) = -\eta \cdot (\mu \cdot \Delta w_{ij}^{(q)}(t) + (1-\mu) \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}) \tag{10}$$

де μ - коефіцієнт інерційності, t - номер поточної ітерації.

Реалізація Backpropogation

Алгоритм навчання багатошарової НМ за допомогою процедури зворотного поширення помилки :

0. Зформувати навчальну вибірку $\{\mathbf{x}_p, \mathbf{d}_p\}$, p=1…P

   ініціалізувати початкові значення вагових параметрів

$$w_{ij} = 0.01*rand(-1,+1).$$

1. Подати на входи мережі один прикладів навчальної вибірки і в режимі звичайного функціонування НМ, коли сигнали поширюються від входів до виходів, розрахувати значення останніх. Нагадаємо, що

$$s_j^{(q)} = \sum_{i=0}^{n_{q-1}} y_i^{(q-1)} \cdot w_{ij}^{(q)}$$

де n_{q-1} - число нейронів в шарі q-1 з урахуванням нейрона з постійним вхідним станом +1, що задає зміщення;

$$y_j^{(q)} = F(s_j^{(q)}),$$

де F() – сигмоїд

$$y_i^{(0)} = x_i$$

де x_i - і -та компонента вектору вхідного образу.

2. Розрахувати для вихідного шару по формулі

$$\delta_j^{(Q)} = (y_j^{(Q)} - d_j) \cdot F'(s_j^{(Q)})$$

Реалізація Backpropogation

3. Розрахувати по формулах відповідні для усіх інших шарів, q=(Q – 1),..1.

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(q+1)} \right] \cdot F'(s_j^{(q)})$$

4. Розрахувати по формулі

$$\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}$$

зміни ваг шару q.

5. Скоригувати усі ваги в НМ

$$w_{ij}^{(q)}(t) = w_{ij}^{(q)}(t-1) + \Delta w_{ij}^{(q)}(t)$$

6. Перейти на крок 1 до вичерпання всіх прикладів навчальної вибірки.

7. Якщо функція помилки мережі E для всіх M прикладів навчальної вибірки істотна, перейти на крок 1. Інакше - кінець.

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{n_Q} \Delta_{jp}^2$$

де $\Delta_{jp} = (y_j^{(Q)} - d_j)$

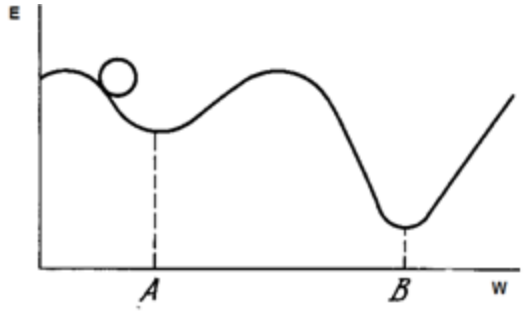
Мережі на кроці 1 поперемінно у випадковому порядку пред'являються усі тренувальні приклади.

Налаштування НМ для вирішення задач

- Локальні мінімуми.
- Параліч НМ
- Розмір кроку.
- Тимчасова нестійкість.
- Попередня обробка вхідних даних.
- Згасання помилки у алгоритмі Backprop.

Локальні мінімуми

BackProp використовує різновид градієнтного спуску, тобто здійснює спуск вниз по поверхні помилки, безперервно налаштовуючи ваги у напрямі до мінімуму. Поверхня помилки складної мережі дуже порізана і складається з пагорбів, долин, складок і ярів в просторі високої розмірності.



Мережа може потрапити в локальний мінімум (неглибоку долину), коли поруч є набагато глибший мінімум. У точці локального мінімуму усі напрями ведуть вгору, і мережа нездатна з нього вибратися.

Статистичні методи навчання можуть допомогти уникнути цієї пастки, але вони повільні. Застосовується також метод «струшування», тобто зміна деяких вагових параметрів для виходу з локального мінімуму і продовження процесу навчання.

Параліч мережі

В процесі навчання мережі значення ваг можуть в результаті корекції стати дуже великими величинами. Це може привести до того, що усі або більшість нейронів функціонуватимуть при дуже великих значеннях S, в області, де похідна функції дуже мала. Оскільки помилка пропорційна цій похідній, то процес навчання може практично завмерти.

Різні евристики використовувалися для оберігання від паралічу або для відновлення після нього, наприклад тимчасове зменшення параметра сигмоїду **a**, оскільки причиною паралічу НМ часто є нульове значення похідної від активаційної функції $F'(S) = \boldsymbol{a}(1-F^2(S))$ для великих значень S і це зупиняє процес навчання. Після відновлення навчання значення параметра ***a*** повертається.

Розмір кроку

Уважний розбір збіжності показує, що корекції ваг передбачаються нескінченно малими. Ясно, що це нездійсненно на практиці, оскільки веде до нескінченного часу навчання.

Розмір кроку **η** повинен братися скінченним, і в цьому питанні доводиться спиратися тільки на досвід.

Якщо розмір кроку **η** дуже малий, то збіжність занадто повільна, якщо ж дуже великий, то може виникнути параліч або постійна нестійкість.

Автоматичне налаштування кроку може здійснюватись по формулі: $\eta = \eta_0/(1+b*t)$

t - номер епохи навчання,
η_0 і **b –** обираються експериментально.

Тимчасова нестійкість

Якщо мережа вчиться розпізнавати букви, то немає сенсу учити "Б", якщо при цьому забувається "А". Процес навчання має бути таким, щоб мережа навчалася на усій навчальній множині без пропусків того, що вже вивчене.

Необхідні зміни ваг повинні обчислюватися **на усій множині прикладів,** а це вимагає додаткової пам'яті; після ряду таких навчальних циклів вони зійдуться до мінімальної помилки.

Цей метод може виявитися даремним, якщо мережа знаходиться в зовнішньому середовищі, що постійно міняється, так що другий раз один і той же вектор може вже не повторитися. В цьому випадку процес навчання може ніколи не зійтися, безцільно блукаючи або сильно осцилюючи. У цьому сенсі зворотне поширення не схоже на біологічні системи.

Попередня обробка вхідних даних.

Процес навчання можна суттєво покращити попередньою обробкою навчальної вибірки.

Приклади для кожного класу повинні бути типовими проте різними. При майже однакових прикладах спостерігається синдром «перенавчання», коли нейронна мережа у процесі функціонування добре розпізнає образи близькі до прикладів і не розпізнають ся віддалені образи.

Разом з тим, дуже «зашумлені» приклади у навчальній вибірці не дозволяють навчити НС взагалі.

Суттєво може прискорити навчання попередня нормалізація навчальної вибірки, наприклад методом природної нормалізації.

Функції активації

- Логістичний сигмоїд $F(s)=\frac{1}{1+e^{-a\cdot s}}$ $F'(s)=a\cdot F(s)\cdot(1-F^2(s))$
- Гіперболічний тангенс $F(s)=th(as)$ $F'(s)=a(1-F^2(s))$
- SoftSign $F(s)=\frac{a\cdot s}{1+|a\cdot s|}$ $F'(s)=\frac{1}{(1+|a\cdot s|)^2}$
- SoftPlus $F(s)=\ln(1+e^{a\cdot s})$ $F'(s)=\frac{a}{1+e^{-a\cdot s}}$
- ReLU(rectified linear units) $F(s)=\max(0,s)$

$$F'(s)=\max(0,1)$$

Згасання помилки у алгоритмі BackProp.

Спостерігається згасання помилки при зворотному розповсюдженні, що зупиняє процес навчання.

Виходом можуть бути застосування інших методів навчання, зокрема, чисельних методів оптимізації чи обмеженої машини Больцмана.

Лабораторна робота № 2

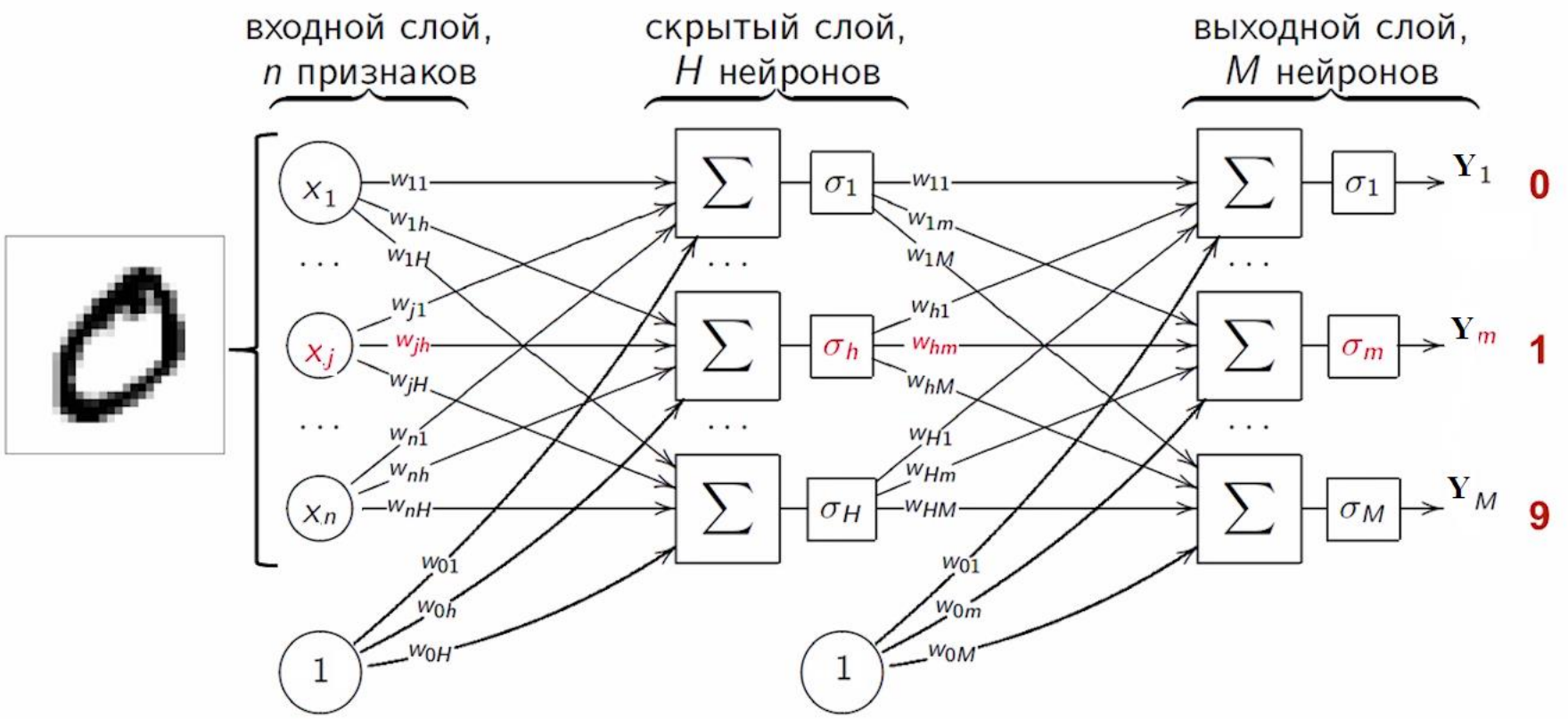
Створити нейро комп’ютерну систему розпізнавання рукописних символів, використавши алгоритм зворотного поширення помилки для навчання багатошарової НМ.

Алгоритм лабораторної роботи №3

1. Створити графічний інтерфейс для введення прикладів і розпізнавання символів.

2. Створити навчальну вибірку рукописних символів або використати стандартний набір MNIST з Internet

Лабораторна робота № 2



Лабораторна робота № 2

3.Спроектувати нейронну мережу, задаючись на вході достатньою кількістю рецепторів для відображення символів через мега пікселі двовимірної сітки, а на виході НС достатньою кількістю нейронів для кодування символів (кількість різних символів <=2^m,

де m – кількість нейронів на виході).

4.Кількість нейронів у проміжних шарах визначити експериментально.

5.Навчити НМ згідно з алгоритмом BackProp.

6. Протестувати роботу НС на контрольній частині навчальної вибірки і при необхідності донавчити НС.

Контрастування НМ

Виробництво явних знань з накопичених даних - проблема, яка набагато старша чим комп'ютери. Навчені нейронні мережі можуть видавати з даних приховані знання: створюється навичка передбачення, класифікації, розпізнавання образів, але її логічна структура зазвичай залишається прихованою від користувача.

Проблема прояву (контрастування) цієї прихованої логічної структури вирішується шляхом приведення нейронних мереж до спеціального "логічно прозорого" розрідженого виду.

Контрастування НМ

Досліджуються два питання:

- Скільки нейронів потрібно для вирішення задачі?

- Яка має бути структура нейронної мережі?

Об'єднуючи ці два питання, ми отримуємо третє:

- Як зробити роботу нейронної мережі зрозумілою для користувача (логічно прозорою) і які вигоди може принести таке розуміння?

Скільки нейронів треба використовувати?

При відповіді на це питання існує дві протилежні точки зору. Одна з них стверджує, що чим більше нейронів використовувати, тим більше надійна мережа вийде.

Прибічники цієї позиції посилаються на приклад людського мозку. Дійсно, ніж більше нейронів, тим більше число зв'язків між ними, і тим більше складні завдання здатна вирішити нейронна мережа. Крім того, якщо використовувати свідомо більше число нейронів, чим необхідно для вирішення завдання, то нейронна мережа точно навчиться.

Якщо починати з невеликого числа нейронів, то мережа може виявитися нездатною навчитися рішенню задачі, і увесь процес доведеться повторювати спочатку з більшим числом нейронів.

Скільки нейронів треба використовувати?

Другий підхід визначає потрібне число нейронів як мінімально необхідне.

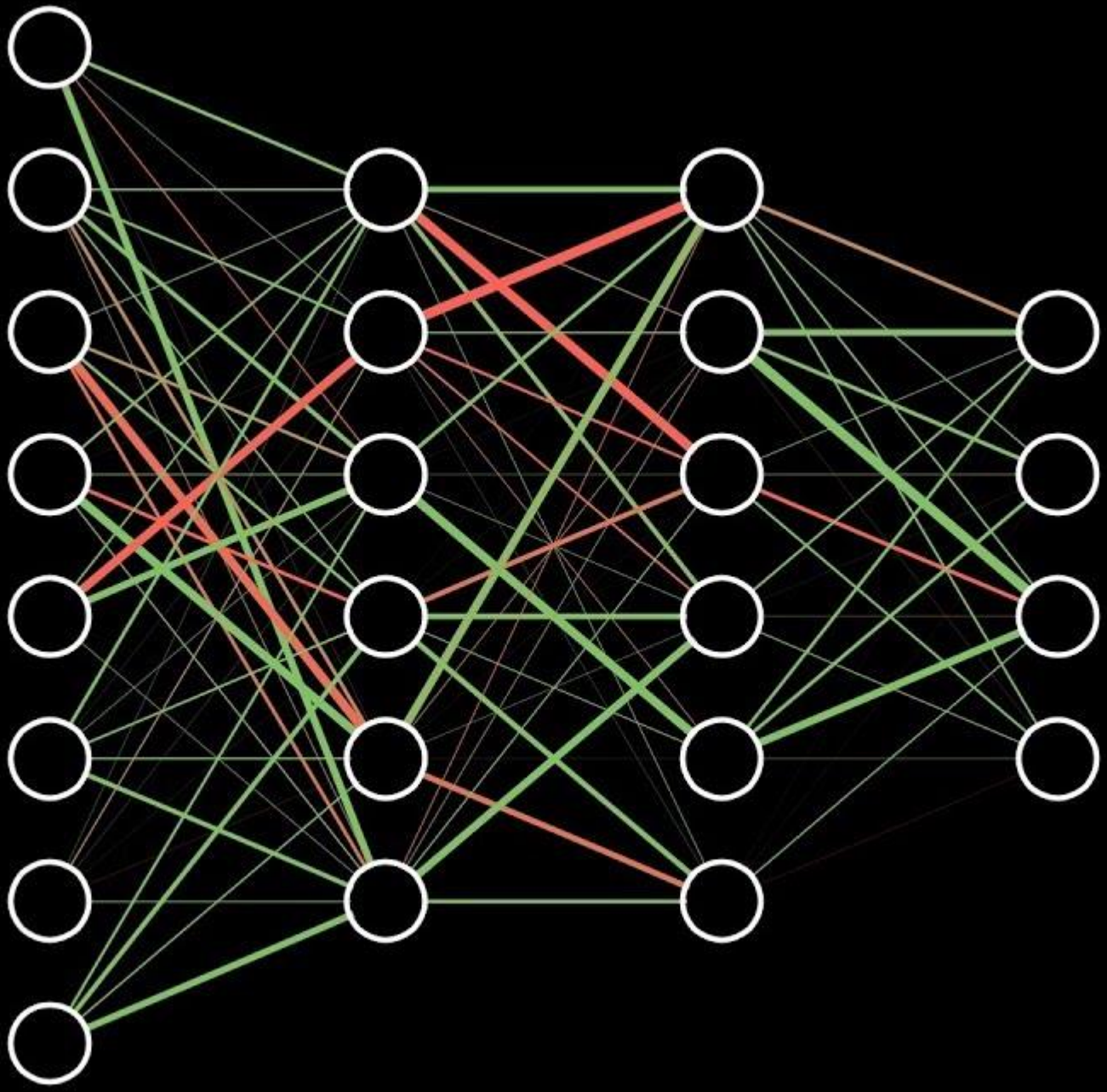
Основним недоліком є те, що це, мінімально необхідне число, заздалегідь невідоме, а процедура його визначення шляхом поступового нарощування числа нейронів дуже трудомістка.

Спираючись на досвід роботи в області медичної діагностики , космічній навігації і психології можна відмітити, що в усіх цих завданнях жодного разу не було потрібно більше декількох десятків нейронів.

Скільки нейронів треба використовувати?

Істина, як завжди буває в таких випадках, лежить посередині: треба вибирати число нейронів більшим чим необхідно, але не набагато. Це можна здійснити шляхом подвоєння числа нейронів в мережі після кожної невдалої спроби навчання.

Проте існує надійніший спосіб оцінки мінімального числа нейронів - використання процедури контрастування. Крім того, процедура контрастування дозволяє відповісти і на друге питання: яка має бути структура мережі.



Контрастування на основі оцінки

Розглянемо мережу, що правильно вирішує усі приклади повчальної великої кількості. Позначимо через w_p, $p=1,\ldots,n$ ваги усіх зв'язків. При зворотному функціонуванні мережі за принципом двоїстості або методом зворотного поширення помилки мережа обчислює вектор градієнта функції оцінки E по вагах зв'язків w_k :

$$\nabla E = \left\{ \frac{\partial E}{\partial w_k} \right\}_{k=1}^{K}$$

Нехай w^0_p- поточний набір ваг зв'язків, а E^0 - помилка поточного прикладу. Тоді в лінійному наближенні можна записати функцію оцінки E в точці w_k як :

$$\mathrm{E}(\mathrm{w}_\mathrm{k}) = E^0 + \sum_{k=1}^{K} \frac{\partial E}{\partial w_k} (w_k - w_k^0)$$

Використовуючи це наближення можна оцінити зміну оцінки при заміні w^0_k на $w_k{}^*$ як:

$$\chi(k, p) = \left| \frac{\partial E}{\partial w_k} \right| \cdot \left| w_k^* - w_k^0 \right|$$

де p - номер прикладу навчальної вибірки, для якого були обчислені оцінка і градієнт.

Контрастування на основі оцінки

Величину $\chi(k, p)$ називатимемо показником чутливості до заміни для прикладу p . Далі необхідно обчислити показник чутливості, незалежний від номера прикладу. Для цього можна скористатися будь-якою нормою. Зазвичай використовується рівномірна норма (максимум модуля):

$$\chi(k) = \max_{p} \chi(k, p)$$

Уміючи обчислювати показники чутливості, можна приступати до процедури контрастування.

Приведемо простий варіант цієї процедури :

1. Обчислюємо показники чутливості для кожної ваги w_k .
2. Знаходимо мінімальний серед показників чутливості для різних прикладів навчальної вибірки.
3. Замінюємо відповідний цьому показнику чутливості вагу на w_k^0 , і виключаємо його з процедури навчання.
4. Пред'явимо мережі усі приклади навчальної вибірки. Якщо мережа не допустила жодної помилки, то переходимо до другого кроку процедури.
5. Намагаємося навчити відконтрастовану мережу. Якщо мережа навчилася безпомилковому рішенню задачі, то переходимо до першого кроку процедури, інакше переходимо до шостого кроку.
6. Відновлюємо мережу в стан до останнього виконання третього кроку. Якщо в ході виконання кроків з другого по п'ятий була відконтрастована хоч би одна вага, (число навчених ваг змінилося), то переходимо до першого кроку. Якщо жодна вага не була відконтрастована, то отримана мінімальна мережа.

Контрастування на основі оцінки

Можливе використання різних узагальнень цієї процедури. Наприклад, контрастувати за один крок процедури не одну вагу, а задане користувачем число ваг.

Найбільш радикальна процедура полягає в контрастуванні половини ваг зв'язків. Якщо половину ваг відконтрастувати не вдається, то намагаємося відконтрастувати чверть і так далі.

Відмітимо, що при описаному методі обчислення показників чутливості, передбачається можливим обчислення функції оцінки і проведення процедури навчання мережі, а також передбачається відомою навчальна множина.

Контрастування без погіршення

Нехай нам дана тільки навчена нейронна мережа і навчальна множина. Допустимо, що вид функції оцінки і процедура навчання нейронної мережі невідомі. В цьому випадку так само можливе контрастування мережі. Припустимо, що ця мережа ідеально вирішує задачу. Тоді нам необхідно так відконтрастувати ваги зв'язків, щоб вихідні сигнали мережі при рішенні усіх завдань змінилися не більше ніж на задану величину.

В цьому випадку контрастування ваг виконується по нейронах. На вході кожного нейрона стоїть адаптивний суматор, який підсумовує вхідні сигнали нейрона, помножені на відповідні ваги зв'язків. Для нейрона найменш чутливою буде та вага, яка при рішенні прикладу дасть найменший вклад в суму. Позначивши через x_k^p вхідні сигнали даного нейрона при рішенні p-го прикладу отримуємо формулу для показника чутливості ваг:

$$\chi(k, p) = \left| x_k^p \right| \cdot \left| w_k^* - w_k^0 \right|$$

Аналогічно раніше розглянутому отримуємо:

$$\chi(k) = \max_p \left| x_k^p \right| \cdot \left| w_k^* - w_k^0 \right|$$

В самій процедурі контрастування є тільки одна відмінність - замість перевірки на наявність помилок при пред'явленні усіх прикладів перевіряється, що нові вихідні сигнали мережі відрізняються від первинних не більше ніж на задану величину.

Логічно прозорі нейронні мережі

Одним з основних недоліків нейронних мереж, з точки зору багатьох користувачів, є те, що нейронна мережа вирішує задачу, але не може розповісти як. Іншими словами з навченої нейронної мережі не можна витягнути алгоритм рішення задачі. Проте спеціальним чином побудована процедура контрастування дозволяє вирішити і це завдання.

Задамося класом мереж, які вважатимемо логічно прозорими (тобто такими, які вирішують задачу зрозумілим для нас способом, для якого легко сформулювати словесне описи у вигляді явного алгоритму). Наприклад зажадаємо, щоб усі нейрони мали не більше трьох вхідних сигналів.

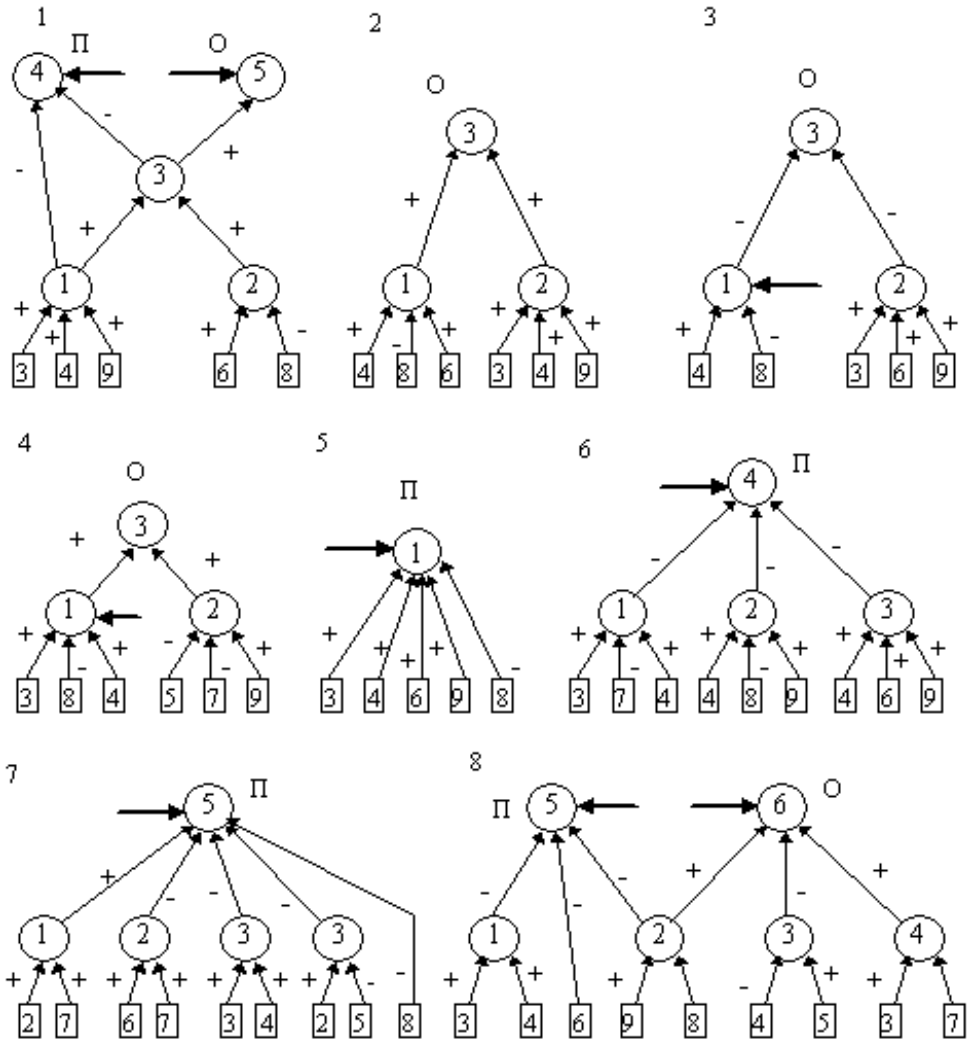
.

Логічно прозорі нейронні мережі

Задамося нейронною мережею у якої усі вхідні сигнали подаються на усі нейрони вхідного шару, а усі нейрони кожного наступного шару приймають вихідні сигнали усіх нейронів попереднього шару. Навчимо мережу безпомилковому рішенню задачі.

Після цього виконуємо контрастування у декілька етапів. На першому етапі контрастуватимемо тільки ваги зв'язків нейронів вхідного шару. Якщо після контрастування у деяких нейронів залишилися більше трьох вхідних сигналів, то збільшимо число вхідних нейронів. Потім аналогічну процедуру виконаємо по черзі для усіх інших шарів.

Після завершення описаної процедури буде отримана логічно прозора мережа. Можна зробити додаткове контрастування мережі, щоб отримати мінімальну мережу



Якщо під логічно прозорими мережами розуміти мережі, у яких кожен нейрон має не більше трьох входів, то усі мережі окрім п'ятої і сьомої є логічно прозорими. П'ята і сьома мережі демонструють той факт, що мінімальність

мережі не спричиняє логічної прозорості.

Логічно прозорі нейронні мережі

Технологія отримання явних знань з даних за допомогою навчених нейронних мереж виглядає досить просто і начебто не викликає проблем - необхідно її просто реалізовувати і користуватися.

Перший етап: навчаємо нейронну мережу вирішувати базову задачу. Зазвичай базовим є завдання розпізнавання, пророцтва (як в попередньому розділі) і тому подібне. В більшості випадків її можна трактувати як завдання про заповнення пропусків в даних. Такими пропусками є і ім'я образу при розпізнаванні, і номер класу, і результат прогнозу, та ін.

Другий етап: за допомогою аналізу показників значущості, контрастування і доучування (усе це застосовується, найчастіше, неодноразово) приводимо нейронну мережу до логічно прозорого виду - так, щоб отриману мережу можна було "прочитати".

Отриманий результат неоднозначний - якщо стартувати з іншої початкової карти, то можна отримати іншу логічно прозору структуру. Кожній базі даних відповідає декілька варіантів явних знань. Можна вважати це недоліком технології, але ми вважаємо, що, навпаки, технологія, що дає єдиний варіант явних знань, недостовірна, а неєдиність результату є фундаментальною властивістю виробництва явних знань з даних.

Логічно прозорі нейронні мережі

Другий етап: за допомогою аналізу показників значущості, контрастування і доучування (усе це застосовується, найчастіше, неодноразово) приводимо нейронну мережу до логічно прозорого виду - так, щоб отриману мережу можна було "прочитати".

Отриманий результат неоднозначний - якщо стартувати з іншої початкової карти, то можна отримати іншу логічно прозору структуру. Кожній базі даних відповідає декілька варіантів явних знань.

Можна вважати це недоліком технології, але ми вважаємо, що, навпаки, технологія, що дає єдиний варіант явних знань, недостовірна, а не єдиність результату є фундаментальною властивістю виробництва явних знань із даних.

Лабораторна робота № 4

1. Створити нейромережу прогнозування результатів президентських виборів США у 2020 р. для заданої навчальної вибірки.
2. Максимально відконтрастувати двошарову нейронну мережу
3. Процес контрастування зобразити графічно для заданої НМ.

Лабораторна робота № 4

Питання для тестування:

1. Правляча партія була при владі більше одного терміну?
2. Правляча партія отримала більше 50% голосів на минулих виборах?
3. У рік виборів була активна третя партія?
4. Була серйозна конкуренція при висуненні від правлячої партії?
5. Кандидат від правлячої партії був президентом у рік виборів?
6. Чи був рік виборів часом спаду або депресії?
7. Чи був зростання середнього національного валового продукту на душу населення більше 2.1%?
8. Чи справив правлячий президент істотні зміни в політиці?
9. Під час правління були істотні соціальні хвилювання?
10. Адміністрація правлячої партії винна в серйозній помилці чи скандалі?
11. Кандидат від правлячої партії - національний герой?
12. Кандидат від опозиційної партії - національний герой?

Питання	1	2	3	4	5	6	7	8	9	10	11	12	перемогла
1860	1	0	1	1	0	0	1	0	1	0	0	0	опозиційна
1864	0	0	0	0	1	0	0	1	1	0	0	0	правляча
1868	1	1	0	0	0	0	1	1	1	0	1	0	правляча
1872	1	1	0	0	1	0	1	0	0	0	1	0	правляча
1876	1	1	0	1	0	1	0	0	0	1	0	0	опозиційна
1880	1	0	0	1	0	0	1	1	0	0	0	0	правляча
1884	1	0	0	1	0	0	1	0	1	0	1	0	опозиційна
1888	0	0	0	0	1	0	0	0	0	0	0	0	правляча
1892	0	0	1	0	1	0	0	1	1	0	0	1	опозиційна
1896	0	0	0	1	0	1	0	1	1	0	1	0	опозиційна
1904	1	1	0	0	1	0	0	0	0	0	1	0	правляча
1908	1	1	0	0	0	0	0	1	0	0	0	1	правляча
1912	1	1	1	1	1	0	1	0	0	0	0	0	опозиційна
1916	0	0	0	0	1	0	0	1	0	0	0	0	правляча
1920	1	0	0	1	0	0	0	1	1	0	0	0	опозиційна
1924	0	1	1	0	1	0	1	1	0	1	0	0	правляча
1928	1	1	0	0	0	0	1	0	0	0	0	0	правляча
1932	1	1	0	0	1	1	0	0	1	0	0	1	опозиційна
1936	0	1	0	0	1	1	1	1	0	0	1	0	правляча
1940	1	1	0	0	1	1	1	1	0	0	1	0	правляча
1944	1	1	0	0	1	0	1	1	0	0	1	0	правляча
1948	1	1	1	0	1	0	0	1	0	0	0	0	правляча
1952	1	0	0	1	0	0	1	0	0	1	0	1	опозиційна
1956	0	1	0	0	1	0	0	0	0	0	1	0	правляча
1960	1	1	0	0	0	1	0	0	0	0	0	1	опозиційна
1964	0	0	0	0	1	0	1	0	0	0	0	0	правляча
1968	1	1	1	1	0	0	1	1	1	0	0	0	опозиційна
1972	0	0	0	0	1	0	1	1	1	0	0	0	правляча
1976	1	1	0	1	1	0	0	0	0	1	0	0	опозиційна
1980	0	0	1	1	1	1	0	0	0	1	0	1	опозиційна
1992	0	1	0	0	1	0	1	0	0	0	0	1	правляча
1996	1	1	0	0	1	0	0	1	0	0	0	0	правляча
2000	1	1	0	0	0	0	1	0	0	1	0	0	правляча
2004	1	1	1	0	1	0	1	0	0	1	0	0	правляча
2008	1	1	1	0	1	0	1	0	0	1	0	0	опозиційна
2012	1	1	0	0	1	0	1	0	0	0	0	0	правляча

Стохастичні методи навчання

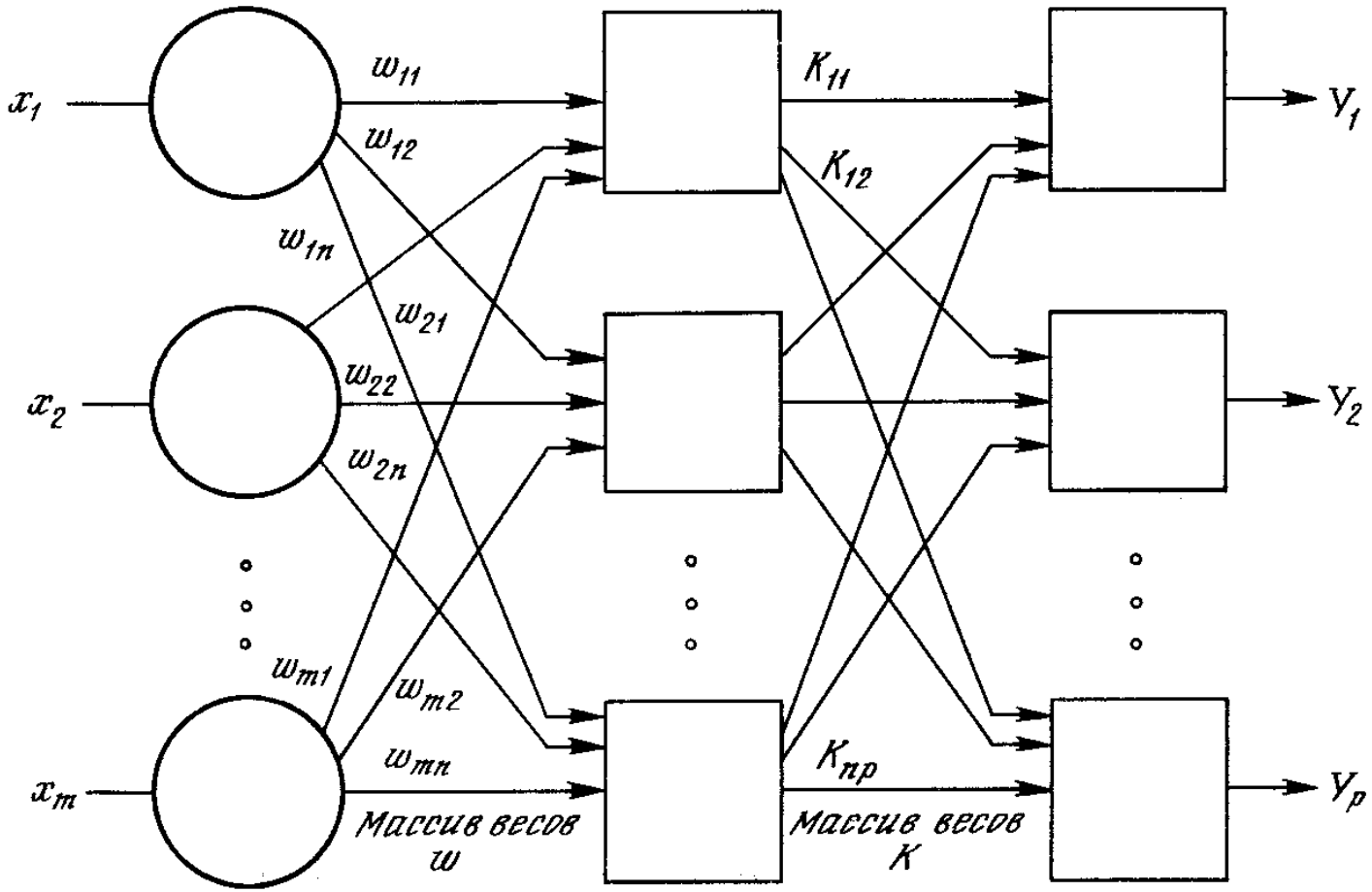
Стохастичні методи корисні як для навчання штучних нейронних мереж, так і для отримання виходу від вже навченої мережі. Стохастичні методи навчання приносять велику користь, дозволяючи виключати локальні мінімуми в процесі навчання. Але з ними також пов'язаний ряд проблем.

Штучна нейронна мережа навчається за допомогою деякого процесу,що модифікує її ваги. Якщо навчання успішне, то пред'явлення мережі безлічі вхідних сигналів призводить до появи бажаної множини вихідних сигналів. Є два класи навчальних методів : детерміністський і стохастичний.

Стохастичні методи навчання

Детерміністський метод навчання крок за кроком здійснює процедуру корекції ваг мережі, засновану на використанні їх поточних значень, а також величин входів, фактичних виходів і бажаних виходів. Навчання персептрона є прикладом подібного детерміністського підходу.

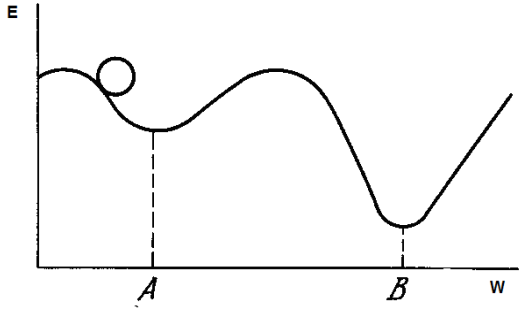
Стохастичні методи навчання виконують псевдовипадкові зміни величин ваг, зберігаючи ті зміни, які ведуть до поліпшень. Щоб побачити, як це може бути зроблено, розглянемо рис., на якому зображена типова мережа, в якій нейрони сполучені за допомогою ваг. Вихід нейрона є тут зваженою сумою його входів, яка, перетворена за допомогою нелінійної функції.


x_1
x_2
x_m
w_{11}
w_{12}
w_{1n}
w_{21}
w_{22}
w_{2n}
w_{m1}
w_{m2}
w_{mn}
Массив весов
w
K_{11}
K_{12}
K_{np}
Массив весов
K
Y_1
Y_2
Y_p

Процедура стохастичного навчання

1. Вибрати вагу випадковим чином і підкоригувати її на невелике випадкове Пред'явити безліч входів і вичислити виходи, що виходять.
2. Порівняти ці виходи з бажаними виходами і вичислити величину різниці між ними. Загальноприйнятий метод полягає в знаходженні різниці між фактичним і бажаним виходами для кожного елементу навченої пари, зведення різниць в квадрат і знаходження суми цих квадратів. Метою навчання є мінімізація цієї різниці, часто званою цільовою функцією.
3. Вибрати вагу випадковим чином і підкоригувати її на невелике випадкове значення. Якщо корекція допомагає (зменшує цільову функцію), то зберегти її, інакше повернутися до первинного значення ваги.
4. Повторювати кроки з 1 до 3 до тих пір, поки мережа не буде навчена достатньою мірою.

Пастки локальних мінімумів докучають усім алгоритмам навчання, заснованим на пошуку мінімуму, включаючи персептрон і мережі зворотного поширення, і представляють серйозну і широко поширену трудність, якої часто не помічають.



Стохастичні методи дозволяють розв'язати цю проблему. Стратегія корекції ваг, що змушує ваги набувати значення глобального оптимуму в точці B, можлива.

- Якщо коробку сильно потрясти в горизонтальному напрямі, то кулька швидко перекочуватиметься від одного краю до іншого. Ніде не затримуючись, в кожен момент кулька з рівною імовірністю знаходитиметься в будь-якій точці поверхні.
- Якщо поступово зменшувати силу струшування, то буде досягнута умова, при якій кулька на короткий час "застряватиме" в точці В. При ще слабкішому струшуванні кулька на короткий час зупинятиметься як в точці А, так і в точці В.
- При безперервному зменшенні сили струшування буде досягнута критична точка, коли сила струшування достатня для переміщення кульки з точки А в точку В, але недостатня для того, щоб кулька могла видертися з В в А.
- Таким чином, остаточно кулька зупиниться в точці глобального мінімуму, коли амплітуда струшування зменшиться до нуля.

- Штучні нейронні мережі можуть навчатися по суті тим же самим чином за допомогою випадкової корекції ваг. Спочатку робляться великі випадкові корекції із збереженням тільки тих змін ваг, які зменшують цільову функцію. Потім середній розмір кроку поступово зменшується, і глобальний мінімум врешті-решт досягається.
- Це дуже нагадує відпал металу, тому для її опису часто використовують термін "імітація відпалу". У металі, нагрітому до температури, що перевищує його точку плавлення, атоми знаходяться в сильному безладному русі.
- Як і в усіх фізичних системах, атоми прагнуть до стану мінімуму енергії (єдиному кристалу в даному випадку), але при високих температурах енергія атомних рухів перешкоджає цьому. В процесі поступового охолодження металу виникають усе більш низько енергетичні стани, поки врешті-решт не буде досягнуто найнижче з можливих станів, глобальний мінімум.
- В процесі відпалу розподіл енергетичних рівнів описується наступним співвідношенням:
  - $P(e) = \exp(-e/kT)$
- де *P(e)* - вірогідність того, що система знаходиться в змозі з енергією e; k - постійна Больцмана; T - температура за шкалою Кельвіна.
- При високих температурах P(e) наближається до одиниці для усіх енергетичних станів. Таким чином, високо енергетичний стан майже так же ймовірний, як і низько енергетичний. У міру зменшення температури вірогідність високо енергетичних станів зменшується в порівнянні з низько енергетичними. При наближенні температури до нуля стає дуже маловірогідним, щоб система знаходилася у високо енергетичному стані.

Навчання Больцмана.

Цей стохастичний метод безпосередньо застосований до навчання штучних нейронних мереж :

Визначити змінну T, що представляє штучну температуру. Надати T велике початкове значення.

Пред'явити мережі безліч входів і вичислити виходи і цільову функцію.

Дати випадкову зміну вазі і перерахувати вихід мережі і зміну цільової функції відповідно до зробленої зміни ваги.

Якщо цільова функція зменшилася (покращала), то зберегти зміну ваги.

Якщо зміна ваги призводить до збільшення цільової функції, то вірогідність збереження цієї зміни обчислюється за допомогою розподілу Больцмана :

$$P(c) = \exp(-c/kT)$$

де P(c) - вірогідність зміни з в цільовій функції; k - константа, аналогічна константі Больцмана, вибирана залежно від завдання; T - штучна температура.

Вибирається випадкове число r з рівномірного розподілу від нуля до одиниці. Якщо P(c) більше, ніж r, то зміна зберігається, інакше величина ваги повертається до попереднього значення.

Це дозволяє системі робити випадковий крок в напрямі, що псує цільову функцію, дозволяючи їй тим самим вириватися з локальних мінімумів, де будь-який малий крок збільшує цільову функцію.

Для завершення больцманівского навчання повторюють кроки 3 і 4 для кожної з ваг мережі, поступово зменшуючи температуру T, поки не буде досягнуто допустимо низьке значення цільової функції. У цей момент пред'являється інший вхідний вектор і процес навчання повторюється.

Мережа навчається на усіх векторах навчальної множини, з можливим повторенням, поки цільова функція не стане допустимою для усіх них.

Величина випадкової зміни ваги на кроці 3 може визначатися різними способами. Наприклад, подібно до теплової системи вагова зміна w може вибиратися відповідно до розподілу Гауса :

$P(w) = \exp(-w2/T2)$

де P(w) - вірогідність зміни ваги на величину w, T - штучна температура.

Такий вибір зміни ваги призводить до системи, аналогічно.

Оскільки потрібна величина зміни ваги Δw, а не вірогідність зміни ваги, що має величину w, то метод Монте-Карло може бути використаний таким чином:

1. Знайти кумулятивну вірогідність, відповідну P(w). Це є інтеграл від P(w) в межах від 0 до w. Оскільки в даному випадку P(w) не може проінтегрувати аналітично, вона повинна інтегруватися чисельно, а результат потрібне затабулировать.
2. Вибрати випадкове число з рівномірного розподілу на інтервалі (0,1). Використовуючи цю величину як значення P(w}, знайти в таблиці відповідне значення для величини зміни ваги.

Властивості машини Больцмана широко вивчалися. У роботі показано, що швидкість зменшення температури має бути обернено пропорційна логарифму часу, щоб була досягнута збіжність до глобального мінімуму. Швидкість охолодження в такій системі виражається таким чином:

$$T(t) = \frac{T_0}{\log(1+t)}$$

де T(t) - штучна температура як функція часу; T0 - початкова штучна температура; t - штучний час.

Цей результат, що розчаровує, передбачає дуже повільну швидкість охолодження (і ці обчислення). Цей вивід підтвердився експериментально. Машини Больцмана часто вимагають для навчання дуже великого ресурсу часу.

Навчання Коші

У цьому методі при обчисленні величини кроку розподіл Больцмана замінюється на розподіл Коші. Розподіл Коші має, як показано на рис. , довші "хвости", збільшуючи тим самим вірогідність великих кроків.

Насправді розподіл Коші має нескінченну (невизначену) дисперсію. За допомогою такої простої зміни максимальна швидкість зменшення температури стає обернено пропорційній лінійній величині, а не логарифму, як для алгоритму навчання Больцмана. Це різко зменшує час навчання. Цей зв'язок може бути виражений таким чином:

$$T(t) = \frac{T_0}{1+t}$$

Розподіл Коши має вигляд

$$P(x) = \frac{T(t)}{T(t)^2 + x^2}$$

де P(x) є вірогідність кроку величини x.

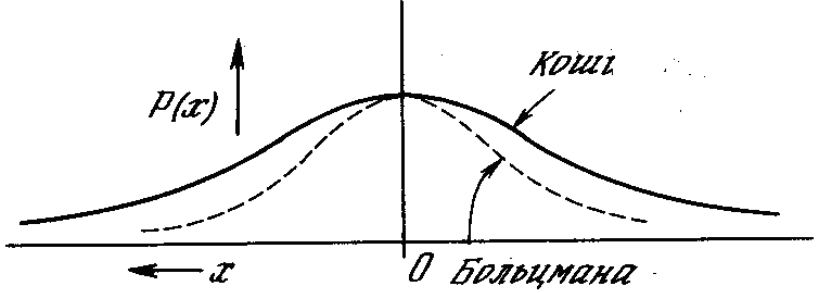


Рис. Розподіл Коші і розподіл Больцмана

У рівнянні P(x) можна про інтегрувати стандартними методами. Вирішуючи відносно x, отримуємо

$$xc = (T(t) tg(P(x)),$$

де xc - зміна ваги.

Тепер застосування методу Монте-Карло стає дуже простим. Для знаходження x в цьому випадку вибирається випадкове число з рівномірного розподілу на відкритому інтервалі $(-\pi/2, \pi/2)$ (необхідно обмежити функцію тангенса). Воно підставляється у формулу як P(x), і за допомогою поточної температури обчислюється величина кроку.

Обме́жена маши́на Бо́льцмана

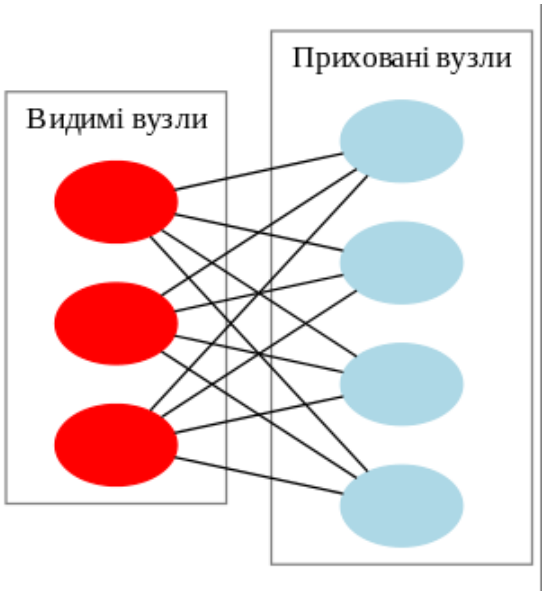
(ОМБ, англ. *restricted Boltzmann machine, RBM*) — це породжувальна стохастична штучна нейронна мережа, здатна навчатися розподілу ймовірностей над набором її входів.

ОМБ було спочатку винайдено під назвою **Гармоніум** (англ. *Harmonium* — фісгармонія) Полом Смоленським у1986 році, а популярності вони набули після винайдення Джефрі Хінтоном зі співавторами у середині 2000-х років алгоритмів швидкого навчання для них.

ОМБ знайшли застосування у зниженн розмірності, класифікації, колаборативній фільтрації, навчанні ознак та тематичному моделювані. Їх може бути треновано як керованим, так і спонтанним чином, в залежності від завдання.

Обме́жена маши́на Бо́льцмана

Як випливає з їхньої назви, ОМБ є варіантом машин Больцмана, з тим обмеженням, що їхні нейрони мусять формувати двочастковий граф: пара вузлів з кожної з двох груп вузлів (що, як правило, називають «видимим» та «прихованим» вузлами відповідно) можуть мати симетричне з'єднання між ними, але з'єднань між вузлами в межах групи не існує.



Обме́жена маши́на Бо́льцмана

Обмежені машини Больцмана можуть також застосовуватися в мережах глибинного навчання. Зокрема, глибинні мережі переконань можуть утворюватися «складанням» ОМБ та, можливо, тонким налаштуванням отримуваної в результаті глибинної мережі за допомогою градієнтного спуску та зворотного поширення.

На противагу, «необмежені» машини Больцмана можуть мати з'єднання між прихованими вузлами. Це обмеження уможливлює ефективніші алгоритми тренування, ніж доступні для загального класу машин Больцмана, зокрема, алгоритм **порівня́льної розбі́жності** (англ. *contrastive divergence*) на основі градієнтного спуску.

Обме́жена маши́на Бо́льцмана

Стандартний тип ОМБ має двійковозначні (булеві/бернуллієві) приховані та видимі вузли, і складається з матриці вагових коефіцієнтів W = ($w_{i,j}$) (розміру *m*×*n*), пов'язаної зі з'єднанням між прихованим вузлом h_j та видимим вузлом v_i, а також вагових коефіцієнтів упереджень (зсувів) a_i для видимих вузлів, і b_j для прихованих вузлів.

З урахуванням цього, *енергія* конфігурації (пари булевих векторів) (*v*,*h*) визначається як

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

або, в матричному записі,

$$E(v,h) = -a^{\mathrm{T}} v - b^{\mathrm{T}} h - v^{\mathrm{T}} W h$$

Ця функція енергії є аналогічною до функції енергії мережі Хопфілда. Як і в загальних машинах Больцмана, розподіли ймовірності над прихованими та/або видимими векторами визначаються в термінах функції енергії:

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)}$$

Обме́жена маши́на Бо́льцмана

де Z є статистичною сумою, визначеною як сума над усіма можливими конфігураціями (іншими словами, просто нормувальна стала для забезпечення того, щоби розподіл імовірності давав у сумі 1). Аналогічно, (відособлена) ймовірність видимого (вхідного) вектора булевих значень є сумою над усіма можливими конфігураціями прихованого шару:

$$P(v) = \frac{1}{Z} \sum_{h} e^{-E(v,h)}$$

Оскільки ОМБ має форму двочасткового графу, без з'єднань усередині шарів, активації прихованих вузлів є взаємно незалежними для заданих активацій видимих вузлів, і навпаки, активації видимих вузлів є взаємно незалежними для заданих активацій прихованих вузлів.

Обме́жена маши́на Бо́льцмана

Тобто, для m видимих вузлів та n прихованих вузлів умовною ймовірністю конфігурації видимих вузлів *v* для заданої конфігурації прихованих вузлів *h* є

$$P(v \mid h) = \prod_{i=1,m} P(v_i \mid h).$$

І навпаки, умовною ймовірністю *h* для заданої *v* є

$$P(h \mid v) = \prod_{j=1,n} P(h_j \mid v).$$

Окремі ймовірності активації задаються як

$$P(h_j = 1 \mid v) = \sigma\left(b_j + \sum_{i=1,m} w_{i,j} v_i\right),$$

$$P(v_i = 1 \mid h) = \sigma\left(a_i + \sum_{j=1,n} w_{i,j} h_j\right),$$

де σ позначає логістичну сигмоїду.

Незважаючи на те, що приховані вузли є бернуллієвими, видимі вузли ОМБ можуть бути багатозначними.

В такому випадку логістична функція для видимих вузлів замінюється багатозмінною логістичною функцією активації (*Softmax function*)

$$P(v_{ik} = 1 \mid h) = \exp\left(a_{ik} + \Sigma_j W_{ijk} h_j\right) \Sigma_{k=1,K} \exp\left(a_{ik} + \Sigma_j W_{ijk} h_j\right),$$

де *K* є кількістю дискретних значень, які мають видимі значення.

Нейронна мережа Хопфілда

Серед різних конфігурацій штучних нейронних мереж (НС) зустрічаються такі, при класифікації яких за принципом навчання, строго кажучи, не підходять ні навчання з учителем , ні навчання без учителя.

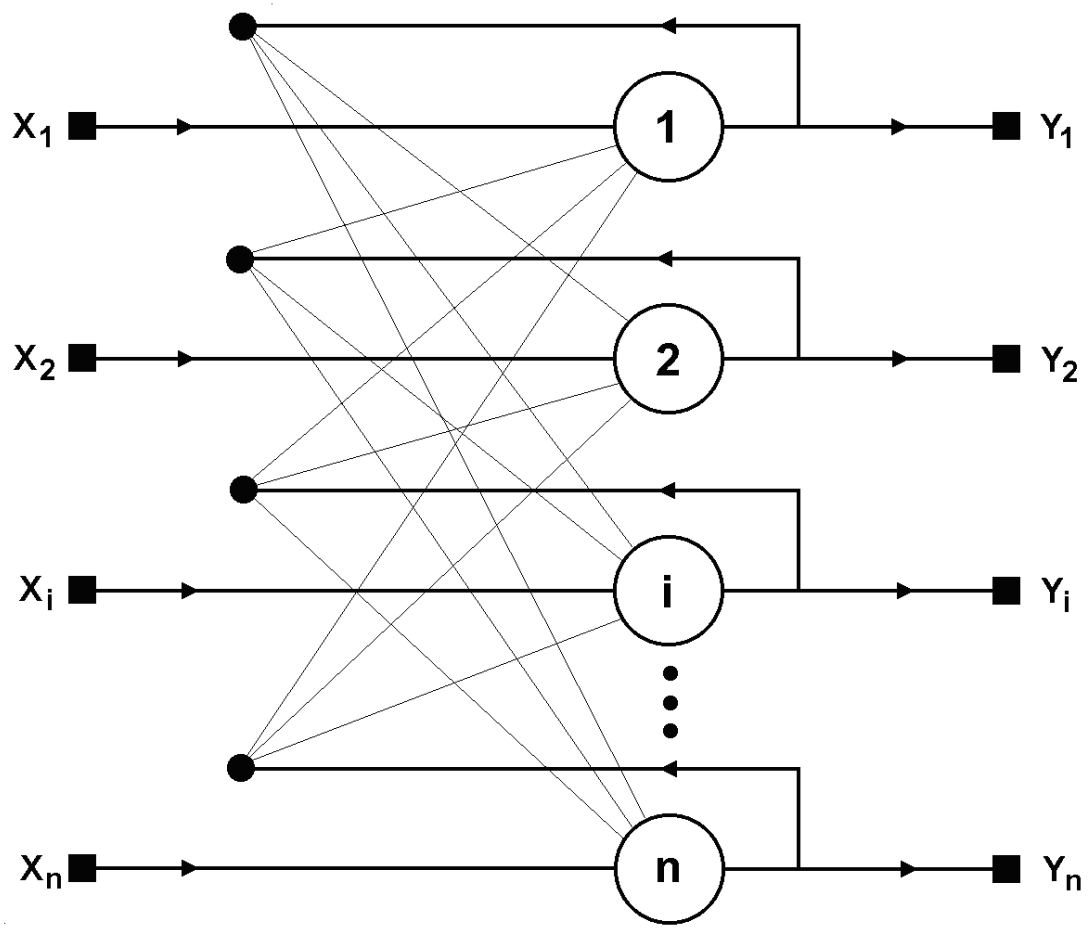
У таких мережах вагові коефіцієнти синапсів розраховуються тільки один раз перед початком функціонування мережі на основі інформації про оброблювані дані, і усе навчання мережі зводиться саме до цього розрахунку.

Нейронна мережа Хопфілда

З одного боку, пред'явлення апріорної інформації можна розцінювати, як допомога учителя, але з іншого - мережа фактично просто запам'ятовує зразки до того, як на її вхід поступають реальні дані, і не може змінювати свою поведінку, тому говорити про ланку зворотного зв'язку зі "світом" (учителем) не доводиться.

З мереж з подібною логікою роботи найбільш відомі мережа Хопфілда і мережа Хеммінга, які зазвичай використовуються для організації асоціативної пам'яті.

Структурна схема мережі Хопфілда приведена на рис. Вона складається з єдиного шару нейронів, число яких є одночасно числом входів і виходів мережі.



Нейронна мережа Хопфілда

Кожен нейрон пов'язаний синапсами з усіма іншими нейронами, а також має один вхідний синапс, через який здійснюється введення сигналу. Вихідні сигнали, як завжди, утворюються на аксонах.

Завдання, що вирішується цією мережею як асоціативна пам'ять, як правило, формулюється таким чином:

1. Відомий деякий набір двійкових сигналів (зображень, звукових оцифрувань, інших даних, що описують деякі об'єкти або характеристики процесів), які вважаються зразковими.
2. Мережа повинна уміти з довільного неідеального сигналу, поданого на її вхід, виділити ("згадати" за частковою інформацією) відповідний зразок (якщо такий є) або “надати висновок" про те, що вхідні дані не відповідають жодному із зразків.

Нейронна мережа Хопфілда

У загальному випадку, будь-який сигнал може бути описаний вектором **X** = { x_i: i=1..n }, n - число нейронів в мережі і розмірність вхідних і вихідних векторів. Кожен елемент x_i рівний або +1, або - 1.

Позначимо вектор, що описує, k-й зразок, через $\mathbf{X}_k$, а його компоненти, відповідно, - x_{ik}, k=1..m , m - число зразків.

Коли мережа розпізнає (або "згадає") який-небудь зразок на основі пред'явлених їй даних, її виходи міститимуть саме його, тобто **Y** = $\mathbf{X}_k$, де **Y** - вектор вихідних значень мережі : **Y** = { y_i: i=1,..n}. Інакше, вихідний вектор не співпаде ні з одним зразковим.

Якщо, наприклад, сигнали є деякими зображеннями, то, відобразив в графічному виді дані з виходу мережі, можна буде побачити картинку, що повністю співпадає з однією із зразкових (у разі успіху) або ж "вільну імпровізацію" мережі (у разі невдачі).

Нейронна мережа Хопфілда

На стадії ініціалізації мережі вагові коефіцієнти синапсів встановлюються таким чином :

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, i \neq j \\ 0, \qquad i = j \end{cases}$$

Тут *i* і *j* - індекси, відповідно, передсинаптичного і постсинаптичного нейронів; x_i^k, x_j^k - і -й і j -й елементи вектору k -го зразка.

Нейронна мережа Хопфілда

Алгоритм функціонування мережі наступний (p - номер ітерації) :

1. На входи мережі подається невідомий сигнал. Фактично його введення здійснюється безпосередньою установкою значень аксонів :

$$y_i(0) = x_i,\ i = 1..n,$$

тому позначення на схемі мережі вхідних синапсів в явному виді носить чисто умовний характер. Нуль в дужці праворуч від y_i означає нульову ітерацію в циклі роботи мережі.

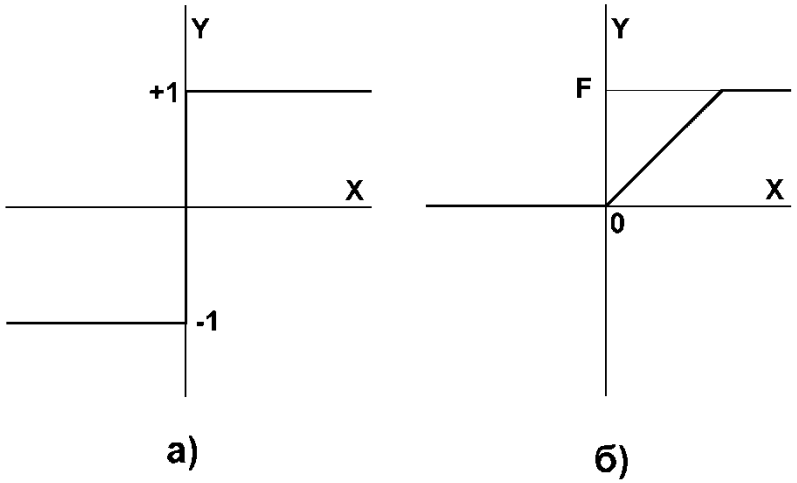
2. Розраховується новий стан нейронів

$$s_j(t+1) = \sum_{i=0}^{n-1} w_{ij} y_i(t) \quad , j=1..n$$

і нові значення аксонів

$$y_j(t+1) = F\left[s_j(t+1)\right]$$

де F - активаційна функція у вигляді стрибка, приведена на рис.



3. Перевірка, чи змінилися вихідні значення аксонів за останню ітерацію. Якщо так - перехід до пункту 2, інакше (якщо виходи стабілізувались) - кінець. При цьому вихідний вектор є зразком, що якнайкраще поєднується з вхідними даними.

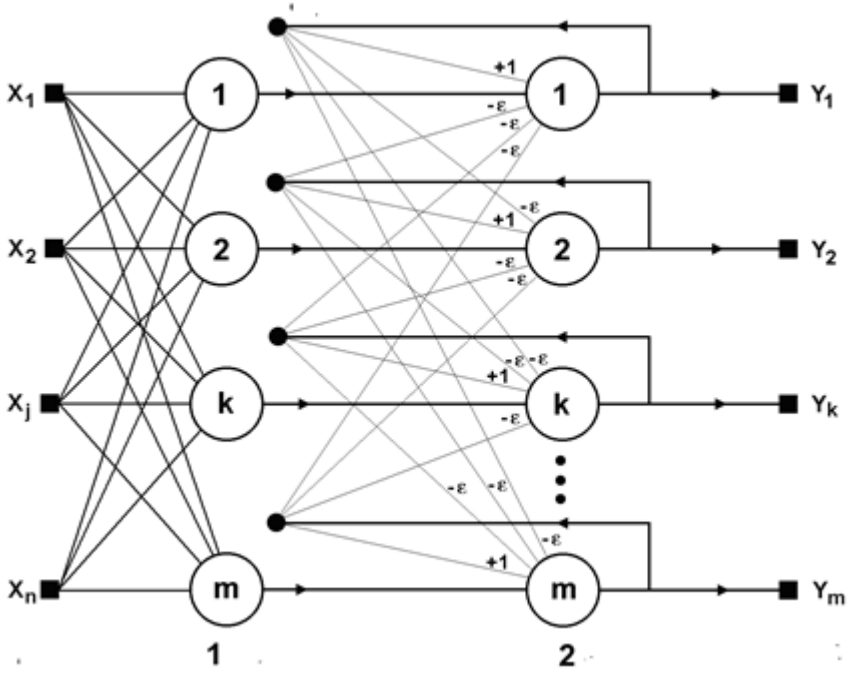
Як говорилося вище, іноді мережа не може провести розпізнавання і видає на виході неіснуючий образ. Це пов'язано з проблемою обмеженості можливостей мережі.

Для мережі Хопфілда число образів m, що запам'ятовуються, не повинне перевищувати величини, приблизно рівної $m<=0.15*n$.

Крім того, якщо два образи А і б дуже схожі, вони, можливо, викликатимуть у мережі перехресні асоціації, тобто пред'явлення на входи мережі вектору А приведе до появи на її виходах вектору б і навпаки.

Нейронна мережа Хеммінга

Коли немає необхідності, щоб мережа в явному виді видавала зразок, тобто досить, скажімо, отримувати номер зразка, асоціативну пам'ять успішно реалізує мережа Хеммінга.



Ця мережа характеризується, в порівнянні з мережею Хопфілда, меншими витратами на пам'ять і об'ємом обчислень, що стає очевидним з її структури

Нейронна мережа Хеммінга

Мережа складається з двох шарів. Перший і другий шари мають по m нейронів, де m - число зразків.

Нейрони першого шару мають по n синапсів, сполучених з входами мережі (що утворюють фіктивний нульовий шар).

Нейрони другого шару пов'язані між собою ингибиторными (негативними зворотними) синаптическими зв'язками. Єдиний синапс з позитивним зворотним зв'язком для кожного нейрона сполучений з його ж аксоном.

Ідея роботи мережі полягає в знаходженні відстані Хеммінга від тестованого образу до усіх зразків. Відстанню Хеммінга називається число бітів, що відрізняються, в двох бінарних векторах.

Мережа повинна вибрати зразок з мінімальною відстанню Хеммінга до невідомого вхідного сигналу, внаслідок чого буде активізований тільки один вихід мережі, відповідний цьому зразку.

Нейронна мережа Хеммінга

На стадії ініціалізації ваговим коефіцієнтам першого шару і порогу активаційної функції привласнюються наступні значення:

$$w_{ik} = \frac{x_i^k}{2}, \qquad i=0..n-1,\ k=0..m-1$$

$$T_k = n / 2, \qquad k = 0..m-1$$

Тут x_{ik} - і -й елемент k -го зразка.

Вагові коефіцієнти гальмівних синапсів в другому шарі беруть рівними деякій величині $0 < \varepsilon < 1/m$. Синапс нейрона, пов'язаний з його ж аксоном має вагу +1.

Алгоритм функціонування мережі Хеммінга

1. На входи мережі подається невідомий вектор X = {xi: i=0..n - 1}, виходячи з якого розраховуються стани нейронів першого шару (верхній індекс в дужках вказує номер шару) :

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=0}^{n-1} w_{ij} x_i + T_j \qquad , j=0..m-1$$

Після цього набутих значень ініціалізувалися значення аксонів другого шару : $y_j^{(2)} = y_j^{(1)}, j = 0..m - 1$

2. Вичислити нові стани нейронів другого шару :

$$s_j^{(2)}(p+1) = y_j(p) - \varepsilon \sum_{k=0}^{m-1} y_k^{(2)}(p), k \neq j, j = 0...m-1$$

і значення їх аксонів :

$$y_j^{(2)}(p+1) = F\left[s_j^{(2)}(p+1)\right], j = 0..m-1$$

Активаційна функція F має вигляд порогу (мал. б), причому величина F має бути досить великою, щоб будь-які можливі значення аргументу не призводили до насичення.

3. Перевірити, чи змінилися виходи нейронів другого шару за останню ітерацію. Якщо так - перейди до кроку 2. Інакше - кінець.

Барт Эндрю Коско (**Bart Andrew Kosko**)

- Письменник, професор електротехніки в Південно-Каліфорнійському університеті (USC).
- Він відомий як науковий і популяризатор нечіткої логіки, нейронних мереж, а також як автор декількох книг по темах, пов'язаних з штучним інтелектом.
- Коско отримав ступінь бакалавра з філософії та економіки в USC, а також магістра математики в UC San Diego, PhD на електротехніку в UC Irvine.
- Самою популярною книгою в Коско на сьогодні є книга "Нечітке мислення", книга про мислення людей і машин у "градаціях сірого", а також широко відомою його книзі "Шум".
- Він також опублікував короткий кібер-триллер "Нанотіме", про Третю світову війні, яка може мати місце в 2030 році.

Двонаправлена асоціативна пам'ять (ДАП) Bi-directional associative memories (BAM)

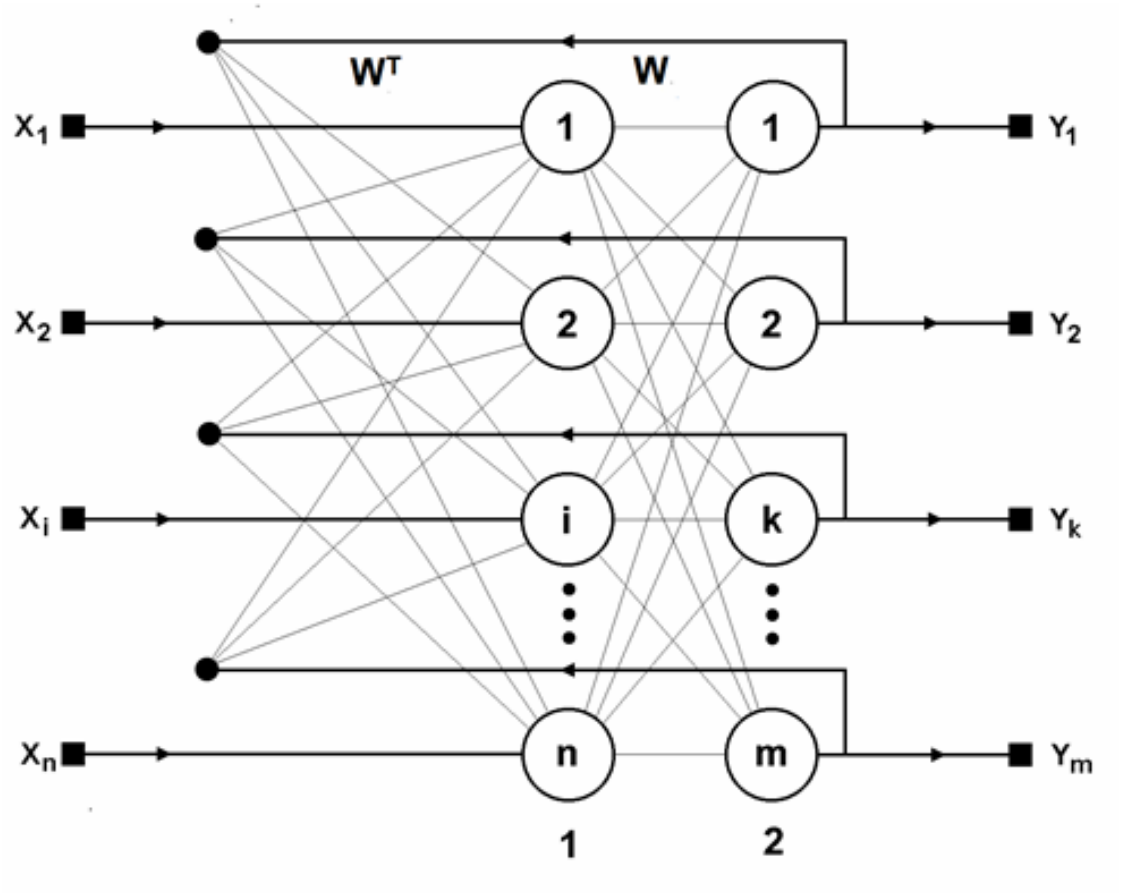
Пам'ять людини часто є асоціативною; один предмет нагадує нам про інше, а цей інший про третє. Якщо дозволити нашим думкам, вони переміщатимуться від предмета до предмета по ланцюжку розумових асоціацій. Крім того, можливе використання здатності до асоціацій для відновлення забутих образів.

Двонаправлена асоціативна пам'ять (ДАП), яку запропонував Бартоломео Коско (Bart Kosko)

, є гетеро асоціативною; вхідний вектор поступає на один набір нейронів, а відповідний вихідний вектор виробляється на іншому наборі нейронів.

Вона є логічним розвитком парадигми мережі Хопфілда, до якої для цього досить додати другий шар. Структура ДАП представлена на рис.

Двонаправлена асоціативна пам'ять



Двонаправлена асоціативна пам'ять (ДАП)

Мережа здатна запам'ятовувати пари асоційованих один з одним образів. Нехай пари образів записуються у вигляді векторів-рядків:

$\mathbf{X_k} = \{x_{ip}: i=1...n\}$ і $\mathbf{Y_k} = \{y_{jp}: j=1...m\}$, $p=1...M$, де M - число пар.

Подача на вхід першого шару деякого вектору $\mathbf{P} = \{p_i: i=1...n\}$ викликає утворення на вході другого шару деякого іншого вектору $\mathbf{Q} = \{q_j: j=1..m\}$, який потім знову поступає на вхід першого шару.

При кожному такому циклі вектори на виходах обох шарів наближаються до пари зразкових векторів, перший з яких - **X** - найбільш схожий на **P**, який був поданий на вхід мережі на самому початку, а другий - **Y** - асоційований з ним на **Q**.

Асоціації між векторами кодуються у ваговій матриці **W** першого шару. Вагова матриця другого шару дорівнює транспонованій першій - $\mathbf{W^T}$.

Процес навчання ДАП, так як і у мережі Хопфілда, полягає в попередньому обчисленні елементів матриці **W** (і відповідно $\mathbf{W}^T$) за формулою:

$$w_{ij} = \sum_k x_i y_j, i = 1 \ldots n, j = 1 \ldots m$$

Ця формула є розгорнутим записом матричного рівняння

$$\mathbf{W} = \sum_{k=1}^{M} \mathbf{X}_k^T \mathbf{Y}_k$$

для окремого випадку, коли образи записані у вигляді векторів, при цьому добуток двох матриць розміром відповідно [n*1] і [1*m] призводить до (11).

Якщо задана активаційна функція F(S), то вихідні вектори обчислюються за формулами:

$$\mathbf{Y}=F(\mathbf{X}\,\mathbf{W}), \qquad \mathbf{X}=F(\mathbf{Y}\,\mathbf{W}^T)$$

При цьому, бінарні вектори рекомендується кодувати -1 та +1 і активаційна функція повинна повертати значення на проміжку[-1,+1].

Двонаправлена асоціативна пам'ять (ДАП)

При функціонуванні ДАП вектори **X** і **Y** обчислюються по черзі до стабілізації значень. Стабілізовані значення і будуть відновленою парою асоціативних образів, що збігається з найближчою парою із навчальної вибірки.

Об’єм ДАП - кількість пар образів, що може відновлюватись, визначається формулою:

$$M \le \frac{\min(m,n)}{4\log_2 \min(m,n)}$$

У висновку можна зробити наступне узагальнення:

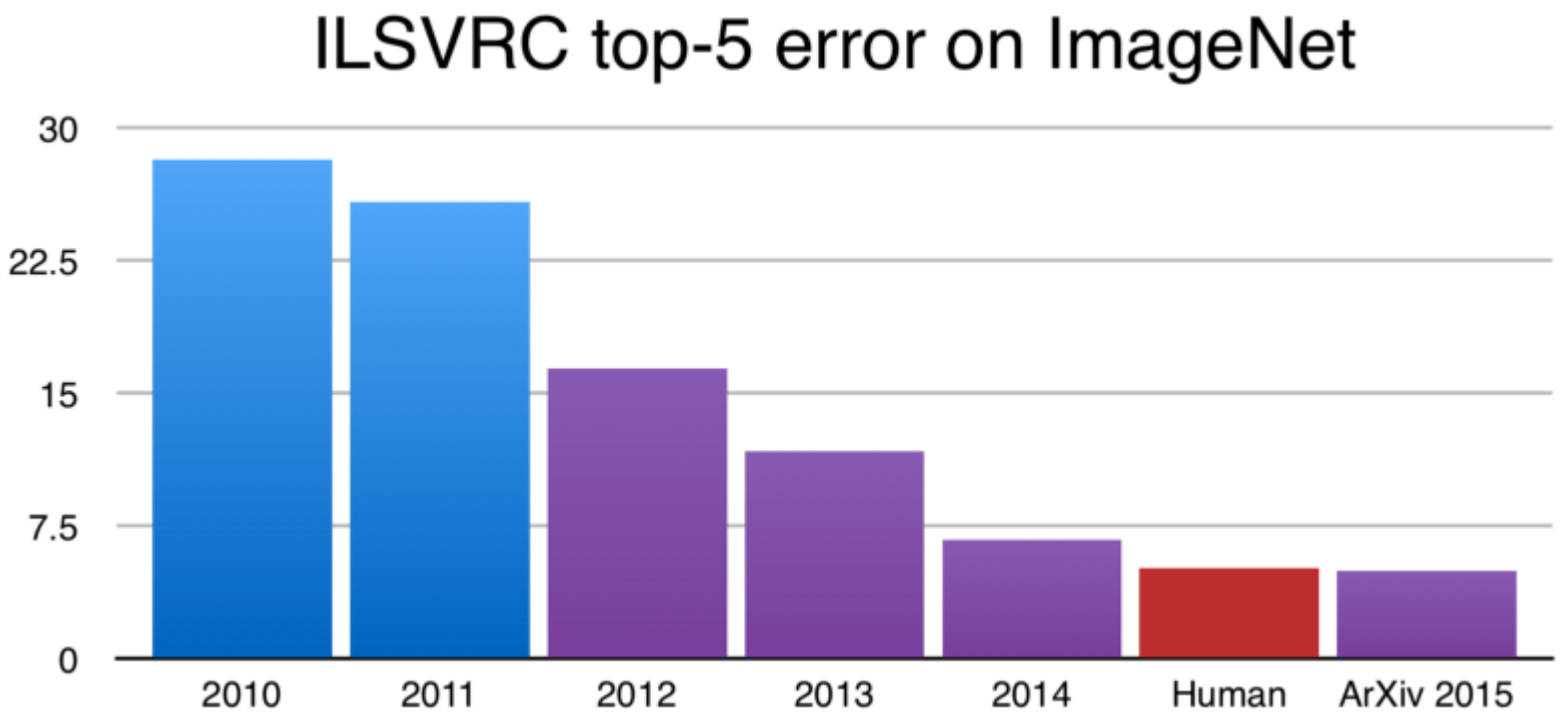
Мережі Хопфілда, Хеммінга і ДАП дозволяють просто і ефективно вирішувати завдання відтворення образів за неповною і спотвореною інформацією.

Невисока місткість мереж (число образів, що запам'ятовуються) пояснюється тим, що, мережі не просто запам'ятовують образи, а дозволяють проводити їх узагальнення.

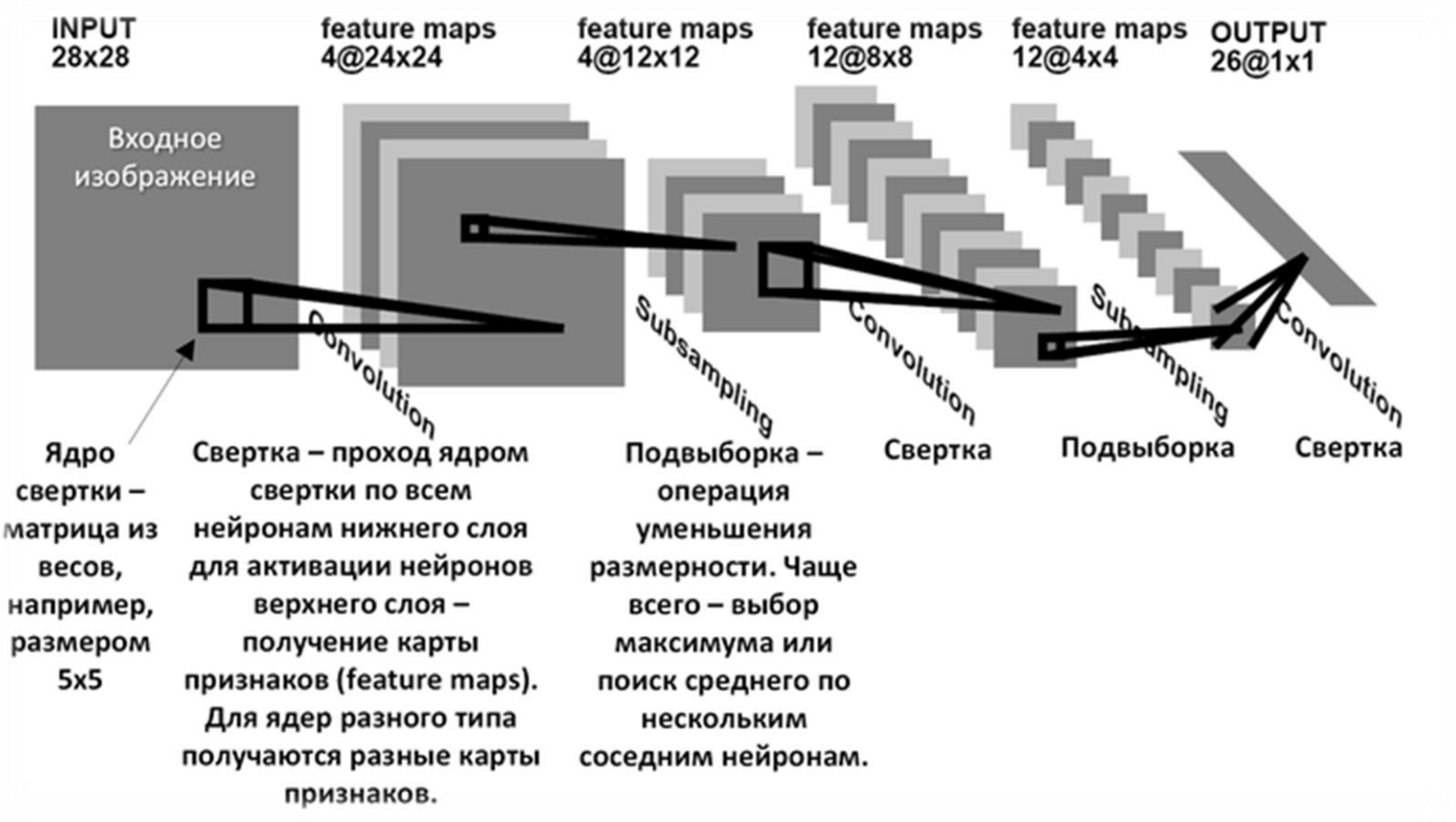
Згорткові нейронні мережі
Convolution Neural Network

Yann LeCun's http://yann.lecun.com/

Professor The Courant Institute of Mathematical Sciences
New York University

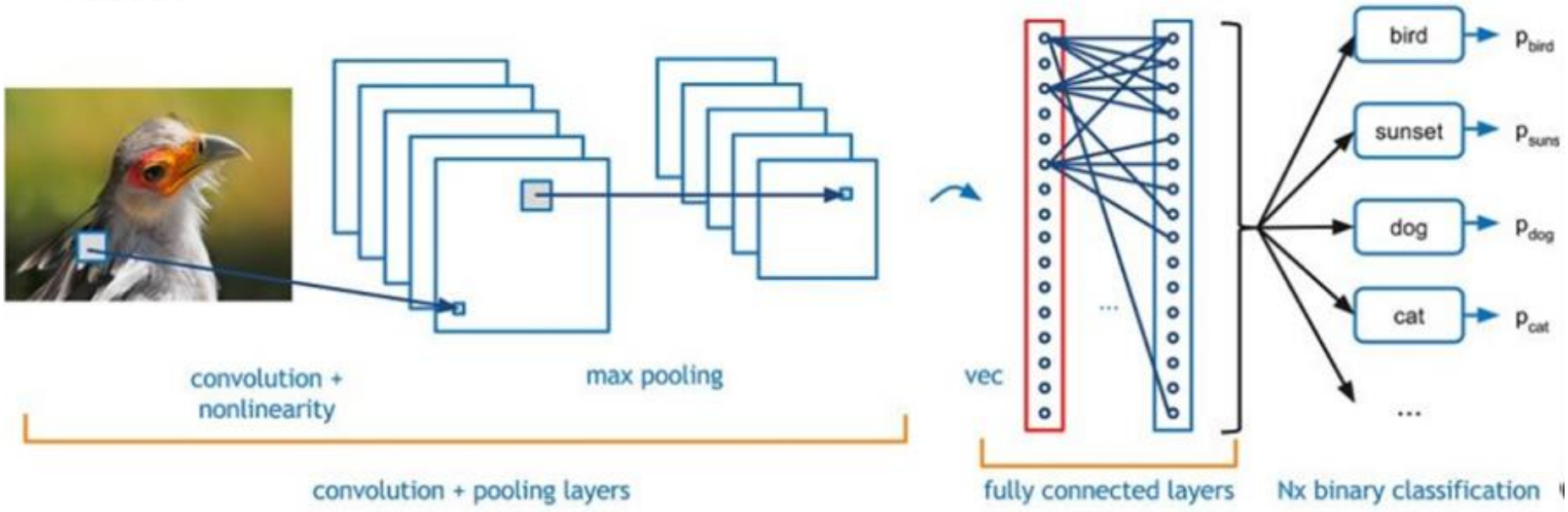

ILSVRC top-5 error on ImageNet
30
22.5
15
7.5
0
2010
2011
2012
2013
2014
Human
ArXiv 2015

Згорткові нейронні мережі



Згорткова нейронна мережа

Свёрточная нейросеть (CNN) — это Feed-Forward сеть специального вида:



У загорткових НМ менше параметрів

CNN

- вход картинка 100x100
- два свёрточных слоя по 100 плоскостей каждый (conv 5x5 и subsampling 2)
- полносвязный слой на 100 нейронов
- выход: 10 классов
- число параметров примерно **70K** (25*100 + 25*100 + 25*25*100 + 100*10)

FNN

- вход: картинка 100x100
- три скрытых слоя по 100 нейронов
- выход: 10 классов
- число параметров примерно **1M** (10000*100 + 100*100 + 100*100 + 100*10)

Згорткові нейронні мережі

Згорткові нейронні мережі працюють на основі фільтрів, які займаються розпізнаванням певних характеристик зображення (наприклад, прямих ліній).

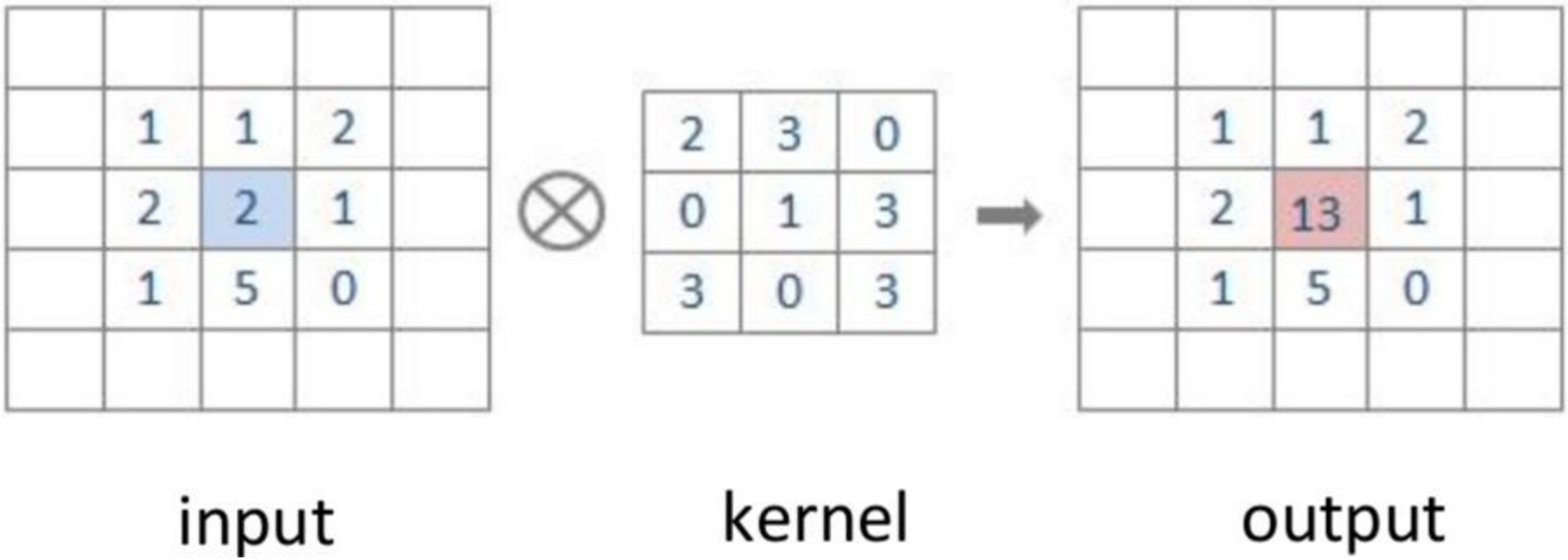
Фільтр — це колекція кернелов; іноді у фільтрі використовується один кернел.

Кернел — це звичайна матриця чисел, званих вагами, які "навчаються" (підлаштовуються, якщо вам так зручніше) з метою пошуку на зображеннях певних характеристик.

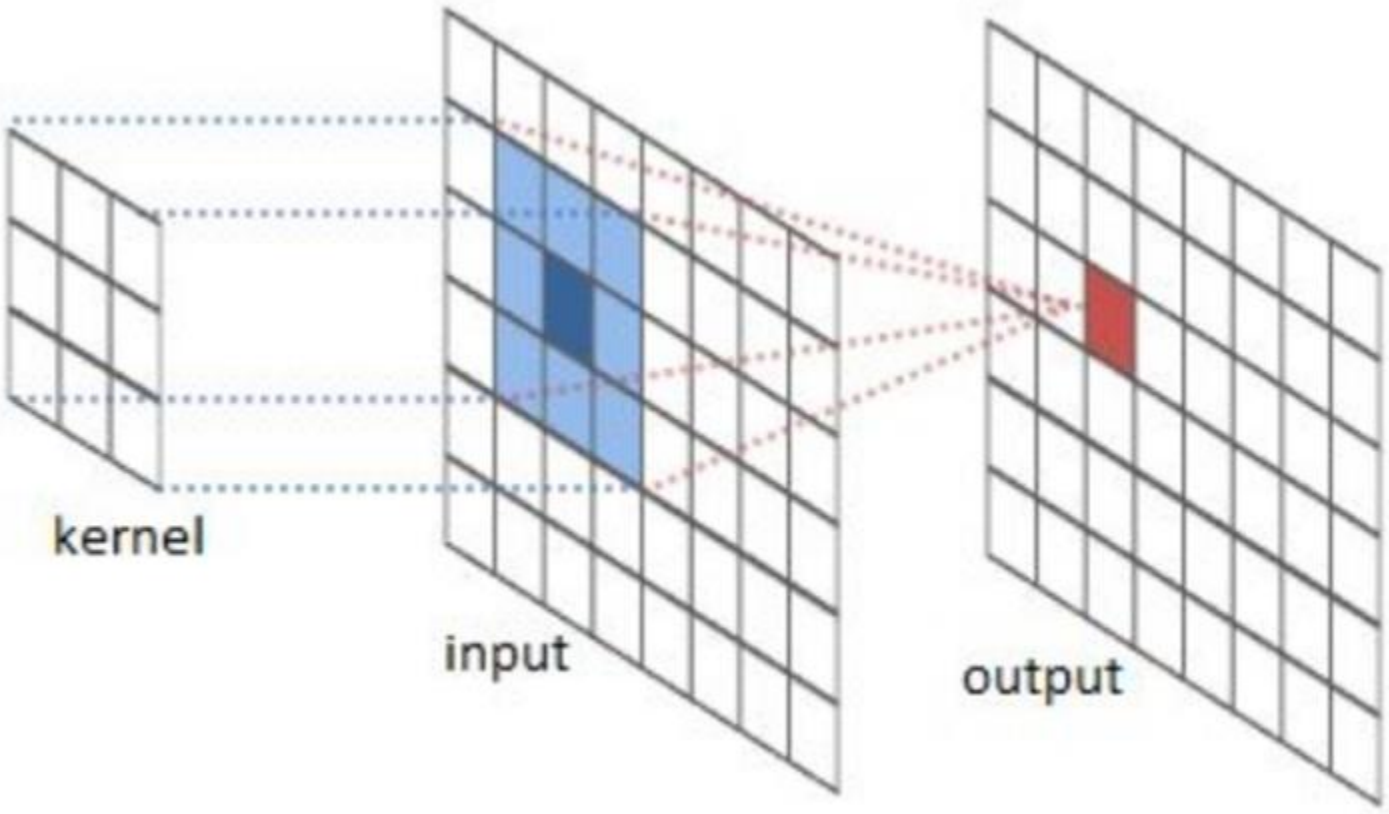
Фільтр переміщається уздовж зображення і визначає, чи є присутньою деяка шукана характеристика в конкретній його частині.

Для отримання відповіді такого роду здійснюється операція свертки, яка є сумою творів елементів фільтру і матриці вхідних сигналів.

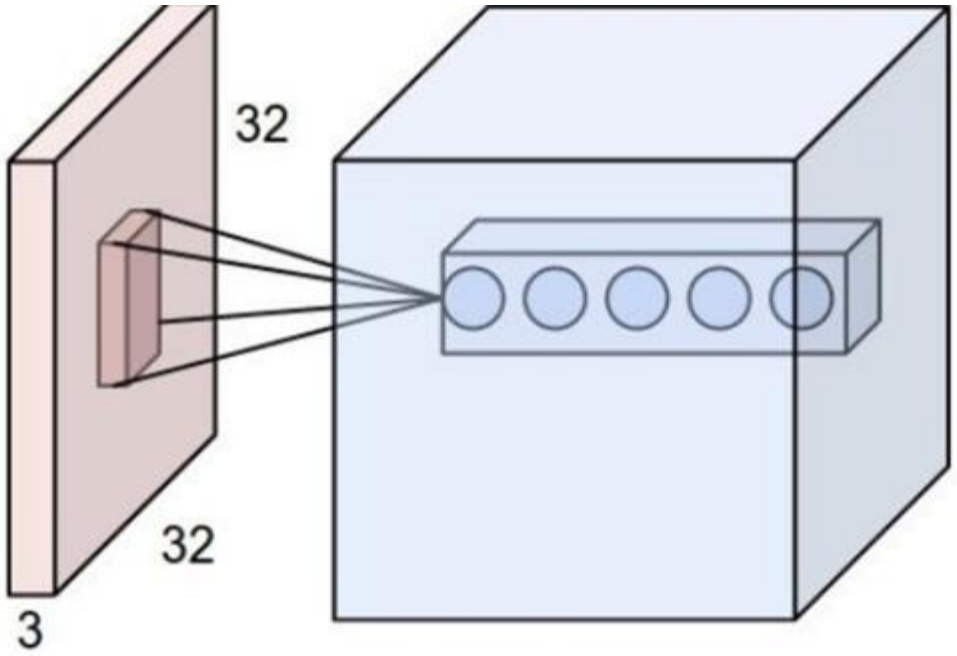
Візуалізація згортки



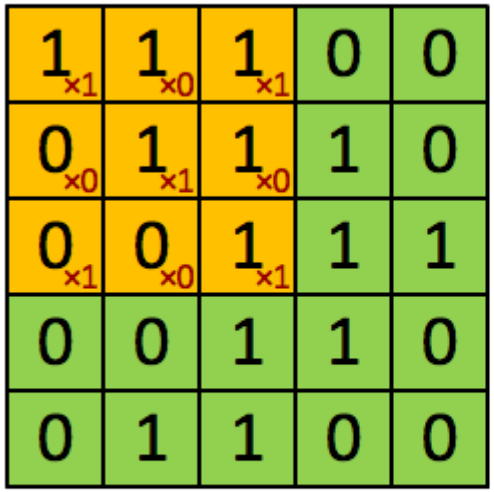
Згортка



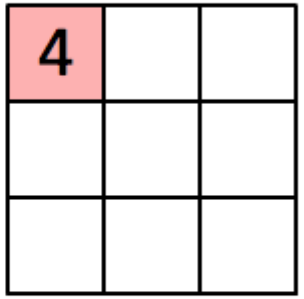
Згортковий шар 5 нейронів



Веса нейронов — это коэффициенты ядра свёртки. Каждая “обучаемая” свёртка выделяет одинаковые локальные признаки во всех частях изображения.



Image



Convolved Feature

Згорткові нейронні мережі

- Фільтр може переміщатися уздовж матриці вхідних сигналів з кроком, відмінним від одиниці. Крок переміщення фільтру називається **страйдом (stride).**
- **Страйд визначає, на яку кількість пікселів повинен зміститися фільтр за один крок.**

$$n_{out} = floor(\frac{n_{in} - f}{s}) + 1 \qquad (1)$$

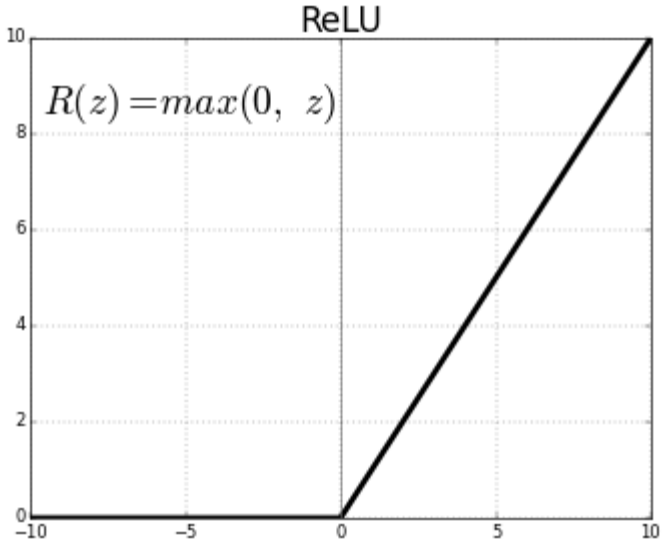
- Кількість вихідних значень після операції згортки може бути розрахована по формулі 1.
- Де n_in — к-ть вхідних пікселів,
- f — к-ть пікселів у фільтрі,
- s — страйд.
- Для прикладу на мал дану формулу слід застосувати таким чином: (25-9)/2+1=3.

Згорткові нейронні мережі

- Для того, щоб навчання ваг, що знаходяться в кернелах, було ефективним, в результати згорток слід ввести деяке зміщення (bias) і нелінійність.
- Зміщення — це статична величина, на яку слід "змістити" вихідні значення. За своєю суттю це звичайна операція складання кожного елементу вихідної матриці з величиною зміщення.
- Якщо пояснювати дуже поверхнево, це треба для того, щоб вивести нейронну мережу з тупикових ситуацій, що мають суто математичні причини.
- Нелінійність представляє з себе **функцію активації**. Завдяки ній картина, що формується за допомогою операції згортки, отримує деяке спотворення, що дозволяє нейронній мережі ясніше оцінювати ситуацію.

Згорткові нейронні мережі

- Цю необхідність дуже грубо можна порівняти з необхідністю людям із слабким зором носити контактні лінзи.
- А взагалі-то така необхідність пов'язана з тим, що вхідні дані за своєю природою нелінійні, тому треба умисне спотворювати проміжні результати, щоб відповідь нейронної мережі була такою, що вирішує проблему.
- Часто як функцію активації використовують ReLU (Rectified Linear Unit).

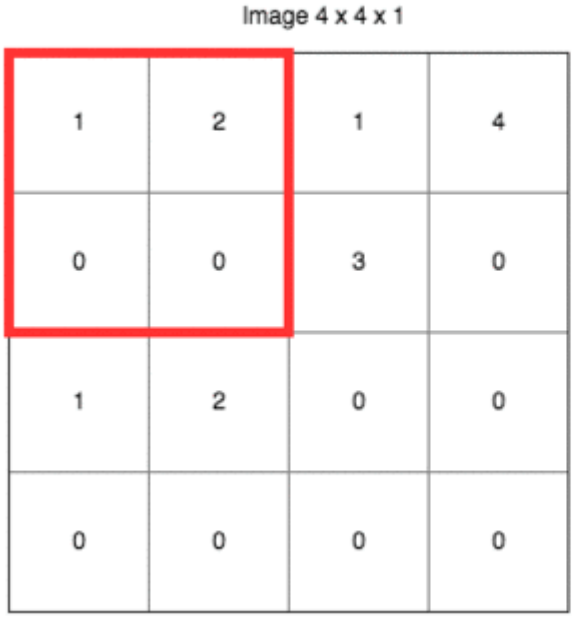


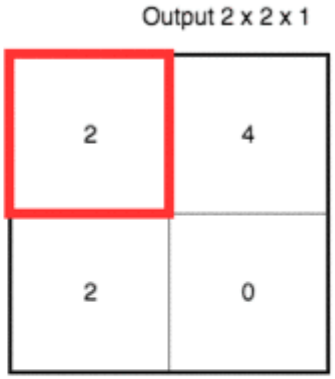
Даунсемплинг

- З метою прискорення процесу навчання і зменшення споживання обчислювальних ресурсів роблять даунсемплинг початкових і проміжних даних
- . Існує декілька способів зробити це. У цій статті ми розглядатимемо найпростіший і поширеніший з них — **максимальне об'єднання (max pooling).**
- Операція максимального об'єднання полягає в тому, що уздовж даних переміщається так зване *вікно просіювання.* З пікселів, що потрапляють в його поле зору, відбирається максимальний і переміщається в результуючу матрицю.
- Страйд для цього вікна може бути відмінним від одиниці; усе залежить від того, якого розміру вихідну матрицю треба отримати, і з якою мірою треба "стиснути" дані.

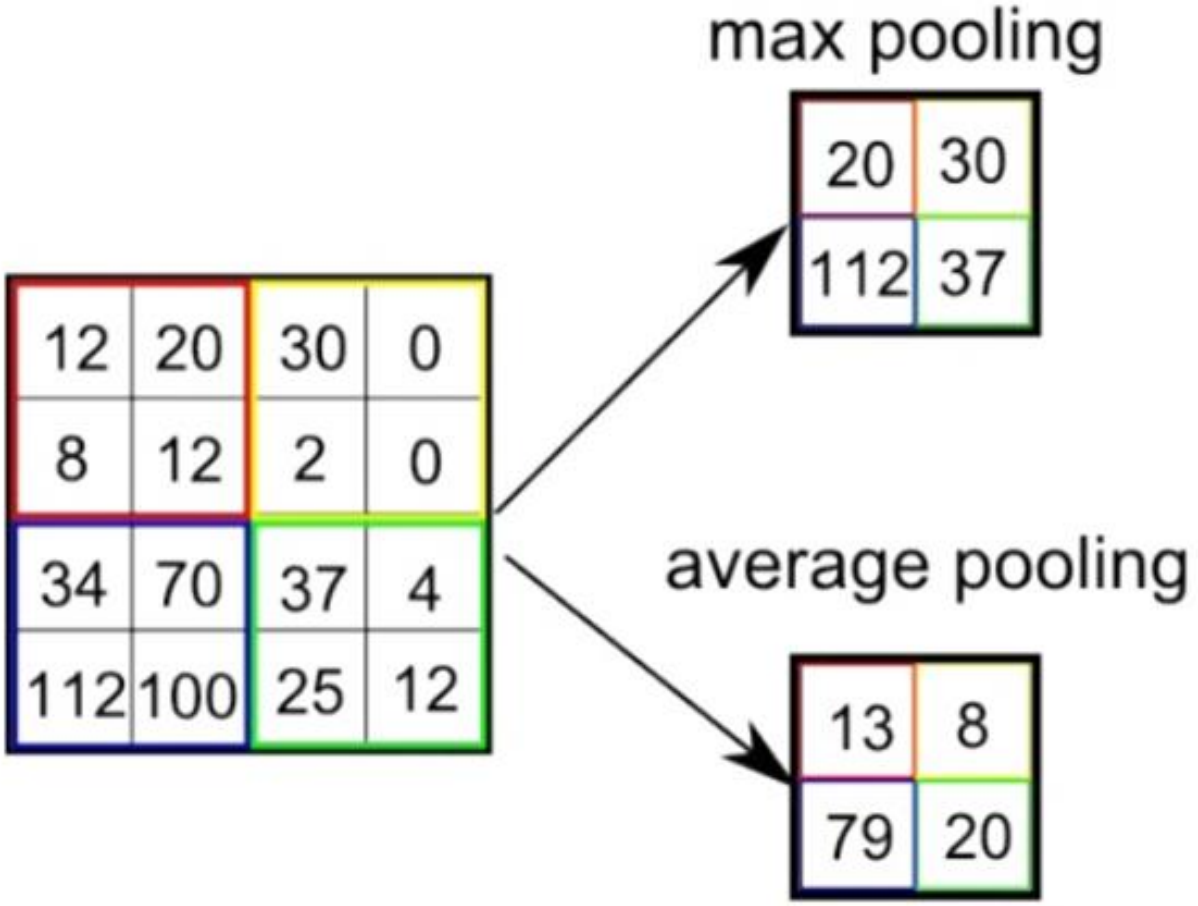
Даунсемплинг

- Подивіться уважно на мал. На нім наочно зображена операція максимального об'єднання : вікно розмірністю 2×2 переміщається уздовж матриці; страйд має значення 2.
- Страйд визначає крок, на який переміщається вікно за один етап. На кожному етапі відбирається максимальне значення. Початкова матриця нібито просіюється через сито, яке видає назовні тільки характерні пікселі.

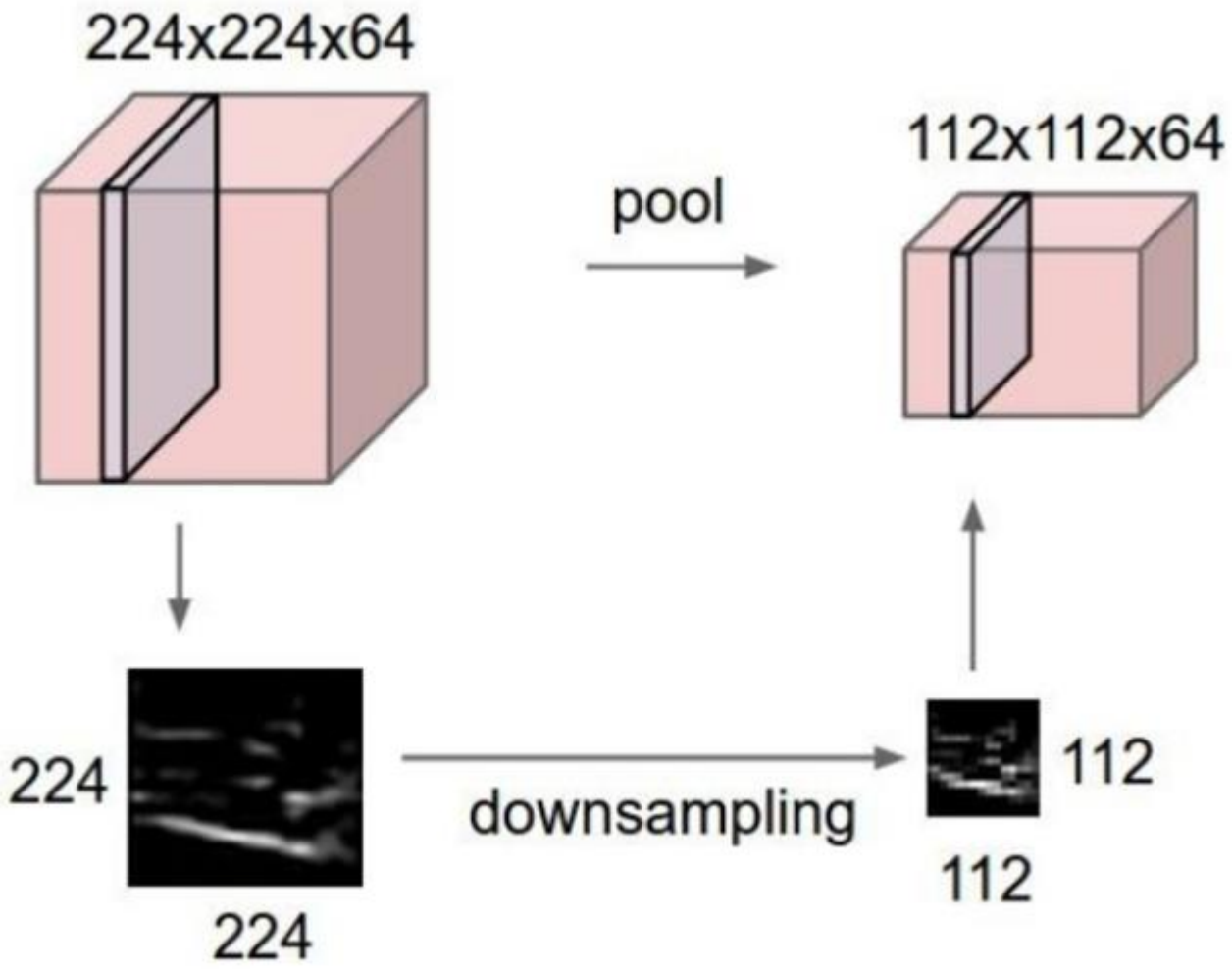




Pooling

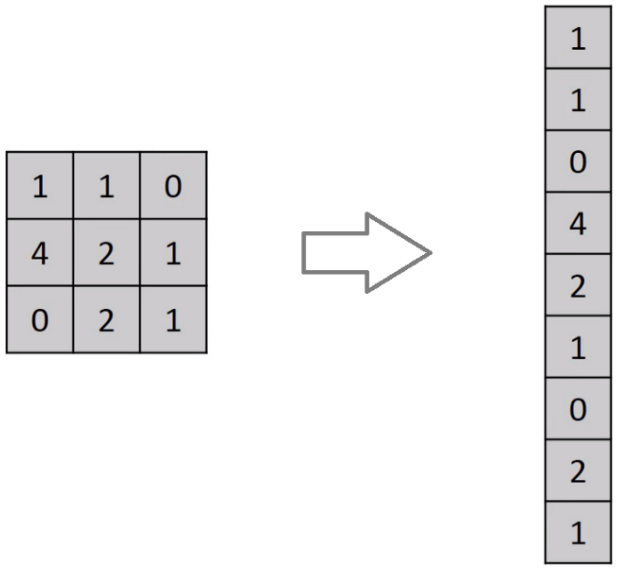

max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

Downsampling



Повнозв'язний шар

- На цьому етапі перетворення даних тривимірна матриця сигналів буде розгорнута у вектор, який буде пропущений через повнозв'язну нейронну мережу. Її завдання полягає в тому, щоб повідомити нам вірогідність того, яку цифру представляє вхідний образ. Мал. демонструє цю операцію.

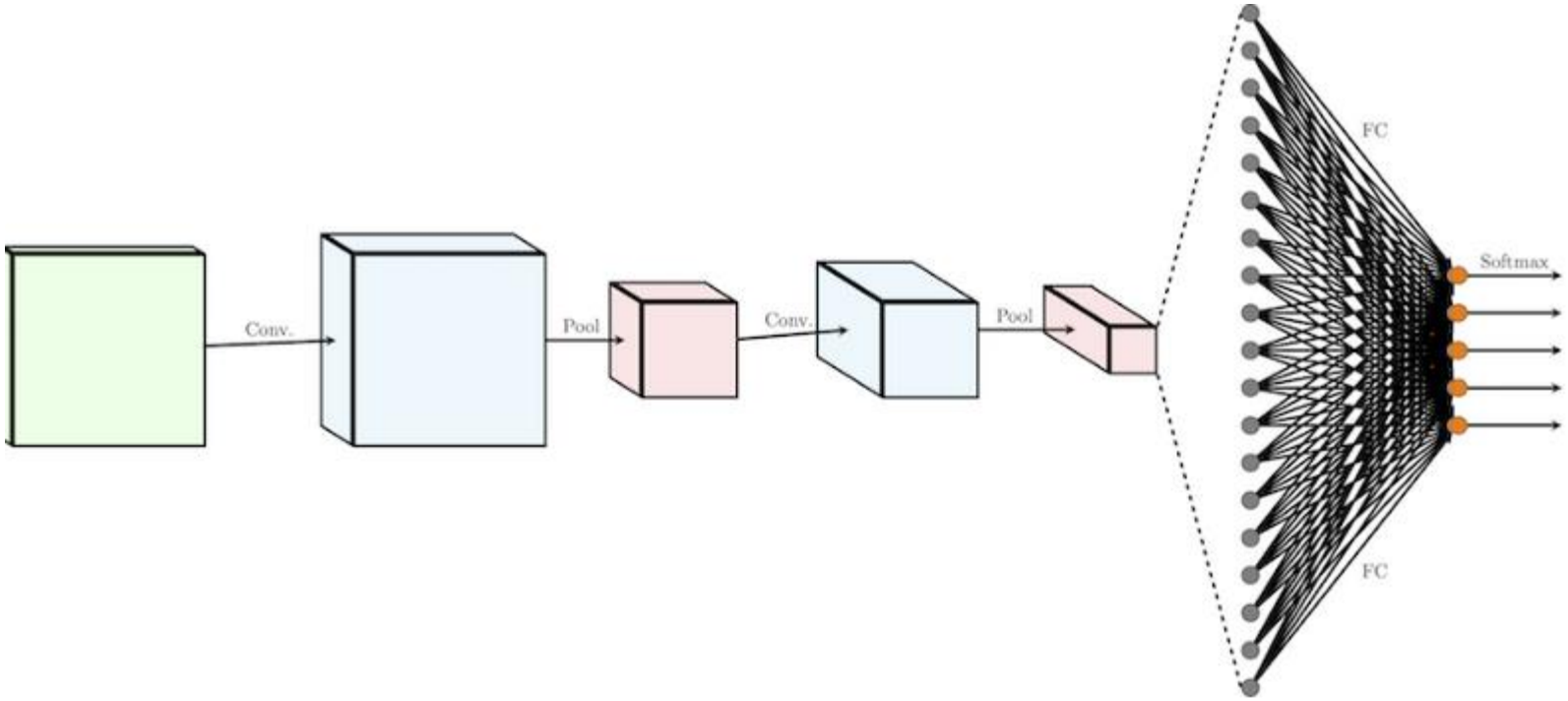


Повнозв'язний шар

- З малюнка видно, що операція розгортання полягає в "склеюванні" рядків в єдиний — величезної довжини — числовий ряд. Це буде воістину великий вектор, який треба буде ще і перетворити за допомогою багатошарової мережі з повними зв'язками!
- Якщо ви не знаєте, як працює повнозв'язний шар, ось простий опис механізму : кожен елемент вектору множиться на вагу зв'язку, ці твори далі підсумовуються між собою і з деяким зміщенням, після чого результат піддається перетворенню за допомогою функції активації. На мал. описане представлене в наочній формі.
- Варто відмітити, що Ян ЛеКун в цьому посту на Фейсбуці сказав, що "в згортальних нейронних мережах немає такого поняття, як повнозв'язний шар". І він абсолютно правий!

Повнозв'язний шар

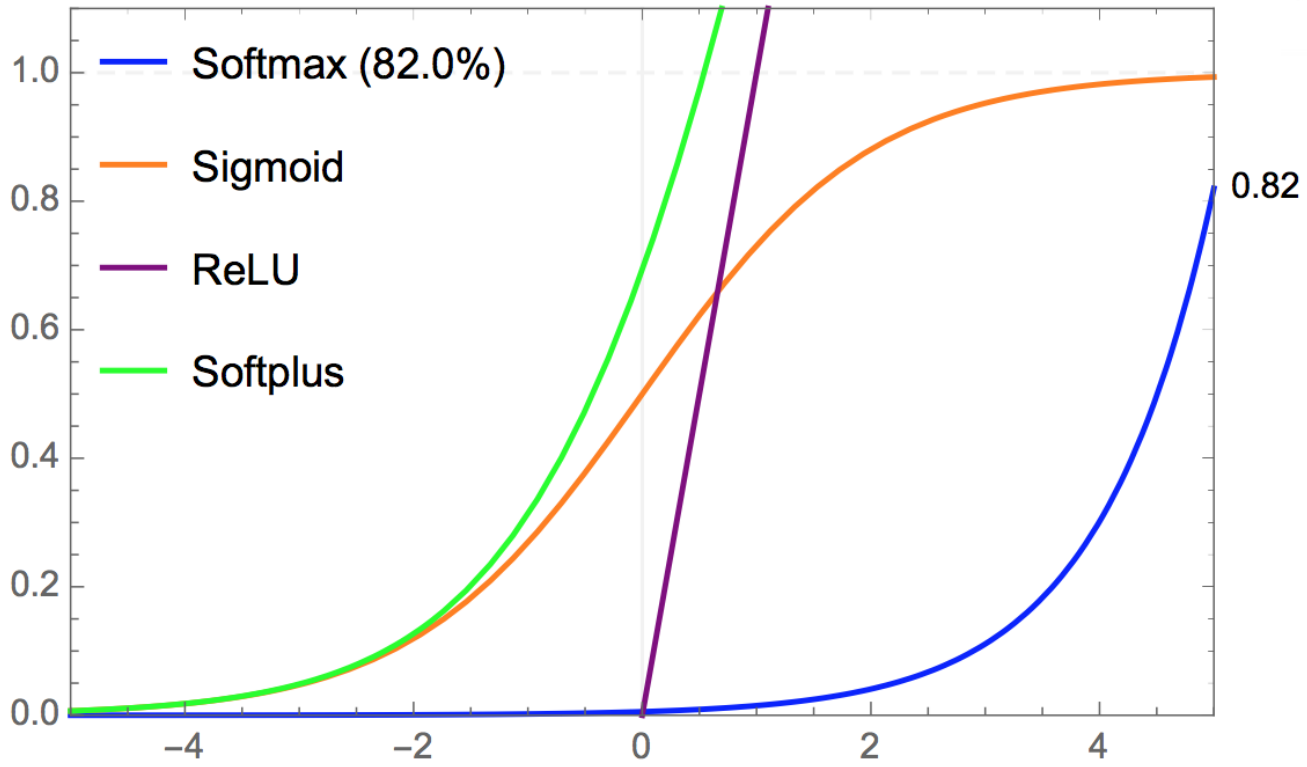
Якщо уважно придивитися, то стане абсолютно очевидно, що принцип роботи повнозв'язного шару аналогічний тому, що має місце в згортальному шарі з кернелом розмірністю 1×1. Тобто, якщо в нашому розпорядженні 128 фільтрів розмірністю n×n, які взаємодіятимуть із зображенням розмірністю n×n, на виході ми отримаємо вектор, в якому буде 128 елементів.



Вихідний шар

- Вихідний шар відповідає за формування вірогідності приналежності вхідного образу тому або іншому класу (деякому числу). Щоб добитися цього, вихідний шар повинен містити кількість нейронів, що відповідають кількості класів. Зважені і підсумовані сигнали далі модифікуються за допомогою **функції активації використовуватимемо softmax**

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



Функція помилки (втрат)

Щоб визначити, наскільки точно нейронна мережа визначає написані від руки цифри, ми використовуватимемо **функцію втрат**.

Відповіддю функції втрат є дійсне число, що характеризують якість відповіді нейронної мережі. Зазвичай для оцінки якості класифікаторів застосовують категоріальну функцію крос-энтропийную втрат (Categorical Cross - Entropy Loss Function, або CCELF).

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

У формулі

$\hat{y}$ — *фактична відповідь нейронної мережі,*

y — бажана відповідь нейронної мережі.

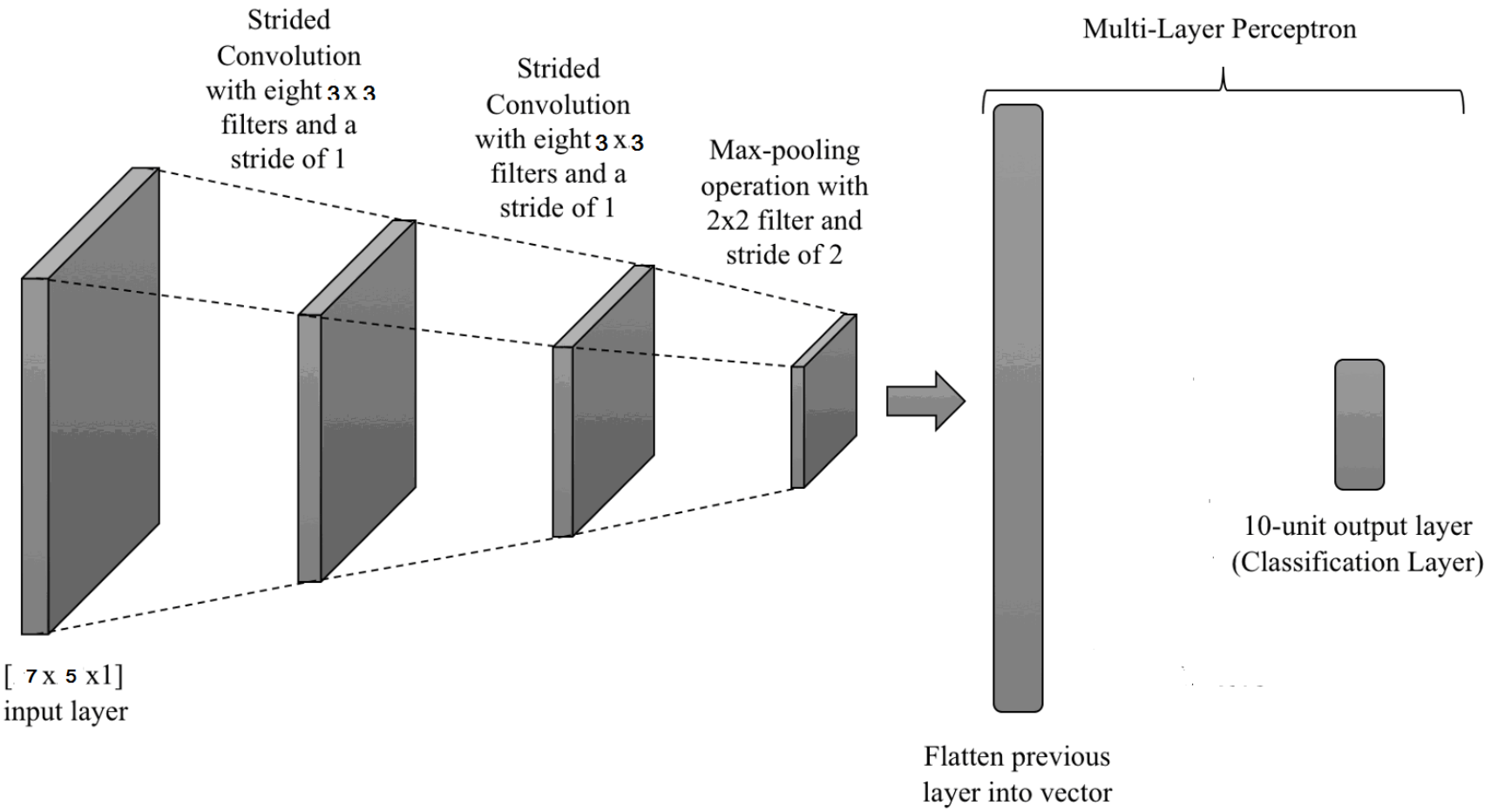
Відповіддю в нашому випадку є деяке число; у ширшому сенсі відповідь представляє категорію. Для отримання загального показника втрат, характерного для усіх категорій в цілому, беруть середнє від значень по кожній категорії.

Приклад CNN

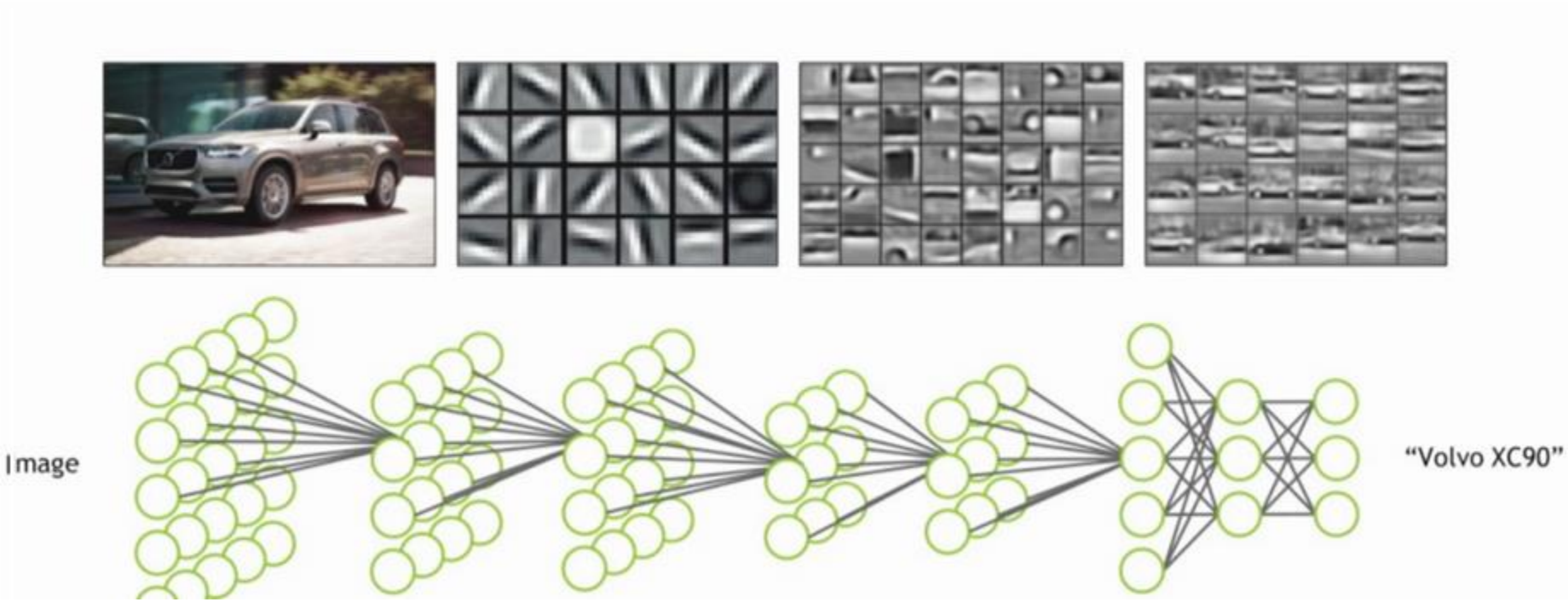
Розглянемо згорткову НМ для розпізнавання цифр.

У нашому розпорядженні відносно невелика кількість класів (10, якщо бути точним). Розмір зображень в навчаній множині складає 7x5 пікселів. З цих причин архітектура нейронної мережі буде досить простою:

- два згортальні шари і один шар максимального об'єднання (max pooling layer) для витягання характеристик початкового зображення;
- багатошаровий перцептрон (Multi - Layer Perceptron, MLP), завданням якого є класифікація отриманих раніше характеристик зображення.


Strided Convolution with eight 3 x 3 filters and a stride of 1
Strided Convolution with eight 3 x 3 filters and a stride of 1
Max-pooling operation with 2x2 filter and stride of 2
Multi-Layer Perceptron
[7 x 5 x 1] input layer
Flatten previous layer into vector
10-unit output layer (Classification Layer)

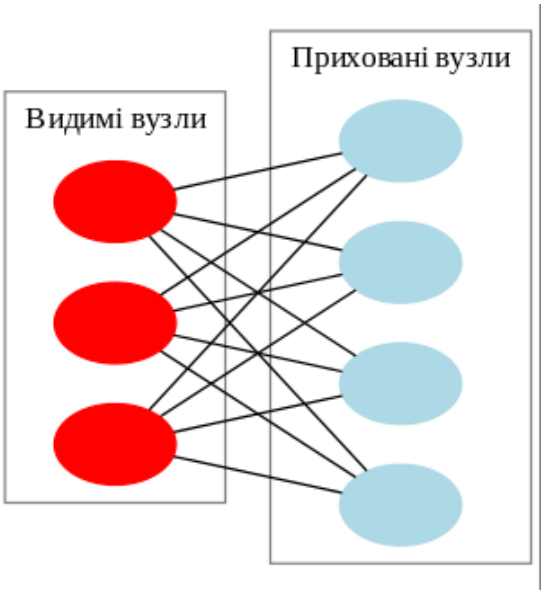
Свёрточные слои учат иерархические признаки для изображений, а spatial pooling даёт некоторую инвариантность к перемещениям.



Попереднє навчання CNN

1. Навчання з допомогою обмеженої машини Больцмана

Restricted Boltzmann machine, RBM (2006)



Для глибокого навчання починаємо із вхідних значень, а потім після навчання значення прихованих вузлів стають вхідними для наступного шару. Процес закінчується на вихідних вузлах.

Навчання відбувається без учителя.

Значення вагових параметрів зв'язків є початковими для подальшого навчання НМ з учителем.

Попереднє навчання CNN

2. Навчання з допомогою ініціалізації Ксав'є

Автори: Ксав'є Глоро і Йошуа Бенджі *(2010)*

Метод застосовується тільки для антисиметричних функцій активації, типу tanh(s).

Формула ініціалізації для чергового шару

$$W_{ij} \approx U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

U- рівномірний розподіл випадкових чисел

n_{in} *– кількість нейронів у попередньому шарі;*

n_{out} *– кількість нейронів у наступному шарі.*

Попереднє навчання CNN

3. Навчання з допомогою ініціалізації Хе

Автор: Каймінг Хе *(2015)*

Метод застосовується тільки для несиметричних функцій активації, типу ReLU*(s).*

Формула ініціалізації для чергового шару

$$W_{ij} \approx N\left[0, \frac{\sqrt{2}}{\sqrt{n_{in}^{(1)}}}\right]$$

N - нормальний розподіл випадкових чисел

$n_{in}^{(1)}$ *– кількість нейронів у вхідному шарі;*

Нормалізація по міні-батчах CNN

Міні- батч – це випадкова частина навчальної вибірки, за якою уточнюється градієнт на кожному кроці навчання. Цей процес називається стохастичним градієнтним спуском. Він займає менше часу і ресурсів, чим навчання по всьому дата сетові.

Виявилось, що таке навчання суттєво прискорюється, якщо проводити нормалізацію вхідних і проміжних даних по міні-батчах.

Сергій Йоффе, Крістіан Сегеді (2015).

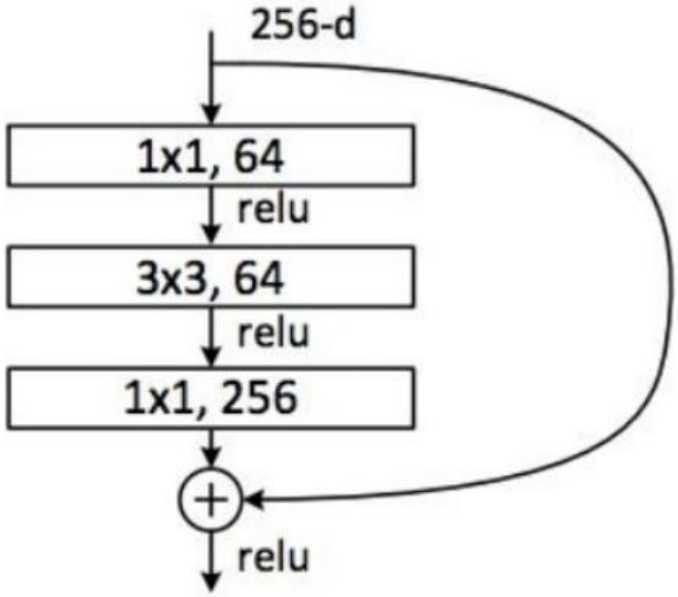
- **Формули нормалізації:**

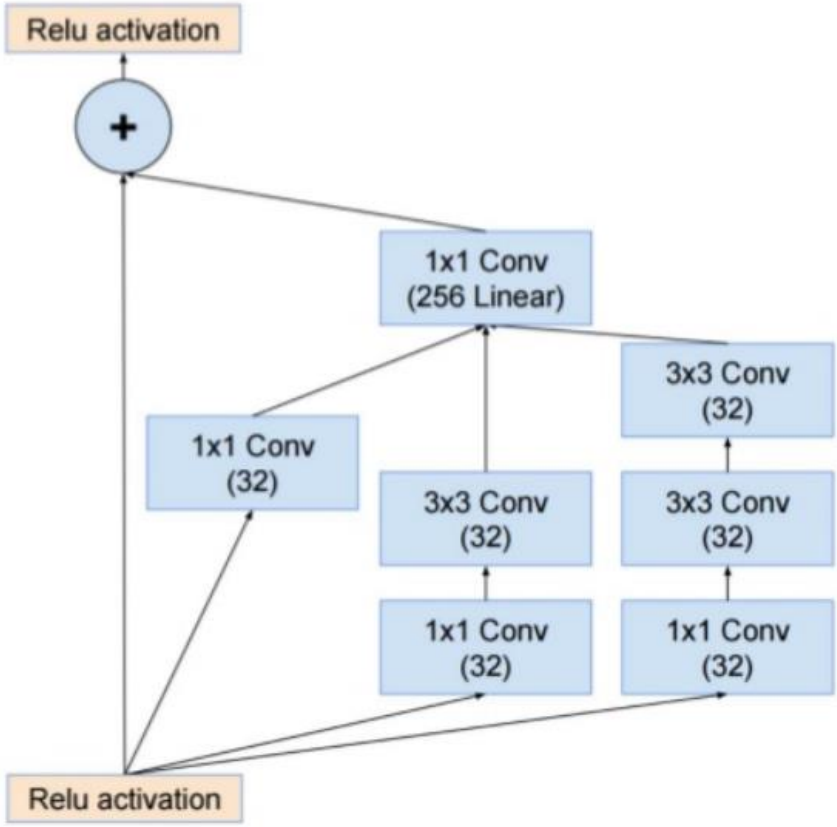
$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

Современные архитектуры

Inception, ResNet и другие современные архитектуры содержат специальные блоки слоёв.

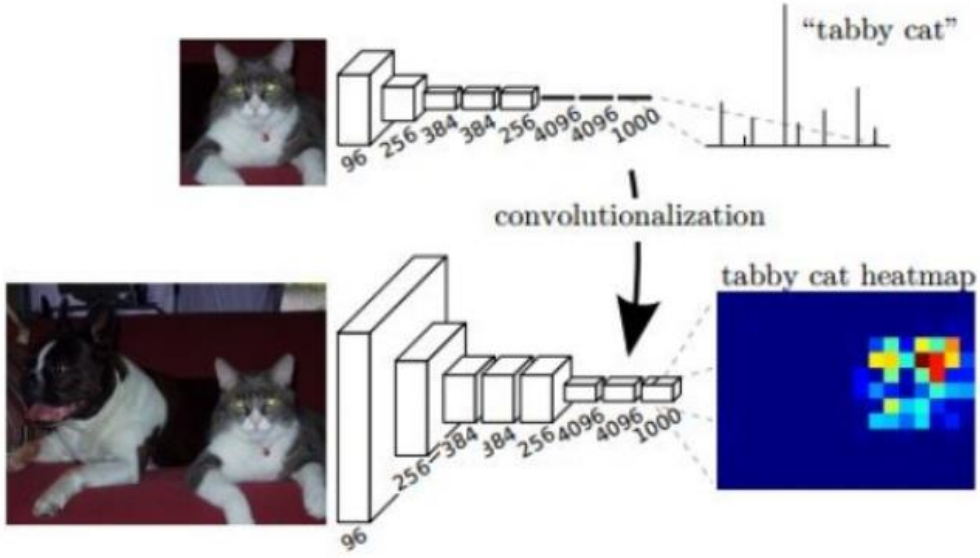




Fully-convolutional networks (FCN)

Обычная свёрточная сеть, но без MLP сверху (нет полносвязных слоёв).

Позволяет работать с изображениями произвольного размера и выдавать на выходе тепловую карту классификации.

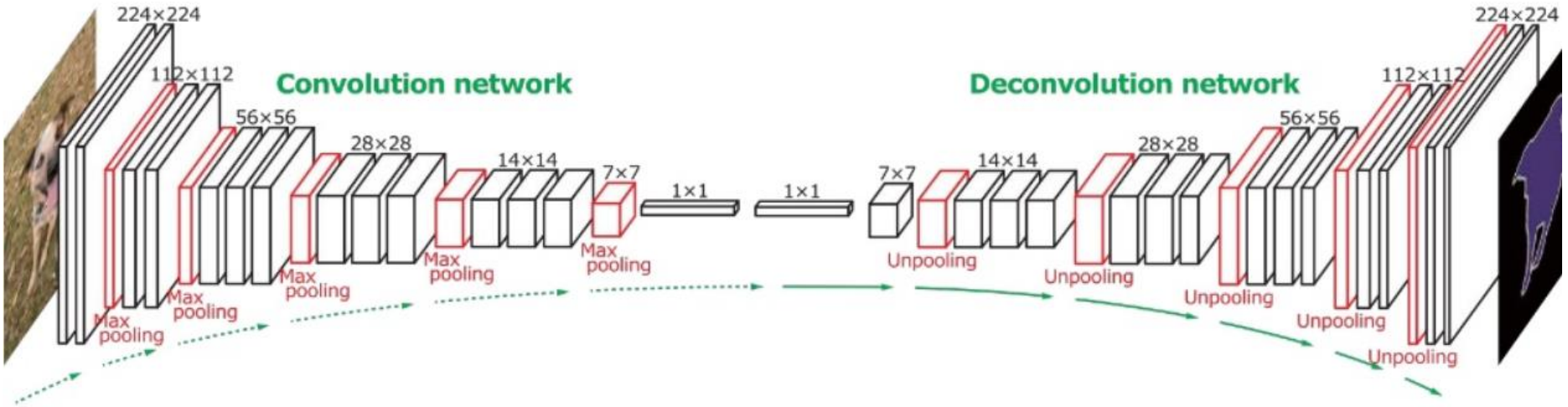


https://arxiv.org/abs/1411.4038

Deconvolution networks

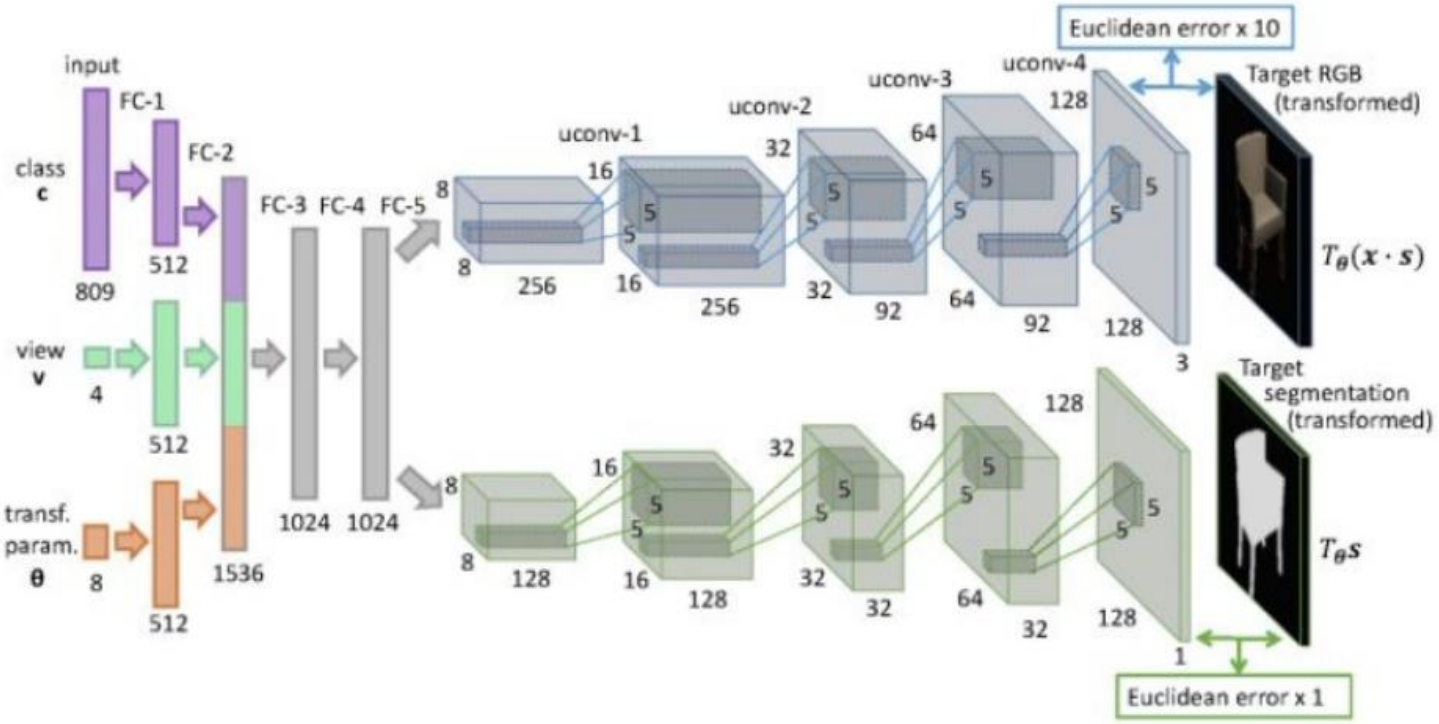
Правильнее называть это Transposed convolution, а не Deconvolution (это слово уже занято в цифровой обработке сигналов для обратной операции).

По сути, реализован обучаемый upsampling.



http://cvlab.postech.ac.kr/research/deconvnet/

Генерация изображений

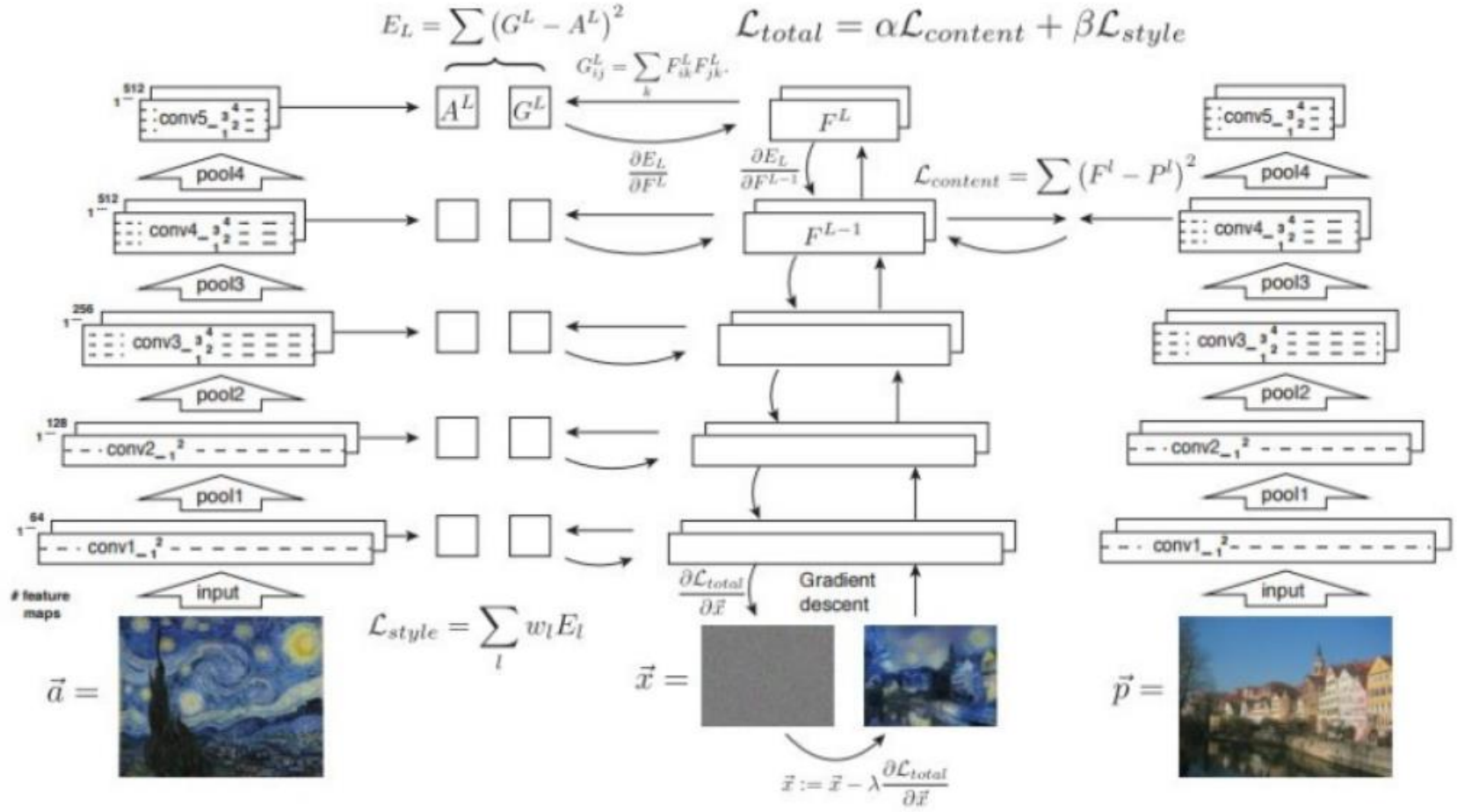


https://arxiv.org/abs/1411.5928

Неклассические задачи: перенос стиля

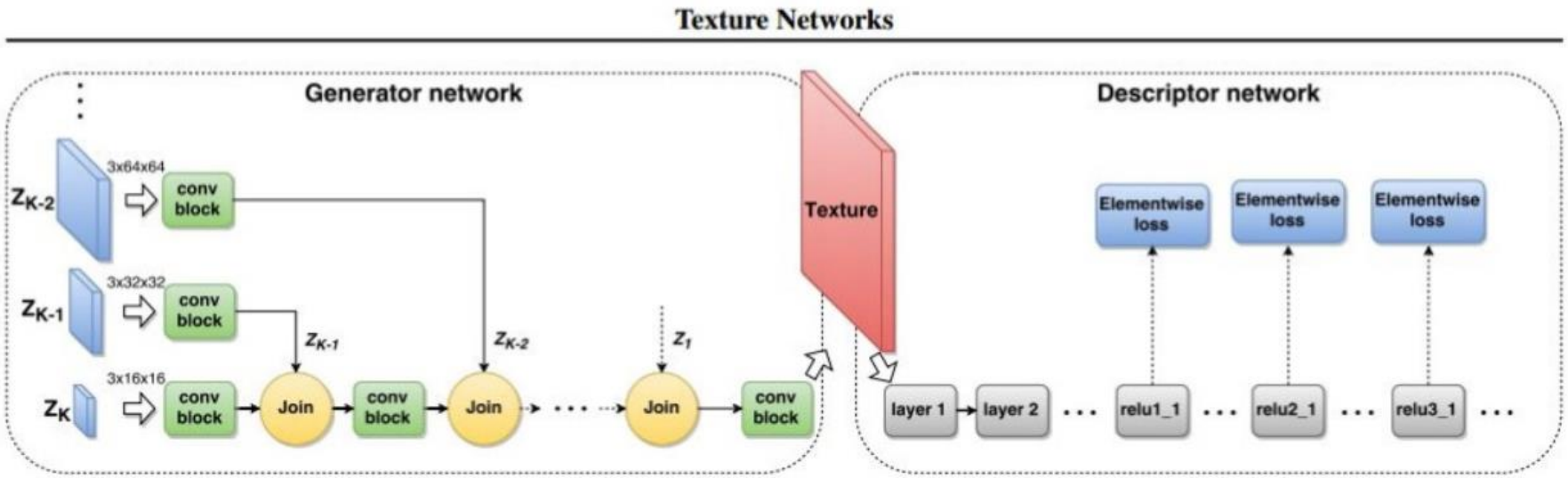
https://arxiv.org/abs/1508.06576

Перенос стиля: оригинальный алгоритм



https://arxiv.org/abs/1508.06576

Перенос стиля: быстрый алгоритм

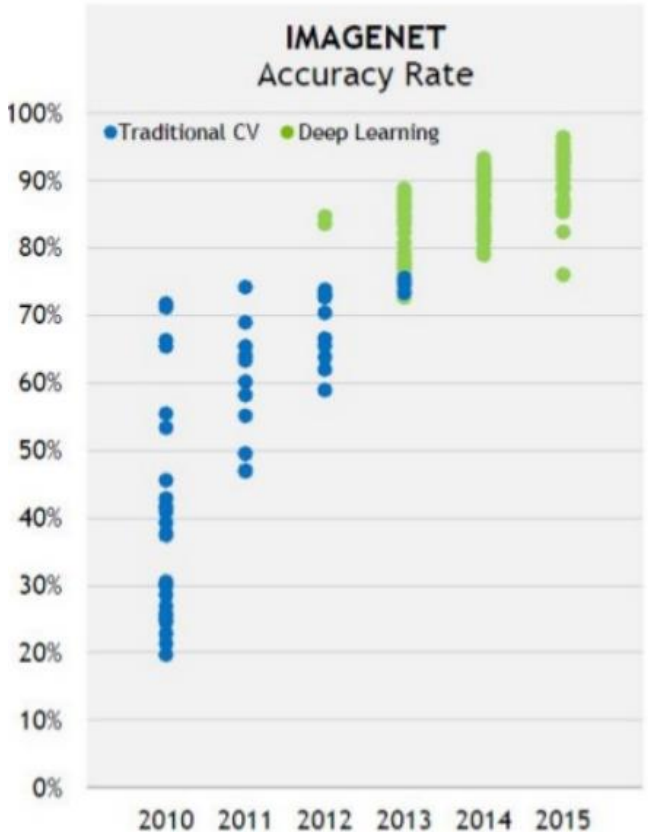


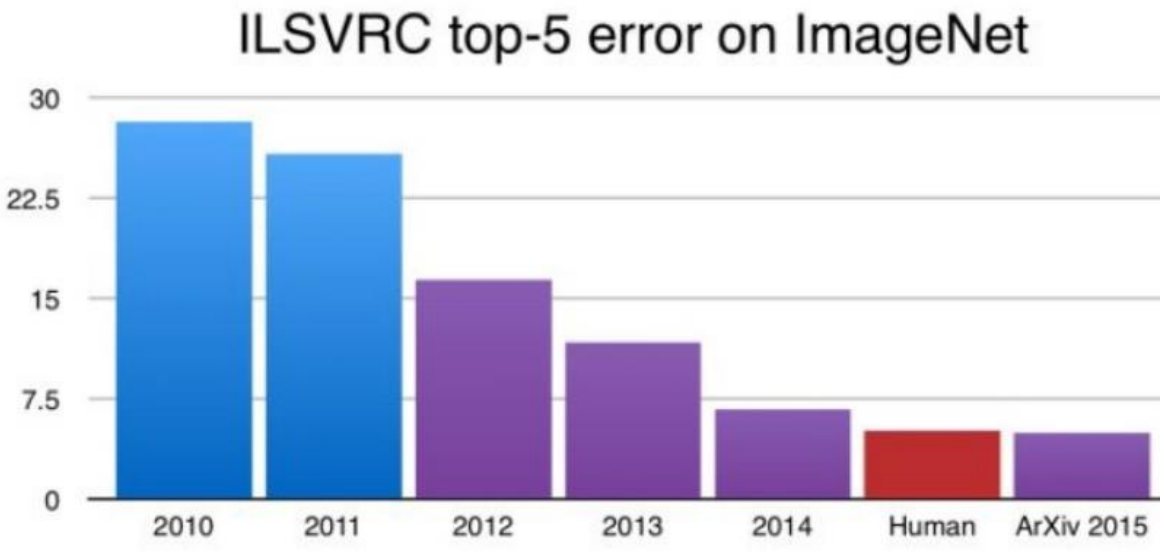
https://arxiv.org/abs/1603.03417

Recap: Важные тренды

Что сейчас происходит с качеством и сложностью моделей

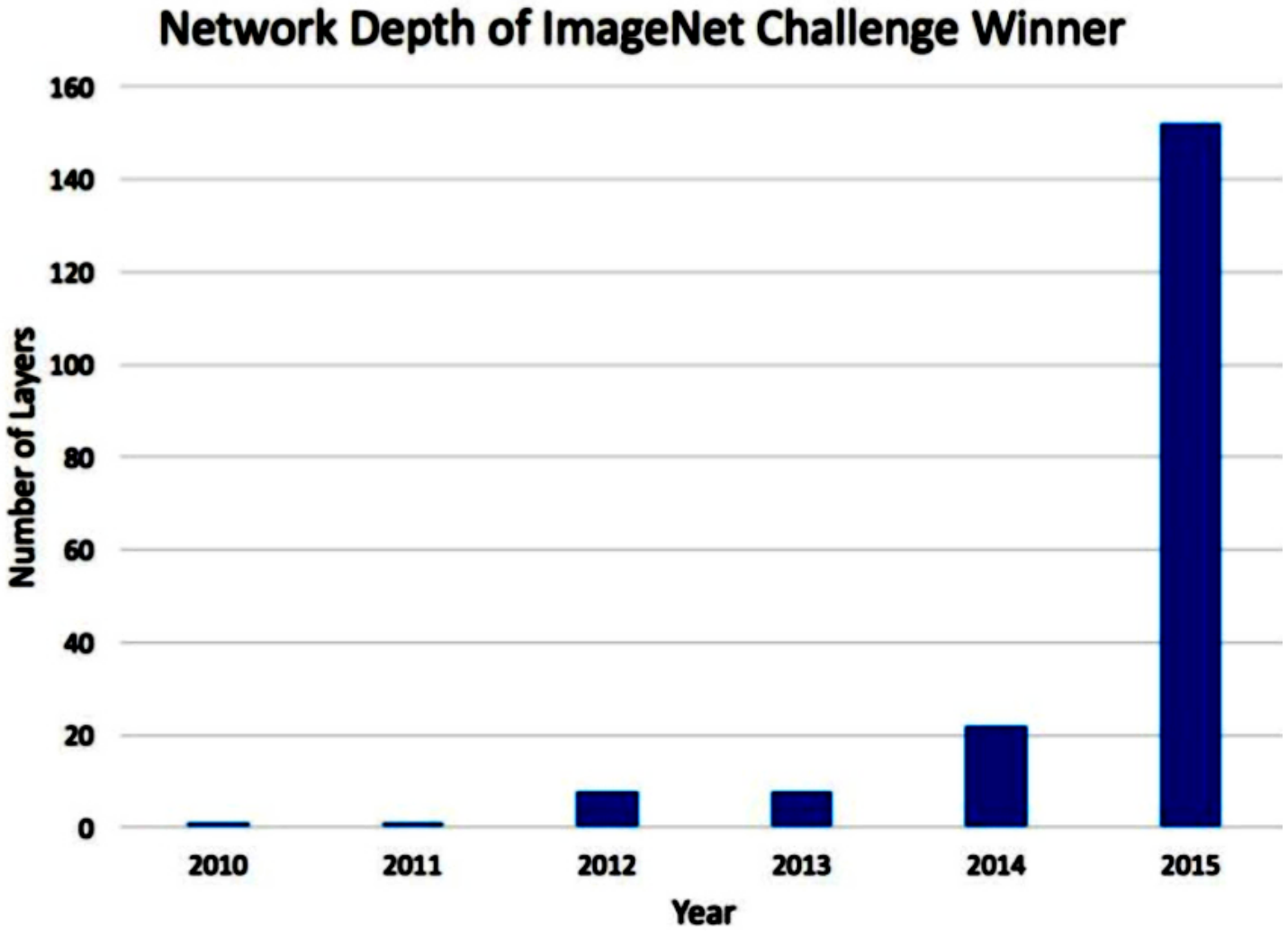
#1. Точность сетей растёт



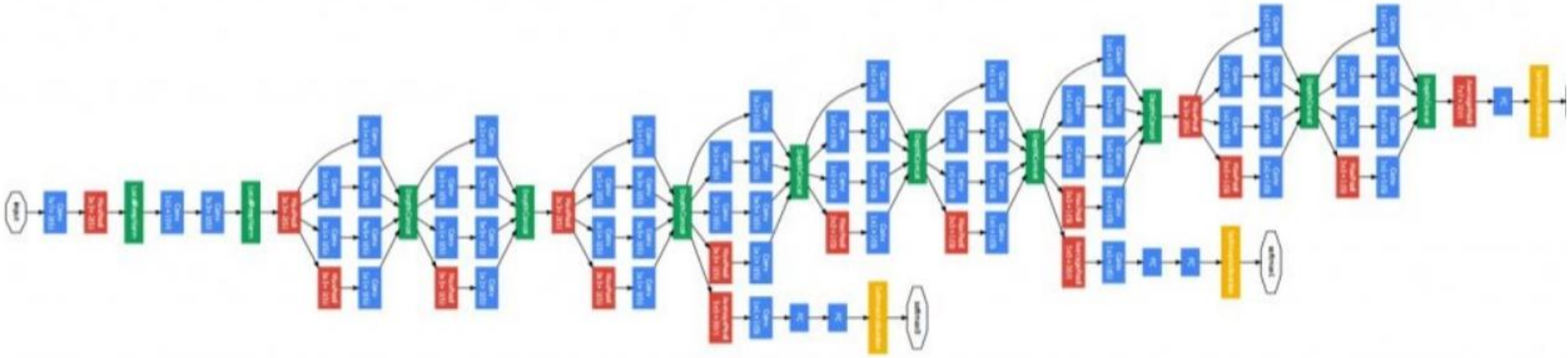


- Blue: Traditional CV
- Purple: Deep Learning
- Red: Human

#2. Сложность сетей растёт

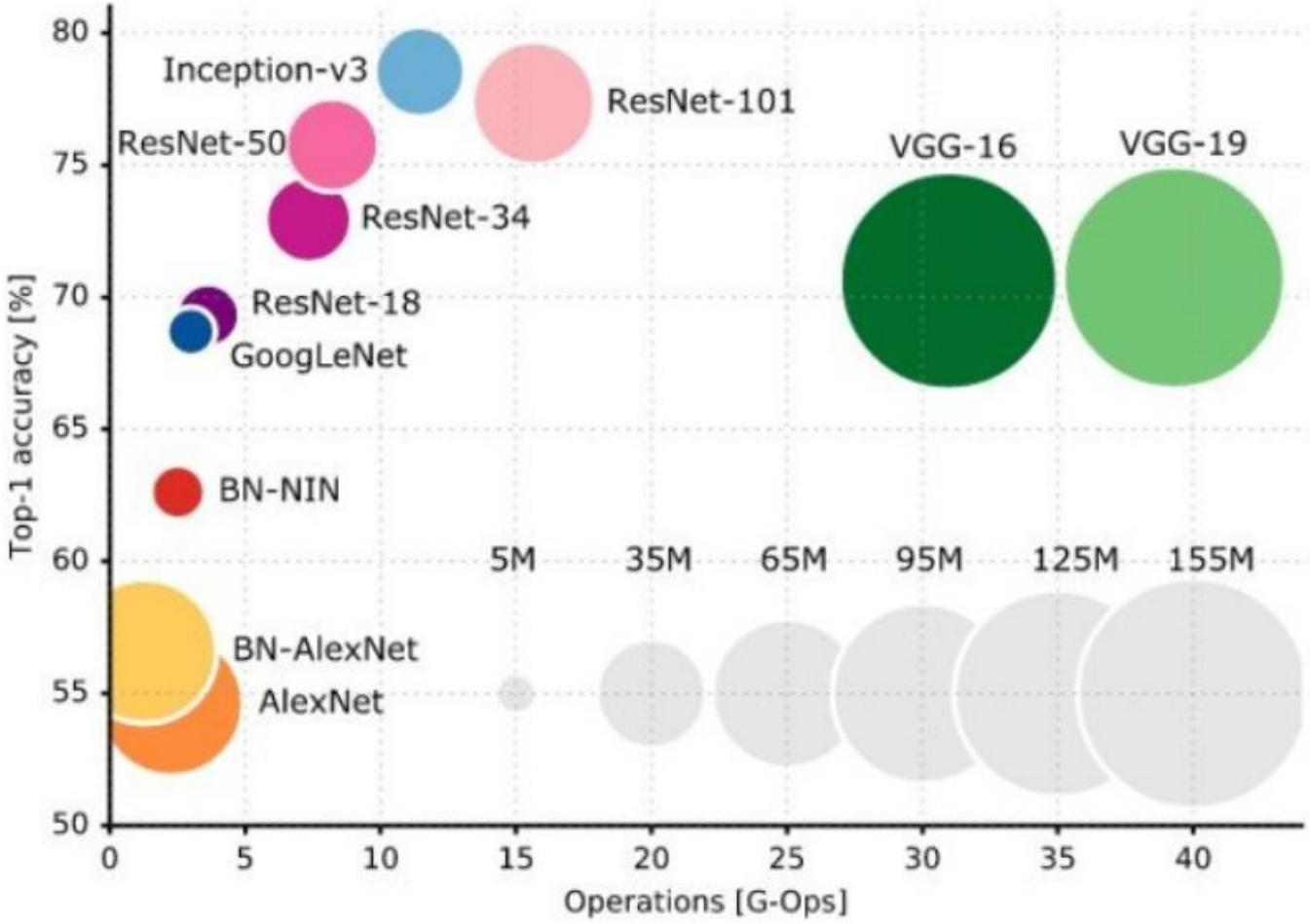


#2. Сложность сетей растёт



Реальная нейросеть: GoogLeNet (2014)
http://cs.unc.edu/~wliu/papers/GoogLeNet.pdf

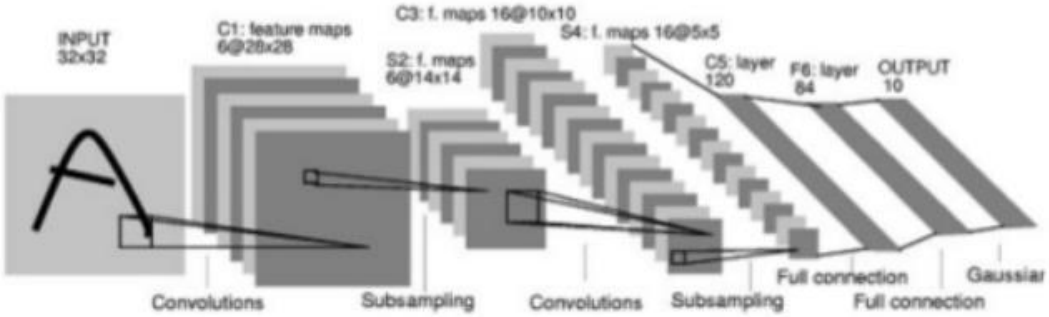
#2. Сложность сетей растёт



#3. Объёмы данных растут

1998

LeCun et al.



# of transistors

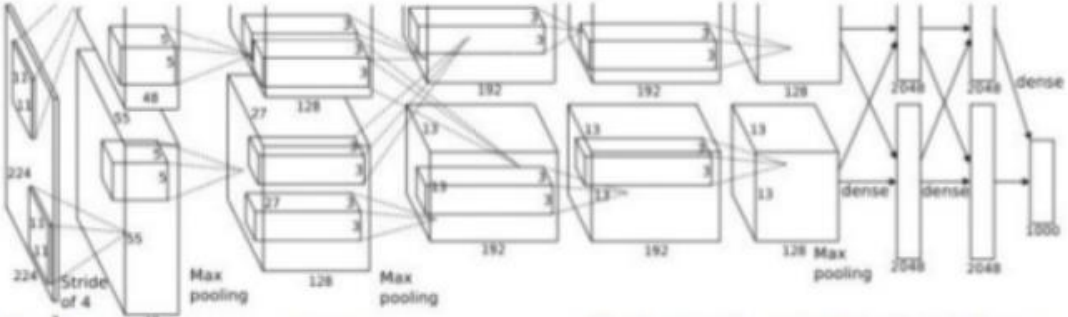
10^6

# of pixels used in training

10^7 NIST

2012

Krizhevsky et al.



# of transistors

10^9

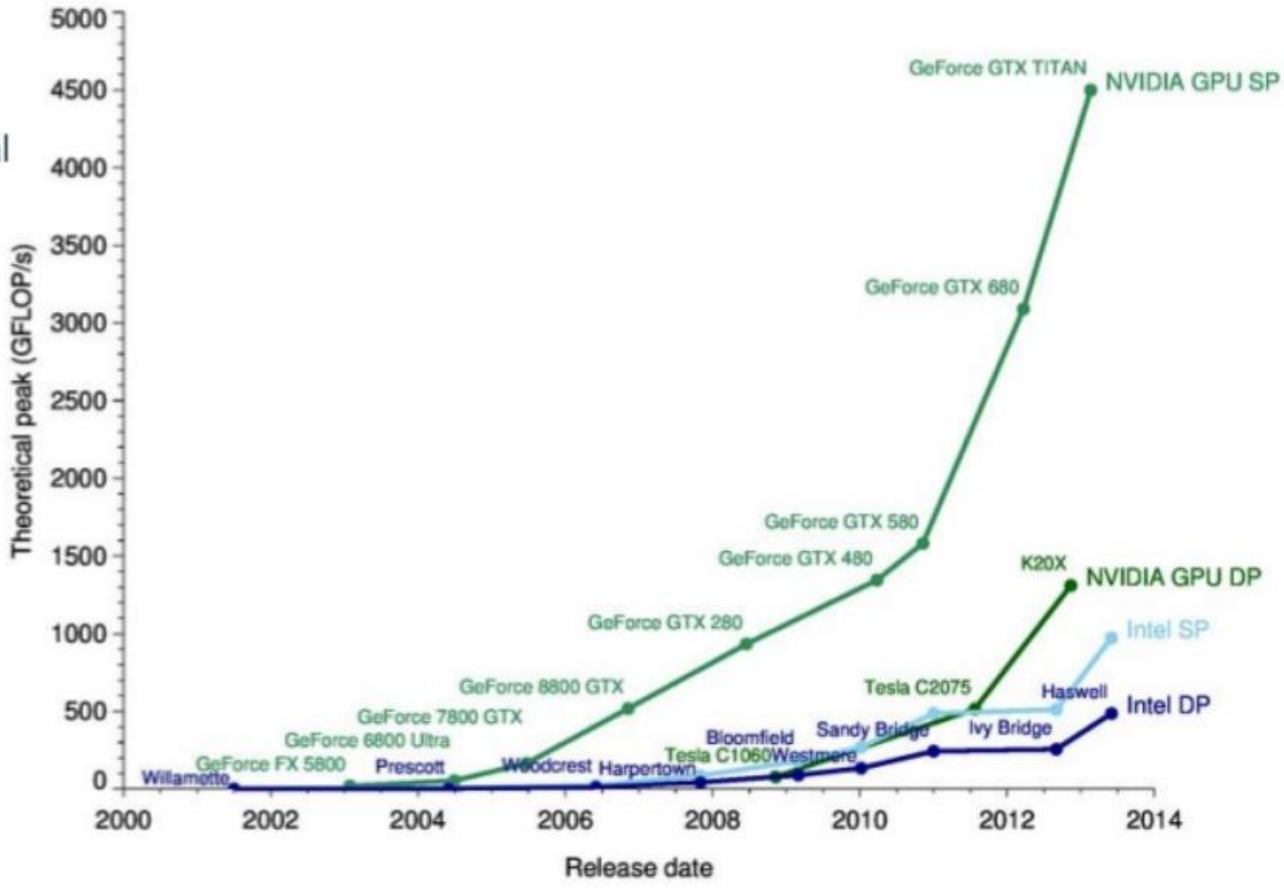
GPUs

# of pixels used in training

10^{14} IMAGENET

#4. Вычислительные мощности растут

- NVIDIA DGX-1 ($129,000)
  - 170 TFLOPS (FP16)
  - 85 TFLOPS (FP32)
- NVIDIA GTX Titan X Pascal ($1000)
  - 11 TFLOPS (FP32)
- NVIDIA GTX 1080
  - 8 TFLOPS (FP32)
- NVIDIA GTX Titan X Old
  - 6.1 TFLOPS (FP32)
- NVIDIA Drive PX-2
  - 8.0 TFLOPS
- NVIDIA Drive PX
  - 2.3 TFLOPS
- Intel Core i7-6700K
  - ~0.1-0.2 TFLOPS



Фреймворки и библиотеки для работы с нейросетями

Подробный список: http://deeplearning.net/software_links/

1. Theano - Компілятор символічних виразів для CPU / GPU в Python (від лабораторії MILA в університеті Монреаля)
2. Torch - забезпечує Matlab-подібне середовище для сучасних алгоритмів машинного навчання в lua (від Ronan Collobert, Clement Farabet і Koray Kavukcuoglu)
3. Pylearn2 - це бібліотека, призначена для легкого вивчення машинного навчання.
4. Blocks - основа Theano для навчання нейронних мереж
5. Tensorflow - бібліотека програмного забезпечення з відкритим кодом для чисельного обчислення з використанням графіків потоків даних.
6. MXNet – це глибоке навчальне середовище, призначене як для ефективності, так і для гнучкості.
7. Caffe - це глибоке навчання, створена з урахуванням виразів, швидкості та модульності.
8. Lasagne –це легка бібліотека для побудови та навчання нейронних мереж у Theano.
9. Keras-Theano based deep learning library.
10. Deep Learning Tutorials - приклади того, як робити глибоке навчання з Theano (від лабораторії LISA у Монреальському університеті)

11. Chainer - Генератор нейронних мереж на базі GPU
12. Matlab Deep Learning - Інструменти глибокого навчання Matlab
13. CNTK - Computational Network Toolkit - це уніфікований набір інструментів для глибокого вивчення, який проводить Microsoft Research.
14. MatConvNet - інструментарій MATLAB, що реалізує Convolutional Neural Networks (CNNs) для комп'ютерних програм зору. Це просто, ефективно, і може запускати і вивчати найсучасніші CNN.
15. DeepLearnToolbox - інструментарій Matlab для глибокого навчання (від Rasmus Berg Palm).
16. BigDL. Розроблена бібліотека з відкритою копією Apache Spark, призначена для ефективного масштабування на декількох вузлах. CPU-оптимізований через MKL. Підтримка Scala та Python.

17. Cuda-Convnet - швидка реалізація C + + / CUDA згорткових (або більш загальних, просунутих) нейронних мереж. Він може моделювати зв'язність довільного рівня та глибину мережі. Буде робити будь-який спрямований ациклічний графік шарів . Тренування проводиться за допомогою алгоритму зворотного поширення.

18. Deep Belief Networks. Код Matlab для глибокого навчання мереж (від Руслана Салахутдінова).

19. RNNLM - Інструментарій для мов на базі періодичних нейромереж Томаса Миколова.

20. RNNLIB- це періодична бібліотека нейронних мереж для проблем, пов'язаних із вивченням послідовності. Застосовується до більшості типів просторових даних, що виявилося особливо ефективним для розпізнавання мови та рукописного тексту.

21. matrbm Спрощена версія коду Руслана Салахутдінова Андрея (Matlab).

22. Deeplearning4Jj- бібліотека з розподіленою нейронною мережею з відкритим кодом, написана на Java та Scala, з ліцензією Apache 2.0.

23. Estimating Partition Functions of RBM's. Код Matlab для оцінки функцій розділів обмежених машин Больцмана з використанням відібраного відбору відбитку важливості (від Руслана Салахутдінова).

24. Learning Deep Boltzmann Machines Matlab для навчання та точного налаштування Deep Boltzmann Machines (від Руслана Салахутдінова).

25. LUSH Мова програмування та середовище розробки LUSH, який використовується @ NYU для глибоких звивистих мереж

26. Eblearn.lsh - це бібліотека машинного навчання на базі LUSH для навчання на основі енергії.

27. Deepmat, Matlab, алгоритми глибокого навчання.

28. MShadow - MShadow - це легка бібліотека шаблонів матриці / тензора CPU / GPU у C ++ / CUDA. Метою mshadow є підтримка ефективної, інваріантної та просторої тензорної бібліотеки для машинного навчання

29. CXXNET - CXXNET - це швидка, лаконічна, розподілена система глибокого навчання, заснована на MShadow. Це легкий і легкий розширюваний набір інструментів для нейронних мереж C ++ / CUDA з дружнім інтерфейсом Python / Matlab для навчання та прогнозування.

30. Nengo- це графічний та сценарійний пакет програмного забезпечення для моделювання великомасштабних нейронних систем.
31. Eblearn - це бібліотека машинного навчання C + + з ліцензією BSD для навчання на основі енергії, конвольні мережі, програми з бачення / розпізнавання тощо. EBLearn перш за все підтримується П'єром Серменетом в NYU.
32. CUDAMat - це матрична бібліотека на базі GPU для Python. Включено приклад коду для навчання нейронних мереж та обмежених машин Boltzmann.
33. Gnumpy - це модуль Python, який інтерфейсу майже такий самий як numpy, але здійснює його обчислення на GPU комп'ютера. Він працює на CUDAMat.
34. Бібліотека CUV Library github link) - це система C ++ з прив'язкою Python для зручного використання функцій Nvidia CUDAMat.
35. 3-way факторинг RBM та mcRBM - це код Python, який викликає CUDAMat для навчання моделей природних зображень (від Marc'Aurelio Ranzato).

35. 3-way факторинг RBM та mcRBM - це код пітона, який викликає CUDAMat для навчання моделей природних зображень (від Marc'Aurelio Ranzato).
36. Код Matlab для навчання умовних РБМ / ДБН та факторингових умовних РВР (від Грем Тейлора).
37. mPoT - це код пітона, що використовує CUDAMat і gnumpy для навчання моделей природних зображень (від Marc'Aurelio Ranzato).
38. neuralnetworks - це Java-бібліотека gpu для глибоких алгоритмів навчання.
39. ConvNet - це інструментарій сверточной нейронної мережі на базі MATLAB.
40. Elektronn - це глибокий навчальний набір інструментів, що робить потужні нейронні мережі доступними для вчених за межами спільноти машинного навчання.
41. OpenNN - це бібліотека класів з відкритим кодом, написана на мові програмування C ++, яка реалізує нейронні мережі - основну область глибокого вивчення досліджень.
42. NeuralDesigner - інноваційний інструмент глибокого навчання для прогнозної аналітики.
43. Theano Generalized Hebbian Learning.

44. Apache Singa - бібліотека глибоких навчальних програм із відкритим кодом, яка забезпечує гнучку архітектуру масштабованої розподіленої підготовки. Це розширюване обладнання для широкого кола обладнання та зосереджене на застосуванні медичних послуг.
45. Lightnet - це легка, універсальна та система суто глибокого навчання на базі Matlab. Мета розробки полягає в тому, щоб забезпечити легко зрозумілу, просту у використанні та ефективну обчислювальну платформу для глибокого вивчення досліджень.
46. SimpleDNN - це машина навчання легкої бібліотеки з відкритим кодом, написана в Kotlin, метою якої є підтримка розробки прямих і рекурентних штучних нейронних мереж.

Универсальные библиотеки и сервисы

- **Torch7** (http://torch.ch/) [Lua]
- **TensorFlow** (https://www.tensorflow.org/) [Python, C++]
- **Theano** (http://deeplearning.net/software/theano/) [Python]
  - **Keras** (http://keras.io/)
  - **Lasagne** (https://github.com/Lasagne/Lasagne)
  - **blocks** (https://github.com/mila-udem/blocks)
  - **pylearn2** (https://github.com/lisa-lab/pylearn2)
- **Microsoft Cognitive Toolkit (CNTK)** (http://www.cntk.ai/) [Python, C++, C#, BrainScript]
- **Neon** (http://neon.nervanasys.com/) [Python]
- **Deeplearning4j** (http://deeplearning4j.org/) [Java]
- **MXNet** (http://mxnet.io/) [C++, Python, R, Scala, Julia, Matlab, Javascript]

Обработка изображений и видео

- **OpenCV** (http://opencv.org/) [C, C++, Python]
- **Caffe** (http://caffe.berkeleyvision.org/) [C++, Python, Matlab]
- **Torch7** (http://torch.ch/) [Lua]
- **clarifai** (https://www.clarifai.com/)
- **Google Vision API** (https://cloud.google.com/vision/)
-

Распознавание речи

- **Microsoft Cognitive Toolkit (CNTK)** (http://www.cntk.ai/)
  [Python, C++, C#, BrainScript]
- **KALDI** (http://kaldi-asr.org/) [C++]
- **Google Speech API** (https://cloud.google.com/)
- **Yandex SpeechKit** (https://tech.yandex.ru/speechkit/)
- **Baidu Speech API** (http://www.baidu.com/)
- **wit.ai** (https://wit.ai/)

Обработка текстов

- **Torch7** (http://torch.ch/) [Lua]
- **Theano/Keras/**… [Python]
- **TensorFlow** (https://www.tensorflow.org/) [C++, Python]
- **Google Translate API** (https://cloud.google.com/translate/)