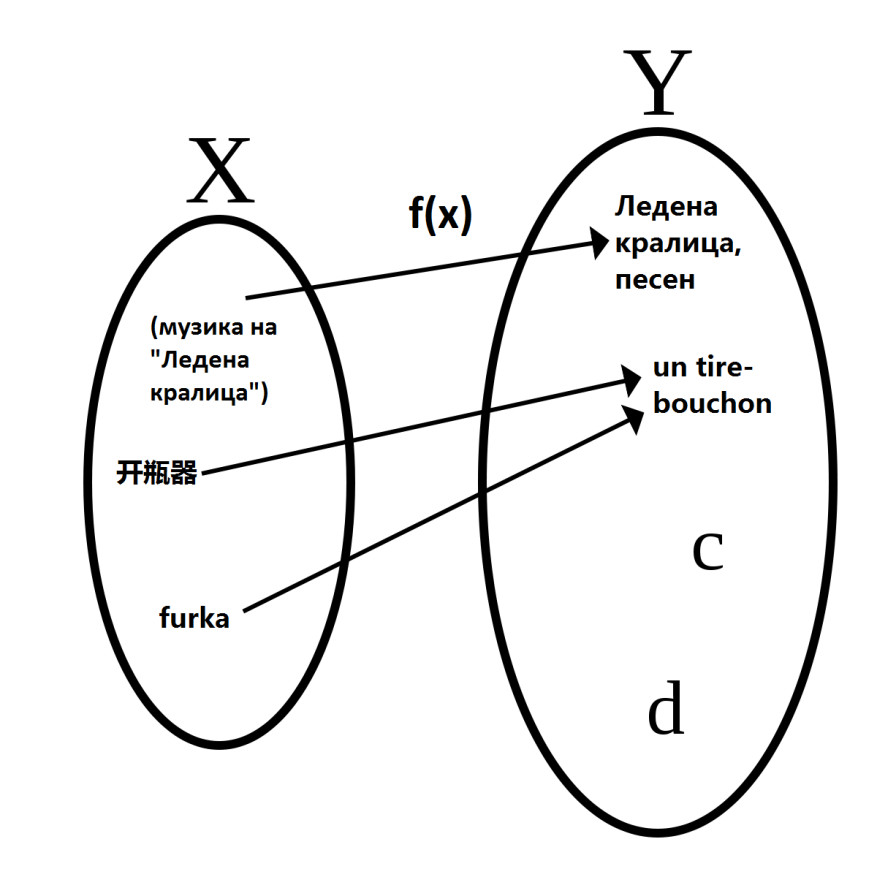
Апроксимация на функции с невронни мрежи.

Доклад към работещ прототип на невронна система за функционална апроксимация.

*Проект на Ал. Бакарски, Дарин и Ян*

***Общ преглед.***

Невронните мрежи в математиката и информатиката са били създадени като аналог на системи от неврони в мозъчната кора на биологични видове. Аналогията не е пълна, но в някои аспекти двете системи са изключително съпоставими. Целта също е заимствана от биологията – през процес на обучение да се постигнат конкретни резултати под формата на поведение. В информатиката процесът общо се нарича машинно обучение, подбраната информация за обучение са тренировъчни данни, а целта е изход под формата на конкретни данни. Всяка невронна система входен лейър (слой), скрити лейъри и изходен лейър. Има само един входен и един изходен в повечето случаи. Всеки слой има определен брой неврони. В този смисъл невронната мрежа е една функция, без значение дали е безкрайно проста или безкрайно сложна. Функцията в математически смисъл съпоставя на дадени входни данни определени изходни по някакъв алгоритъм, а невронната мрежа е пример точно за това. Всяка НМ има своя цел – например превод на даден текст от китайски на френски. Входни данни: китайски, изходни: френски. Това е функция, в математически смисъл. Невронна система, тренирана да разпознава музика, също е просто една комплексна функция. На дадени входни аудио данни съпоставя изходни данни от датабаза със заглавия на музики.



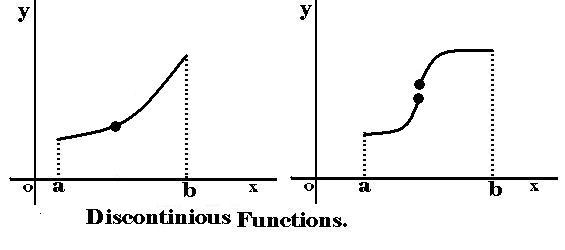
Невронните мрежи и машинното обучение в последните години претърпяха изключителен подем. Изкуственият интелект, който познаваме днес, чето е основан на някакъв вид невронна система. Примерите са много: от Шазам, Гугъл асистент, Фотомат, до GPT-3, която може да ви напише есето, да програмира уебсайт по дадено вербално описание, да рисува картини по вербално описание и още впечатляващи неща, и AlphaFold, която разреши 100-годишен проблем със структурата на протеините. Не случайно някои хора наричат 21 век златната ера на ИИ, именно защото потенциалът в сферата е практически неограничен.

1. ***Апроксимация на функция и теорема за универсалност.***

Ще се фокусираме на частен случай от невронните системи: целта е да обучим мрежа, която да предсказва изхода от дадена функция с голяма точност. Това се нарича функционална апроксимация и се използва широко в съвременната наука. Нека обаче да започнем с една теорема, на базата на която решихме да работим в тази посока, защото кой би тръгнал да прави нещо, ако не е сигурен, че то е възможно?

Теорема за универсалност:

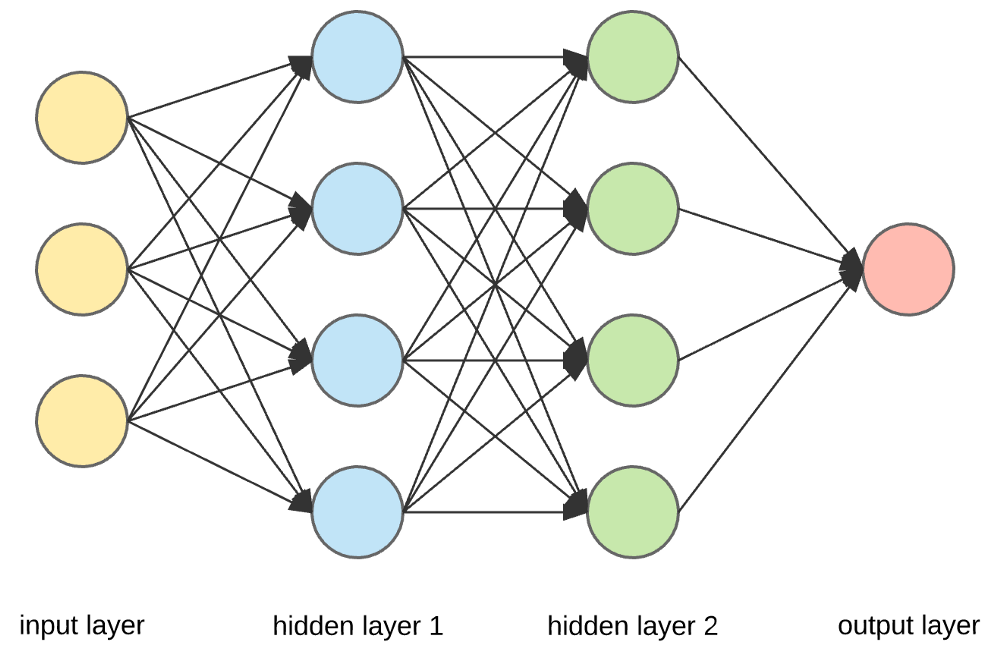
1. Невронни мрежи не могат да бъдат използвани за ТОЧНО изчисление на функции, а само за апроксимация.
2. Винаги можем да съставим невронна мрежа (при дадени достатъчно данни за обучение и компютърна мощ) за всяка изисквана точност ε>0, за която |g(x)-f(x)|< ε, където g(x) е изходът на функцията от невронната мрежа за някое х в интервала от данните за обучение и f(x) е реалната стойност на функцията за същото х. Това важи и за функции с повече от една променлива f(x, y, z…).
3. Невронни мрежи могат да апроксимират само непрекъснати функции.



Разбирането на матрици и векторни произведения, както и частни производни и мултипроменливи функции ще бъде в полза, но не е строго необходимо за схващането на главния механизъм на действие на невронните мрежи от този тип.

1. ***Как функционират невронните мрежи?***

**f(x)=2x**



* Всеки неврон съдържа число от 0 до 1 (0.8975395 е валидна стойност)
* Във входния лейър подаваме входните данни, в нашия случай х, както във всяка функция. Но понеже невроните работят с числа от 0 до 1, то десетична система не ни устройва. Конвертираме числото х в двоична система.

Х=3 -> Х(2)=001

Тогава в първия лейър (входния) винаги числата са 0 или 1, и са двоична репрезентация на числото Х. Колкото повече неврони имаме във входния лейър, толкова по-големи числа можем да вкарваме в невронната мрежа. В случая с три неврона: можем да вкараме 2^3 или цели осем числа: в случая от 0 до 7 в десетична. Ако х ни е 7, то числата и в трите неврона щяха да бъдат 1: 111. Числата над 8 и под 0 няма как да използваме. Решението: Правим системата 64 битова, а не 3 битова. Така имаме 2^64 варианти за числа и няма да имаме притеснения с това колко големи (или малки) числа вкарваме. Как да различим в бинарен код числото 42 от -42? Заделяме един бит (неврон) от 64-те за знака. Ако невронът е 0, то знакът е +, ако е 1, то е -. Ако решим този неврон да е първия, то числото -7 ще представим така:

100000000000000000000000000000000000000111 (нулите са 64-4=60)

Така „превеждаме“ входната информация за работа на невронната мрежа.

* Изходният лейър също съдържа неврони, които имат стойност от 0 до 1. Понеже резултатът от математическата функция е число в десетична система, отново трябва да се конвертира, но този път от двоична в десетична, за да получим отговора. Тоест, след като вкараме входното число в първия лейър, НМ си върши работата и връща стойност в последния лейър, които ние конвертираме в десетична система, по-лесно разбираема от нас.

В заключение „превеждането“ от десетична в двоична и обратното се случва в първия и последния лейър.

* Всяка от свързващите линии между невроните на последователни слоеве представляват уейтове (тежести) – реални числа, без значение дали са положителни или отрицателни и в какъв интервал. Докато обучаваме невронната система, алгоритъм променя всеки уейт, за да може крайният резултат да е по-добър с всяка следваща итерация на обучителния процес.
  1. ***Прехвърляне на стойности напред в мрежата***

Нека вземем НМ с 2 лейъра (входен и изходен), всеки с по един 1 неврон (най-простата НМ):



Взимаме числото в неврон 1 (между 0 и 1), умножаваме го по уейта w1 (някакво реално число) и към произведението добавяме трето число, наречено байъс (склонност). То също може да бъде всякакво реално число.

Y1 = X1\*w1 + b1

Тук имаме проблем. Числото Y1 е стойност на неврон, значи трябва да е между 0 и 1. Затова по някакъв начин трябва да „вкараме“ произведението X1\*w1 + b1 в интервала (0; 1). Това най-лесно става с функцията сигмоид, която винаги връща стойност между 0 и 1:

D:\Users\User\Desktop\NumberedEquation1.gif

Значи



За удобство ще изписваме:



Така невронът в следващия лейър получава своята стойност. Тук трябва да отбележим, че сигмоид може да бъде заменена с друга активираща функция, която да дава резултат между 0 и 1. В нашия случай сме използвали сигмоид.

В по-сложни НМ, всеки неврон е свързан с всеки друг от следващия слой с уейт . Така стойността на един неврон от лейър, ако предишния е с 64 неврона, ще бъде:



Забелязваме, че не всеки уейт има собствен байъс, а за всички уейтове към един неврон от следващ слой има един байъс.

Какъв е смисълът от уейтовете? Те определят колко значение за крайната стойност има даден неврон в предходния лейър. Да кажем (в горния пример с 64) w1 e 5, a w2 e 25. Тогава неврон 2 от предходния лейър ще има 5 пъти повече тежест за резултата, тоест стойността на изследвания неврон от следващия лейър. Това е класически пример за претеглена сума.

Какъв е смисълът на байъса? Той определя колко голяма трябва да е претеглената сума, за да се активира достатъчно много неврона. Ако б1 е много малко число (-150), невронът ще се активира слабо (да кажем 0.05) дори при голяма претеглена сума, тоест дори предходните неврони да са силно активирани. Ако е голямо число (150), то изследваният неврон ще се активира с вискоа стойност (0.95) дори при малък сбор от предходните неврони.

* 1. ***Инициализация.***

Преди да започне обучението, всички уейтове и байъси се взимат на произволен принцип. Да вземем функцията f(x)=2x. Подаваме на невронната мрежа числото 7. Да проследим:

1. Числото се конвертира в 000000000000000000000000111. Всеки от 64те неврони в първия лейър получава стойност: първия 0, втория 0…. 62 получава 1, 63 и 64 също.
2. На базата на случайните уейтове и байъи, тези числа преминават през още 2 (скрити) лейъра, да кажем всеки с 16 неврона , и стигат в изходния 64-невронен лейър.
3. Получаваме за първия неврон числото 0.0672345, за втория 0.6782435, за третия 0.988835235 и т.н. Как да го преведем в десетична? Ако числото е под 0.5, то се закръгля на 0, ако е над 0.5, се закръгля на 1. Така имаме краен резултат в бинарен код:

011…………………………

1. Конвертираме го към десетична. Числото се оказва 8723578295728385978293576920304010458892468.

А не 14. Какъв ужас.

Време е да накажем НМ за това чудовищно число, оскверняващо математическите принципи. Но с произволни уейтове и байъси – толкова! Предстои обучение на мрежата.

* 1. ***Обучение на НМ – Кост функция***

Първо трябва да установим математически, че имаме проблем и колко голям е той. Това става с Кост функция, която изчислява колко „далече“ е била НМ в своето предсказание.

Разбира се, това предполага обучение като при малко дете: трябва да му дадеш готов и решен пример, за да успее в бъдеще само да си решава задачите. Тук обаче става въпрос за проблемно дете: трябва да му приготвиш стотици хиляди примери понякога. Тези примери се наричат обучителни данни, изглеждат по този начин:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Х | 5 | 3 | 6 | 7 | 18 | -4 |
| У | 10 | 6 | 12 | 14 | 36 | -8 |

Ние лесно съобразяваме, че става въпрос за функцията у=2\*х, но проблемното дете НМ ще му трябват много повече решени примери.

Но то има едно предимство:

|  |  |  |  |
| --- | --- | --- | --- |
| Х | 5 | 3.14 | -2 |
| У | 2453.87 | 1223.87 | -2356 |

Коя е тази функция? Убеден съм, че дори след 10000 примера няма да разберете коя е. Но проблемното дете – НМ ще може да заключи убедително, че това е всъщност:

-1000\*sin(x)+2000\*cos(3\*x^2-487x+69)

За пример да вземем НМ с един неврон за изходен лейър. Да кажем, че в конкретния случай искаме неврона да е активиран с единица. Ние знаем какво сме подали на неронната мрежа и знаем какво искаме тя да изкара. Понеже не сме я обучили още, тя ще върне 0.02834756 или може би 0.98562954 за този един неврон. Искаме 1. Тогава големината на проблема ни е:

, където у1 е получената стойност (0.0283424…), а у е исканата стойност: 1.

В случая имаме Cost = (0.0283-1)^2 =0.944. Колкото по-далече е получената стойност от исканата стойност, толкова по-голямо е числото от Кост функцията, толкова по-голям проблем имаме.

Когато имаме многоневронен аутпут лейър (64), Кост функцията ще е просто сборът на всички тези квадрати на разликата от исканата и върната стойност. В нашият случай с числото 7:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Неврон | 1 | 2 | … | 64 |
| Какво ни е върнало? | 0.0241642 | 0.98356134 | … | 0.446324246 |
| Какво искаме | 0 | 0 | … | 1 |

Изчисляваме Кост функцията:

Cost = (0.0241642-0)^2 + (0.983-0)^2+ ……………. +(0.4463-1)^2

В случая получаваме 28.04. Толкова голям ни е проблемът. Ясно е, че целта е да намалим стойността, тогава системата ще е много по-точна.

Как да я намалим?

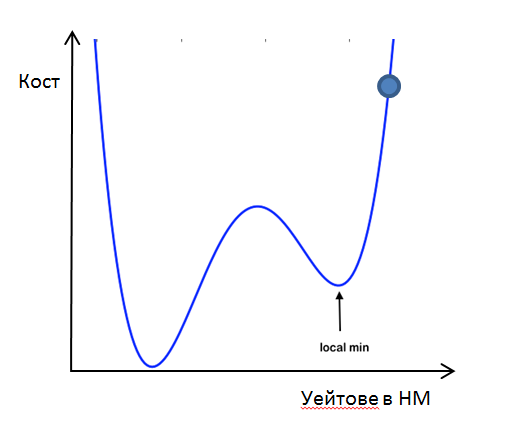
* 1. ***Обучение на НМ – градиентно спускане***

Мултипроменлива функция:

f(x,y) = 2x+8y => f(2,3) = 2.2 + 8.3 = 28

Функциите с една променлива най-лесно се представят с графики в 2 измерения, функциите с 2 променливи: в графики с 3 измерения (една равнина за двете променливи и съответно височина (обем) за резултатът от функцията). Ако функцията беше с 3 променливи, няма как да съберем всичко на една графика (късмет с виденията в четвъртото измерение!).

В случаят нашата Кост функция е с променливи ВСИЧКИ уейтове в НМ. Защо? Когато променим уейтовете, се променя и резултатът, който връща НМ, оттам и Кост стойността („променлива“ – променя стойността на дадена функция). В нашия случай имаме 768 уейта, значи 768 променливи във функцията. Това не е особено обнадеждаващо, имайки предвид, че не можем физически да погледнем в 3 измерени, още повече в 769. Затова има един трик – набутваме всички 768 променливи в една „много сложна“ променлива, която изобразяваме като абсцисата на нашата проста, двуизмерна графика на Кост функцията (изобразена на ординатата):



Разбира се, компютърът работи с всички 768 уейтове поотделно, а не като една „много сложна“ променлива. Но за целта ни и тази графика ще свърши работа.

Каква е целта оттук нататък? За да направим нашата НМ по-точна, тоест да апроксимира функции по-точно, трябва да намалим стойността на Кост функцията. Това се нарича градиентно спускане. Да погледнем част от графиката на нашата Кост функция. Да кажем, че при първото произволно избиране на уейтове, след първата итерация, Костът излиза 28.04. Това е синята точка на графиката. Наблюдавайки тази 768-пъти опростена графика, на нас ни е ясно, че трябва да се движим „наляво“, за да понижим стойността на Кост функцията. Представяме си топка, която пада по наклон. Крайната цел е тя да се озове на най-ниското място, защото там е най-ниска Кост функцията.

Тук трябва да се отбележи, че това „движение наляво“ става, като променяме стойностите на променливите, тоест уейтовете, по някакъв умен начин. За функцията f(x) = x^2 за да получим по-нисък резултат и сме отдясно на ординатата трябва да понижаваме х, ако сме отляво: трябва да увеличаваме х. Тоест трябва някакъв алгоритъм, който „умно“ да променя стойностите на променливите, за да понижава Кост функцията. Това може би е очевидно при функция с 1 променлива, но при 768 положението е друго.

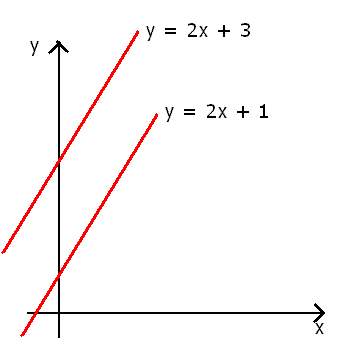
Тоест въпросът става: Какъв алгоритъм да ползваме, че да понижаваме стойността на Кост функцията, като променяме уейтовете на НМ? Отговорът: бакпропагейшън. Математически се открива накъде на функцията (търкалянето на топката) е наклона и какъв е ъгълът му, тоест колко голяма „стъпка“ наляво или надясно да направи. Разбира се, това е опростено. Всъщност трябва да се реши накъде в 768-измерното пространство трябва да се поеме стъпка, и колко голяма трябва да е тя.

Остава един проблем. Както от графиката става ясно, когато попаднем в „дупка“, често тя не е глобалният минимум на функцията. Тоест, ако алгоритъмът ни докара до локален минимум, е добре, но не е най-ниската точка в цялата графика. А най-ниската точка в цялата графика, разбира се, е 0. Когато Костът ни е точно 0. Точно 0 означава, че предсказаният резултат от системата съвпада точно с реалната функция. Както знаем от теоремата за универсалност, това е невъзможно. Вече можем да си представим защо: в 768 измерения (една сравнително проста НМ) намирането на глобален минимум е практически невъзможна задача.

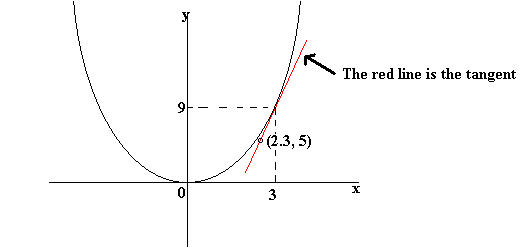
* 1. ***Обучение на НМ – бакрпропагейшън***

Бакпропагейшънът е най-сложната част от математиката в невронните мрежи. На всяка итерация в обучителния процес, той ни дава точно с колко да променим всяка една тежест в системата.

Първата стъпка е да изчислим градиента на Кост функцията. В математиката, градиентът на една функция ни дава „най-стръмната“ посока. Логично, ние ще вземем отрицателната стойност, за да намерим посоката за най-голямо спускане. Ще го демонстрираме с еднопроменлива линейна и квадратна функция. Принципът е същия за всички променливи във функцията, така че с 768 нямаме проблем.



Както вече сме учили, при линейни функции (графика на права), наклонът на функцията е коефициентът к пред първата степен на х. В илюстрирания случай, това е 2. Както виждаме, без значение от свободния коефициент, двете прави са успоредни и мат еднакъв наклон. Също така знаем, че той е равен на тангенса на ъгъла на наклона, което също сме доказвали. Нека погледнем наклона на една квадратна графика.



Това е у = х^2. Всъщност „наклона“ на функцията тук е различен във всяка точка и не е нищо друго, освен наклона на допирателната към графиката. Както в илюстрацията, нека намерим наклона на графиката в точката х=3. Очевидно у=9. Точката от графиката е А(3,9). Построяваме допирателна в тази точка към параболата. Намираме и формулата (което вече знаем) и така откриваме ъгловия коефициент. Съответно така намираме ъгъла на квадратната графика в тази определена точка. В точката В(2,4) е ясно, че той ще е различен.

Откриването на уравнението на права на допирателна всеки път не е ефективно. Затова ползваме по-прост метод: производни. Всъщност първата производна на дадена функция ни дава наклона на графиката в някоя точка!

За да стане по-ясно, ще решим един лесен пример. Нека намерим наклона на функцията у = х^2 в точката х=3, но този път без допирателни. Първо намираме първата производна на функцията:



Или друг начин за записване: (х^2)‘ = 2х.

Сега, за да намерим наклона в точка х=3, просто заместваме в производната стойност:

2х=2.3=6. Следователно наклона на функцията в точката х=3 е 6. Веднага можем да кажем какъв ще е в точката х=24. Това ще е 2.24 = 48.

Сега да намерим градиента на функцията:



Означава се с набла (∇). Представлява вектор, в нашия случай едноизмерен. (можем да движим х само наляво и надясно). Този вектор ни показва накъде и колко да променим х, за да претърпим най-голямо „изкачване“, тоест сочи най-стръмната посока. Да кажем, че искаме да изчислим градиента на у = х^2 в точката х=3. Тогава



Разбира се, ние търсим посоката на най-голямо спускане, значи взимаме отрицателната стойност:



Нека обаче разгледаме градиента на функция с 2 променливи, да кажем f(x,y)=x^2+y^2. Тогава отрицателния градиент ще бъде двуизмерен вектор (можем да променяме х И у):



В този случай за всяка променлива, която участва, трябва да намерим накъде да я „преместим“, за да разберем посоката на най-стръмния наклон. Следователно векторът, посочващ тази посока, е двуизмерен, за х и у. За да намерим координатите на вектора, взимаме частната производна на функцията спрямо съответната променлива. Ето как това става за у (аналогично за х):





Значи, ако разглеждаме точката за х и у 2 и 3, за да намерим най-стръмното спускане, просто заместваме:



Какво ни казва този резултат? Че в точката А(2,3), за да намалим стойността на функцията максимално, трябва да намалим х с някакво число, и да намалим у с по-голямо число. Тоест, че да променим у ни е по-важно и ще ни доведе по-близо до крайната цел, отколкото х. Крайната цел, да припомним, е локален минимум на функцията. Нека сега да се върнем към нашия 768-измерен пример. Отрицателният градиентът на Кост функцията, зависеща от 768 променливи, ще изглежда така:



Крайният вектор е 768-измерен. В случая показва, че да променим втората тежест има много по-голямо значение да се движим надолу по склона, отколкото да променим първата. И също ни дава с колко точно по-голямо е значението, както и посоката (+).

* 1. ***Обучение на НМ – процесът на обучение и еволюция***

Остана последната стъпка. Тя е да променим уейтовете, както ни инструктира отрицателния градиент. Но, тъй като Кост функцията е много сложна, не можем просто от един тренировъчен пример да правим заключения за цялата мрежа, а и не е компютърно ефективно. Както вече споменахме, трябва да имаме приготвени хиляди примери, с които да обучим мрежата. Да проследим как става това:

1. Подбираме 1000 такива примери (тренировъчни данни), както досегашната 7->14.

Например -2->-4, 32->64, и т.н.

1. Изчисляваме за всяка от тях отрицателния градиент, но вместо след всеки пример да нагласяваме уейтовете, взимаме средната стойност на всяка векторна координата през тези 1000 примера. Тоест, след 1000 примери, имаме следния вектор:



1. Правим промените за уейтовете с този вектор. Просто добавяме към всяка тежест съответната за нея „инструкция“ – за първата тежест, добавяме числото на първия ред от вектора, за втората втория ред и т.н. Казано просто, събираме вектора на уейтовете с вектора от средния отрицателен градиент. С това завършваме една итерация/генерация в обучението на НМ.
2. Повтаряме процеса стотици, може би хиляди пъти. Не ние, разбира се, всичко става автоматично чрез програмиран алгоритъм. Всяка следваща генерация е една идея по-добра от предходната, Кост функцията дава малко по-ниска стойност. Повтаряме това, докато не получим задоволително ниска Кост стойност или не попаднем в локален минимум (Кост стойността спира да пада след всяка следваща генерация).

Самият процес по обучение може да трае секунди за прости функции, или часове за по-сложни. Такива прости невронни мрежи са работели още през 90-те на миналия век. Съвременни ИИ като АлфаФолд за предсказване на структурата на протеините се обучават седмици, и то на суперкомпютри. Веднъж получиш ли достатъчно ниска грешка на апроксимация, ползите са много и мъките са си стрували.

1. ***Заключения на базата на експериментални данни с НМ***

* За броя на невроните в двата скрити лейъра: Можем да заключим, че колкото по-близко се броя на невроните в двата лейъра, толкова по-добре се обучава НМ. Спряхме се на оптималното 16Х16. Финалният вариант за структурата е 64-битов за И/О и 16-битов за двата скрити лейъра: 64Х16Х16Х64. Така общо уейтовете стават 2304. С по-малки числа системата работи перфектно и с 16-битовия вариант 16Х16Х16Х16, с 768 уейта.
* Тестване на мрежата извън множеството на обучителната дата: изключително неточни резултати. Това означава, че по-добре да имате малко примери, които да обхождат цялото множество за работа, отколкото много примери в малък интервал, с крайна цел по-голямо работно множество.
* Относно типа на изследваната функция: справя се отлично с линейни, квадратни и кубични, много добре с прости периодични, задоволително със сложни периодични.