МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
"КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ"

ФАКУЛЬТЕТ ІНФОРМАТИКИ
КАФЕДРА МУЛЬТИМЕДІЙНИХ СИСТЕМ

КУРСОВА РОБОТА

Напрям підготовки: 122 «Комп'ютерні науки та інформаційні технології»

на тему:

"Застосування нейронних мереж для прогнозування випадкових процесів"

Виконав студент 4 курсу:
Гічва Назар Русланович

Науковий керівник:
кандидат технічних наук, доцент
Олецький Олексій Віталійович

Київ-2019

Зміст

Вступ

Курсова робота присвячена для дослідження, як застосовуються нейронні мережі для прогнозування випадкових процесів. Метою цієї курсової роботи є дослідження існуючих інтелектуальних методів та систем прогнозування і розробка нових для покращення якості прогнозу. Об’єктом дослідження є системи прогнозування в цілому. У цій роботі ми розберемо що таке штучна нейронна мережа, з чого вона складається. Розберемо ваги та з’єднання, розберемо як можна навчити мережу. Дізнаємось про парадигми навчання штучної нейронної мережі. Розберемо де використовуються та застосовуються штучні нейронні мережі. Розглянемо типи задач на прогнозування та підходи до їх вирішення. Розберемо задачу на прогнозування цін будинків та отримаємо побудовані діаграми моделей навчання та тренування нашої нейронні мережі із заданою вибіркою.

Нейронні мережі

Штучна нейронна мережа

Штучні нейронні мережі (ШНМ) – це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Штучна нейронна мережа навчає та покращує себе. Візьмемо для прикладу розпізнавання фотографії тварини, а якщо бути точнішим, то собаки. У нас є мічені фотографії такі як, «собака» та «не собака». Нейронна мережа використовує результати для ідентифікування. ШНМ нічого не знає про характеристики, а вона сама розвиває свій набір характеристик з навчального матеріалу, який обробляє. Також нейронні мережі використовують для прогнозування даних. Перші нейронні мережі пробували створити Пітс та Маккалюк.

Штучна нейронна мережа складається з штучних нейронів. Кожне з'єднання передає сигнал до іншого нейрона, аналогічно імпульсу. Нейрон який отримує сигнал, може його обробити та передати всім нейронам, які з'єднані з ним.

В більш поширених штучних нейронних мережах сигнал на з'єднанні є дійсне число, а вихід кожного сигналу обчислюється нелінійною функцією суми його входів. З'єднання та нейрони зазвичай мають якусь вагу(цінність, пріорітет). Ця вага під час навчання підлаштовується. Тобто ця вага, може збільшити або зменшити силу сигналу на з'єднанні. Деякі штучні нейрони можуть мати такий поріг, що сигнал буде надісланий лише тоді, якщо сукупний сигнал перетне цей поріг. Зазвичай штучні нейрони організовані в шари. Кожен шар може виконати різні види перетворень для своїх входів. Сигнали проходять від першого до останнього, після того як проходження декілька разів. Якщо з'єднати штучні нейрони у велику мережу з керованою взаємодією, тоді такі прості нейрони разом можуть виконувати досить складні завдання. Дивлячись з точки зору машинного навчання, то нейронна мережа являється окремим випадком методів розпізнавання образів, методи кластеризації. А от з математичної точки зору, багатопараметричне завдання нелінійної оптимізації можна назвати навчанням нейронних мереж. У кібернетиці нейронна мережа використовується для керування робототехніки, а також як алгоритми для них. Так як обчислювальна техніка стрімко розвивається

й програмування не стоїть та місці, тоді штучна нейронна мережа – це спосіб розв'язку проблеми ефективного паралелізму.

Спочатку основною метою штучної нейронної мережі було розв’язання задач, таким способом як це робить людський мозок. Але з часом увага зосередилась на відповідності до певних розумових здібностей. Це відхилилось від біологічних процесів. Штучну нейронну мережу використовували в розв’язуванні різноманітних задач. Приклади таких задач: розпізнавання мовлення, машинним баченням, соціально-мережевим фільтруванням, грою в настільні та відеоігри, машинний переклад та медичне діагнозування.

Складові штучної нейронної мережі

Давайте досконало розберемось з чого складається штучна нейронна мережа та як вона працює. Яке значення мають ваги та сигнали і як це можна описати з математичної точки зору.

Нейрон

З біологічної точки зору нейрон – це електрично збудлива клітина, яка обробляє та передає інформацію у вигляді хімічного або електричного сигналу. Але нам потрібно розібратись з математичної точки зору.

Візьмемо нейрон з міткою j, який отримує вхід $p_j(t)$, від нейронів попередників. Такий нейрон складається з наступних складових:

- збудження $a_j(t)$, що залежить від дискретного параметру часу;
- можливо, порогу θ_j, який залишається незмінним, до тих пір поки його не змінить функція навчання;
- функція збудження f, яка обчислює нове збудження в заданий час $t + 1$ з $a_j(t)$, θ_j та мережевого входу $p_j(t)$, даючи в результаті відношення
  $a_j(t + 1) = f(a_j(t), p_j(t), \theta_j)$
- та функції виходу f_{out}, яка обчислює вихід з активації
  $o_j(t) = f_{out}(a_j(t))$

Дуже часто функція виходу є тотожною функцією.

У нейрона входу немає попередників, він слугує інтерфейсом входу для всієї мережі. Так само аналогічно для нейрона виходу. Він слугує виходом для всієї мережі і в нього немає наступників.

Ваги та з’єднання

Кожна нейронна мережа складається з з’єднань. Кожне таке з’єднання передає вихід нейрону i до входу нейрону j. В такому випадку i є попередником j, а от j є наступником i. Кожне таке з’єднання має свою вагу – w_{ij}.

Поширення та навчання

Штучна нейронна мережа поширює з’єднання та навчається сама. Існує певна функція для поширення та правило для навчання.

Отже, функція поширення обчислює вхід $p_j(t)$ до нейрону j з виходів $o_i(t)$ нейронів попередників. Така функція зазвичай має вигляд:

$$p_i(t) = \sum_i o_i(t)$$

Давайте розглянемо правило для навчання нейронної мережі. Якщо сказати простими словами, то це правило або алгоритм, який міняє параметр нейронної мережі, для того щоб вхід який ми задаємо видавав вихід який буде придатний. Цей процес зазвичай полягає в зміні ваг та порогів змінних мережі.

Нейронні мережі як функції

Ми розглянули, що таке штучна нейронна мережа, нейрон, з чого вони складаються та що потрібно для того щоб передати з’єднання. Давайте розглянемо нейронну мережу як функцію.

Нейронну мережу можна розглянути як просту математичну модель. Така модель визначає функцію $f: X \to Y$. Ще може бути як розподіл над X, або над X та Y.

Давайте розглянемо з математичної точки зору. Нейромережеву функцію $f(x)$ визначають як композицію інших функцій $g_i(x)$. Вони можуть бути розкладені на інші функції. Для нас це зручно представляти як структуру мережі. Досить вживаним способом є нелінійна зважена сума:

$$f(x) = K\sum_i w_i g_i(x)$$

Давайте розберемось що це за параметри. K – є визначеною функцією. Наприклад як сигмоїдна функція, або нормована експоненційна функція, або випрямляльна функція, або гіперболічний тангенс. Досить важливим аспектом для функції збуджування є те, що вона забезпечує плавний

перехід при зміні значень входу. Це означає, що невелика зміна входу призводить до невеликої зміни виходу.

Можливість навчання викликала найбільше зацікавлення нейронними мережами. Для конкретної задачі для розв'язання та класу функцій F навчання означає використання набору спостережень для знаходження $\boldsymbol{f^* \in F}$, яка розв'язує цю задачу в певному оптимальному сенсі. Це все спричиняє визначення такої функції, як функція витрат – $\boldsymbol{C : F \rightarrow \mathbb{R}}$, яка для оптимального розв'язку $\boldsymbol{f^* \in C(f^*) \leq C(f) \forall f \in F}$. Це означає, що жоден розв'язок немає витрат, менших ніж витрати оптимального розв'язку. Функція витрат $\boldsymbol{C}$ є важливим поняттям у навчанні. Вона є мірою того, наскільки далеким є певний розв'язок від оптимального розв'язку задачі, яку потрібно розв'язати. Алгоритми навчання здійснюють пошук простором розв'язків, для того щоб знайти функцію, що має найменші можливі витрати.

Де розв'язок залежить від даних, для тих застосувань витрати обов'язково мусять бути функцією від спостережень. В іншому випадку модель не буде мати зв'язку з даними.

Навіть тоді коли можливо визначити функцію витрат, досить часто використовують конкретні витрати. Навіть через те що вони мають бажані властивості, або через те, що вони природно виникають з певного формулювання задачі. Отже, функція витрат залежить від задачі.

Давайте розглянемо як відбувається зворотне поширення. Що ж таке зворотне поширення? Зворотне поширення – це метод градієнту функції втрат по відношенню до ваг в штучній нейронній мережі. Уточнення ваг зворотного поширення можливо здійснювати за допомогою стохастичного градієнтного спуску із застосуванням наступного рішення:

$$\boldsymbol{w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} + \varepsilon(t)}$$

Розглянемо параметри цього рішення. $\boldsymbol{\eta}$ є темпом навчання нейронної мережі, $\boldsymbol{C}$ функція втрат, $\boldsymbol{\varepsilon(t)}$ – стохастичним членом. Вибір функції витрат залежить від різних чинників, таких як тип навчання та функції збудження.

Парадигми навчання

Давайте розглянемо, які є парадигми навчання штучної нейронної мережі. Існує три основні парадигми навчання. Кожна з них відповідає якійсь певній навчальній задачі. Такі парадигми є: кероване навчання, спонтанне

навчання, навчання з підкріпленням та алгоритм збіжного рекурсивного навчання.

Кероване навчання

Кероване навчання використовує набір пар $(x, y), x \to X, y \to Y$ і має на меті пошук функції $f : X \to Y$ в дозволеному класі функцій. Якщо сказати іншими словами, то ми хочемо вивести відображення, на яке натякають ці дані.

Найбільш широко вживаними витратами є середньоквадратична похибка. Вона намагається звести до мінімуму усереднену квадратичну похибку між виходом мережі, $f(x)$, та цільові значення y над усіма прикладами пар. Алгоритм зворотного поширення для тренування нейронних мереж зводить до мінімуму ці витрати за допомогою градієнтного спуску для класу нейронних мереж. Задачі для цього типу є розпізнавання образів та регресія. Також ця парадигма застосовується до послідовних даних. Можна розглядати, як навчання з «учителем» у вигляді функції, яка забезпечує постійний зворотний зв'язок.

Спонтанне навчання

Спонтанне навчання на вході має якісь дані x та функцію витрат для зведення до мінімуму, якою може бути будь-яка функція від даних x та вироду мережі f. Від задачі та наявних апріорних припущень залежить функція витрат. Давайте розглянемо приклад. Візьмемо модель $f(x) = a$. У цій моделі a є сталою, а витрати $C = E[(x - f(x))^2]$.

Середнє значення даних, це і є значення a, яке зводить до мінімуму витрати. Витрати можуть бути набагато складнішими. Задачі для цього типу парадигми є задачами оцінювання, кластерування, оцінювання статичних розподілів, стиснення та фільтрування.

Навчання з підкріпленням

У цій парадигмі дані зазвичай не надаються. Вони породжуються взаємодією агента з середовищем. В кожен момент часу t агент виконує дію y_t, а середовище породжує спостереження x_t та миттєві витрати c_t відповідно до якоїсь динаміки. Основною метою є визначити стратегію вибору дій, що зводить до мінімуму якусь міру довготривалих витрат. Зазвичай динаміка та витрати для кожної з стратегій є невідомими, але вони можуть бути оцінені.

Часто штучна нейронна мережа використовується у навчанні з підкріпленням як частину загального алгоритму. Задачі для цього типу

парадигми – це задачі керування, а також ігри та інші задачі послідовного ухвалювання рішень.

Збіжне рекурсивне навчання

Цей алгоритм розроблений спеціально для нейронних мереж. Особливість алгоритму в тому, що він може збігатись та уточнювати всі ваги за один крок навіть якщо будуть нові вхідні дані. Спочатку він мав обчислювальну складність $\boldsymbol{O(N^3)}$, а на основі QR-розкладу алгоритм спростився до **O(N)**.

Алгоритми навчання

Коли ми тренуємо нейронну мережу, тоді це означає, що ми вибираємо одну модель з множини дозволених моделей, і це зводить витрати до мінімуму. Існує багато алгоритмів для навчання нейронної мережі, та більшість з них можна розглядати як безпосереднє застосування теорії оптимізації.

Більшість з цих алгоритмів використовують градієнтний спуск, застосовують зворотне поширення для обчислення фактичних градієнтів. Ми просто беремо похідну від функції витрат по відношенню до параметрів мережі, з наступною змінною цих параметрів у пов'язаному з градієнтом напрямку. Так алгоритми тренування зворотним поширенням поділяються на три категорії:

- найшвидший спуск
- квазі-ньютонові
- спряжені градієнти

Еволюційні методи, імітування відпалювання, метод рою часток, непараметричні методи, очікування-максимізація та генно-експресійне програмування є іншими методами для тренування нейронних мереж.

Використання та застосування ШНМ

Щоб використовувати штучну нейронну мережу, потрібно розуміти її характеристики:

- Модель
- Алгоритм навчання
- Робастність

Можливості штучної нейронної мережі підпадають під наступні широкі категорії:

- Класифікація
- Обробка даних
- Робототехніка
- Наближення функцій
- Автоматичне керування

Штучні нейронні мережі знайшли своє використання в широкому діапазоні дисциплін.

Отже, де ми можемо застосувати нейронну мережу. Ми можемо її застосувати для ідентифікації систем, керування системами, розпізнавання образів, розпізнавання послідовностей, добування даних, унаочнення, машинний переклад, соціально-мережеве фільтрування. Також нейронну мережу використовують в діагностуванні раку, для побудови чорноскринькових моделей, моделювання океану. А також використовується для прогнозування випадкових процесів.

Задачі прогнозування

Типи задач прогнозування та підходи до них

Розглянемо типи задач на прогнозування. Вони мають особливе значення, серед яких виділяють завдання з набором певних специфічних ознак, тому вартує провести класифікацію таких задач. Задачі дослідження явищ, де розвиток є поєднаний з часом, можна розділити на декілька класів:

За характером ознак об’єкту:

- прогнозування процесів, реалізація яких представлена у вигляді детермінованих рядів
- прогнозування процесів, де реалізація представлена у вигляді індетермінованих часових рядів
- стаціонарного часового ряду та він характеризується однорідністю в часі
- нестаціонарного часового ряду та він характеризується певною танденцією розвитку в часі

Кількість ознак об’єкту досліджень:

- одновимірна задача
- багатовимірна задача

Види прогнозів:

- згладжування, R = 0;
- короткотерміновий прогноз, R = 1…2;
- середньотерміновий прогноз, R = 3…7;
- довготерміновий прогноз, R = 10…15;

Отже, вид прогнозу досить сильно впливає на вибір засобів і метод реалізації.

Коли ми маємо моменти часу ***t = 1, 2 … n***, тоді дані за якими ми спостерігаємо набувають виду $\boldsymbol{x(t_1), x(t_2), \ldots, x(t_n)}$. До моменту ***n*** інформація про значення часового ряду дозволяє давати оцінки параметрів ***x(tn+1),x(n+2),…,x(n+m)***.

Передбачення

Коли ми маємо задачу з передбаченням, або по-іншому з однокроковим прогнозуванням, то тоді задача зводиться до задачі відображення, коли один вхідний вектор відображається у вихідний.

Передбачення застосовують також для моделювання дискретних послідовностей, що не пов'язані з часом. Враховуючи специфіку часових рядів, такий тип прогнозу не завжди є доцільним, але для певних випадків короткотермінових прогнозів ним можливо скористатись.

Багатокрокове прогнозування

Ті процеси які представлені у вигляді числових рядів, застосовують багатокрокове прогнозування. Багатокрокове прогнозування дозволяє робити коротко- та середньотермінові прогнози, так як суттєвий вплив на точність має накопичення похибки на кожному кроці прогнозування. Коли ми будемо застосовувати довготермінового багатокрокового прогнозування, то тоді буде спостерігатись характерне для багатьох прогнозуючих систем поступове затухання процесу і інші спотворення картини прогнозу. Цей тип прогнозування підходить для часових рядів, що підпадають під означення стаціонарного процесу з невеликою випадковою складовою.

Задача на прогнозування ціни будинків

Давайте розглянемо задачу на прогнозування ціни будинків. На вході ми маємо вибірку даних, в форматі ***csv***. Нам потрібно переробити дані в цей формат, який нам зручний. Задамо вибірку в масив. Розділимо масив даних на функції входу ***x*** та мітки ***y***. Масштабуємо дані або нормалізуємо, щоб вхідні функції мали аналогічні порядки. Я розділив наші дані на

навчальний набір, набір перевірки та тестовий набір. На вході ми маємо такі колонки даних як:

- Площа ділянки
- Загальна якість
- Загальний стан
- Загальна площа
- Кількість повних ванних кімнат
- Кількість половини ванних кімнат
- Кількість спалень над землею
- Загальна кількість номерів над землею
- Кількість камінів
- Площа гаража

В останньому стовпці ми маємо особливість, яку ми хотіли б передбачити:

- Ціна на будинок вище середньої чи ні?

Тепер ми розділили наші дані на вхідні ***X*** та функцію, яку ми хочемо передбачити ***Y***. Для цього розбиття ми просто присвоюємо перші 10 стовпців нашого масиву змінній під назвою ***X***, а останній стовпець нашого масиву – змінній під назвою ***Y***. Наступним кроком у нашій обробці даних є переконання, що масштаб вхідних функцій схожий. Зараз такі функції, як площа партії, складають порядки тисяч, оцінка за загальну якість коливається від 1 до 10, а кількість камінів, як правило, становить 0, 1 або 2. Це ускладнює ініціалізацію нейронної мережі, що спричиняє деякі практичні проблеми. Один із способів масштабування даних - це використання наявного пакету ***scikit-learn***. З цього пакету я використаю функцію ***min*** та ***max*** яка масштабує набір даних, щоб усі функції вводу лежали між 0 і 1 включно. Тепер наш масштабований набір даних зберігається в масиві ***X_scale***.

Тепер ми перейшли до нашого останнього кроку в обробці даних, який полягає в розділенні нашого набору даних на навчальний набір, набір перевірки та тестовий набір. Ми будемо використовувати код з ***scikit-learn*** під назвою ***train_test_split***, який, як випливає з назви, розділив наш набір даних на навчальний набір і тестовий набір. Розмір тестових даних складе 30% від загального набору даних. На жаль, ця функція допомагає нам розділити наш набір даних на два. Оскільки ми хочемо окремий набір перевірки і тестовий набір, ми можемо використовувати ту саму функцію, щоб повторно розділити порівну на набір перевірки та тестовий набір.

Підводячи підсумок, зараз у нас є шість змінних для наших наборів даних, які ми будемо використовувати.

- ***X_train*** (10 функцій вводу, 70% повного набору даних)
- ***X_val*** (10 функцій вводу, 15% від повного набору даних)
- ***X_test*** (10 функцій вводу, 15% від повного набору даних)
- ***Y_train*** (1 мітка, 70% від повного набору даних)
- ***Y_val*** (1 мітка, 15% від повного набору даних)
- ***Y_test*** (1 мітка, 15% від повного набору даних)

Як бачимо, навчальний набір містить 1022 бали даних, тоді як набір перевірки та тестування має 219 точок даних кожен. Змінні ***X*** мають 10 функцій введення, тоді як змінні ***Y*** мають лише одну особливість для передбачення.

Отже, наші дані готові!

Давайте підведемо підсумки, що ми зробили для того, щоб отримати дані в такому вигляді, які потрібні для прогнозування:

- Прочитали фай csv
- Конвертували його в масиви
- Розділили набір даних на функції введення та мітку
- Масштабували дані так, щоб функції введення мали подібні порядки
- Розділили набір даних на навчальний, перевірки та тестовий набори

Дальше ми створюємо архітектуру. Ми використаємо таку архітектуру:

- Прихований шар 1: 32 нейрони, активація ReLU
- Прихований шар 2: 32 нейрони, активація ReLU
- Вихідний шар: 1 нейрон, активація сигмоїдів

Тепер нам потрібно описати цю архітектуру ***kerasu***. Ми будемо використовувати послідовну модель, а це означає, що нам просто потрібно описати шари, описані вище, послідовно.

Дальше ми заповнюємо найкращі цифри. Тепер, коли ми уточнили свою архітектуру, нам потрібно знайти найкращі номери для неї. Перш ніж розпочати наше навчання, ми повинні налаштувати модель.

Нарешті, ми хочемо відстежувати точність поверх функції втрат. Тепер, коли ми запустили цю клітинку, ми готові до тренувань!

Дальше ми описали навчання даних. Функція називається ***fit***, оскільки ми підганяємо параметри до даних. Ми повинні вказати, за якими даними ми навчаємось, а це ***X_train*** та ***Y_train***. Потім ми визначаємо розмір нашої міні-партії та скільки часу ми хочемо її тренувати. Нарешті, ми визначаємо, які наші дані валідації, щоб модель розповіла нам, як ми робимо дані валідації в кожній точці. Ця функція виведе історію, яку ми зберігаємо під змінною ***hist***. Ми будемо використовувати цю змінну трохи пізніше, коли перейдемо до візуалізації.

Тепер ми бачимо, що модель тренується. Переглядаючи цифри, ви зможете побачити зменшення втрат і збільшення точності з часом. На цьому етапі ви можете експериментувати з гіпер-параметрами та архітектурою нейронної мережі. Запустивши клітинки ще раз, ми побачили, як змінилося наше навчання, коли ми підкорегували свої гіперпараметри. Після того, як ви будете задоволені остаточною моделлю, ми можемо оцінити її на тестовому наборі. Через випадковість того, як ми розділили набір даних, а також ініціалізацію ваг, цифри та графіки будуть дещо відрізнятися щоразу, коли ми запускаємо наш ноутбук. Тим не менш, ми повинні отримати точну перевірку десь від 80% до 95%.

Отже, ми навчили нашу нейронну мережу!

Давайте підсумуємо, що ми зробили, для того щоб навчити та керувати нейронною мережею:

- Задали архітектуру
- Налаштували оптимізатор, функцію втрати, показники для відстеження, за допомогою ***module.compile***.
- Навчаємо нашу модель з навчальними даними з ***model.fit***
- Оцінюємо нашу модель на тестовому наборі за допомогою ***model.evaluate***

Побудуємо графік втрати тренувань та втрати валу на кількість пройдених епох. Щоб відобразити кілька приємних графіків, ми будемо використовувати пакет ***matplotlib***. Ми використовуємо ***matplotlib*** для візуалізації втрат та точності навчання та валідації з часом, щоб побачити, чи є в нашій моделі надмірне обладнання.

Для того, щоб внести регуляризацію в нашу нейронну мережу, давайте сформулюємо нейронну мережу, яка погано переграє наш навчальний набір.

Тепер давайте спробуємо кілька стратегій щодо зменшення надмірного розміщення. Ми включимо регуляризацію та випадання L2. А тепер давайте побудуємо графіки втрат та точності. Ви помітите, що втрата на початку значно більша, і це тому, що ми змінили функцію втрат.

Порівняно з нашою моделлю в моделі 2, ми значно зменшили надмірний розмір! І саме тому ми застосовуємо наші методи регуляризації, щоб зменшити перевитрату до навчального набору.

Висновок

Отже, ми розібрали що таке штучна нейронна мережа, з чого складається нейрон, як він передає сигнали та взаємодіє з іншими нейронами. Ми дізнались про ваги, поширення та навчання нейронної мережі. Дізнались про парадигми навчання штучної нейронної мережі. Також ми розібрали де використовуються та застосовуються штучні нейронні мережі. Ми розглянули типи задач на прогнозування та підходи до них. Переглянули передбачення та багаторазове прогнозування. За допомогою Python та допоміжних бібліотек, ми розібрали задачу на прогнозування цін будинків та отримали побудовані діаграми моделей навчання та тренування нашої нейронної мережі із заданою вибіркою.

Лістинг коду

```
import pandas as pd

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from Keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras import regularizers

import matplotlib.pyplot as plt


df = pd.read_csv('housepricedata.csv')
```

```
dataset = df.values


X = dataset[:,0:10]
Y = dataset[:,10]
min_max_scaler = preprocessing.MinMaxScaler()


X_scale = min_max_scaler.fit_transform(X)
X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, Y, test_size=0.3)
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)


# model 1
model = Sequential([
    Dense(32, activation='relu', input_shape=(10,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid'),
])


model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist = model.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))


model.evaluate(X_test, Y_test)[1]
```

```
# loss 1
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()

# training 1
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()

# model 2
model_2 = Sequential([
    Dense(1000, activation='relu', input_shape=(10,)),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1, activation='sigmoid'),
])
```

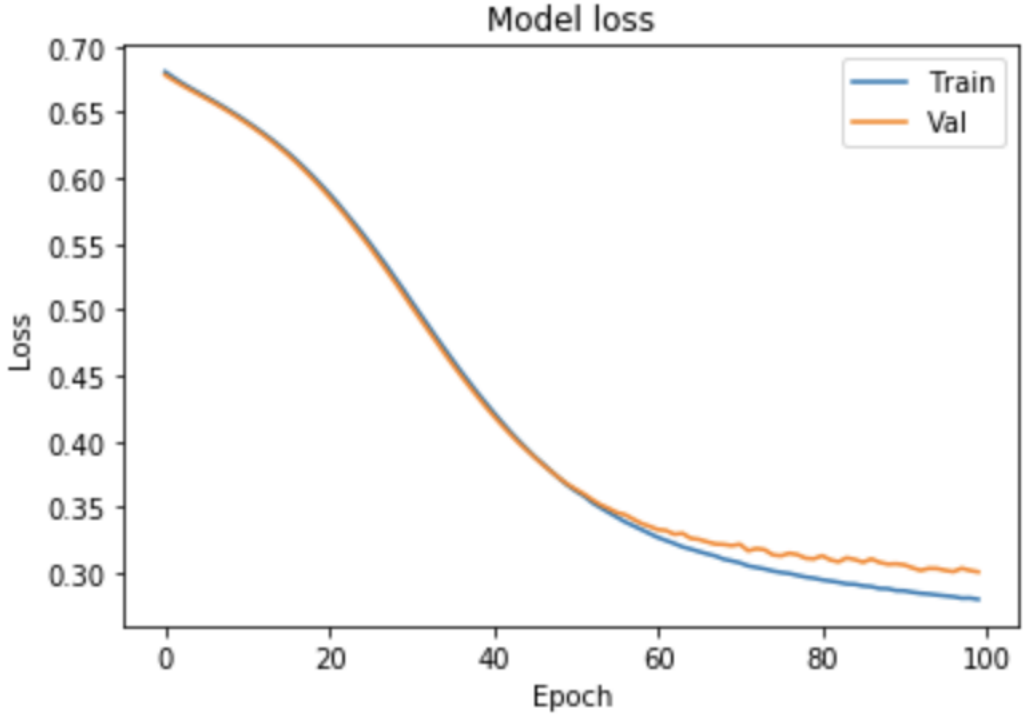
```
model_2.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

hist_2 = model_2.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))

# loss 2
plt.plot(hist_2.history['loss'])
plt.plot(hist_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()

# training 2
plt.plot(hist_2.history['acc'])
plt.plot(hist_2.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()

# model 3
```

```
model_3 = Sequential([
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01),
input_shape=(10,)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])

model_3.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

hist_3 = model_3.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))

# loss 3
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
```

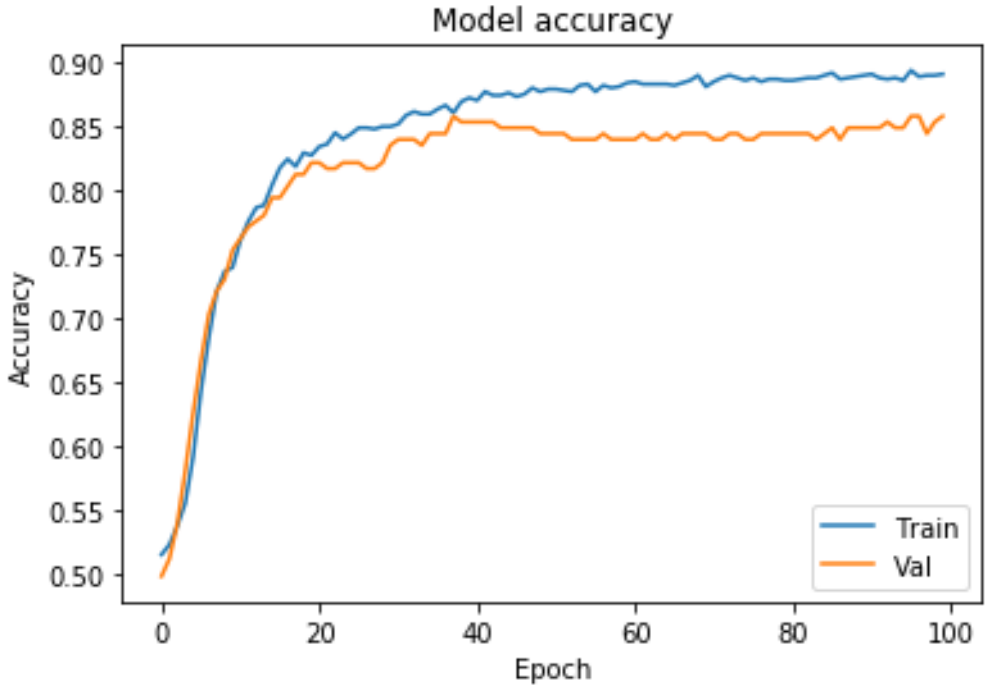
```
plt.ylim(top=1.2, bottom=0)
plt.show()

# training 3
plt.plot(hist_3.history['acc'])
plt.plot(hist_3.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```

Ілюстрації

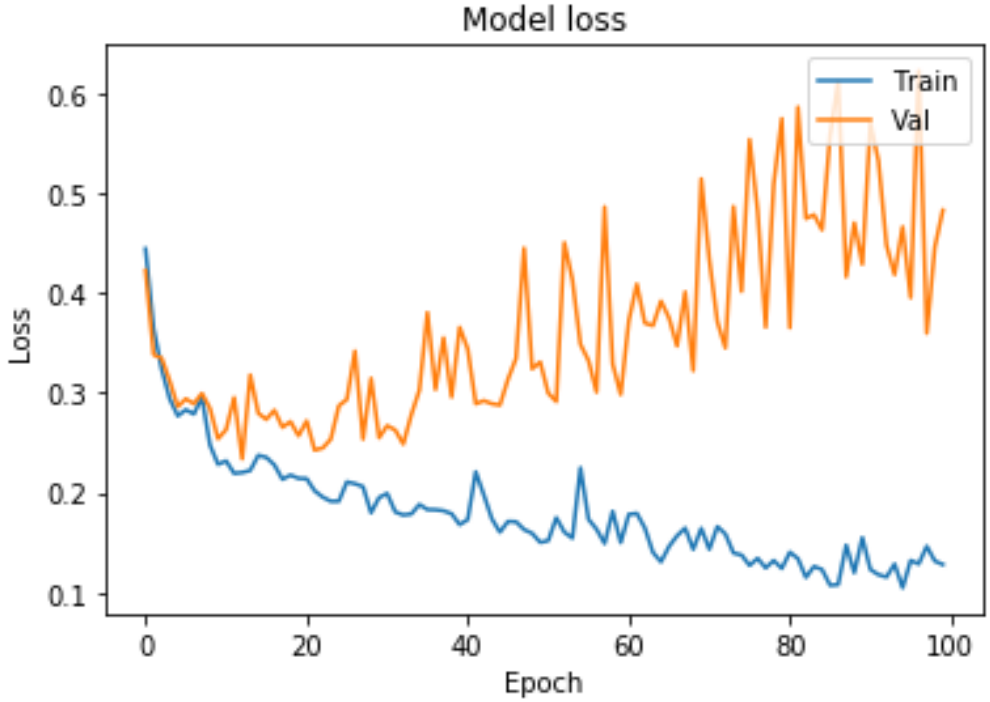
Діаграма втрати моделі 1



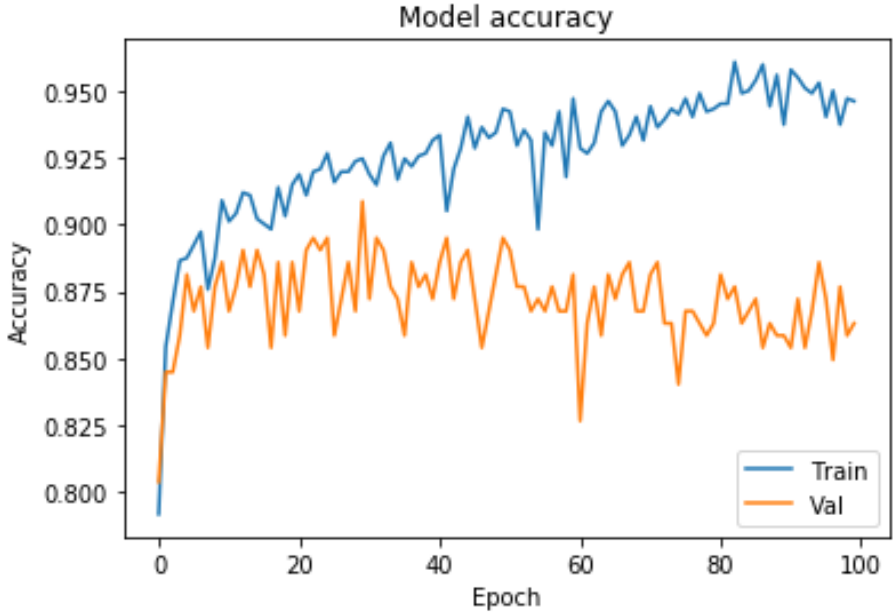
Діаграма навчання моделі 1



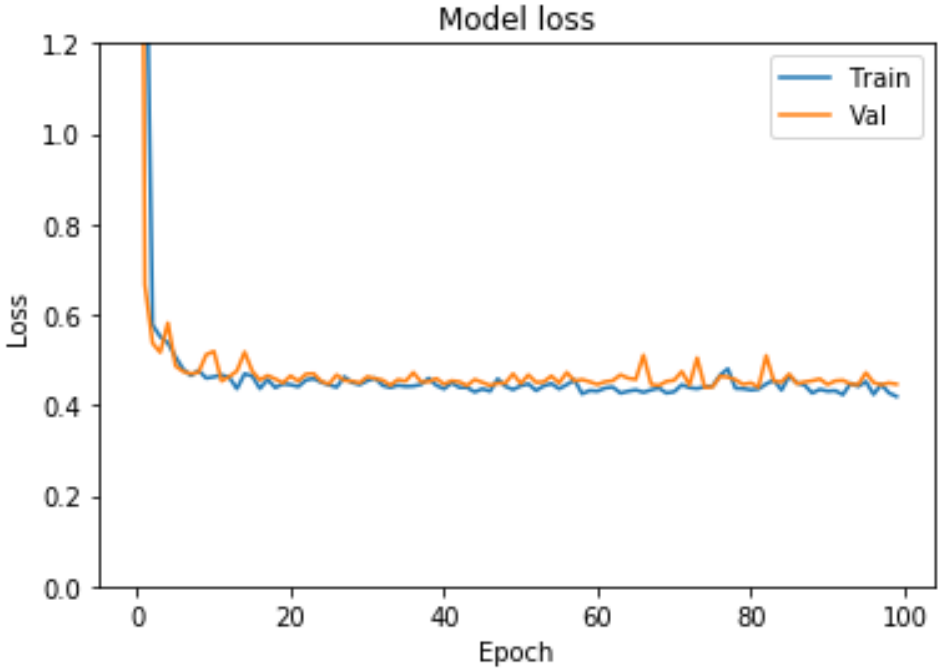
Діаграма втрати моделі 2



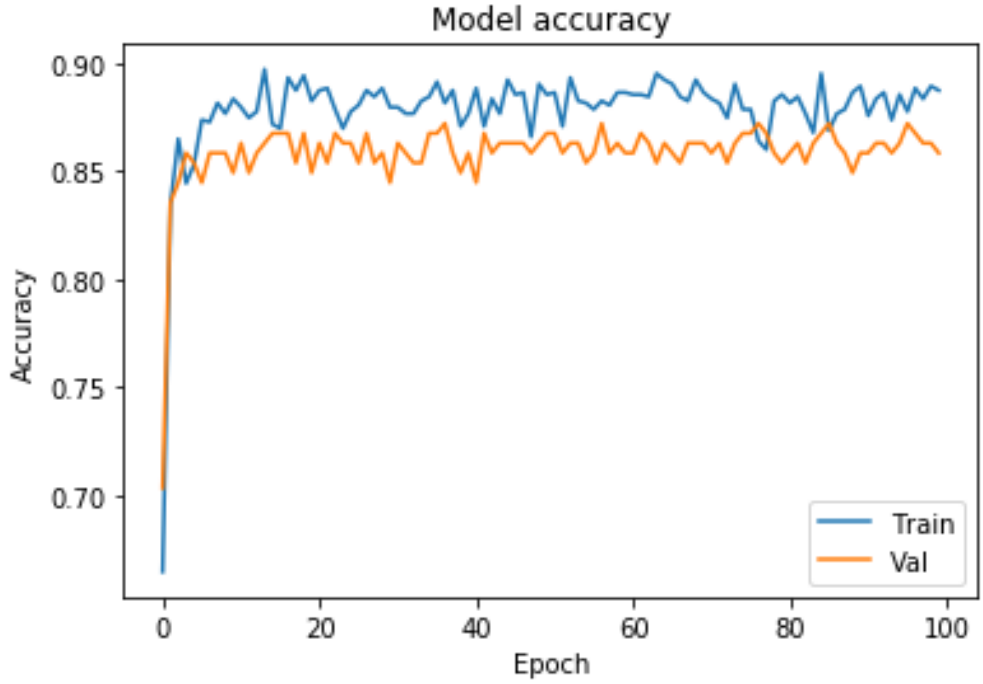
Діаграма навчання моделі 2



Діаграма втрати моделі 3



Діаграма навчання моделі 3



Джерела

https://wiki.tntu.edu.ua/%D0%9F%D1%80%D0%BE%D0%B3%D0%BD%D0%BE%D0%B7%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%B0_%D0%B4%D0%BE%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%BE%D1%8E_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D1%85_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6

https://cyberleninka.ru/article/n/neyronni-merezhi-ta-yihne-vikoristannya-dlya-prognozuvannya-tendentsiy-rinku-neruhomosti

http://ena.lp.edu.ua:8080/bitstream/ntb/9913/1/25.pdf

https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B0_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0