Пояснювальна записка

до дипломного проекту

на тему: Інструментарій моделювання згорткових нейронних мереж

Київ – 2019 року

ЗМІСТ

Перелік умовних позначень, скорочень і термінів

ІМЗНМ – інструментарій для моделювання згорточних нейронних мереж

ЗНМ – згорточні нейронні мережі

ОС – операційна система

СКБД – система керування базами даних

CPU – central processing unit

GPU – graphics processing unit

GPGPU – General-purpose graphics processing units

ORM – Object-relational Mapping

Python – мова програмування Python

UI – User Interface

Вступ

У наш час, коли майже у кожному портативному пристрої є цифрова камера з можливістю зйомки фото та відео матеріалів, обсяги інформації зростають з величезною швидкістю, тому виникає проблема у автоматичному розпізнаванні зображень та їх класифікації. Глобально ці проблеми відносяться до терміну «машинне навчання» («Machine Learning»). Постійні публікації у даній сфері підтверджують актуальність даної проблеми.

Як приклад, у жовтні 2012 року відбувся конкурс ImageNet [2], що був присвячений класифікації об'єктів на фотографіях. У конкурсі було потрібно розпізнавання образів в 1000 категорій. Команда переможця Джеффрі Хинтона використовувала методи поглибленого вивчення (Deep Learning) та згорточних нейронних мереж (Convolutional Neural Networks) [3].

У березні 2013 року Google інвестував в проект Хинтона, заснований при університеті Торонто. Протягом шести місяців був розроблений сервіс пошуку по фотографіях photos.google.com. Напевно, у кожного є величезний архів фотографій, тепер Ви можете з легкістю їх класифікувати, що допоможе у майбутньому користуванні цих матеріалів.

Та повернімось до методів, якими ця можливість тепер є досяжною. Зокрема, це згорточні нейронні мережі – окремий клас нейронних мереж, який найкращим чином підходить для інтелектуальної обробки візуальних даних. ЗНМ були розроблені професором Яном Лекуном в кінці 1990-х років. Вже тоді ця технологія дозволяла надійно вирішувати задачі розпізнавання рукописних текстів. З тих пір значно збільшилася потужність комп'ютерів, і з'явилися нові алгоритми навчання нейронних мереж.

Також варто зазначити, що Джеффрі Хинтон був одним з дослідників, хто запропонував використовувати метод зворотного поширення помилки для тренування нейронних мереж.

Об’єктом дослідження даного проекту є процеси розпізнавання та класифікації зображень, а предметом дослідження – використання ЗНМ для розпізнавання та класифікації зображень.

Мета даного проекту – розробка інструментарію для моделювання ЗНМ. Для досягнення поставленої мети був виконаний пошук і порівняння існуючих бібліотек для роботи зі ЗНМ, наступним кроком був вибір додаткових компонент, а після цього – розробка зручного інтерфейсу, що об’єднує обрані інструменти.

1 Постановка задачі

1.1 Загальні вимоги

Дана робота полягає у розробці інструментарію для моделювання ЗНМ. Необхідно створити архітектуру, яка дозволить виконувати моделювання ЗНМ для користувача без особливих проблем, реалізувати зрозумілий та простий інтерфейс.

Так як майбутньою сферою використання даного ІМЗНМ є використання у навчальному класі для початкового знайомства зі ЗНМ, виникає наступна вимога – можливість використовування кількома користувачами одночасно. Але це не виключає використання ІМЗНМ для більш серйозних задач, таких як знаходження оптимальної архітектури ЗНМ для конкретних наборів даних, вдосконалення тренувальних алгоритмів та інше.

1.2 Вимоги до навчальної частини

Можливість використання у навчальному класі зобов'язує реалізувати декілька ролей користувача, такі як Адміністратор (Administrator), Вчитель (Instructor), Користувач (User). На діаграмі прецедентів зображені трьома акторами (рисунок 1.2.1):

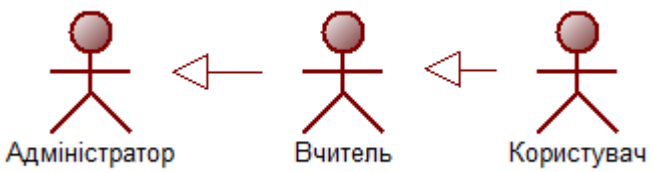

Рисунок 1.1 – Ієрархія користувачів ІМЗНМ

Оскільки моделювання потребує великих ресурсів системи і неможливо одночасно моделювати безліч ЗНМ, є необхідність реалізації черги. Параметри черги

встановлює Адміністратор, основний параметр – кількість одночасних процесів, максимальний час виконання, максимальна кількість використаної пам’яті RAM. Адміністратор також має можливість зміни черги (рисунок 1.2).

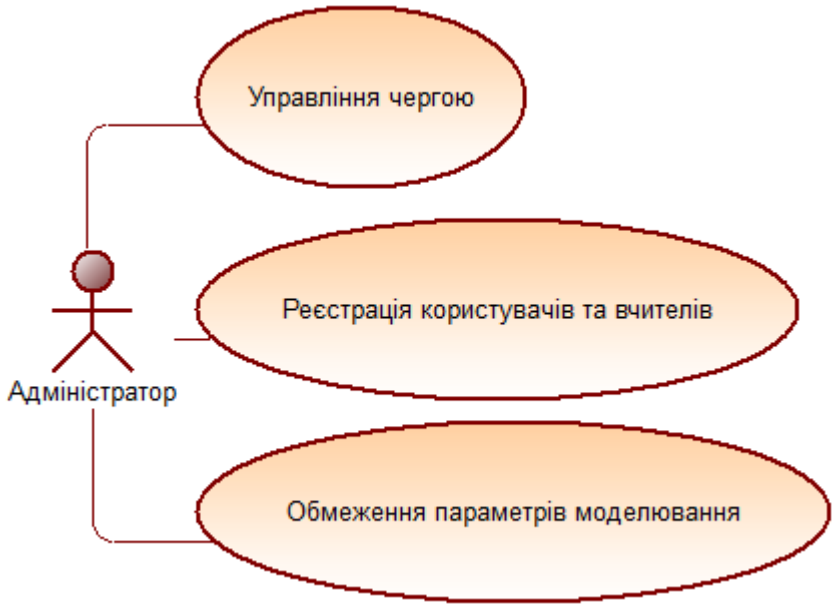

Рисунок 1.2 – Діаграма прецедентів для ролі Адміністратор

Користувачі мають можливість обрати клас навчання, кількість обраних класів необмежена. Класи створює Вчитель, для легкості перевірки отриманих Користувачем результатів до одного класу може належати кілька Вчителів.

Інші функції, що доступні для Користувача: створення ЗНМ, встановлення у чергу, видалення з черги, запуск на моделювання, відміна моделювання, перегляд своїх результатів моделювання (рисунок 1.3):

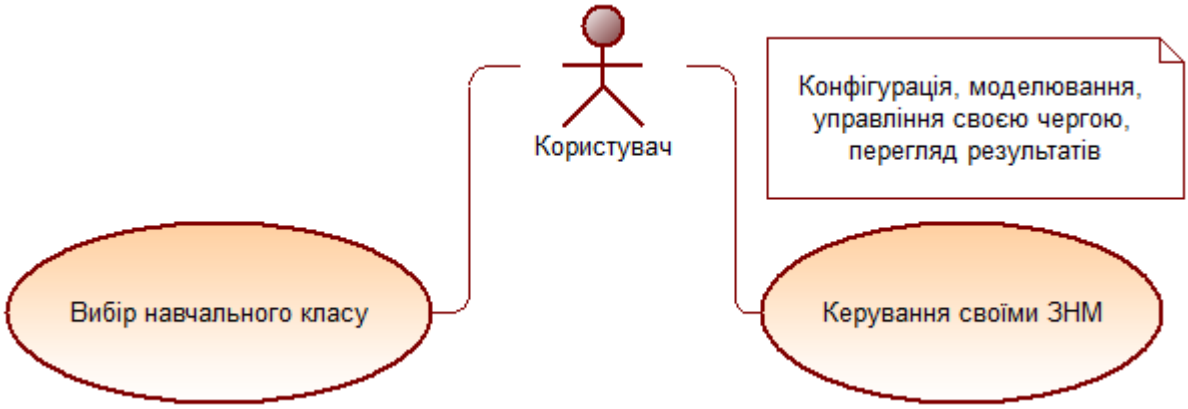

Рисунок 1.3 – Діаграма прецедентів для ролі Користувача

Вчитель має можливість перегляд та користування усіх ЗНМ у межах свого класу (рисунок 1.4). Також Вчитель має можливість створювати та моделювати свої ЗНМ. Для можливості виконувати Вчителю власні дослідження, обов'язкова реалізація параметра, що дозволить обчислювати ЗНМ без обмежень Адміністратора та з пріоритетом черги.

Рисунок 1.4 – Діаграма прецедентів для ролі Вчитель

ЗНМ може перебувати у наступних станах (рисунок 1.5): Конфігурація, Черга, Моделювання, Результати (перегляд результатів моделювання).

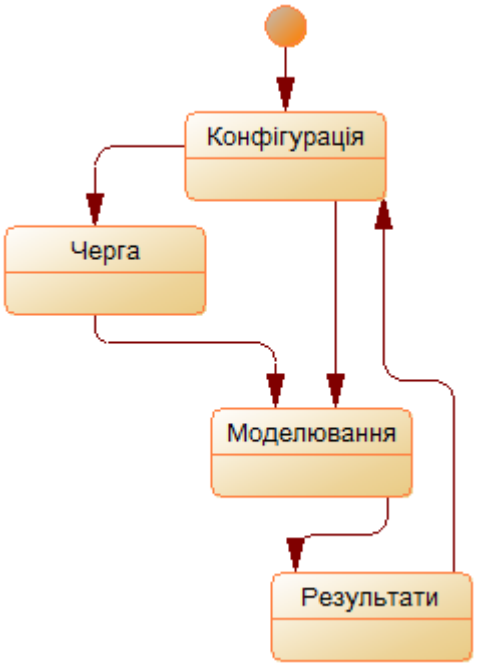

Рисунок 1.5 – Діаграма станів ЗНМ

1.3 Вимоги до бібліотеки для роботи зі ЗНМ

Для виконання Вчителю власних досліджень необхідна гнучка структура бібліотеки, що дозволить легко розробляти нові моделі та алгоритми тренування, тому виникає наступна вимога – зрозуміла структура коду та підтримка модульної системи.

Також необхідною є можливість обирати для тренування різні датасети (datasets), та розробляти власні.

1.4 Технічні ресурси

Оскільки моделювання ЗНМ є доволі ресурсномісткою задачею, отримане рішення має мінімально залежати від технічних ресурсів користувача.

Велике навантаження на CPU при обчисленні подібних задач ставить наступне побажання – використання GPU для складних обчислень. Дана техніка має назву GPGPU [4] – GPU загального призначення. Перевага даного рішення над використанням CPU наступні:

- висока пропускна здатність пам'яті;
- велика кількість малих, але швидких ядер (наприклад, Nvidia GTX 580 має 512 ядер), а отже велика кількість потоків;
- велика швидкість створення потоків;
- добре підходять для виконання математичних операцій, таких як множення матриць та інше.

Найпопулярнішими реалізаціями техніки GPGPU є CUDA та OpenCL. CUDA надає можливість використання GPU лише від компанії Nvidia.

1 ЗНМ та їх застосування

2.1 Історія виникнення

Згорточна нейронна мережа представляє особливий клас нейронних мереж, який найкращим чином підходить для інтелектуальної обробки візуальних даних.

У 1981 році нейробіологи Торстен Візел та Девід Хабель досліджували зорову кору головного мозку кішки і виявили, що існують так звані прості клітини, які особливо сильно реагують на прямі лінії під різними кутами і складні клітини, що реагують на рух ліній в одному напрямку.

Пізніше Ян Лекун запропонував використовувати так звані згорточні нейронні мережі, як аналог зорової кори головного мозку для розпізнавання зображень [5, 6, 7].

Ян Лекун і його співробітники зробили дійсно гарну роботу по розпізнанню почерку, використовуючи ЗНМ, що були застосовані на практиці (детальніше у підрозділі 2.6). Це один з небагатьох на той час прикладів з застосуванням комп’ютера для тренування нейронних мереж, що мали добрий результат.

2.2 Основні механізми

ЗНМ засновані на ідеї поступового виділення характерних ознак («the idea of replicated features») [7, 8]. Якщо об'єкт переміщується та у нас є детектор ознак («feature detector»), який корисний в одному місці в зображенні, цілком ймовірно, що той самий детектор буде корисний в іншому місці. Отже, ідея полягає у створенні різних копій одного детектора ознак для різних позицій.

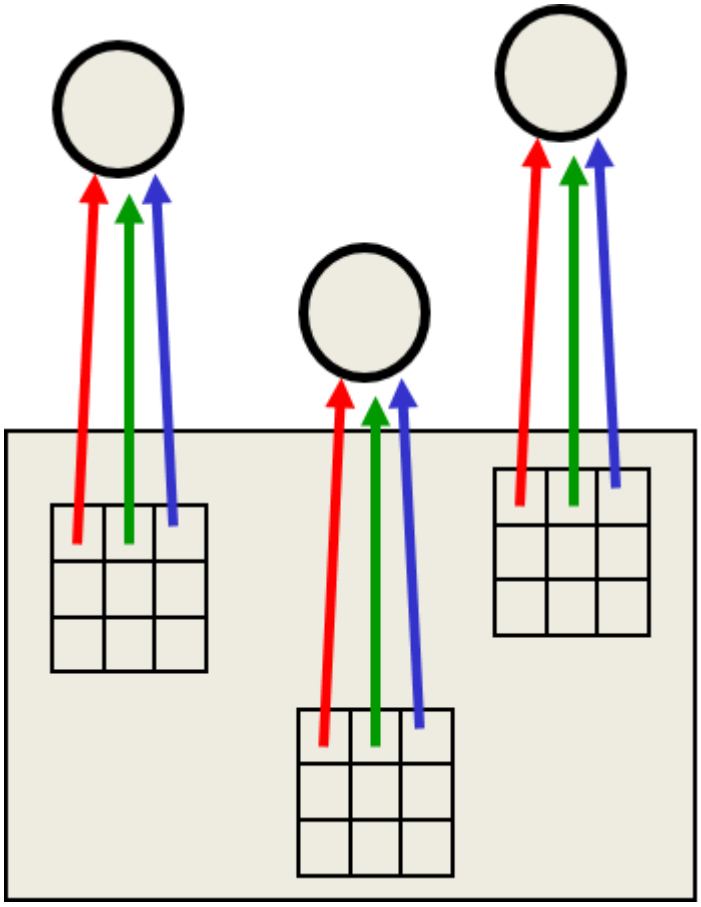

Рисунок 2.1 – Три детектора ознак, що є копіями один одного

Приклад наведений на рисунку 2.1. Кожна ознака має вагу до дев'яти пікселів, ваги ідентичні між трьома різними детекторами. Так що червона стрілка має ту ж вагу для всіх трьох детекторів ознак. Але червоні і зелені стрілки матимуть різну вагу. Отже ці 27 пікселів, що ми бачимо для трьох детекторів ознак, виділяють 9 різних ваг.

Ми можем використовувати детектори ознак не тільки одного типу, кожен тип об’єднується у карти, так звані карти ознак. Потім різні карти навчаться виявляти різні ознаки. Це дозволяє кожну ділянку зображення представити ознаками різних типів.

Чергування шарів нейронної мережи дозволяє складати карти ознак з карт ознак, що на практиці дає можливість розпізнавати складні ієрархії ознак.

Зазвичай після проходження декількох шарів карта ознак вироджується у вектор або навіть скаляр, але таких карт ознак стає сотні. У такому вигляді вони подаються на один-два шари повнозв'язної мережі. Вихідний шар такої мережі може

мати різні функції активації. У найпростішому випадку це може бути тангенціальна функція. Також успішно використовуються радіальні базисні функції.

Згорточна нейронна мережа складається з шарів згортки та субдискретизації. Зображення, що надходить на вхід піддається згортці з деяким ядром згортки (рисунок 2.2) [9].

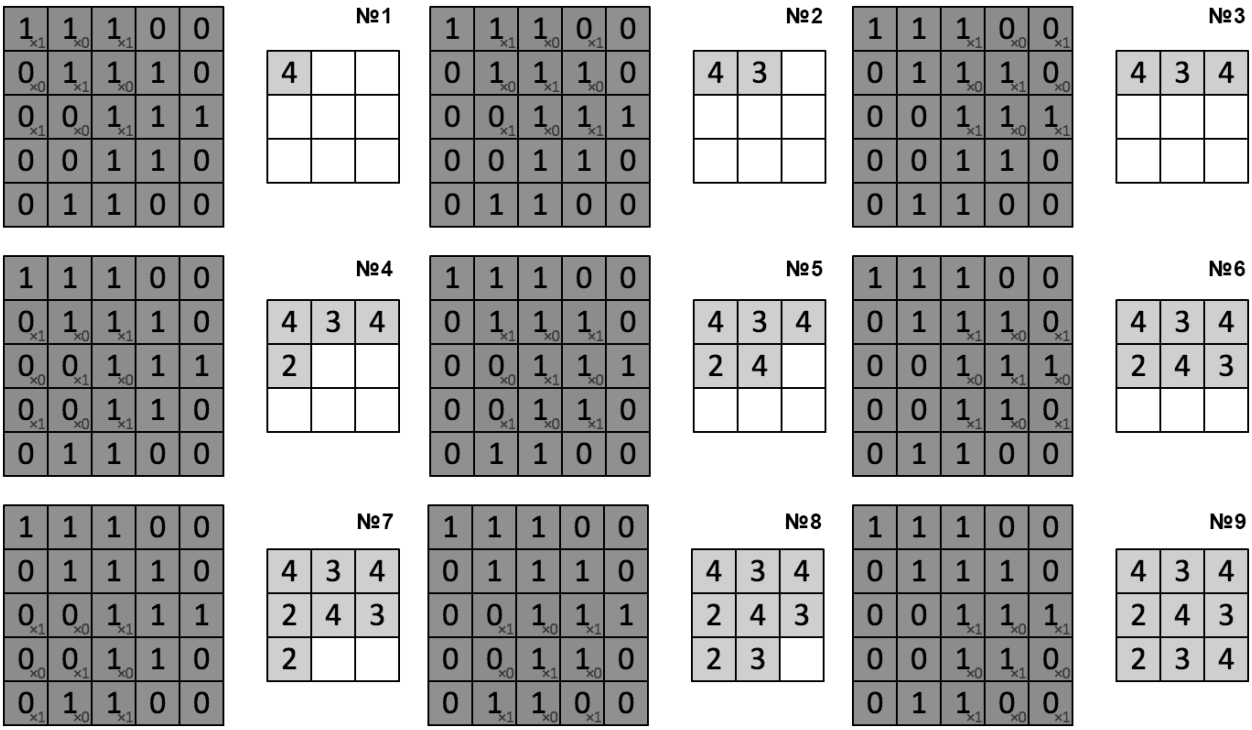

Рисунок 2.2 – Приклад виконання операції згортки. Ліворуч – вхідне зображення, праворуч – ядро згортки

Ядро згортки є набором колективних вагових коефіцієнтів. Результатом операції згортки є також деяке зображення, яке називається картою ознак. Залежно від обраного ядра згортки, карта ознак буде виділяти ті чи інші характеристики вхідного зображення. Для найбільш повного виділення характеристик вхідного зображення використовується декілька різних ядер згортки так, що на виході шару згортки виходить декілька карт ознак (рисунок 2.3).

Рисунок 2.3 – Структура згорточної нейронної мережі

За шаром згортки йде шар усереднення та субдискретизації, який знижує розмірність карти ознак, тим самим знижуючи чутливість виходів до зсувів та поворотів (рисунок 2.4).

Таке чергування згорточних та субдискретизуючих шарів приводить до поступового збільшення кількості карт ознак при зменшенні їх розмірності від шару до шару.

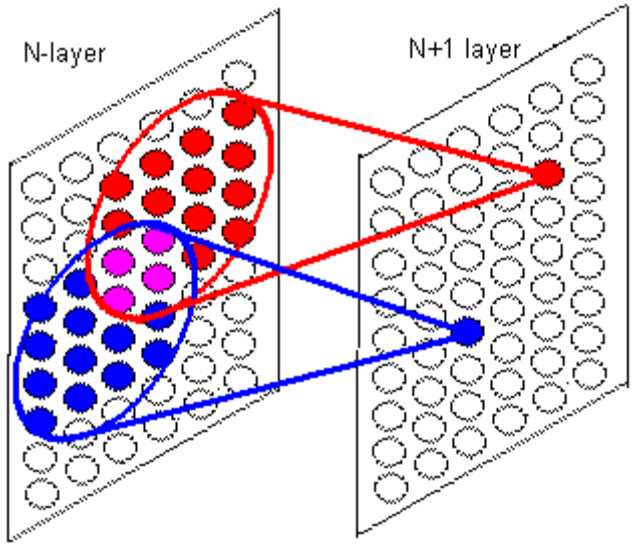

Рисунок 2.4 – Чергування шарів згортки та підвибірки

Навчання мережі починається з пред'явлення образу та обчислення відповідної реакції. Порівняння з бажаною реакцією дає можливість змінювати ваги зв'язків таким чином, щоб мережа на наступному кроці могла видавати більш точний

результат. Навчальне правило забезпечує налаштування ваг зв'язків. Інформація про виходи мережі є вхідною для нейронів попередніх шарів. Ці нейрони можуть налаштовувати ваги своїх зв'язків для зменшення похибки на наступному кроці.

Коли неналаштованій мережі пред'являється вхідний образ, вона видає деякий випадковий вихід. Функція помилки представляє собою різницю між поточним виходом мережі та ідеальним виходом, який необхідно отримати. Для успішного навчання мережі потрібно наблизити вихід мережі до бажаного виходу, тобто послідовно зменшувати величину функції помилки. Це досягається настройкою міжнейронних зв'язків. Кожен нейрон в мережі має свої ваги, які настроюються, щоб зменшити величину функції помилки.

В основі алгоритму зворотного поширення помилки лежить методика, що дозволяє швидко обчислювати вектор часткових похідних складної функції багатьох змінних, якщо структура цієї функції відома. В якості такої функції в алгоритмі розглядається функція помилки мережі та враховується той факт, що структура функції помилки мережі повністю визначається архітектурою нейронної мережі, яка вважається відомою.

Початкова ініціалізація нейронної мережі має величезний вплив на кількість ітерацій навчання. Від того, наскільки вдало вибрані початкові значення ваг залежить, як довго мережа за рахунок навчання та підстроювання буде шукати їх оптимальні величини, і чи знайде вона їх.

Ваги повинні бути обрані випадково, але в такий спосіб, щоб функція активації насамперед активізувалася у своїй лінійної області. Для досягнення такого ефекту необхідне узгодження між нормалізацією вхідних значень нейронної мережі, вибором функції активації та вибором початкових значень вагових коефіцієнтів.

В основі згорточних мереж лежать три механізми, використовувані для досягнення інваріантності до переносу, масштабування, незначних викривлень:

– *Локальне формування ознак*. Кожний нейрон одержує вхідний сигнал від локального рецептивного поля в попередньому шарі, витягаючи, таким чином, його

локальні ознаки. Як тільки ознака витягнута – її точне розташування вже не має значення, оскільки встановлене місцезнаходження відносно інших ознак.

– *Формування шарів у вигляді набору карт ознак*. Кожний обчислювальний шар складається з безлічі карт-ознак – площин, на яких усі нейрони повинні використовувати той самий набір синаптичних ваг. Така форма ускладнює структуру мережі, однак має дві важливі переваги: інваріантність до зсуву, який досягається за допомогою згортки з ядром невеликого розміру, і скорочення числа вільних параметрів, яке досягається за рахунок спільного використання синаптичних ваг нейронами однієї й тієї ж карти.

– *Підвибірка*. За кожним шаром згортки розташований обчислювальний шар, що здійснює локальне усереднення й підвибірку. За рахунок цього, досягається зменшення розмірності для карт ознак. Така операція приводить до зниження чутливості вихідного сигналу оператора відображення ознак до незначного зсуву й іншим формам деформації. У якості такого оператора виступає одна з сигмоїдальних функцій, що використовується при побудові нейронних мереж, наприклад гіперболічний тангенс.

Слід зазначити, що послідовне застосування згортки й підвибірки приводить до так званого підвищення рівня ознак: якщо перший шар витягає локальні ознаки з областей зображення, то наступні шари витягають загальні ознаки, які також називаються ознаками вищого порядку.

2.3 Принцип роботи ЗНМ на прикладі

Розглянемо архітектуру (рисунок 2.5) та принцип роботи згорточної нейронної мережі на прикладі мережі для пошуку облич людей на фотографії.

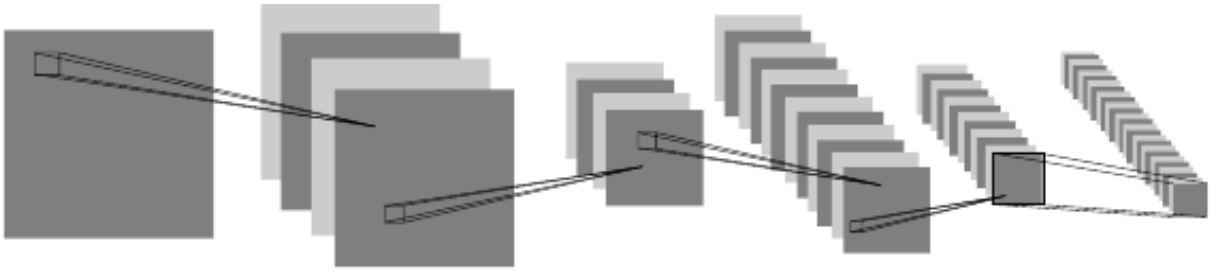

Рисунок 2.5 – Архітектура згорточної нейронної мережі

У вхідний шар надходить зображення розміру 35 на 35 пікселів, яке може бути або не бути зображенням обличчя людини. Значення вхідних пікселів нормалізуються для прискорення навчання. Точні нормалізовані значення обчислюються залежно від типу використовуваної активаційної функції.

Перший схований шар є шаром згортки. Він складається з 8 карт ознак розміром 16 на 16.

Розглянемо процес формування цього шару, оскільки інші згорточні шари формуються подібним чином.

Кожний елемент карти ознак з'єднаний з областю розміром 5x5 на вхідному зображенні. Отже, кожний елемент карти має 25 вільних коефіцієнтів.

Рецептивні поля сусідніх елементів на карті ознак з'єднані з сусідніми областями попереднього шару, отже ці поля частково накладаються один на одного. Як було сказано раніше, усі елементи карти ознак мають загальний набір з 25 ваг та зрушення, тому можна сказати, що вони витягають ту саму ознаку в різних областях попереднього шару. У цьому випадку, 8 різних карт другого шару витягають 8 різних ознак для різних областей зображення. Значення всіх елементів карти ознак обчислюються шляхом послідовного проходу по вхідному шару й застосуванню формули прямого поширення сигналу до областей, які є локальними рецептивними полями для даних елементів карти ознак. Цікавою особливістю згорточних шарів є той факт, що при зсуві вхідного зображення значення карт ознак будуть зсунуті на ту ж саму величину. За рахунок цього згорточні мережі мають інваріантність до зсувів та викривлень вхідного сигналу.

Отже, перший схований шар є шаром згортки, який містить 8 карт ознак розміром 16x16. Кожний його елемент з'єднаний з областю розміром 5x5 на вхідному зображенні. Перший схований шар містить 122,304 зв'язків та 156 параметрів, яких навчають. Така економія пам'яті й, що головне, обчислювальних витрат досягається за рахунок спільного використання ваг елементами однієї карти.

Другий схований шар є шаром підвибірки. Він складається з 8 карт ознак розміром 8x8, причому кожний з елементів карт цього шару з'єднано з областю 2x2 у відповідній карті ознак попереднього шару. Значення елементів шарів згортки також обчислюються за формулою прямого поширення. Важливо розуміти, що рецептивні поля для елементів шару підвибірки не перетинаються, отже, карти ознак цього шару містять в 2 рази менше рядків та стовпців, ніж в попередньому шарі. Слід зазначити, що оскільки завданням даного шару є підвибірка (тобто локальне усереднення), не обов'язково зберігати для карт 4 ваги й зрушення, а досить обійтися однією загальною вагою й зрушенням. Таким чином, другий схований шар містить 5,880 зв'язків та 12 вільних параметрів.

Третій схований шар є згорточним шаром з 20 картами ознак розміром 6x6. Кожний елемент у кожній карті ознак пов'язаний з декількома областями розміром 5x5 певних карт попереднього шару. Такому незвичайному способу зв'язку цих двох шарів є ряд пояснень. По-перше, з'єднання всіх карт другого шару з усіма картами третього шару значно збільшило б кількість зв'язків. З'єднання карт «одна до одної» (за умови, що в третьому шарі всього 8 карт) не дало б ніяких результатів – це стало б ще одним повторенням згортки, яке вже було присутнє між першим та другим шарами. З іншого боку, з'єднання карт третього шару з певними картами другого шару порушує симетрію мережі, і змушує її витягати зовсім інші ознаки, які будуть доповнювати раніше витягнуті. Як правило, архітектор мережі сам ухвалює рішення щодо того, за яким принципом організовувати з'єднання карт другого й третього шарів. Загалом, третій шар містить 151,600 зв'язків та 1,516 вільних параметрів.

Четвертий шар реалізує підвибірку й складається з 20 карт ознак розміром 3x3. Цей шар мало в чому відрізняється від другого шару: кожний його елемент пов'язаний

з областю 2x2 на відповідній карті попереднього шару. Четвертий схований шар містить 2,000 з'єднань та 32 вільних параметра.

П'ятий схований шар є повнозв'язним згорточним шаром і містить 20 елементів, кожний з них з'єднано областями розміру 3x3 з усіма 20 картами четвертого шару. У цьому змісті він є гібридом згорточного й повнозв'язного шарів – оскільки розмір його карт рівний 1x1, він може ухвалювати сигнал рецептивного поля розміром 3x3 і при цьому, він зв'язаний з усіма картами попереднього шару. П'ятий шар містить 48,120 вільних параметрів. Шостий шар містить 2 нейрона, які використовуються як вихідні.

Як і у випадку з класичними нейронними мережами, на вхід кожного елемента мережі надходить зважена сума – скалярний добуток між вхідним вектором сигналів та ваговим вектором, який потім подається в якості аргументу функції активації. У цьому випадку, такою функцією є сигмоїдальна функція.

В якості вихідних величин використовуються виходи двох нейронів останнього шару. Навчання проводиться таким чином, щоб при поданні на вхід мережі обличчя мережа видавала значення *1; 0*, а при поданні на вхід зображення, що не є обличчям — навпаки, *0; 1*.

Для оптимізації швидкості та стабільності навчання згорточних мереж можна використовувати стандартні механізми оптимізації нейронних мереж: пакетне навчання, використання матриці Гессе, пропуск зворотного ходу в випадку малої помилки й інші. На практиці, краща здатність мережі до узагальнення досягається поповненням навчальної вибірки деформованими прикладами, а також вилученням надмірно перекручених прикладів, на вивчення яких мережа витрачає багато часу.

2.4 Недоліки згорточних нейронних мереж

Згорточні нейронні мережі широко використовуються для рішення задач розпізнавання об’єктів, у тому числі обличчя людини. Не дивлячись на те, що цей клас штучних нейронних мереж успішно виконує покладене на них завдання, ці нейронні мережі не позбавлені недоліків:

1. *Висока складність архітектури*. Сама по собі складність будови нейронної мережі не становить проблеми, але згорточні нейронні мережі для нормальної роботи повинні мати великий розмір. Згорточні нейронні мережі можуть займати багато місця, що робить важкою задачу використання цих мереж на комп'ютерних архітектурах малої потужності. Хоча згорточна нейронна мережа навчається тими самими алгоритмами (з деякими змінами), що й інші нейронні мережі, але навчання великої мережі може бути досить довгим, що знову ж таки обмежує їх використання.
2. *Повнозв'язність*. У попередньому розділі обґрунтовано переваги згорточних нейронних мереж над класичними багатошаровими персептронами, а також дається пояснення недолікам повно зв'язних нейромережевих архітектур. Нажаль класична згорточна нейронна мережа також є повнозв'язною. Звісно існують способи скоротити кількість зв'язків у багатошарових персептронах за рахунок розрідження їх до 70-80%, але для згорточних нейронних мереж ці евристичні підходи не прийнятні. Далеко не всі міжнейронні зв'язки можна розріджувати у згорточній нейронній мережі.
3. *Фіксована площа вікна шару згортки дозволяє знаходити ознаки першого порядку лише одного певного розміру*. Розглянемо цей пункт на прикладі.

На рисунку 2.6 схематично показаний процес пошуку ознаки на вхідному зображенні. Для спрощення пояснення приймаємо, що такою примітивною ознакою є коло.

Згідно з архітектурою згорточної нейронної мережі вхідний шар нейронів поділено на вікна розміром 5x5. Для простоти зображення на схемі не наведено суміжні вікна, що перетинаються, а також представлено лише одну карту ознак з шару згортки.

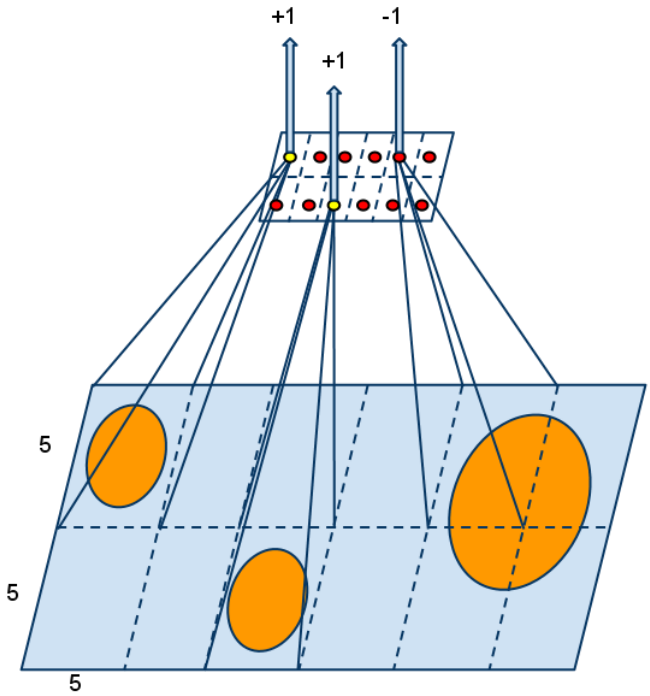

Рисунок 2.6 – Пошук ознак першого порядку мережею з вузьким вікном

Оскільки розмір вікна становить 5x5, то така карта ознак спроможна знайти лише ті кола, які можуть бути вписані у відповідне вікно. Більше того, якщо коло буде замалим, то відповідний нейрон карти ознак може не прийняти його за коло та ознака буде втраченою.

Проблема очевидна – усі кола, що не можуть бути вписані у вікно, будуть проігноровані у першому шарі згортки. Це не означає, що така ознака не буде приймати участі у процесі ідентифікації, але вона буде розкладена на сукупність дуг і у першому шарі згортки можливо буде знайдена як сукупність ознак, але мережа на першому етапі ідентифікації не зможе отримати інформацію про те, що ця ознака – коло. Нейронна мережа зможе відтворити коло лише на вищих шарах, але це може негативно вплинути на процес розпізнавання.

Проблему можна було б вирішити збільшенням вікна, але в такому разі проблема з’являється з іншого боку. На рисунку 2.7 зображено карту ознак з занадто широкими розмірами вікна – 10x10.

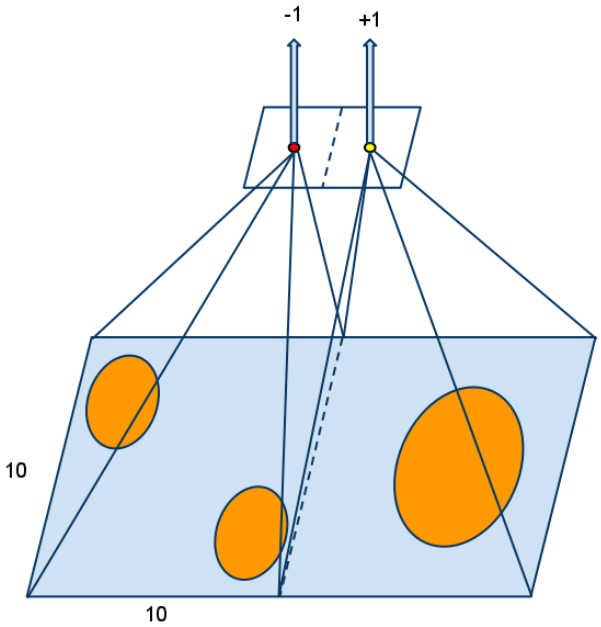

Рисунок 2.7 – Пошук ознак першого порядку мережею з широким вікном

При розпізнаванні облич людей правильно підібраний розмір вікна може знаходити 80-90% наявних ознак у правильному вигляді. Інші ознаки декомпонуються та спотворюють процес розпізнавання.

У даній роботі запропоновано декілька модифікацій до згорточної нейронної мережі, що роблять спробу виправити частину з зазначених недоліків.

2.5 Шляхи подолання недоліків згорточних нейронних мереж

Вище було вказано на основний недолік згорточних нейронних мереж щодо розпізнавання облич людей. Протягом роботи з згорточними нейронними мережами було запропоноване наступне рішення: розмістити у першому шарі згортки карти ознак з різною кількістю нейронів і, відповідно, такі карти ознак повинні мати вікна згортки різних розмірів.

Приклад такої компоновки шару згортки схематично зображено на рисунку 2.8.

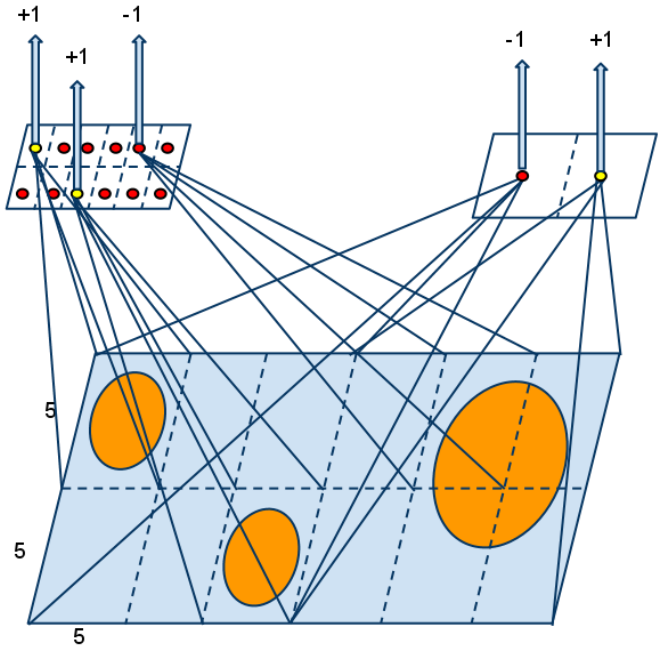

Рисунок 2.8 – Пошук ознак першого порядку мережею з комбінованим вікном

Ознака, пропущена першою картою ознак (велике коло) сприймається вікном більшої площі і таким чином не втрачається при обробці ознак першого порядку.

Така структура нейронної мережі повинна збільшити кількість інформації, що отримується з вхідного шару нейронів і, таким чином, позитивно вплинути на точність роботи згорточної нейронної мережі.

Але, нажаль, така структура порушує симетричність нейронної мережі. Схематично це можна пояснити так: на рисунку 2.9 зображено згорточну нейронну мережу з однією парою шарів згортки-субдискретизації та трьома вихідними нейронами. В реальній згорточній мережі з більшою кількістю шарів на місці цих нейронів знаходяться карти ознак вищих порядків.

З'єднання додаткових шарів з шаром субдискретизації (*SS*) вимагає симетрії нейронної мережі, тобто однакового розміру площин *SS* (*N/2 x N/2*).

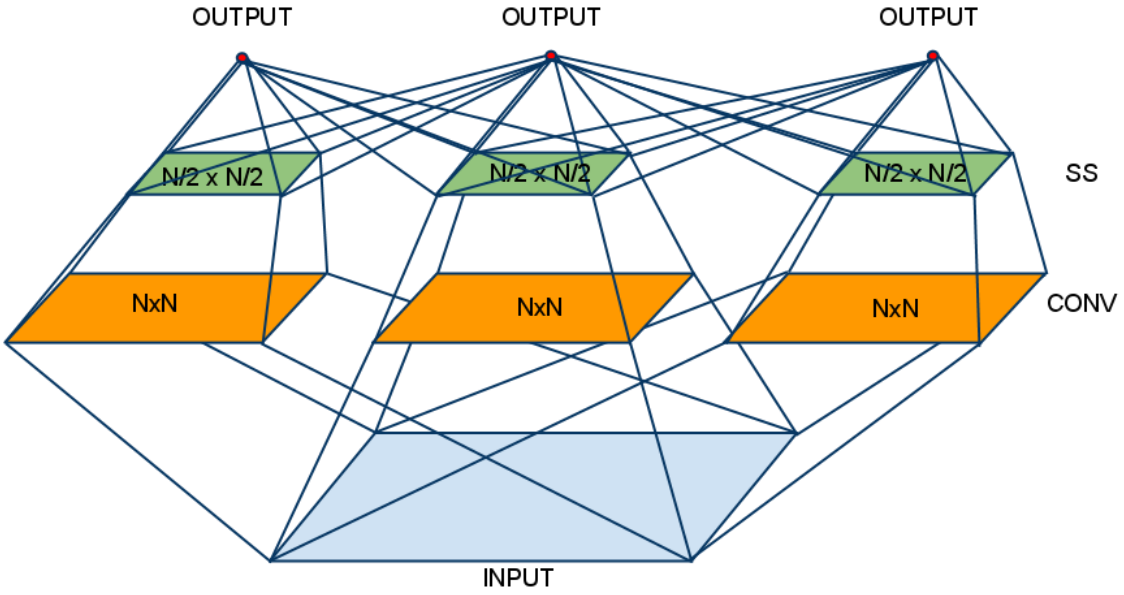

Рисунок 2.9 – Структура простої згорточної нейронної мережі

Запропонована модифікація нейронної мережі, яка передбачає наявність у першому шарі згортки (CONV) карт ознак різного розміру, призведе до ситуації, коли значення розміру карти *N* не є однаковими для сусідніх карт ознак, тобто порушиться симетрія нейронної мережі.

Це призведе до значного ускладнення зв'язків з наступним шаром згортки і зробить роботу нейронної мережі некоректною. Цілком можливо, що така архітектура буде потребувати внесення змін в алгоритм навчання штучної нейронної мережі.

Для подолання цих труднощів було запропоноване наступне вдосконалення – зробити можливість додавати до шару згортки карти ознак однакового розміру, але з різним розміром вікна.

Схематичне зображення такої топології представлене на рисунку 2.10:

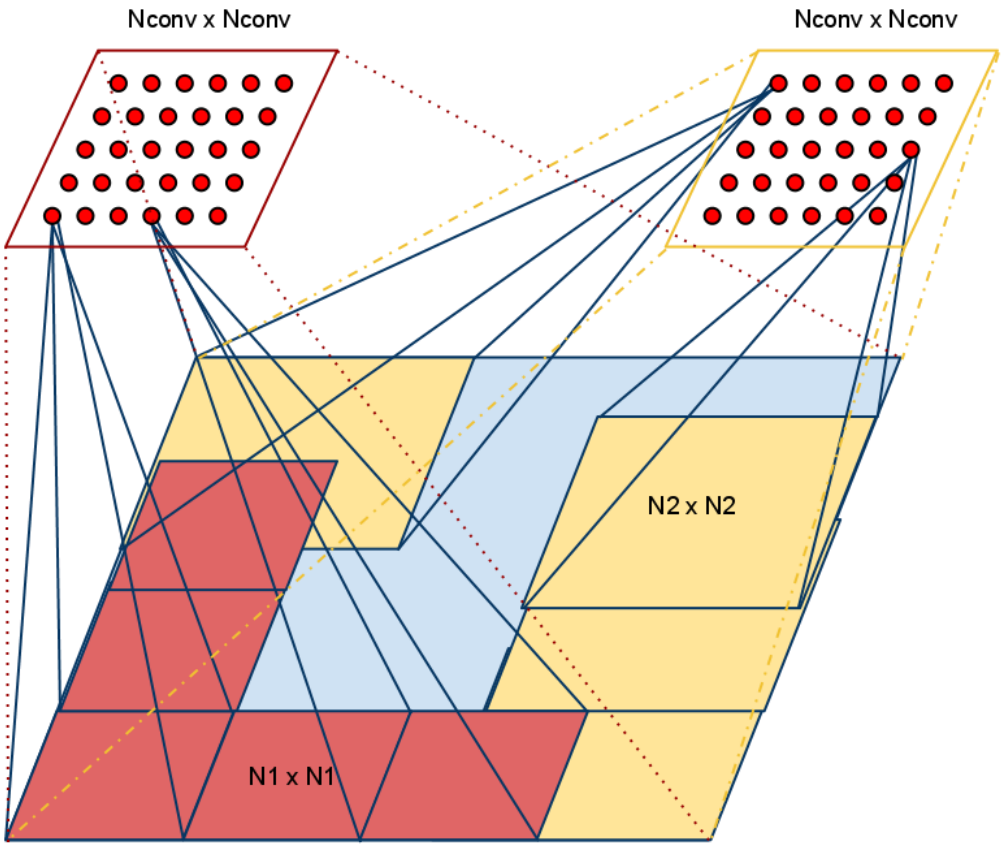

Рисунок 2.10 – Структура простої згорточної нейронної мережі

Змінена згорточна нейронна мережа забезпечує одночасну роботу декількох карт ознак різних розмірів, але кожна з таких карт може мати свій розмір вікна згортки. Кількість таких вікон фіксована для всіх карт ознак та складає N^2_{conv}. З огляду на це зрозуміло, що з таким розташуванням вікон зі збільшенням розміру вікна різко зростає площа перекриття.

Оптимальна площа перекриття сусідніх вікон становить 50-60% від розміру вікна, тобто з такою будовою різко зростає кількість зв’язків між шаром вхідних нейронів та першим шаром згортки.

Для подолання цієї проблеми було прийняте рішення застосувати диференційне зменшення щільності зв’язку.

Зрозуміло, що в звичайній згорточній нейронній мережі не можна застосовувати зменшення щільності міжнейронних зв’язків довільним чином так, як це можливо у багатошарових персептронах.

Наприклад, не рекомендується розріджувати зв’язки всередині пари шарів згортки-субдискретизації, оскільки кожен нейрон шару субдискретизації отримує вхідні дані лише з незначної кількості нейронів попереднього шару (зазвичай 4-9 нейронів).

Так само, не можна розріджувати зв’язки між вхідним шаром та першим шаром згортки, якщо розмір вікна обрано досить малим.

На відміну від класичної згорточної нейронної мережі, для узагальненої згорточної нейронної мережі можливе скорочення кількості зв’язків за рахунок нейронів згортки з великими розмірами вікна.

Пояснюється це тим, що в архітектурі узагальненої згорточної нейронної мережі може бути наявна велика кількість взаємно перетинаючихся сусідніх вікон. Суть диференційного розрідження полягає в тому, що відсоток надлишкових зв’язків встановлюється пропорційно до розміру вікна.

Наприклад, при значенні параметру щільності зв’язку рівним 0.7, видаляється 30% зв’язків. Для нейрона з розміром вікна 3x3 це означає видалення лише 3-х зв’язків, а для вікна 7x7 – це вже 15 зв’язків.

Резюмуючи все вищесказане, можна підкреслити, що узагальнена згорточна нейронна мережа відрізняється від звичайної згорточної нейронної мережі завдяки наявності:

- вікон згортки різного розміру в одному шарі згортки;
- алгоритму диференційного розрідження зв’язків.

2.6 LeNet

Перший вражаючий приклад використання ЗНМ належить Яну Лекуну та його співробітниками, які створили дійсно добрий розпізнавач написаних від руки цифр. Ця мережа була використана для читання ~10% чеків в Північній Америці [8, 10].

Розроблена ЗНМ мала наступні особливості:

- велика кількість схованих шарів;
- багато карт ознак у кожному шарі;
- об’єднання виходів близьких виділених ознак;
- виконувала розпізнавання, навіть, якщо символи перекривались;
- навчання виконувалось не окремих символів, а повного напису;
- у якості тренувального алгоритму використовувався алгоритм зворотного розповсюдження помилки.

Нижче наведена архітектура LeNet-5 (2.11):

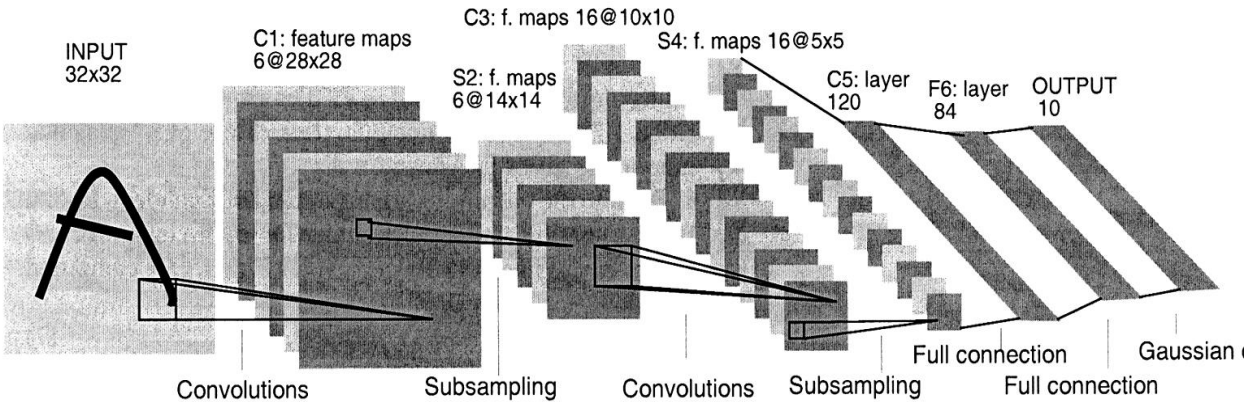

Рисунок 2.11 – Архітектура LeNet-5

При проходженні тесту на 10000 зображень помилка виникла лише у 82 випадках, це більш ніж 99% правильних відповідей. Тим не менше, більшість помилок, що робить LeNet-5, люди досить легко розпізнають. Таким чином, мабуть існує певний спосіб покращити результат, до чого і потрібно прагнути.

3 Огляд існуючих рішень

Для порівняння було обрано 5 найпопулярніших бібліотек машинного навчання: TensorFlow [10], MXNet [11], PyTorch [13] і PyLearn2 [14].

3.1 TensorFlow

TensorFlow [10] – це система машинного навчання, яка працює у широкомасштабних та в неоднорідних середовищах. TensorFlow використовує графи потоку даних (dataflow graphs) для представлення обчислень, спільного стану та операцій, які змінюють цей стан. TensorFlow дозволяє розробникам експериментувати з новими алгоритмами оптимізації та навчання. TensorFlow підтримує безліч додатків зосереджуючись на тренуванні та висновку щодо роботи глибоких нейронних мереж. Кілька служб Google використовують TensorFlow у виробництві, він випущений як проект із відкритим кодом, і він широко використовується для дослідження машинного навчання. Приклад класифікатора зображень, написаного на TensorFlow зображено на рисунку 3.1.

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                 # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                  # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

Рисунок 3.1 - Класифікатор зображень, написаний на TensorFlow [10]

TensorFlow забезпечує просте програмування на основі абстракції потоку даних, яка дозволяє користувачам розгортати додатки на розподілених кластерах, локальних робочих станціях, мобільних пристроях і спеціально розроблених прискорювачів. Високорівневий інтерфейс для написання скриптів обгортає конструювання графів потоку даних і дозволяє користувачам експериментувати з різними архітектурами моделі та алгоритмами оптимізації не змінюючи основну систему.

Основні принципи проектування TensorFlow:

Графи потоку даних примітивних операторів (Dataflow graphs of primitive operators). TensorFlow використовує представлення потоку даних для його моделі, яка являє собою індивідуальні математичні оператори (наприклад, множення матриць, згортка тощо) як вузли в графі потоку даних. Цей підхід полегшує створення користувачами нових шарів, використовуючи високорівневий інтерфейс для написання скриптів. На додаток до функціональних операторів, TensorFlow має змінний стан та операції, що оновлюють його, як вузли на графу потоку даних, що дозволяє експериментувати з різними правилами оновлення.

Відкладене виконання (Deferred execution). Типова програма TensorFlow має дві чіткі фази: перша фаза визначає програму (наприклад, нейронну мережу, яку слід навчати, та правила оновлення) як символічний граф потоку даних із змінними, які будуть заповнюватись вхідними даними та змінними, що представляють стан; і друга фаза виконує оптимізовану версію програми на наборі доступних пристроїв. Відклавши виконання, поки не буде доступна вся програма, TensorFlow може оптимізувати фазу виконання за допомогою глобальної інформації про обчислення. Наприклад, TensorFlow досягає високої утилізації GPU, використовуючи структурну залежність графу, щоб видати послідовність ядер для GPU не чекаючи проміжних результатів.

Загальна абстракція для гетерогенних прискорювачів На додаток до універсальних пристроїв, таких як багатоядерний процесор і графічний процесор, спеціальних прискорювачі для глибокого машинного навчання може досягти значних

покращень у продуктивності і енергозбереження. У Google побудовано спеціальний блок обробки тензора (Tensor Processing Unit, TPU) для машинного навчання. Для підтримки цих прискорювачів в TensorFlow, визначено загальну абстракцію для пристроїв. Як мінімум, пристрій повинен реалізувати методи для створення ядра для виконання, розподілу пам'яті для входів і виходів і передачі буферу з і до пам’яті хоста. Кожен оператор (наприклад, матричне множення) може мати кілька спеціалізованих реалізацій для різних пристроїв. У результаті та ж сама програма може легко націлюватися на графічні процесори, TPU або мобільні процесори як це потрібно для тренувань.

TensorFlow використовує тензори примітивних значень як загальний формат обміну, який розуміють всі пристрої. На найнижчому рівні, всі тензори в TensorFlow щільні. Це рішення гарантує, що на найнижчих рівнях система має просту реалізацію для виділення пам'яті і серіалізації даних, тим самим зменшуючи основні накладні витрати.

Модель виконання TensorFlow. TensorFlow використовує єдиний граф потоку даних для представлення всіх обчислень та стану в алгоритмі машинного навчання, включаючи окремі математичні операції, параметри та їх правила оновлення, а також попередню обробку вхідних даних (рисунок 3.2.).

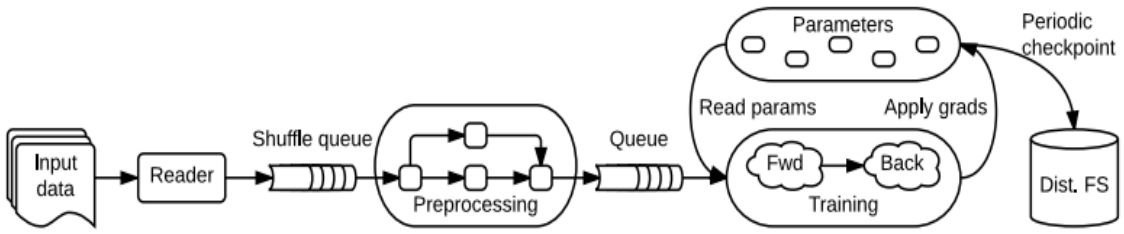

Рисунок 3.2 - Схематичний граф потоку даних TensorFlow для тренування, що містить підграфи для зчитування вхідних даних, попередньої обробки, тренування та періодичне збереження [10]

Граф потоку даних виражає комунікацію між підобчисленнями явно, таким чином, що дозволяє легко виконувати незалежні обчислення паралельно і розділити обчислення на кількох пристроях. TensorFlow відрізняється від пакетних систем потоку даних у двох аспектах:

1. Модель підтримує кілька одночасних виконань на перекритті підграфів загального графа.

2. Індивідуальні вершини можуть мати змінюваний стан, який може бути розподіленим між різними виконаннями графа.

Основне спостереження в архітектурі сервера параметрів полягає в тому, що змінюваний стан має вирішальне значення для тренування дуже великих моделей, тому що стає можливим робити оновлення для дуже великих параметрів на місці та поширювати ці оновлення для паралельних обчислень, настільки швидко, як це можливо. Потік даних із змінним станом дозволяє TensorFlow імітувати функціональність сервера параметрів, але з додатковою гнучкістю, оскільки це стає можливим для виконання довільних підграфів потоків даних на машинах, що приймають загальні параметри моделі. В результаті, користувачі змогли експериментувати з різними алгоритми оптимізації, схеми узгодженості та стратегії розпаралелювання.

Елементи графу потоку даних. У графі TensorFlow кожна вершина являє собою одиницю локального обчислення, і кожне ребро представляє вихід від або вхід до вершини. Розрахунки у вершинах – це операції, а значення, які проходять уздовж ребер – це тензори.

Тензор. У TensorFlow всі дані моделюються як тензори (n-мірні масиви) з елементами, які мають примітивний тип, таких як *int32*, *float32* або *string* (де *string* може бути довільними бінарними даними). Тензори, природно, представляють входи і результати загальних математичних операцій в багатьох алгоритмах машинного навчання: наприклад, множення матриць приймає два 2-D тензори і віддає 2-D тензор; пакет 2-D згортки приймає два тензори 4-D і віддає ще один 4-D тензор.

Операції. Операція приймає $m \geq 0$ тензорів на вхід і віддає $n \geq 0$ тензорів на виході. Операція має іменований "тип" (наприклад, Const, MatMul або Assign) і може мати нуль або більше атрибутів під час компіляції, які визначають її поведінку. Операція може бути поліморфічною і варіативною під час компіляції: її атрибути визначають обидва очікувані типи та арність її входів і виходів.

Наприклад, найпростіша операція *Const* не має вхідних даних і має єдиний вихід; її значення є атрибутом під час компіляції. Наприклад, *AddN* сумує декілька тензорів того ж примітивного типу, і вона має атрибут типу *T* і цілочисельний атрибут *N*, який визначає його тип.

Операції з відслідковуванням стану: змінні (*variables*). Операції можуть містити змінний стан, що читається і / або записується під час кожного виконання. Операція *Variable* містить змінний буфер, який може бути використаний для зберігання спільних параметрів моделі, яку тренують. *Variable* не має вхідних даних; і віддає посилання обробника (*reference handle*), який діє як типізована можливість читання та запису буфера. Операція Read приймає посилання обробника *r* як вхід і вихід значення змінної *(State [r])* як щільний тензор. Інші операції змінюють буфер: наприклад, *AssignAdd* приймає посилання обробника *r* та значення тензору *x*, а при виконанні оновлює $State'[r] \leftarrow State[r] + x$. Подальші операції *Read(r)* віддають значення $State'[r]$.

Операції з відслідковуванням стану: черги (queues). TensorFlow включає кілька реалізації черги. Найпростіша черга – це *FIFOQueue*, що володіє внутрішньою чергою тензорів, і підтримує одночасний доступ у порядку first-in-first-out. Інші типи черг віддають тензори у випадковому та пріоритетному порядку, які гарантують, що вхідні дані відбираються відповідно. Як *Variable*, операція *FIFOQueue* віддає посилдання обробника, який може бути прийнятий однією з стандартних операцій черги, таких як *Enqueue* і *Dequeue*. *Enqueue* буде блокувати, якщо його задана черга повна, і *Dequeue* блокуватиметься, якщо ця черга порожня. [10]

3.2 MXNet

Можливі парадигми програмування варіюються від *імперативних*, де користувач вказує "як" повинно бути виконане обчислення, а також *декларативних*, де користувач фокусується на тому, "що" буде зроблено.

Що стосується проблеми парадигм програмування, то це те, як здійснюється обчислення. Виконання може бути *конкретним*, де результат одразу ж повертається у тому самому потоці, або *асинхронним* чи *відкладеним*, де операції збираються та перетворюються в граф потоку даних як проміжне представлення, перш ніж перейти до виконання на доступних пристроях. Основні відмінності між імперативним і декларативним підходом зображено у таблиці 3.1.

Таблиця 3.1 – Відмінності між імперативним та символічним підходами

	Імперативний підхід	Декларативний підхід
a = b + 1	Одразу ж виконує та зберігає результати в *a* того ж самого типу як і *b*	Повертає граф обчислень; прив'язує дані до b і виконує обчислення пізніше
Переваги	Концепутально є прямолінійним підходом, і часто працює без проблем з вбудованими структурами даних, функціями, дебагером і сторонніми бібліотеками конкретної мови програмування	Отримує повний граф обчислень перед виконанням, корисний для оптимізації і звільнення пам'яті. Також зручно реалізовувати функції такі як завантаження, збереження і візуалізація

MXNet має на меті комбінацію переваг цих різних підходів. Декларативне програмування пропонує чітку межу на глобальному графі обчислень, відкриваючи більше можливостей для оптимізації, тоді як імперативні програми пропонують більшу гнучкість. В контексті глибокого навчання, декларативне програмування є

корисним для визначення структури обчислень в конфігураціях нейронної мережі, тоді як імперативне програмування є більш природним для параметрів оновлення та інтерактивного налагодження.

Незважаючи на підтримку декількох мов та поєднання різних парадигм програмування, MXNet здатний поєднати виконання з єдиним движком на бекенді. Движорк відстежує дані залежностей через обчислювальні графи та імперативні операції, а також ефективно розподіляє їх за часом. MXNet сильно знизили об'єм пам'яті, виконуючи оновлення на місці та перевикористовують пам'ять всюди, де це можливо. MXNet розробили компактний API для того, щоб запустити програму MXNet на кількох машинах з невеликими змінами.

Інтерфейс програмування

Декларативні символічні вирази. MXNet використовує символічні вирази, що мають багато виходів, *Symbol*, що визначають обчислювальний граф. Символи складаються у оператори, такі як прості матричні операції (наприклад, "+"), або складні шари нейронних мереж (наприклад, шар згортки). Оператор може приймати кілька входів, віддавати більше одного вихідного значення і мають внутрішні змінні стану. Змінна може бути або вільна, яку ми можемо прив’язати із значенням пізніше, або з вихідом іншого символу. На рисунку 3.3 показано побудову багатошарової мережі і деяких операцій з шарами нейронної мережі.

```
using MXNet
mlp = @mx.chain mx.Variable(:data) =>
  mx.FullyConnected(num_hidden=64) =>
  mx.Activation(act_type=:relu)    =>
  mx.FullyConnected(num_hidden=10) =>
  mx.Softmax()
```

```
>>> import mxnet as mx
>>> a = mx.nd.ones((2, 3),
... mx.gpu())
>>> print (a * 2).asnumpy()
[[ 2.  2.  2.]
 [ 2.  2.  2.]]
```

Рисунок 3.3 – Багатошарова мережа та операції із шарами за допомогою MXNet [11]

Для виконання символу нам потрібно пов'язати вільні змінні з даними та визначити потрібні результати. Крім виконання ("прямий прохід"), символ підтримує

автоматичне символічне диференціювання ("зворотній прохід"). Інші функції, такі як завантаження, збереження, оцінка пам'яті та візуалізація, також надаються для символів.

NDArray: імперативне обчислення тензора. MXNet пропонує *NDArray* з імперативним обчисленням тензора для заповнення прогалини між декларативним символічним виразами та мовою хоста. На рисунку вище наведено приклад, який виконує множення константних матриць на GPU, а потім виводить результати за допомогою *numpy.ndarray*.

NDArray абстракція працює без проблем з виконанням, оголошеними *Symbol*, оскільки MXNet дозволяє змішувати імперативне тензорне обчислення з іншим. Наприклад, дано символічну нейронну мережу та функцію оновлення ваги, наприклад $w = w - \eta g$. Тоді ми зможемо реалізувати градієнтний спуск по

*while (1) {net.foward_backward (); net.w - = eta * net.g};*

Вищезгадана реалізація така ж ефективна, як і реалізація, що використовує єдиний але часто набагато складніший символічний вираз/ Причиною є те, що MXNet використовує ліниву ініціалізацію NDArray та движка, що може правильно вирішувати залежність даних між двома підходами.

Графи обчислень. Прив’язаний символічний вираз представляється у вигляді графу обчислень для виконання. На рисунку 3.4 показано пряме і зворотнт проходження графа. Перед виконанням MXNet перетворює граф, щоб оптимізувати ефективність і виділяє пам’ять для внутрішніх змінних.

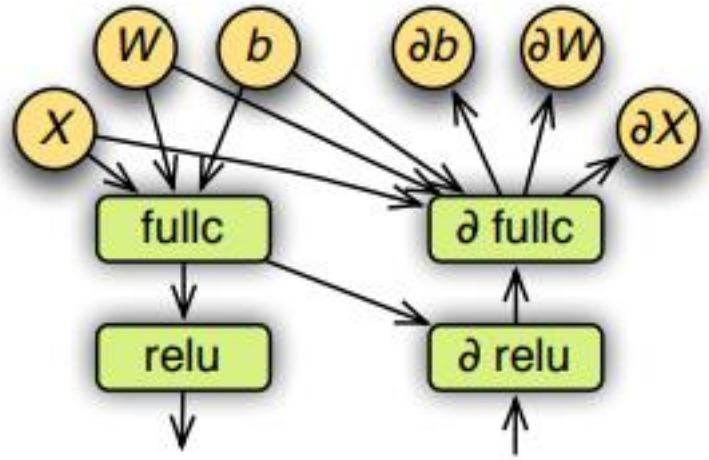

Рисунок 3.4 – Пряме та зворотне проходження графа [11]

Оптимізація графа. Спочатку обраховується тільки необхідний підграф, що потрібен для результатів, отриманих під час прив’язки. Наприклад, при прогнозуванні потрібний лише граф при прямому проходженні, тоді як при виділенні ознак з внутрішніх шарів, останні шари можуть бути пропущені. По-друге, оператори можуть бути згрупован. Нарешті, у MXNet вручну оптимізовано "Великі" операції, такі як шари в нейронній мережі.

Розподіл пам'яті. Час життя кожної змінної, а саме період між створенням і останнім часом використання, відоме графу обчислень. Таким чином стає можливим повторно використати пам'ять для цих змінних. Однак для ідеального розподілу стратегія вимагає $O(n^2)$ складность часу, де n - це число змінних. [11]

3.3 PyTorch

PyTorch – це пакет python, який надає дві основні функції:

1. Тензорні обчислення (наприклад, numpy) з сильним прискоренням GPU
2. Глибокі нейронні мережі, побудовані на потоковій системі Autodiff

Зазвичай PyTorch використовують як заміна numpy, щоб використати можливості графічного процесора; і платформу для машинного навчання, яка є дуже гнучкою.

PyTorch надає тензори (рисунок 3.5), які можуть жити як на центральному процесорі, так і на графічному процесорі, і прискорити обчислення з великою кількістю даних.

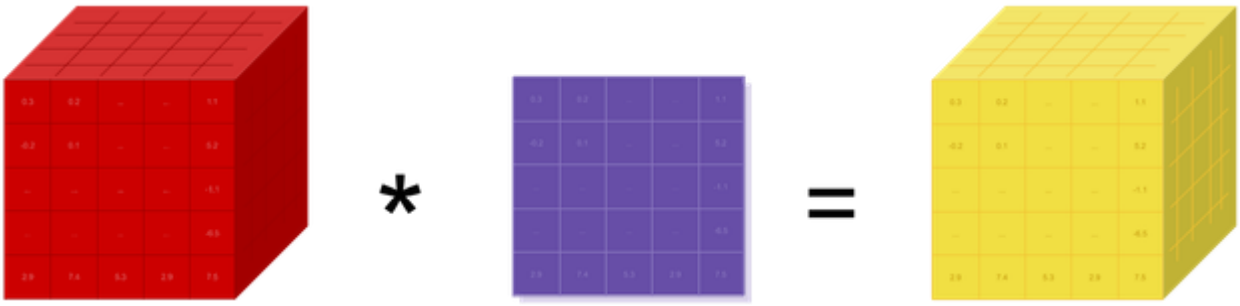

Рисунок 3.5 – Схематичне зображення тензору [13]

Одні з переваг PyTorch:

1. Немає необхідності писати код обгортки для PyTorch.
2. Легко налагоджувати та розуміти код
3. Має стільки типів шарів, як Torch (Unpool, CONV 1,2,3D, LSTM, Grus)
4. Має вбудовані фукнції втрат.
5. Може розглядатися як Numpy розширення для GPU
6. Дозволяє створювати мережі, структура яких залежить від самого обчислення (корисно для навчання з підсиленням)

PyTorch має унікальний спосіб створення нейронних мереж: використання та відтворення стрічки запису.

Більшість бібліотек, таких як TensorFlow, Theano, Caffe та CNTK, мають статичний вигляд. Потрібно побудувати нейронну мережу і повторно використовувати ту саму структуру знову і знову. Зміна способу поведінки мережі означає, що треба починати з нуля.

PyTorch використовує техніку, яка називається зворотньою автодиференціацією, яка дозволяє змінювати те, як працює мережа без додаткових зусиль. Ця техніка не є унікальною для PyTorch, але це одна з найшвидших його

реалізацій на сьогодні. PyTorch дозволяє отримати найкращу швидкість і гнучкість для досліджень. Граф обчислень будується динамічно під час проходження мережі (рисунок 3.6).

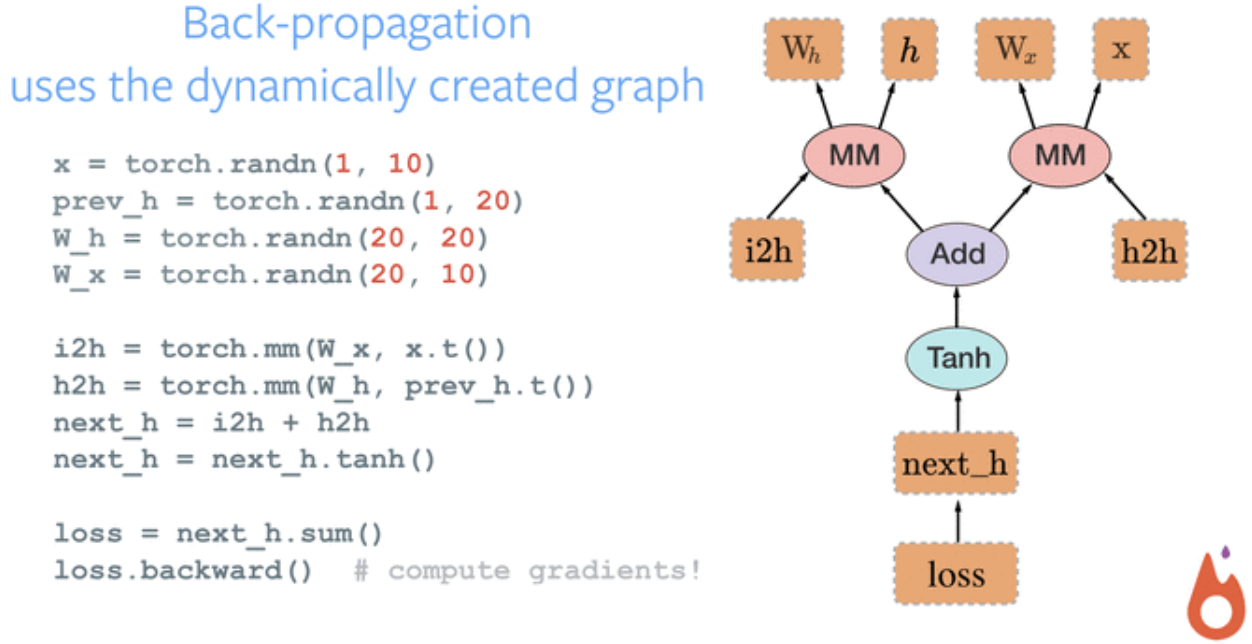

Рисунок 3.6 – Граф обчислень, побудований PyTorch [13]

PyTorch - це не обгортка Python над C++. Він побудований так, щоб бути глибоко інтегрованим в Python. Він може бути використаний це природно, як numpy / scipy / scikit-learn і т. д. Є можливість написати нові шари для нейронної мережі в Python, використовуючи існуючі бібліотеки та використовувати такі пакети, як Cython та Numba.

PyTorch розроблений, щоб бути інтуїтивно зрозумілим, лінійним та легким у використанні. Він використовує імперативний підхід, що означає, що коли виконується рядок коду, то він запускається одразу. Немає асинхронного погляду на світ. Якщо використовувати налагоджувач або отримути повідомлення про помилки та стеки виконання, то розуміння їх є прямим. Стек вказує на те, де саме код був визначений.

PyTorch має мінімальні накладні витрати. Він інтегрує прискорювальні бібліотеки, такі як Intel MKL і NVIDIA (CuDNN, NCCL), щоб максимально збільшити швидкість. PyTorch є досить швидким - чи використовуєте ви малі чи великі нейронні мережі.

Використання пам'яті в PyTorch є надзвичайно ефективним у порівнянні з Torch або деякими альтернативами. Було створено спеціальні розподільники пам'яті для GPU, щоб переконатися, що глибокі моделі навчання максимально ефективні для пам'яті. Це дає змогу навчати більші глибинні моделі навчання, ніж раніше.

Написання нових модулів для нейронних мереж або взаємодія з PyTorch's Tensor API було розроблено так, щоб бути прямим та з мінімальним абстракціями.

Модель на PyTorch представлена на рисунку 3.7. [13]

```
def forward(self, x):
    #Computes the activation of the first convolution
    #Size changes from (3, 32, 32) to (18, 32, 32)
    x = F.relu(self.conv1(x))
    #Size changes from (18, 32, 32) to (18, 16, 16)
    x = self.pool(x)
    #Reshape data to input to the input layer of the neural net
    #Size changes from (18, 16, 16) to (1, 4608)
    #Recall that the -1 infers this dimension from the other given dimension
    x = x.view(-1, 18 * 16 *16)
    #Computes the activation of the first fully connected layer
    #Size changes from (1, 4608) to (1, 64)
    x = F.relu(self.fc1(x))
    #Computes the second fully connected layer (activation applied later)
    #Size changes from (1, 64) to (1, 10)
    x = self.fc2(x)
    return(x)
```

Рисунок 3.7 – Модель нейронної мережі на PyTorch [13]

3.4 Pylearn2

Pylearn2 – бібліотека машинного навчання написана на мові Python [15]. Містить у собі так моделі: ЗНМ, багатошаровий перцептрон, softmax-регресія (як випадок багатошарового перцептрона), обмежена машина Больцмана та інші.

Підтримує наступні методи навчання: стохастичний градієнтний спуск, пакетний градієнтний спуск.

Використовує додаткову бібліотеку Theano для оптимізації та обчислення математичних виразів, що включають багатовимірні масиви ефективно [16]. Також ця бібліотека дозволяє використовувати GPU для обчислення складних операцій. Розробник заявляє, що у майбутньому для цього буде використовуватись не лише CUDA, а й OpenCL. Це означає можливу підтримку GPU не тільки компанії Nvidia, а й інших компаній.

Також варто зазначити, що Theano містить у собі реалізацію ЗНМ, та навіть існує бібліотека pynnet [17], що реалізує інтерфейс для роботи з ними, та вона не підтримується розробником з 2011 року. Сам розробник тепер приймає участь у покращенні та розробці Theano.

Заявленими перевагами Pylearn2 є зрозуміла структура з можливістю створення під-компонентів та доступність для використання в навчальному процесі (використовується в Монреальському університеті). Бібліотека знаходиться у стадії «бурхливої» розробки.

Мінусом даної бібліотеки є відсутність повної документації, але цей недолік виправляється. Також для основних моделей існують навчальні матеріали у форматі ipynb (IPython Notebook).

3.5 Обґрунтування вибору бібліотеки

PyTorch. Це фреймворк, який дозволяє проводити обчислення швидко на графічному процесорі та будувати нейронні мережі. Переваги: підтримка попередньо натренованих моделей, швидка зміна архітектури нейронної мережі завдяки техніці reverse-mode auto-differentiation (автоматичне диференціювання з рухом назад), підтримка візуалізації побудованої архітектури, опис моделі за допомогою методів, що мають назви шарів згорткової нейронної мережі, підтримка візуалізації побудованої архітектури, опис моделі за допомогою методів, що мають назви шарів

згорткової нейронної мережі (conv, relu, max_pool), що пришвидшує початок роботи з цим фремворком, підтримка CUDA. До недоліків можна віднести те, що код для тренування необхідно писати самому, що сповільнює розробку.

TensorFlow. Фреймворк від Google, який замінив Theano, став простішим і швидшим на відміну від попередника, є найпопулярнішим серед розробників. Переваги: архітектура мережі описується у термінах, що використовуються у CNN, має детальну візуалізацію TensorBoard, що будує граф під час навчання, готова реалізація тренування, використовує auto-differentiation для переналаштування архітектури, підтримує паралельні обчислення, підтримка CUDA. Недоліки: значно повільніший за інші фреймворки (наприклад, MxNet), невелика кількість попередньо натренованих моделей, складний для початківця.

MXNet. Фреймворк для машинного навчання від Microsoft і Amazon. Переваги: набагато швидший за інші фреймворки (швидше навчання, менше використання оперативної пам’яті), підтримує попередньо натреновані моделі (у прикладах пристуній готовий код для fine-tuning), опис моделі проводиться у термінах CNN, присутня візуалізація, можливе швидке переналаштування завдяки auto-differentiation, готовий код для тренування, підтримка CUDA. Недоліки: необхідно встановлювати окремі версії фреймоврка для GPU та CPU, невелика кількість документації.

Основні критерії порівняння винесені в таблицю 3.2. Варто додати, що усі фреймворки підтримують навчання на GPU за допомогою CUDA, та мають візуалізацію побудованої мережі. В результаті порівняння кожного фреймворку за обраними критеріями, їх було оцінено по калі від 0 до 5 (від найгіршого результату до найкращого). Результат цієї оцінки представлено в таблиці 3.4.

Таблиця 3.2 – Порівняння бібліотек за обраними критеріями

Бібліотека	Архітектура	Переналаштування	Підтримка попередньо натренованих моделей	Наявність прикладів	Трену-вання	ОС	Пара-лельні обчис-лення
PyTorch	Методи, що мають назву шарів CNN	Auto-differentiation	Присутня велика кількість моделей, легке тренування	Сайт фреймворку, репозиторії на github	Треба писати самому	Windows, Linux	+
TensorFlow	Методи, що мають назву шарів CNN	Auto-differentiation	Не так розповсюджено	Сайт фреймворку, репозиторії на github, статті	Готові методи	Windows, Linux	+
MXNet	Методи, що мають назву шарів CNN	Auto-differentiation	Присутня велика кількість моделей, легке тренування	Сайт фреймворку, репозиторії на github	Готові методи	Windows, Linux	+
PyLearn2	Методи, що мають назву шарів CNN	Auto-differentiation	Присутня велика кількість моделей, легке тренування	Сайт фреймворку, репозиторії на github	Готові методи	Windows, Linux	+

Таблиця 3.3 - Оцінка кожного фреймворку за обраними критеріями

Бібліотека	Архітек-тура	Перенала-штування	Підтримка попередньо натренованих моделей	Наявність прикладів	Трену-вання	ОС	Пара-лельні обчи-слення	**Сума**
PyTorch	4.5	5	4	4	3	5	5	30.5
TensorFlow	4.5	5	3	5	5	5	5	32.5
MXNet	4.5	5	4.5	4	5	5	5	33
PyLearn2	4.5	5	5	5	5	5	5	34.5

Усі наведені бібліотеки містять можливість використовувати GPU для складних математичних операцій, але тільки для Pylearn2 ведеться розробка для підтримки не лише CUDA, а й OpenCL. Також у останньої бібліотеки є досвід впровадження для навчальних занять. З іншого боку інші бібліотеки мають плюс у якості повної документації. Але Pylearn2 має зручну та гнучку структуру, для подальшого розширення, як алгоритмів тренування, датасетів, так і досліджуваних моделей. Отже, Pylearn2 повністю задовольняє вказані вимоги у минулому розділі.

4 Архітектура інструментарію для моделювання ЗНМ

4.1 Архітектура системи

Система передбачає кліент-серверну архітектуру (рисунок 4.1). Моделювання ЗНМ покладено на сервер, де виконуються необхідні обчислення. Користувач працює з ІМЗНМ через web-браузер, що встановлений на персональному комп'ютері. Через web-браузер задаються параметри моделювання, та переглядається отриманий результат. Необхідний доступ до мережі Інтернет.

Рисунок 4.1 – Приклад архітектури

Дана архітектура має такі переваги:

- для роботи користувачу необхідний лише web-браузер;
- відсутність жорстких технічних вимог до компьютера користувача;

- модернізація та оновлення програмного продукту відбувається без втручання користувача;
- безпека та цілісність даних покладена на сервер.

Та наступні недоліки:

- повна залежність від підключення до Інтернет;
- обмеженність серверних ресурсів.

4.2 Вибір сервера

Враховуючи обрану бібліотеку Pylearn2 для ЗНМ, та компоненти, необхідні для її роботи, операційну систему для сервера було обрано CentOS 6.2 64-bit.

CentOS частіше всього використовують, як серверну операційну систему для веб-хостингу. Багато великих хостингових компаній використовують CentOS для забезпечення стабільної роботи для своїх веб-застосунків. Даний дистрибутив Linux базується на основі комерційного дистрибутиву Red Hat Enterprise Linux компанії Red Hat, що гарантує безпеку і стабільність роботи. Життєвий цикл CentOS складає 10 років, підтримка 6-ої версії буде відбуватись до 2020 року.

CentOS віддав перевагу програмному забезпеченню для оновлення на основі yum, хоча підтримка up2date також присутня. Можна використовувати для завантаження та встановлення як додаткові пакунки і залежності, так і спеціальні та періодичні оновлення безпеки з репозиторію на CentOS Mirror Network.

4.3 Фреймворк для розробки web-систем

Враховуючи, що обрана бібліотека для роботи зі ЗНМ написана на Python, фреймворк був обраний теж написаний на цій мові.

Один з варіантів – Flask, що є мікрофреймворком для створення web - сайтів мовою Python. Основна причина чому Flask називається "мікрофреймворком" - це ідея зберегти ядро простим, але розширюваним. У ньому немає абстрактного рівня бази даних, немає валідації форм або всього такого, що вже є в інших бібліотеках до яких ви можете звертатися. Однак Flask підтримує розширення, які можуть додати необхідну функціональність.

Але вибір був зупинений на Django, що має усі необхідні компоненти «з коробки» та такі основні особливості [18]:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами на більшість мов;
- диспетчер URL на основі регулярних виразів;
- розширювана система шаблонів з тегами та наслідуванням
- система кешування;
- інтернаціоналізація;
- авторизація та аутентифікація, підключення зовнішніх модулів аутентифікації: OpenID та ін.;
- бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД);
- вбудована автоматична документація по тегам шаблонів та моделями даних, доступна через адміністративний застосунок.

Сайт на Django будується з однієї або декількох частин, які рекомендується робити модульними. Це одна з істотних архітектурних відмінностей цього фреймворку від деяких інших.

Архітектура Django подібна на «Модель-Вид-Контролер» (MVC). Однак, те що називається «контролером» в класичній моделі MVC, в Django називається «відображення», а те, що мало б бути «відображенням», називається «шаблон». Таким чином, MVC розробники Django називають MTV («Модель-Шаблон-Відображення»).

Початкова розробка Django, як засобу для роботи новинних ресурсів, досить сильно позначилася на його архітектурі: він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований модуль для керування вмістом, який можна включити в будь-який сайт, зроблений на Django, і який може керувати відразу декількома сайтами на одному сервері. Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючі всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав).

У дистрибутив Django також включені програми для системи коментарів, синдикації RSS і Atom, «статичних сторінок» (якими можна управляти без необхідності писати контролери та відображення), перенаправлення URL та інше.

4.4 Вибір СКБД

Обрана бібліотека для web-системи Django дозволяє використовувати наступні СКБД: PostgreSQL (8.2 та вище), MySQL (5.0 та вище), SQLite, Oracle Database Server (9i та вище). Оскільки очікуване навантаження на СКБД має бути невеликим та функції покладені на неї не для моделювання ЗНМ, а тільки для забезпечення роботи з web-системою, було обрано найпоширеніше рішення для web-платформ – MySQL 5.5 [19].

Обрана СКБД використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

Гілка MySQL 5.5 містить ряд значних поліпшень, пов'язаних з підвищенням масштабованості та швидкодії, серед яких:

- використання рушія InnoDB;

– підтримка напівсинхронного (semi-synchronous) механізму реплікації (заснований на патчах від компанії Google);

– поліпшення функцій з партіціювання даних; розширений синтаксис для розбиття великих таблиць на кілька частин, розміщених в різних файлових системах (partitioning); додані операції RANGE, LIST і метод оптимізації «partition pruning»;

– новий механізм оптимізації вкладених запитів та операцій JOIN;

– перероблена система внутрішніх блокувань;

– інтегровані патчі Google з оптимізацією роботи InnoDB на CPU з великою кількістю ядер.

4.5 Web-сервер

Для обраної ОС розглядалось два найпопулярніші web-сервери: Apache та nginx. Обидва рішення надають надійність та гнучкість у конфігурації. Але головна особливість, в першу чергу в тому, що Apache працює за принципом: новий запит - новий потік; nginx працює з використанням моделі подій, відповідно новий запит не породжує нових потоків. Завдяки цій особливості, web-сервер потребує менше оперативної пам'яті, а швидкість його роботи стає вищою.

В nginx робочі процеси обслуговують одночасно безліч з'єднань, мультиплексуючи їх викликами операційної системи select, epoll (Linux) і kqueue (FreeBSD). Робочі процеси виконують цикл обробки подій від дескрипторів. Отримані від клієнта дані розбираються з допомогою кінцевого автомата. Розібраний запит послідовно обробляється ланцюжком модулів, що задається конфігурацією. Відповідь клієнту формується в буферах, які зберігають дані або в пам'яті, або вказують на відрізок файлу. Буфери об'єднуються в ланцюжки, що визначають послідовність, в якій дані будуть

передані клієнтові. Якщо операційна система підтримує ефективні операції вводу-виводу, такі як writev і sendfile, то nginx застосовує їх при нагоді.

Конфігурація HTTP-сервера nginx розділяється на віртуальні сервери (директива server). Віртуальні сервери поділяються на локації (директива location). Для віртуального сервера можливо задати адреси і порти, на яких будуть прийматися з'єднання, а також імена, які можуть включати * для позначення довільній послідовності в перший і останній частині, або задаватися регулярним виразом. Локації можуть задаватися точним URI, частиною URI, або регулярним виразом. Location можуть бути налаштовані для обслуговування запитів зі статичного файлу, проксування на http, fastcgi чи memcached сервер.

Для ефективного управління пам'яттю nginx використовує пули. Пул — це послідовність попередньо виділених блоків динамічної пам'яті. Довжина блоку змінюється в діапазоні від 1 до 16 кілобайт. Спочатку під пул виділяється лише один блок. Блок розділяється на зайняту область і незайняту. Виділення дрібних об'єктів виконується шляхом просування покажчика на незайняту область з урахуванням вирівнювання. Якщо незайнятої області в усіх блоках бракує для виділення нового об'єкта, то виділяється новий блок. Якщо розмір виділеного об'єкта перевищує значення константи NGX_MAX_ALLOC_FROM_POOL, або довжину блоку, то він повністю виділяється з купи. Таким чином, дрібні об'єкти виділяються дуже швидко і мають накладні витрати тільки на вирівнювання.

Вже тривалий час nginx обслуговує сервери багатьох високонавантажених російських сайтів, таких як Яндекс, Mail.Ru, ВКонтакте і Рамблер. Згідно зі статистикою Netcraft, nginx був застосований у 14.08% самих навантажених сайтів у червні 2013 року [20, 21].

4.6 Gateway Interface HTTP-сервера

Враховуючи обрані вище рішення, в якості Python web-сервера, до якого відбувається звернення з nginx, було вирішено використовувати Gunicorn [22]. Він має наступні особливості:

- Підтримка Django з коробки;
- Автоматичне управління робочим процесом;
- Проста конфігурація Python;
- Можливість використовувати кілька конфігурацій;
- Різні hooks для розширюваності;
- Сумісність з Python 2.x >= 2.5.

4.7 Фоновий процес моделювання

Оскільки процес моделювання досить ресурсномісткий, його необхідно виконувати у фоні та асинхронно, бажано навіть на окремому сервері. Тому було обрано бібліотеку ZeroMQ [23].

ZeroMQ (також називають OMQ) – високопродуктивна бібліотека, що призначена для асинхронного обміну повідомленнями. Ця бібліотека призначена для використання в масштабованих розподілених системах чи в паралельних додатках. Вона забезпечує чергу повідомлень, але на відміну проміжного програмного забезпечення, що орієнтоване на обробку повідомлень, система OMQ може працювати без спеціалізованого брокера повідомлень. ZeroMQ розроблена під програмний інтерфейс під назвою Сокети Берлі, що підтримує міжпроцесну взаємодію.

OMQ розроблена корпорацією iMatix разом з великим співтовариством учасників. Є сторонні прив'язки для найбільш популярних мов програмування, таких як: Java, Python, Erlang і Haskell.

Бібліотека без особливих зусиль дозволяє створювати складні комунікаційні рішення. Спочатку ця програмна компонента розроблялась як інтерфейс для обміну повідомленнями (messaging middleware), потім - як

легкий комунікаційний протокол, заснований на TCP / IP, а нині ZeroMQ позиціонується як нова компонента в стеку мережевих протоколів.

Для дипломної роботі ця бібліотека була обрана через такі якості:

1. *Продуктивність* - ZeroMQ дійсно працює істотно швидше ніж більшість реалізацій AMQP, і це досягається завдяки:

- відсутності підтримки AMQP та відповідних до уього протоколу недоліків,
- використанню ефективних транспортів, наприклад широкомовного протоколу з гарантованою доставкою або оригінальної розробки ZeroMQ - набору викликів для многопотокового розсилання повідомлень декільком адресатам,
- використанню агрегованої відправки кількох повідомлень в одному TCP-пакеті, що є теж розробкою ZeroMQ, що дозволяє не тільки мінімізувати витрати мережевого протоколу, а й зменшити кількість системних викликів.

2. *Простота використання* - За допомогою API ZeroMQ передача повідомлення дійсно простіша, ніж при використанні сокетів, де потрібно, наприклад, стежити за довжиною сокетного буфера, а в ZeroMQ - просто ініціювати відправлення повідомлення, а дроблення (або агрегація) і відправка робиться API в окремому потоці, асинхронно з виконанням користувацького коду. Асинхронна природа методів ZeroMQ особливо зручна для реалізації механізмів подієвої обробки. Важливою зручністю в ZeroMQ є відмова від типізації повідомлень переданих інтерфейсом - повідомлення не інтерпретуються інтерфейсом і є BLOB (областю пам'яті). Таким чином, через ZeroMQ можна передавати що завгодно, наприклад повідомлення JSON або виконавчі відформатовані дані типу BSON, Protocol Buffers або Thrift, не відчуваючи при цьому ніяких незручностей.

3. *Масштабованість* - Будучи низькорівневим інтерфейсом, ZeroMQ, тим не менш, надає безліч опцій, наприклад сокет ZeroMQ може бути

підключений до декількох адресатів і рівномірно розподіляти навантаження по мережі. Інша можливість - це вхідне мультиплексування, коли один сокет може отримувати повідомлення від безлічі відправників. У ZeroMQ реалізована децентралізована схема обміну повідомленнями. Це, в комбінації з високою продуктивністю, дає можливість побудови розподілених систем будь-якої складності.

4.8 UI користувача

Для побудови простого, ненавантаженого та зрозумілого інтерфейса для користувача, було обрано CSS-фреймворк Twitter Bootstrap [24]. Він включає CSS та HTML для типографії, форм, кнопок, таблиць, сіток, навігації тощо.

При виборі цього інструменту було враховано популярність даного рішення, високу кількість необхідних UI-елементів, а також широку підтримку сучасними web-браузерами.

Також даний фреймворк має дуже зручний конструктор (рисунок 4.2):

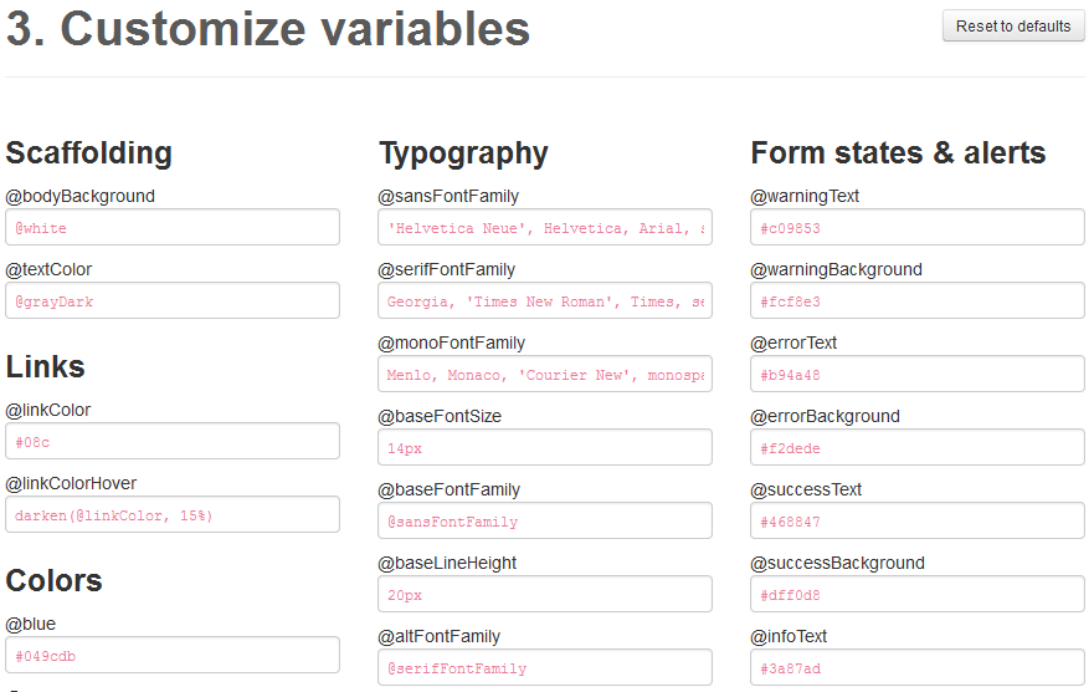

Рисунок 4.2 – Конструктор Twitter Bootstrap

Основні інструменти Twitter Bootstrap:

- сітки — наперед задані розміри колонок, які можна відразу ж використовувати, наприклад ширина колонки 90px відноситься до класу .span2, який ми можемо використовувати в CSS описі документа;
- шаблони — фіксований або гумовий шаблон документа;
- типографіка — опис шрифтів, визначення деяких класів для шрифтів таких як код, цитати і т.п.;
- медіа — представляє певне управління зображеннями та відео;
- таблиці — засоби оформлення таблиць, дозволяє додавати функціональність сортування;
- форми — класи для оформлення не тільки форм але і деяких подій;
- навігація — класи оформлення для табів, вкладок, сторінок, меню і тулбара;
- алерт — оформлення діалогових вікон, підказок і спливаючих вікон.

5 Створення інструментарію для моделювання ЗНМ

5.1 Встановлення обраних засобів

Приймемо за умову, що обрана ОС (CentOs 6.2 64-bit) вже встановлена, а також має встановлені пакети Git, pip, mysql-server, ZeroMQ.

5.1.1 Встановлення Python та налаштування Virtualenv

Спочатку встановимо Python версії 2.7, у обраному дистрибутиві наявна версія 2.6.6.

Також ми будемо використовувати Virtualenv, дана утиліта призначена для створення віртуального середовища проекту [25]. Наприклад, як у нашому випадку, ми бажаємо використовувати у проекті версію Python, відмінну від встановленого Python або які-небудь бібліотеки відмінних версій. Тому ми створимо окреме віртуальне середовище для нашого проекту. Тим самим ми зможемо встановлювати, змінювати і видаляти пакети, і це не вплине на інші проекти або системне середовище. Дана особливість допоможе нам з використанням бібліотек, що знаходяться у активній розробці.

5.1.2 Встановлення Pylearn2

Обрана нами бібліотека для роботи зі ЗНМ, Pylearn2, використовує бібліотеку Theano, яка має ряд вимог, а саме:

- g++ та python-dev - технічно не обов’язково, але настійно рекомендується для компіляції згенерованого коду на C;
- NumPy та SciPy – python бібліотека, що використовується для математичних обчислень;

- BLAS - встановлений де факто стандарт інтерфейсу бібліотек підпрограм, призначених для виконання основних операцій лінійної алгебри, таких як, наприклад, множення матриць та векторів.

Рекомендується використовувати версію Theano прямо з git-репозиторію, для цього виконаємо наступну команду [26]:

pip install git+git://github.com/Theano/Theano.git

Після цього можна встановити інші необхідні для Pylearn2 пакети, а саме:

- PyYAML - бібліотека мови Python для роботи з даними формату YAML.
- PIL - бібліотека мови Python, призначена для роботи з растровою графікою;
- Matplotlib - бібліотека на мові програмування Python для візуалізації даних двовимірної графікою.

Pylearn2 не міститься у каталозі PyPI, тому необхідно використовувати файли з git-репозиторію:

git clone git://github.com/lisa-lab/pylearn2.git

Потім обов’язково необхідно створити змінну середовища з назвою PYLEARN2_DATA_PATH, у якій буде знаходиться шлях до фізичного розташування усіх датасетів:

export PYLEARN2_DATA_PATH=/home /yourname/datasets

5.1.3 Налаштування Django

Для ввімкнення інтерфейсу адміністратора та механізму аутентифікації використовуємо наступну конфігурацію (рисунок 5.1):

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'imznm.frontend',
    'imznm.backend',
)
```

Рисунок 5.1 – Конфігурація Django

Тепер ми маємо можливість зайти до сайту (рисунок 5.2):

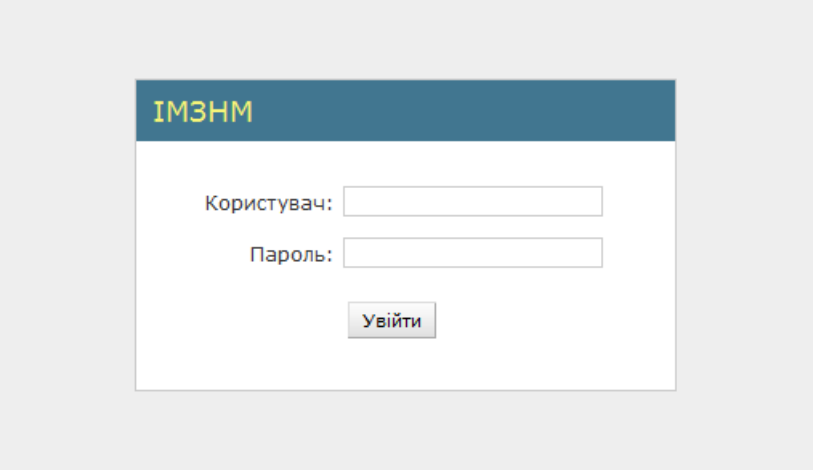

Рисунок 5.2 – Вхід до ІМЗНМ

Користування ІМЗНМ дозволено тільки зареєстрованим користувачам, оскільки задачі, які можна виконувати на сайті дуже ресурсномісткі.

Згідно поставленої задачі, ми маємо 3 ролі (групи) користувачів: Адміністратор, Вчитель, Користувач. Налаштування прав для ролей (Groups) користувача відбувається за наступним посібником [18].

Перейдемо до опису моделей нашого django-додатка:

```
class Dataset(models.Model):
    title = models.CharField()
    folder = models.CharField()
    start = models.IntegerField(default=False, blank=True)
    stop = models.IntegerField(default=False, blank=True)
    one_hot = models.BooleanField(default=True)

    def __unicode__(self):
        return self.title
```

Рисунок 5.3 – Django-модель датасету

Варто зазначити, що параметр one_hot (рисунок 5.3) має тип Boolean. Значення True, вказує на те, що змінна «y» має бути представлена у вигляді вектору. Більш детальну інформацію можна знайти за наступним посиланням [27].

```
class ConvRectifiedLinear(models.Model):
    name = models.CharField()
    output_channels = models.IntegerField()
    irange = models.FloatField()
    kernel_shape = models.CommaSeparatedIntegerField()
    pool_shape = models.CommaSeparatedIntegerField()
    pool_stride = models.CommaSeparatedIntegerField()

    def __unicode__(self):
        return self.name
```

Рисунок 5.4 – Django-модель шару ЗНМ

Для моделі шару ЗНМ (рисунок 5.4), що реалізована класом ConvRectifiedLinear, параметр irange означає випадкове розподілення ваги в діапазоні від -irange до +irange.

```
class CNN(models.Model):
    name = models.CharField()
    num_channels = models.IntegerField()
    shape = models.CommaSeparatedIntegerField()
    dataset = models.OneToOneField(Dataset)
    layers = models.ForeignKey(ConvRectifiedLinear)

    def __unicode__(self):
        return self.name
```

Рисунок 5.5 – Django-модель ЗНМ

5.2 Інтерфейс Користувача

Після входу на сайт ви можете переглянути ЗНМ, що ви використовуєте для моделювання. В залежності від статуси користувачу доступні різні дії (рисунок 5.6).

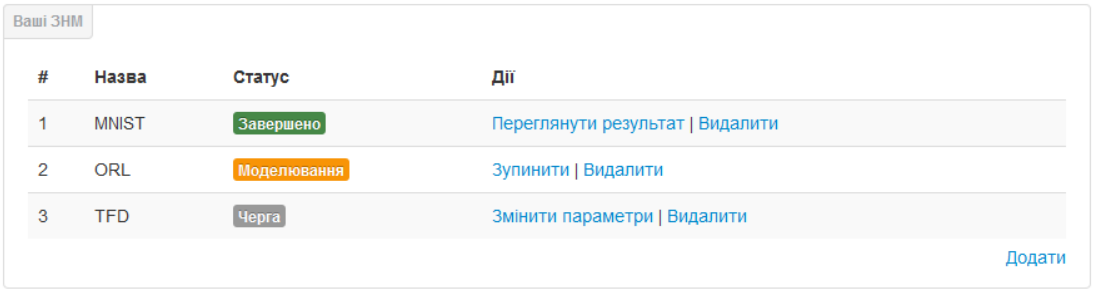

Рисунок 5.6 – Ваші ЗНМ

Зміна статусу ЗНМ описана у діаграмі станів на рисунку 2.3.

Додавання нової ЗНМ (рисунок 5.7) для моделювання відбувається наступним чином: користувачу необхідно обрати датасет, що буду використовуватись для моделювання. Опціонально можна задати інтервал, над яким будуть проводитись обчислення. Це необхідно оскільки датасет може бути різного розміру: від кількох сотень, до мільйона та більше. Швидкість моделювання залежить від розміру датасета. Після того, як ці параметри вказані, можна перейти до архітектури ЗНМ, обрати тип шару, вказати його назву, кількість вихідних каналів, параметри ядра згортки та інше.

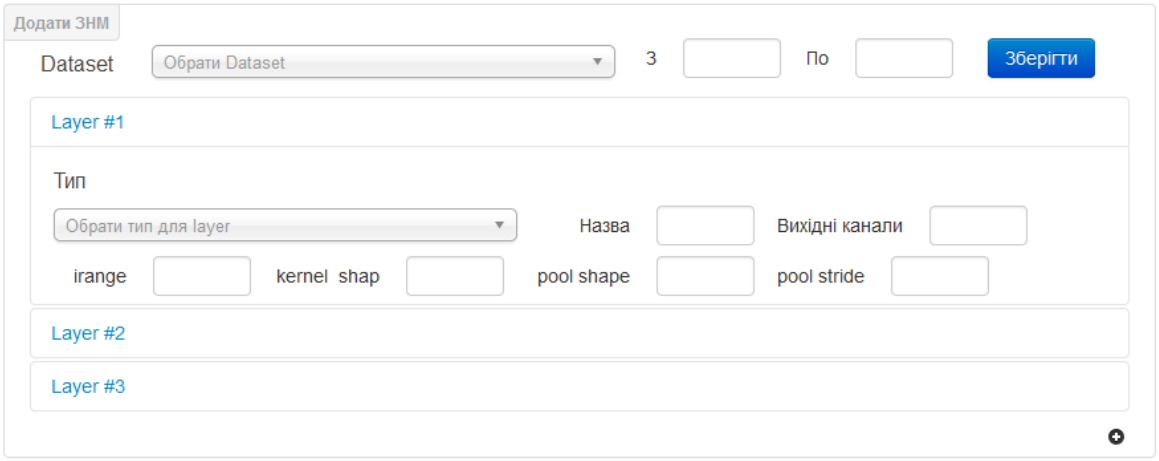

Рисунок 5.7 – Додавання ЗНМ

Після моделювання можна переглянути результати, вони доступні наступним чином (рисунок 5.8):

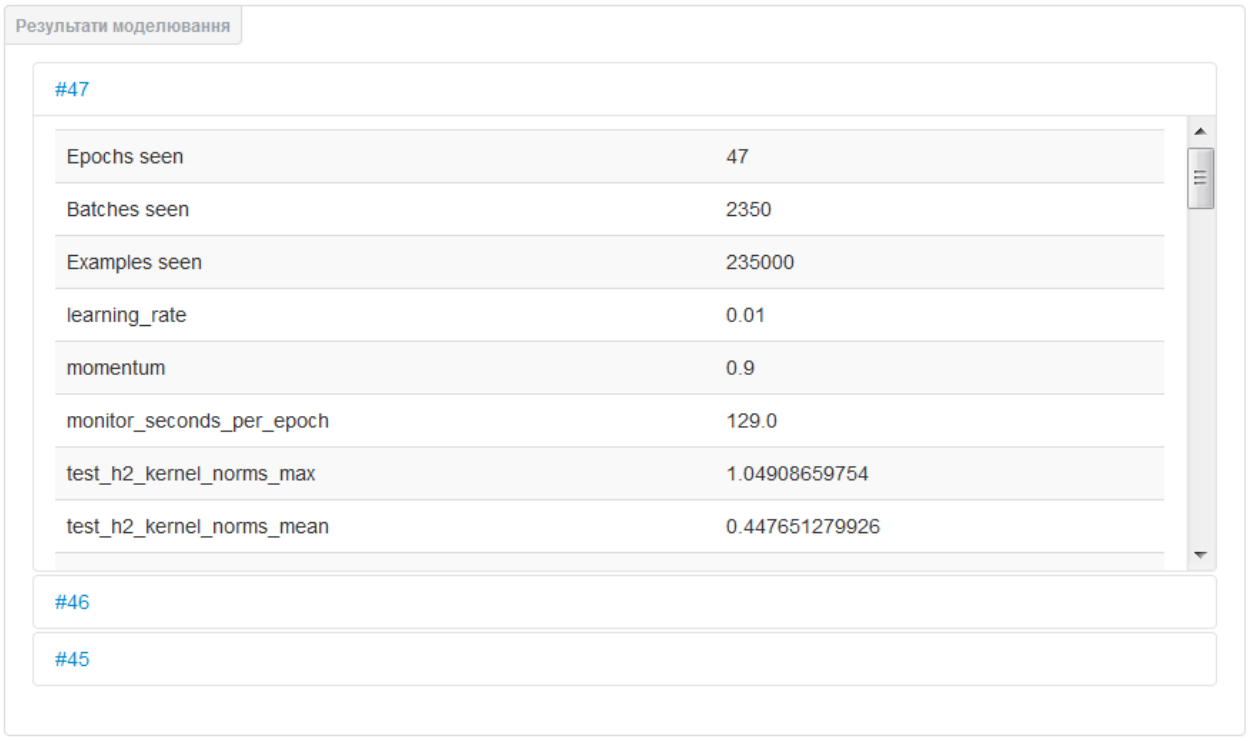

Рисунок 5.8 – Приклад результатів моделювання

Для кожного параметра можна побудувати графік. Наприклад, графік помилки розпізнання (рисунок 5.9):

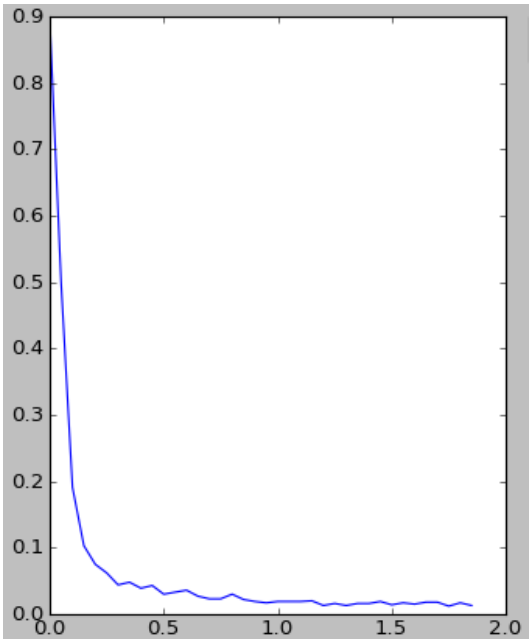

Рисунок 5.9 – Приклад, графік помилки розпізнання

Методом градієнтного спуску був отриманий паттерн (рисунок 5.10), який максимізує значення вихідний активації класифікатора при заданих параметрах нейронної мережі:

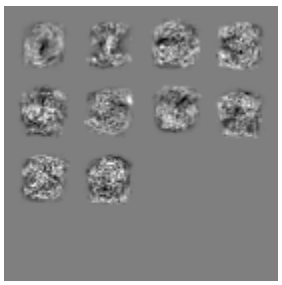

Рисунок 5.10 – Приклад паттерну

Повний приклад результатів моделювання наведений у додатку А.

Висновки

В даному бакалаврському проекті було розроблено інструментарій для моделювання згорткових нейронних мереж.

В перших трьох розділах увагу присвячено опису предметної області, огляду існуючих методів, способів та рішень для моделювання ЗНМ.

На основі виконаного огляду було обрано бібліотеку pylearn2, написану ну мові Python. Враховуючи поставлену задачу, у 4 розділі було обрано допоміжні компоненти для реалізації поставленої мети.

Отримане рішення має переваги у вигляді доступності та простоти у користуванні, але існує недолік, такий як залежність від ресурсів сервера. З іншого боку, сервером може виступати будь-який комп’ютер з ОС Linux, або навіть GPU-кластер.

Список літератури

1. Методичні вказівки до виконання дипломних робіт освітньо-кваліфікаційного рівня «бакалавр» для студентів напряму підготовки 126 «Інформаційні системи» кафедри «Автоматики та управління у технічних системах» – К.: «Політехніка», 2018.
2. ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012) [Электронний ресурс]. – Режим доступу: http://www.image-net.org/challenges/LSVRC/2012/
3. Команда Джеффри Хинтона победила в конкурсе компьютерного зрения ImageNet с двукратным преимуществом / Хабрахабр [Электронний ресурс]. – Режим доступу: http://habrahabr.ru/post/183380/
4. GPGPU — Википедия [Электронний ресурс]. – Режим доступу: http://ru.wikipedia.org/wiki/GPGPU
5. LeCun Y. A theoretical framework for backpropagation // Proc. of IEEE. - 1998. - P.21-28.
6. LeCun Y., Bottou L., Bengio Y., Haffne P. Gradient-Based Learning Applied to Document Recognition // Proc. IEEE. - 1998. - P.59-67.
7. Дорогий Я.Ю. Модифицированный алгоритм обучения сверточных нейронных сетей. //Сборник материалов. VII Международная научно-практическая конференция “ПЕРСПЕКТИВЫ РАЗВИТИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ”. 18 апреля 2012, г. Новосибирск: Издательство НГТУ. – с 34-38.
8. Convolutional nets for digit recognition [16 min] | Neural Networks for Machine Learning [Электронний ресурс]. – Режим доступу: https://class.coursera.org/neuralnets-2012-001/lecture/73
9. Feature extraction using convolution - Ufldl [Электронний ресурс]. – Режим доступу: http://ufldl.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

10. Yann LeCun's Home Page [Электронний ресурс]. – Режим доступу: http://yann.lecun.com

11. Convolutional neural network class - Mikhail Sirotenko's home page [Электронний ресурс]. – Режим доступу: https://sites.google.com/site/mihailsirotenko/projects/convolutional-neural-network-class

12. cuda-convnet - High-performance C++/CUDA implementation of convolutional neural networks - Google Project Hosting [Электронний ресурс]. – Режим доступу: http://code.google.com/p/cuda-convnet/

13. nnForge [Электронний ресурс]. – Режим доступу: http://milakov.github.io/nnForge/

14. dwf/convolupy [Электронний ресурс]. – Режим доступу: https://github.com/dwf/convolupy

15. Welcome - Pylearn2 dev documentation [Электронний ресурс]. – Режим доступу: http://deeplearning.net/software/pylearn2/

16. Welcome - Theano 0.6rc3 documentation [Электронний ресурс]. – Режим доступу: http://deeplearning.net/software/theano/

17. pynnet - Neural Network Library in Python/theano - Google Project Hosting [Электронний ресурс]. – Режим доступу: http://code.google.com/p/pynnet/

18. Using the Django authentication system | Django documentation | Django [Электронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/dev/topics/auth/default/#topic-authorization

19. MySQL — Вікіпедія [Электронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/MySQL

20. nginx — Вікіпедія [Электронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/Nginx

21. June 2013 Web Server Survey [Электронний ресурс]. – Режим доступу: http://news.netcraft.com/archives/2013/06/06/june-2013-web-server-survey-3.html

22. Gunicorn (HTTP server) - Wikipedia, the free encyclopedia [Электронний ресурс]. – Режим доступу: http://en.wikipedia.org/wiki/Gunicorn_%28HTTP_server%29

23. ØMQ - Wikipedia, the free encyclopedia [Электронний ресурс]. – Режим доступу: http://en.wikipedia.org/wiki/ZeroMQ

24. Bootstrap [Электронний ресурс]. – Режим доступу: http://twitter.github.io/bootstrap/

25. How to install Python 2.7 and 3.3 on CentOS 6 | Too Much Data [Электронний ресурс]. – Режим доступу: http://goo.gl/WgVlI

26. Easy Installation of an optimized Theano on CentOS 6 - Theano 0.6rc3 documentation [Электронний ресурс]. – Режим доступу: http://deeplearning.net/software/theano/install_centos6.html#install-centos6

27. pylearn2/pylearn2/scripts/tutorials/softmax_regression.ipynb at master [Электронний ресурс]. – Режим доступу: https://github.com/lisa-lab/pylearn2/blob/master/pylearn2/scripts/tutorials/softmax_regression.ipynb

Додаток А Приклад результатів моделювання

…

Epochs seen: 46

Batches seen: 2300

Examples seen: 230000

learning_rate: 0.01

momentum: 0.99

monitor_seconds_per_epoch: 128.0

test_h2_kernel_norms_max: 1.55053091512

test_h2_kernel_norms_mean: 0.624761569171

test_h2_kernel_norms_min: 0.147479767746

test_h3_kernel_norms_max: 1.83767522423

test_h3_kernel_norms_mean: 1.5324467745

test_h3_kernel_norms_min: 0.973684229374

test_objective: 0.0517527083304

test_term_0: 0.0407800294774

test_term_1_weight_decay: 0.0109726788529

test_y_col_norms_max: 1.87857623558

test_y_col_norms_mean: 1.85864577076

test_y_col_norms_min: 1.82825909148

test_y_max_max_class: 0.999999999993

test_y_mean_max_class: 0.994979275961

test_y_min_max_class: 0.820620397804

test_y_misclass: 0.008

test_y_nll: 0.0407800294774

test_y_row_norms_max: 0.521019400834

test_y_row_norms_mean: 0.217996451477

test_y_row_norms_min: 0.0143365579701

valid_h2_kernel_norms_max: 1.55053091512

valid_h2_kernel_norms_mean: 0.624761569171

valid_h2_kernel_norms_min: 0.147479767746

valid_h3_kernel_norms_max: 1.83767522423

valid_h3_kernel_norms_mean: 1.5324467745

valid_h3_kernel_norms_min: 0.973684229374

valid_objective: 0.104301869617

valid_term_0: 0.0933291907639

valid_term_1_weight_decay: 0.0109726788529

valid_y_col_norms_max: 1.87857623558

valid_y_col_norms_mean: 1.85864577076

valid_y_col_norms_min: 1.82825909148

valid_y_max_max_class: 1.0

valid_y_mean_max_class: 0.994425815782

valid_y_min_max_class: 0.70111017439

valid_y_misclass: 0.017

valid_y_nll: 0.0933291907639

valid_y_row_norms_max: 0.521019400834

valid_y_row_norms_mean: 0.217996451477

valid_y_row_norms_min: 0.0143365579701