УДК 004.855.5

Мазурік Олексій Юрійович
студент
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Мазурик Алексей Юрьевич
студент
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Mazurik Oleskii
Student of the
National Technical University of Ukraine
«Igor Sikorsky Kyiv Polytechnic Institute»

ПОБУДОВА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ВІДЕО ЗА ДОПОМОГОЮ КЛАСУ МОДЕЛЕЙ WORD2VEC

ПОСТРОЕНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ С ПОМОЩЬЮ КЛАССА МОДЕЛЕЙ WORD2VEC

BUILDING A RECOMMENDER SYSTEM USING WORD2VEC MODELS

Анотація. *Досліджено алгоритм побудови моделей класу Word2Vec. Побудовано рекомендаційну систему на основі моделі SkipGram класу Word2Vec та досліджені результати її роботи.*
Ключові слова: *рекомендаційні системи, прогнозування, нейронні мережі, word2vec.*

Аннотация. *Исследован алгоритм построения моделей класса Word2Vec. Построено рекомендательную систему на основе модели SkipGram класса Word2Vec и исследованы результаты ее работы.*
Ключевые слова: *рекомендационные системы, прогнозирование, нейронные сети, word2vec.*

Summary. *The algorithm for building Word2Vec models was investigated. A recommender system based on the SkipGram model of Word2Vec class was constructed and the results of its work were investigated.*
Key words: *recommendation systems, forecasting, neural networks, word2vec.*

Постановка задачі. Рекомендаційні системи з'явилися на сучасному ринку IT як механізм для заміни статичному списку рекомендацій при пошуку або покупках на веб-сайтах. Ці системи формують рейтинговий перелік об'єктів (товарів, фільмів, музичних композицій) на основі різних критеріїв: релевантність, популярність, історія оцінок тощо.

Але, незважаючи на те, що рекомендаційні системи досить недавно з'явилися на ринку, вже існує безліч способів їх покращити. З розвитком досліджень у сфері Big Data та Machine Learning, все більше компаній починає використовувати нейронні мережі для покращення якості рекомендацій. Використовуються всі дані, які користувач залишає на сайті — історія переглядів, кліків, оцінок тощо. Зараз набуває популярність алгоритм побудови моделі Word2Vec (алгоритм пошуку семантичних зв'язків між словами). Його починають використовувати для пошуку семантичних зв'язків між різними типами медіа-даних. Саме експеримент з дослідження можливості використання цього алгоритму для медіа-даних буде основним предметом дослідження даної роботи [5].

Поняття Word Embeddings. Основним інструментом сучасної обробки текстів наразі є розподілені представлення слів (distributed word representations, вони ж word embeddings), графічне інтуїтивне представлення яких можна побачити на рисунку 1. В цих представленнях кожному слову ставиться

у відповідність вектор із дійсних чисел, елемент евклідового простору R^d для якогось d (зазвичай кілька сотень). Ці вектори далі служать входами для наступних моделей, а базове припущення полягає в тому, що геометричні відношення в просторі R^d будуть відповідати семантичним відношенням між словами, наприклад, найближчі сусіди слова в цьому просторі виявляються його синонімами або іншими тісно пов'язаними словами і т.д. [2]

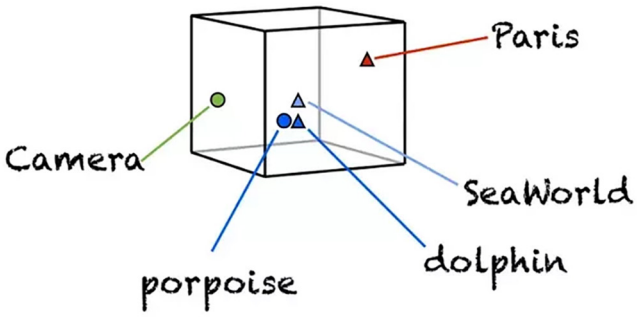

Рис. 1. Графічне представлення слів у 3-вимірному просторі [1]

Ідея Word2Vec. Основне завдання даного алгоритму полягає в побудові нейронної мережі, яка буде за заданим словом у середині речення (вхідне слово) шукати слова поряд та обирати одне випадковим чином. Мережа буде обчислювати ймовірність кожного слова зі словника стати цим «близьким» словом до того, яке ми обрали. Коли мова йде про «близькість», то мається на увазі параметр алгоритму «розмір вікна». Типовий розмір вікна може бути 2, який означає, що 2 слова до та 2 слова після даного будуть «близькими» до даного [3].

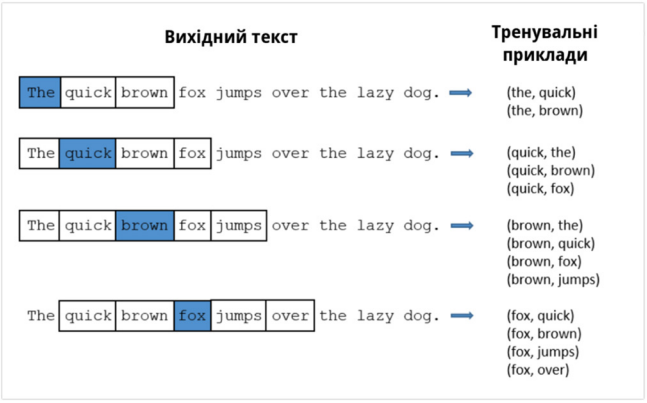

Рис. 2. Побудова тренувальних прикладів для мережі [3]

Нейронна мережа буде вивчати статистику кількості зустрічальності кожної з пар. Отже, для прикладу, нейронна мережа ймовірно буде мати набагато більше прикладів типу «United» — «States», ніж «United» — «Roads». Тож, після тренування якщо на вхід подано слово «United», то на виході ймовірності для слів «States» та «America» будуть у рази вищими, ніж для слова «Roads».

Архітектура нейронної мережі для Word2Vec. По-перше, неможливо подати слово на вхід до нейронної мережі як String. Необхідно знайти спосіб репрезентації його у якийсь тип ідентифікатору. Щоб цього досягти необхідно спочатку побудувати словник з наших тренувальних документів. Нехай ми вже маємо словник із 10000 унікальних слів [3].

Далі ми закодуємо кожне слово (наприклад «trees») як one-hot вектор за допомогою алгоритму OHE (One-Hot Encoding). Цей вектор буде містити 10000 координат (по одній координаті для кожного слова зі словника). На позиції, що відповідає за слово «trees» буде стояти 1, на всіх інших — 0.

Виходом нейронної мережі буде звичайний вектор з 10000 координатами, в якому будуть міститися для кожного слова зі словника ймовірності того, що випадково взяте слово буде знаходитись поряд з даним.

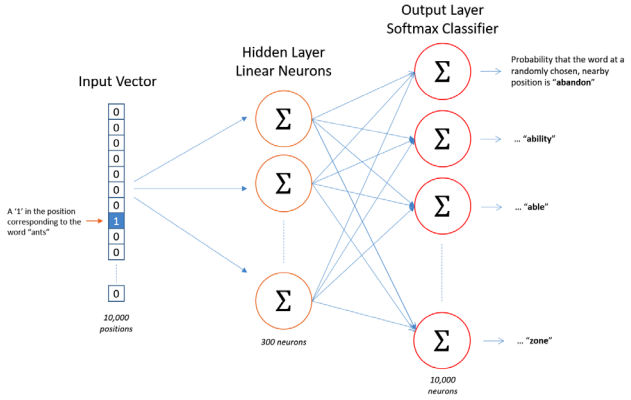

Рис. 3. Архітектура нейронної мережі моделі Word2Vec [3]

У прихованих шарах немає функції активізації, але вихідні нейрони використовують softmax. Коли відбувається тренування мережі на парах слів, вхідним словом буде one-hot вектор, що презентує це слово, а вихідним очікується також one-hot вектор, що представляє вихідне слово. Але при прогоні слова через мережу насправді ми отримуємо вихідний вектор, який є розподілом ймовірностей (набором чисел з плаваючою комою) [3].

Підсумки моделі Word2Vec. Отже, якщо два різних слова мають дуже схожі «контексти» (слова, що дуже ймовірно з'являться коло них), тоді наша модель має на виході дуже схожі результати для цих двох слів. І одним зі способів нашою моделлю цього досягти — це видати на виході подібні векторні представлення слів. Тож, якщо два слова мають схожий контекст, то наша мережа намагатиметься видати схожі векторні представлення слів для них.

Побудова рекомендаційної системи. Надана теоретична інформація Word2Vec та архітектура типової нейронної мережі для моделі Skipgram підводить до думки, що ID відео можна використовувати в якості слів, а сессії користувачів як речення або

документи. Тому, теоретично, ця ідея має спрацювати схожим чином.

Підготовка даних. Для побудови нейронної мережі рекомендацій необхідно зібрати гарний датасет з сесіями користувачів, який можна буде використовувати для навчання нейронної мережі. Після проведення роботи з пошуку такого набору даних було вирішено використовувати набір даних з сервісу Kaggle.

Знайдений набір містить в собі анонімізовані активні сесії переходів користувачів за різними відео-файлами. Виглядає він як показано на рисунку 4.

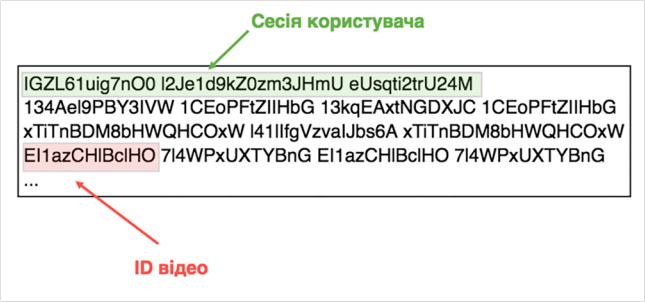

Рис. 4. Структура даних підготовленого датасету

Побудова SkipGram моделі Videoid2Vec за допомогою PyTorch. Пропустивши блок з імпортом бібліотек і даних звичайна модель для побудови embeddings для рекомендації схожих відео матиме наступні параметри:

- ***emb_size:*** Розмір векторного представлення.
- ***emb_dimention:*** Розмірність простору embedding, зазвичай — 32.
- ***u_embedding:*** Векторне представлення центрального відео.
- ***v_embedding:*** Векторне представлення для сусідніх відео у сесії.

Ініціалізація моделі буде застосована для двох шарів нейронної мережі, що лежить в основі рекомендаційної системи. На цьому етапі будуть ініціалізовані ваги нейронів на прихованому шарі та на вихідному. Векторне представлення (***u_embedding***) центрального відео — це рівномірний розподіл в проміжку $[-0.5/em\_size, 0.5/emb\_size]$, де вектори сусідніх відео (***v_embedding***) будуть заповнені нулями [6].

Тренування моделі SkipGram. Процес тренування полягає в поділу вхідних даних на пари відео для тренування моделі. Після формування таких пар формуються набори (batches) відео для створення кількох епохів тренування, оскільки неможливо одночасно обробити велику кількість даних. Код тренування представлений на рисунку.

Звичайно, для запуску даного коду та ефективній роботі необхідно використовувати ресурси GPU, тому недивно, що для їх задіяння було використано бібліотеку CUDA.

Побудова веб-додатку рекомендаційної системи. Отже, після опрацювання всіх вхідних даних ми побудували векторне представлення у 32-мірному просторі для кожного відео. Такий словник вже можна використовувати для побудови рекомендаційної системи. Звичайно, що її структура буде надзвичайно простою. На вхід буде подаватися відео з корпусу відео, а на виході ми матимемо 50 схожих відео. Схожі відео будуть обиратися з корпусу відео шляхом пошуку найближчого — існує безліч інструментів пакету PyTorch для цього.

Простий веб-додаток для наочної демонстрації ефективності експерименту було побудовано за допомогою фреймворку Flask (back-end) та React/Redux (front-end). Результат побудови додатку можна побачити на рисунку 7.

Результати роботи рекомендаційної системи. Після побудови додатку було проведено тестування системи. 90 % корпусу відео було надано для тренувальних цілей, а 10 % — для тестування. Це звичайний крок при побудові експериментів глибокого навчання. Графік на рисунку 3.15 демонструє результати тестування, а саме на якій позиції виявилися рекомендовані відео відносно тестового прикладу. Тобто, якщо позиція відео в рекомендації збігається

```
def forward(self, pos_u, pos_v, neg_v):
    """Forward process.
    As pytorch designed, all variables must be batch format, so all input of this method is a list of gif id.
    Args:
        pos_u: list of center gif id indexes for positive gif pairs.
        pos_v: list of neibor gif id indexes positive gif pairs.
        neg_u: list of center gif id indexes negative gif pairs.
        neg_v: list of neibor gif id indexes negative gif pairs.
    Returns:
        Loss of this process, a pytorch variable.
    """
    emb_u = self.u_embeddings(pos_u)
    emb_v = self.v_embeddings(pos_v)
    score = torch.mul(emb_u, emb_v).squeeze()
    score = torch.sum(score, dim=1)
    score = F.logsigmoid(score)
    neg_emb_v = self.v_embeddings(neg_v)
    neg_score = torch.bmm(neg_emb_v, emb_u.unsqueeze(2)).squeeze()
    neg_score = F.logsigmoid(-1 * neg_score)
    return -1 * (torch.sum(score) + torch.sum(neg_score))
```

Рис. 5. Код прямого ходу нейронної мережі моделі SkipGram

```
def train(self):
    """Multiple training.
    Returns:
        None.
    """
    pair_count = self._data.evaluate_pair_count(self._window_size)
    batch_count = self._iteration * pair_count / self._batch_size
    process_bar = tqdm(range(int(batch_count)), smoothing=1.0)

    for i in process_bar:
        pos_pairs = self._data.get_batch_pairs(self._batch_size, self._window_size)
        neg_v = self._data.get_neg_v_neg_sampling(pos_pairs, 5)
        pos_u = [pair[0] for pair in pos_pairs]
        pos_v = [pair[1] for pair in pos_pairs]

        pos_u = Variable(torch.LongTensor(pos_u))
        pos_v = Variable(torch.LongTensor(pos_v))
        neg_v = Variable(torch.LongTensor(neg_v))
        if self._use_cuda:
            pos_u = pos_u.cuda()
            pos_v = pos_v.cuda()
            neg_v = neg_v.cuda()

        self._optimizer.zero_grad()
        loss = self._skip_gram_model.forward(pos_u, pos_v, neg_v)
        loss.backward()
        self._optimizer.step()

        process_bar.set_description(f'Loss: {loss.data[0]:.4f}, lr: {self._optimizer.param_groups[0]["lr"]:.6f}')
        if i * self._batch_size % 100000 == 0:
            lr = self._initial_lr * (1.0 - 1.0 * i / batch_count)
            for param_group in self._optimizer.param_groups:
                param_group['lr'] = lr
    self._save_embedding()
```

Рис. 6. Код тренування моделі SkipGram (SkipGramTrainer)

з тестовою, то конкретна колонка, що відповідає за дану позицію, буде збільшена на 1. Якщо позиція >= 50 — останню колонку буде збільшено на 1.

Як можна побачити з даного графіку (рис. 8) ми отримали дуже гарний результат, оскільки дуже багато даних вдалося рекомендувати правильним чином. Звичайно, те, що остання колонка виявилася досить великою не є дуже добрим знаком. Але це лише є знаком того, що ми просто не вгадали позицію відео.

Подальший розвиток дослідження рекомендаційних систем. Звичайно, як можна побачити, дана модель вже поводить себе належним чином — рекомендації є релевантними. Дану тему можна розвивати нескінченно, покращуючи якість рекомендацій. Для цього, по-перше, можна збільшити корпус відео та краще розпаралелити навчання. По-друге, можна використовувати більше даних, оскільки окрім ID відео можна використовувати як мета-інформацію, так і сам контент відео. Тоді задача з NLP алгоритмів буде ускладнена Computer Vision складовою, що, звичайно, збільшує час розробки, але, в теорії, може покращити якість рекомендацій [4].

Висновки. Було побудовано рекомендаційну систему для великого корпусу відео на основі SkipGram моделі. Після побудови моделі було створено рекомендаційну систему, вбудовану у веб-додаток, створений за допомогою Flask (back-end API) та

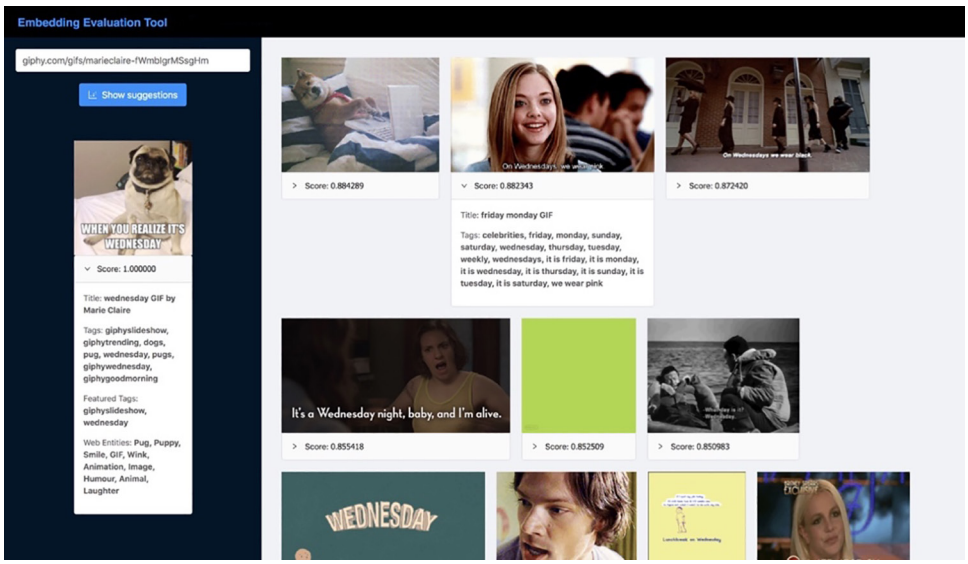

Рис. 7. Приклад веб-додатку рекомендацій коротких відео

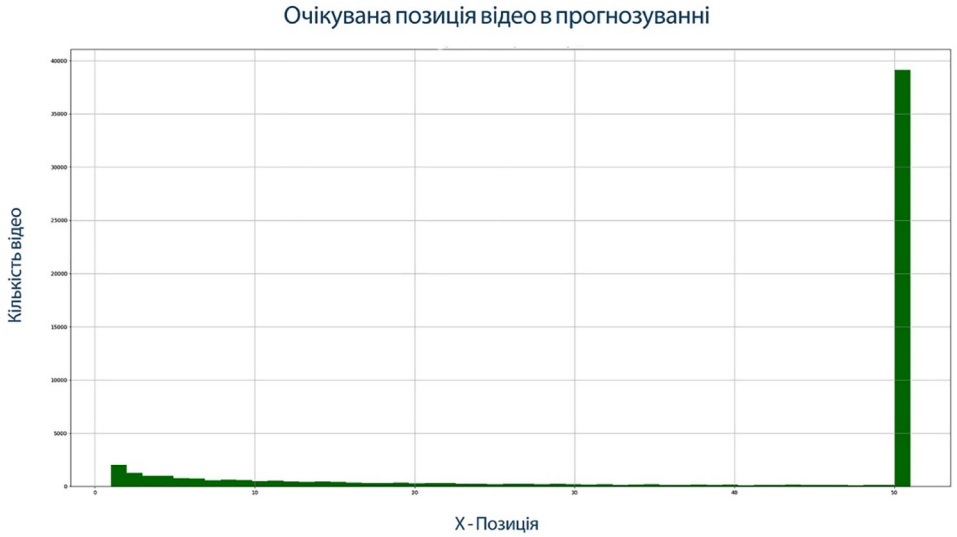

Рис. 8. Графік очікуваної позиції відео в рекомендаціях

React/Redux (UI front-end) для наочної демонстрації результатів дослідження. З результатів роботи рекомендаційної системи було зроблено висновок, що система поводить себе коректно, рекомендації — релевантні. Тому можна використовувати поточне рішення для різного роду медіа-платформ для збільшення конверсії тощо.

Література

1. Hu S. Video2Vec: Learning semantic spatio-temporal embeddings for video representation / S. Hu, Y. Li, B. Li. // International Conference on Pattern Recognition (ICPR). — 2016. — № 23. — С. 811–816.

2. Распределенные представления слов: word2vec. / С. Николенко, А. Кадурин, Е. Архангельская // Распределенные представления слов: word2vec // Глубокое обучение. Погружение в мир нейронных сетей. — 2018. — Санкт-Петербург. — С. 285–305.

3. Word2Vec Tutorial — The Skip-Gram Model [Електронний ресурс]. — Режим доступу до ресурсу: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

4. Word2Vec Tutorial Part 2 — Negative Sampling [Електронний ресурс]. — Режим доступу до ресурсу: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

5. Listing Embeddings for Similar Listing Recommendations and Real-time Personalization in Search Ranking [Електронний ресурс]. — Режим доступу до ресурсу: https://medium.com/airbnb-engineering/listing-embeddings-for-similar-listing-recommendations-and-real-time-personalization-in-search-601172f7603e

6. PyTorch. Documentation [Електронний ресурс]. — Режим доступу до ресурсу: https://pytorch.org/