Pontificia Universidad Católica de Chile

Escuela de Ingeniería

Departamento de Computación

Sistemas Recomendadores IIC-3633

Avance

Francisco Pérez Páez

Francisco Rencoret Domínguez

28 de Octubre, 2018

Problemas Encontrados en Entrega Pasada

Luego de la entrega de la propuesta anterior y de recibir el feedback del profesor y ayudantes, se nos sugirió modificar el *dataset* utilizado. Las recomendaciones de hoteles predecían un cluster de hoteles en vez de un hotel en específico, por lo que el profesor consideraba que no era suficiente. El hecho de que los hoteles fuesen agrupados según un criterio externo y no se pudiese saber en especifico cual era el que se predecía, empobrecia la recomendación.

Luego de esta noticia empezamos a buscar nuevos datasets para reemplazar el que habíamos propuesto, pero no fuimos exitosos.

Finalmente decidimos ampliar la frontera de búsqueda y terminamos encontrando un nuevo tipo de modelo llamado Modelos de *Self-Attention* que habían tenido resultados de estado del arte en traducción de textos. Encontramos un paper muy reciente (20 de Agosto de 2018) que presentan SASRec, una aplicación de este tipo de modelos en el área de recomendación. Su ventaja principal está en obtener mejores resultados tanto en *datasets* densos como *sparse* con un rendimiento 9,6% mejor en NDCG y con tiempos de entrenamiento de alrededor de un orden de magnitud menor a los métodos del estado del arte.

Decidimos transformar nuestro proyecto en la implementación propia de este modelo. Lo probaremos con el *dataset* de MovieLens 1M y realizaremos comparaciones con otros baselines utilizando la librería OpenRec.

Nueva Propuesta

Contexto del Problema

Las películas son una fuente de diversión para muchas personas. Existen muchos tipos y variedades de películas y cada vez resulta más difícil para una persona encontrar una película en un mar de películas que siempre crece.

Este es un problema que afecta a muchas personas en todo el mundo. El problema afecta tanto a los servicios que ofrecen contenido como Netflix o Hulu que no saben cómo clasificarlo y ofrecerlo eficientemente, como a los consumidores de contenido que deben sufrir intentando escoger una película que sea de su gusto o el de su familia.

Problema y Justificación

El problema de recomendación de una película es crucial en el modelo de negocios de empresas como Netflix o Hulu, ya que se basan en recomendar películas y series a sus usuarios con el objetivo de que su experiencia al consumir el contenido sea lo más

placentero posible. Una buena recomendación hace que los usuarios siempre encuentren valor en el producto y manténganse usandolo.

Dada la gran variedad de películas que se ofrecen, la tarea de encontrar una pelicula o serie que sea de gusto del cliente cuando nunca la ha visto es cada vez más difícil. Es por esto que el problema de recomendación de una película a un cliente es uno muy importante de resolver. Además, dada la gran cantidad de usuarios que estas empresas tienen, la tarea de recomendar una película de manera personalizada y manual, resulta impracticable.

Objetivos

Hay tres principales objetivos que esta investigación pretende cumplir.

En primer lugar, generar un sistema recomendador que, basado en secuencias de consumo de películas por parte de un usuario, prediga el siguiente ítem que el usuario consumirá.

En segundo lugar, explorar conocimientos de Deep Learning del estado de arte para realizar implementar y experimentar con las tecnologías más novedosas.

En tercer lugar, se aspira a que la recomendación tenga un rendimiento que sea comparable o mejor con los resultados del métodos en el estado del arte para recomendación.

Solución propuesta

Los modelos de atención generalmente constan de tres componentes claves: *key, query* y *value*. La gracia es que dado un *query* (el objeto en cuestión) se calcula la atención con respecto a todo el *key* obteniendo así un coeficiente de atención (todos los coeficientes suman uno) para cada valor del *key*. Luego, se multiplican los coeficientes de los *key* con los *value* (generalmente el key y value son los mismos objetos) y se suman todos para obtener un *embedding* nuevo para ese *query*. Es decir, ahora ese *query* contiene información de todo el resto del dato.

El paper Self-Attentive Sequential Recommendation implementa un modelo de *Self-Attention* para recomendar. *Self-Attention* es una versión particular de los modelos de atención.

Ellos toman un largo fijo de secuencias de ítems y para cada ítem de esa secuencia ellos lo pasan por un *Embedding* (para poder representarlo con un vector descriptivo). Luego hacen atención de ese embedding (*query*) con todos los otros ítems de la secuencia (*key*) para así obtener los coeficientes de atención. Ahora, el nuevo *embedding* para ese ítem sería la suma ponderada de los los ítems por sus coeficientes. La gracias es que ahora ese

embedding del ítem no solo contiene información de él (ese ítem está dentro de los *key* a pesar de ser la *query*) sino que contiene información de todo el resto de la secuencia.
Notamos que un bloque de atención retorna un vector por cada ítem de la secuencia, pero esa atención es una atención de un nivel de profundidad (solo relaciones entre pares). Para poder captar relaciones más profundas, en el paper plantean usar varios de estos bloques para poder captar esas secuencias profundas (un *stack* de bloques). Después de todos los bloques de atención le insertan una *feed-forward network* para poder hacer la clasificación.

Con esto, tomando una secuencia de n elementos, podemos predecir el n+1 más probable. Dado que el output de la *feed-forward network* es una *softmax* sobre todos los posibles ítems, podemos ordenar ese resultado para obtener una recomendación más extensa (obtenemos una probabilidad para cada ítem).

El gran problema de los modelos de *self-attention* es que pierden la noción de secuencialidad, es decir, no consideran el factor temporal de los datos. Para solucionarlo, ellos le suman un *positional embedding* (que es un parámetro aprendible) a cada *embedding* inicial de la secuencia. Tenemos una intuición de que esto puede no ser suficiente, por lo que luego explicaremos experimentos a crear.

Descripción de experimentos

El primer desafío es implementar el modelo. No existe ni una implementación en internet del paper en PyTorch, por lo que la implementación es completamente nuestra. Cuando lo tengamos implementado planeamos entrenarlo sobre el dataset de MovieLens 1M para poder comparar los resultados (que deberían ser similares si es que lo implementamos bien) con los obtenidos en el paper.

Luego, pensamos en probar con *positional embeddings* fijos: el índice del ítem en la secuencia o con un positional embedding sinusoidal como lo hacen en el Universal Transformer [4].

Probaremos distintas cantidades de bloques de atención juntos, para poder entender la relación de costo de entrenamiento y rendimiento a medida que se hace más profunda la atención.

Para intentar de considerar la secuencialidad, planeamos poner una RNN encima del *stack* de bloques de atención, de tal manera que el modelo de atención sea un potenciador del embedding simplemente. Es decir, en vez de calcular un embedding corriente para los ítems y luego pasarle eso a una RNN corriente, queremos que ese embedding sea el output de los modelos de atención. Nuestra intuición nos dice que esto podría funcionar bien. Vamos a partir con una RNN corriente, y luego vamos a ir iterando sobre ella. Vamos a considerar distintas dimensiones para el estado oculto y profundidades de la RNN.

Si el modelo RNN llega a funcionar y todavía tenemos tiempo sobrante, vamos a implementar un modelo de atención sobre esa RNN para captar mejor la información de la secuencia. En ese caso, usaremos la atención típica de Bahdanau [5].

Para obtener *baselines* con los que comparar el modelo SASRec, realizaremos varios experimentos utilizando la librería OpenRec. Una ventaja de OpenRec es que una vez que tengamos el primer modelo funcionando, podremos realizar varios experimentos de muchos otros recomendadores con pocas modificaciones[1]. Pensamos comenzar implementando un modelo Collaborative Deep Learning (CDL, por sus siglas en inglés) y luego ver que resultados nos da en el mismo dataset, pero con los siguientes modelos:

1. Positive Matrix Factorization (PMF)
2. General Matrix Factorization (GMF)
3. Bayesian Personalized Ranking (BPR)

Análisis Exploratorio de los datos

El dataset que utilizaremos es el de MovieLens[2]. Este dataset contiene identificadores de usuarios, identificadores de películas, un rating en una escala 1 a 5 y un *timestamp*. La versión que utilizaremos para hacer pruebas iniciales es la de 100.000 ratings. Las principales características del dataset son:

- 100.836 Ratings
- 9742 Películas
- 610 Usuarios
- Más de 160 ratings por usuario en promedio.

Una vez que se obtengan resultados en este dataset, pretendemos utilizar el dataset más reciente de 27 millones de ratings.

Para utilizar el modelo propuesto, se debe hacer un pre procesamiento y asumir que en vez de predecir el rating, lo que se busca en realidad es buscar el siguiente ítem. Entonces, dado un usuario, se utiliza el timestamp de los ratings para ordenar la secuencia de películas vistas y se entrena con esa secuencia. La anterior se realiza para todos los usuarios. Consideramos que el usuario consumió un ítem si es que le puso un rating.

Se define una constante n que es el largo de una secuencia a aprender. El paper no explicitan exactamente el caso para los usuario que han visto más de n ítems. Si bien hay que hacer un *sliding window* que vaya captando secuencias de largo n (dentro de todos los ratings del usuario), no se menciona con qué *stride* se usa por lo que nosotros vamos a usar con 2 o 5 dependiendo si estamos con el dataset reducido para prueba o con el para entrenar el modelo final (varía mucho la cantidad de datos).

Por el otro lado, si un usuario tiene pocos ratings, se rellena la secuencia por la izquierda con un *padding* de cero.

---

[1] https://openrec.readthedocs.io/en/latest/index.html Revisado el 28 de Octubre
[2] https://grouplens.org/datasets/movielens/ Revisado el 25 de Octubre

Notamos que dado que estamos implementando modelos de atención donde no se le entrega el usuario al modelo, este *data augmentation* ayuda al modelo pero no beneficia a los usuarios que han visto más ítems.

Siguiendo la intuición del paper, el ítem más reciente es usado para test, el segundo más reciente para val y todo el resto para training. Es decir, si un usuario tiene 55 ratings, de ese usuario podemos obtener dos datos para el *train* ([0: 50], [1: 51]), uno para el *val* ([2:52]) y uno para el *test* ([3: 53]). Usando un *sliding window* con *stride=5,* obtenemos 15225 datos para el *train*, 610 para el *val* y 610 para el *test*. Dado que el con un stride de 5 no obtenemos tantos datos, encontramos que está bien tener solamente 610 para *val* y *test*, pero si redujeramos el *stride* (o cambiamos a la versión completa del dataset) aumentando la cantidad de datos podría hacer sentido excluir más usuarios solamente para el *val* y *test* para compensar un poco. Es decir, si el *train* es chico, esta bien tener 5% *val* y 5% *test*, pero si el *test* crece mucho sería bueno aumentarlo a 10% y 10% respectivamente.

El archivo MovieLens.ipynb contiene información del dataset y un breve preprocesamiento visual.

Estado de Avance

Actualmente tenemos implementado el preprocesamiento de la data junto con el *data loader*. El archivo preprocess_dataset.py prepara la data para poder ser usada fácilmente por el *data loader*. Lo que hace es agrupar los ratings por usuario y almacenar unos json que contienen los ratings (ordenados por timestamp) de cada usuario. El *data loader* provee una clase que luego se la pasamos a torch.utils.data.DataLoader para poder cargar los datos y usarlos para el entrenamiento.

En el archivo model.py escribimos el código del modelo donde separamos clases distintas para las distintas partes del modelo. Embedding Layer obtiene el embedding inicial de los objetos (con el positional embedding), el AttentionBlock presenta un bloque de atención y luego el SASRec presenta el modelo completo. El modelo esta funcionando para un bloque de atención y estamos terminando de implementar el modelo completo.

El archivo main.py carga el *data loader* y el modelo y lo entrena. Estamos trabajando todavía en la estructura del entrenamiento, pero luego le agregaremos funciones para que grafique los resultados del entrenamiento.

Resultados Preliminares

Dado que cambiamos de tema hace solo unos días, no hemos podido obtener resultados significativos del entrenamiento. Estamos justo terminando la implementación y empezando a entrenar dentro de poco.

Problemas Encontrados

No tenemos mucha experiencia en PyTorch pero hicimos unos tutoriales [6] que nos han ayudado a entender cómo funciona. Estamos aprendiendo sobre la marcha.

El tema del hardware podría ser un tema, pero tenemos GPU's a nuestra disposición. Con las pruebas que hemos hecho de entrenamiento, las épocas no deberían tomar tanto tiempo y además el modelo de Self-Attention es paralelizable completamente por lo que podríamos acelerar más aún el entrenamiento.

Una duda que sería muy útil responder es si ¿Es justo comparar nuestro modelo que no se basa en el valor del rating para estimar el siguiente ítem con otros modelos que si utilizan esa información? ya que de alguna manera un modelo tendría "ventaja" sobre el otro.

Plan de Avance

En primer lugar queremos terminar de implementar bien el código de SASRec e intentar lograr los resultados del paper.

En segundo lugar queremos generar los baselines contra los cuales comparar nuestros resultados. Para esto utilizaremos la librería OpenRec con la que generaremos los modelos y métricas de rendimiento para comparar. Con esto, vamos a tener harto margen de comparación y saber bien cuando efectivamente mejoremos.

En tercer lugar, vamos a probar los experimentos mostrados para ver cómo podríamos mejorar el rendimiento de SASRec. Vamos a probar cosas y entrenar con el dataset de 1M, cuyo entrenamiento debería ser rápido, y cuando ya obtengamos la configuración óptima que hayamos encontrado, planeamos entrenar el modelo con el MovieLens 27M (dataset completo). Dado que no hemos terminado la implementación completa para paralelización en GPU, no podemos estimar cuánto demoraría ese entrenamiento.

Por último, debemos agrupar la información y escribir formalmente el *paper* y preparar la presentación final.

Bibliografía

1. Documentacion PyTorch: https://pytorch.org/
2. Dataset MovieLens: https://grouplens.org/datasets/movielens/
3. Link al paper en que nos basamos: https://arxiv.org/pdf/1808.09781.pdf
4. Universal Transformer: https://arxiv.org/pdf/1807.03819.pdf
5. Neural Machine Translation: https://arXiv:1409.0473v7
6. PyTorch tutorials: https://pytorch.org/tutorials/