Dwell-Time aplicado a recomendaciones basadas en sesiones utilizando redes neuronales recurrentes

Proyecto Final - IIC3633 - Sistemas Recomendadores

Patricio Cerda
Pontificia Universidad Católica de Chile
Santiago, Chile
pcerdam@uc.cl

Bastián Mavrakis
Pontificia Universidad Católica de Chile
Santiago, Chile
bamavrakis@uc.cl

1 ABSTRACT

In this paper, we present a Keras implementation for a session-based RNN recommendation system in *e-commerce* that replicates the original work's result. The framework permits rapid prototyping and extending for future work. Additionally, we implement Dwell-Time as a preprocessing technique with a mild improvement over the baseline RNN model.

2 INTRODUCCIÓN

En este trabajo, se introduce una nueva implementación de un modelo basado en redes neuronales recurrentes para generar recomendaciones en contexto de *e-commerce* sobre sesiones de usuarios anónimos. Para esto se trabaja con un framework de alto nivel, Keras, lo que permite código de fácil mantención, extensión y lectura. Por otro lado, se realizan trabajos exploratorios que algunas líneas de investigación planteadas en la literatura sugieren para extender el modelo base, con diversos grados de completitud. En particular, se implementa satisfactoriamente "Dwell Time" como feature implícita que permite dar con un set de entrenamiento más numeroso, obteniendo resultados superiores al obtenido por el modelo recurrente base.

3 TRABAJO RELACIONADO

La primera publicación en la que se utiliza una red neuronal recurrente para realizar recomendaciones basadas en sesiones es [1]. El modelo propuesto recibe de entrada el *one-hot encoding* del item actual, que pasará por una capa de Gated Recurrent Units ([5]), para finalmente llegar a una capa densa con activación *softmax*, cuyo output es un vector que señala el siguiente elemento más probable en la sesión de un usuario. Dada la naturaleza del dominio en el cual se trabaja, en donde el largo de sesiones puede ser muy variable, se utiliza un método de generación de batches en paralelo, que considera de forma paralela elementos de múltiples sesiones en un mismo batch al momento de entrenar la red.

El trabajo anterior validó las RNN's como herramienta para predecir el comportamiento de usuarios en sus sesiones de *e-commerce*, generando interés en el área, y dando lugar a extensiones del modelo en trabajos posteriores.

En [2] se propone realizar el entrenamiento sobre batches secuenciales (compensando la pérdida de eficiencia con un proceso de *data augmentation* mediante *zero-padding*), además de incorporar una capa de *embedding* para dar con un modelo más compacto y fácil de entrenar.

Otro trabajo interesante es [3], en donde se plantea aplicar un preprocesamiento al conjunto de datos de entrenamiento para obtener información de importancia relativa entre items a partir del tiempo que el usuario permaneció en la página de cada uno, concepto conocido como *Dwell-Time*. El modelo es idéntico al utilizado en [1].

Los autores de [1] presentan en un trabajo posterior ([7]) funciones de pérdida para *ranking* (TOP1-max, BPR-max) que presentan un mejor desempeño respecto a las originales (TOP1, BPR).

Finalmente, [4] plantea un dominio nuevo sobre el cual probar sistemas recomendadores de este tipo: MovieLens 20M. Este *dataset* es muy utilizado en el área de recomendaciones, y lleva registro del ranking que usuarios asignan a películas que han visto. El trabajo hace el supuesto de que un usuario equivale a una sesión, lo que claramente no es del todo correcto: muchos usuarios califican películas a lo largo de años, y no necesariamente consumen los items a una frecuencia constante. Más aún, el hecho de que haya una pausa en esta frecuencia no implica que la última película vista haya sido especialmente de su agrado. Por lo mismo, utilizar Dwell-Time aquí no es sensato, pero de todos modos el dataset sirve para comparar el rendimiento de modelos similares, hasta cierto punto.

4 METODOLOGÍA

4.1 Objetivos

Se definen dos objetivos. Primero, realizar una re-implementación del modelo original propuesto por Hidasi [1] en un framework activamente mantenido como Keras, con TensorFlow como *backend*. Esto a diferencia del modelo original, implementado en el framework Theano. Nos aprovechamos de que Keras sea un wrapper sobre el *backend* de elección para obtener un mayor grado de abstracción y simpleza. Segundo, se implementa Dwell-Time según lo propuesto en [3] como técnica de preprocesamiento aplicada en el *dataset* de entrenamiento. Finalmente, se compara el rendimiento de nuestra implementación (con y sin *Dwell-Time*) respecto a ItemKNN como *baseline*, y a los modelos presentados en [1] y [3].

4.2 Dwell Time

Esta técnica[1] consiste en otorgar mayor importancia a los items que el cliente pasó mayor parte de su tiempo visitando. Para esto se realiza el supuesto de que mientras más tiempo se encuentre un usuario sobre un determinado item, más interesado estará ese usuario en ese elemento. Se busca incorporar esta información implícita al dataset original (pues se dispone de las *timestamps* del inicio de cada visita) mediante un proceso artificial de *boosting* sobre las sesiones. Para esto, se define el tiempo de permanencia de

[1] En español, un análogo podría ser *"Permanencia temporal"*

un elemento i en una determinada secuencia como dt_i. A su vez, se especifica un threshold t por sobre el cual se agregará artificialmente a la sesión s'_i el i-ésimo elemento ($\frac{dt_i}{t}$) veces[2] adicionales, como se muestra a continuación:

$$s'_i = \{x_{i1} \cdot (\frac{dt_{i1}}{t} + 1), ..., x_{in} \cdot (\frac{dt_{in}}{t} + 1)\}$$

De esta forma, al momento de entrenar se da mayor importancia relativa a los elementos en los que el usuario permaneció más tiempo.

Se debe notar que nuestro proceso de *data augmentation* utilizando esta técnica otorgó resultados idénticos a los expuestos en [3] (figura 1), validando nuestro procedimiento.

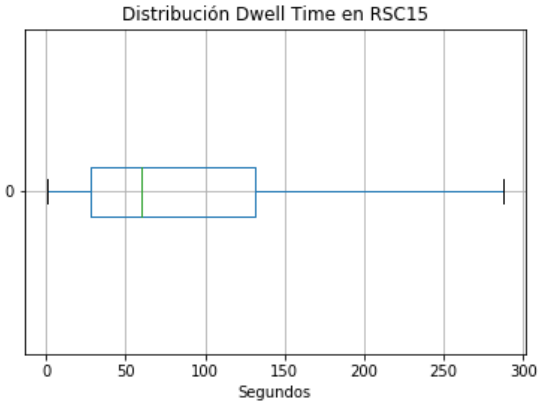

Figure 1: Resultados preprocesamiento *dataset* RSC15

4.3 Sistema recomendador

Como se explica en [1], la diferencia más grande entre redes neuronales recurrentes y redes neuronales tradicionales es la presencia de un estado oculto h_t, que confiere a una celda recurrente el potencial de utilizar información tanto de elementos anteriores en la secuencia como del input temporal actual. Las redes neuronales recurrentes tradicionales actualizan el valor de sus estados ocultos mediante la siguiente función:

$$h_t = g(W \cdot x_t + U \cdot h_{t-1})$$

en donde W y U corresponden a las matrices de pesos del modelo, x_t es el input actual, h_{t-1} es el valor del estado oculto en el tiempo anterior, y g es una función no lineal.

El modelo tradicional de RNN's sufre un problema importante: el aporte de elementos temporalmente muy lejanos con respecto al actual deja de ser significante, lo que lleva a una incapacidad de modelar dependencias temporales de gran extensión. Una de las arquitecturas de celdas recurrentes diseñada para solucionar lo anterior es la Gated Recurrent Unit (o GRU, [5]), que ofrece dos innovaciones: ahora cada celda posee una *update gate* y una *reset gate*.

La update gate z_t permite determinar qué parte de la información pasada será utilizada para la iteración actual. Para actualizar este valor se considera la información del input además del estado oculto en el tiempo anterior:

$$z_t = \sigma(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1})$$

Por otro lado, la reset gate r_t determina qué trozos olvidar de la información pasada.

$$r_t = \sigma(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1})$$

Se define $\hat{h}_t$ como un elemento de memoria que almacena información relevante del pasado que no debe ser olvidada. Esto se logra mediante la utilización de la reset gate. Su función de actualización es la siguiente (con $\odot$ el producto Hadamard, elemento a elemento):

$$\hat{h}_t = tanh(W \cdot x_t + r_t \odot U \cdot h_{t-1})$$

Por último, para calcular el valor del estado oculto para un determinado tiempo t, se considerará tanto la información pasada que se decidió mantener así como la información que aporta el input actual:

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \hat{h}_t$$

4.3.1 Implementación en Keras. Para la implementación del modelo original de [1], se consideró el diagrama completo propuesto por Hidasi, expuesto en la figura 2.

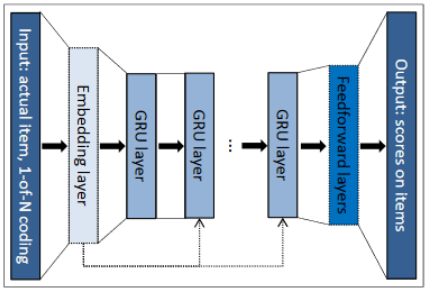

Figure 2: Modelo original, *Hidasi et al.* ([1])

Sin embargo, se probó el uso de una capa de embedding además de un *stack* de capas GRU, obteniendo resultados pobres (al igual que en [1]), por lo que el modelo final simplemente consta de los siguientes elementos:

(1) Capa de input: *one-hot encoding* para los elementos del input, de dimensionalidad n_{items}.
(2) Una capa de 100 Gated Recurrent Units.
(3) Capa de Dropout, con probabilidad 0.25.
(4) Capa densa con activación $softmax$.

Además, se implementó el *batcher* de sesiones en paralelo propuesto en [1], cuyo funcionamiento se ejemplifica en la figura 3.

El input de cada mini-batch se compone de elementos de distintas sesiones, y el output correspondiente (es decir, lo que queremos que la red aprenda a predecir) es el elemento siguiente de cada una de esas sesiones. Es importante destacar que cuando se acaba una sesión durante la generación de mini-batches, ésta debe ser reemplazada por una sesión nueva. En el ejemplo de la figura, cuando se acaba la sesión 2, ésta se reemplaza con los elementos correspondientes de la sesión 4 para el siguiente instante.

[2] División entera.

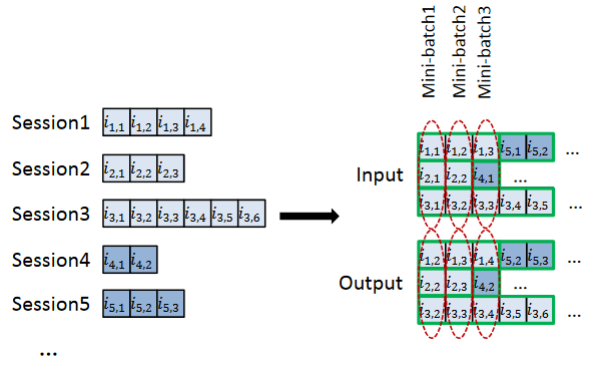

Figure 3: Batches en parelelo

Es imperativo *resetear* el estado oculto asociado a una determinada sesión cuando ella termina, ya que de otro modo se estaría asumiendo que existe dependencia entre sesiones.

4.3.2 Dwell Time. Para la implementación de Dwell Time, se utilizó el threshold propuesto en [3] de $t = 75$. El entrenamiento posterior prosigue de manera idéntica al modelo que trabaja con el set de datos original. Se ejemplifica el efecto de esta técnica en la figura 4, en donde el elemento $i_{2,1}$ es extendido temporalmente en dos unidades dado el tiempo que el usuario pasó en su página.

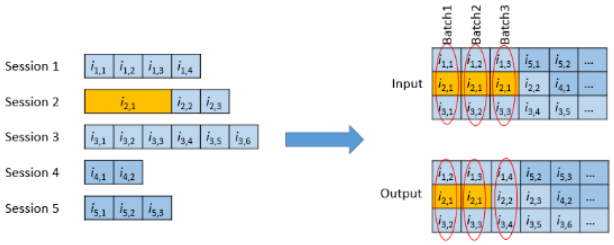

Figure 4: Batches en parelelo con Dwell Time

5 EXPERIMENTOS

5.1 Métricas de desempeño

Se utilizaron 2 métricas de desempeño para la evaluación de rendimiento de todos los experimentos realizados en el trabajo: Recall@20 y MRR@20. Esto, para poder comparar de manera directa a los resultados obtenidos en [1] y [3].

- Recall@20: corresponde a la proporción de casos en los que la label real está dentro de los 20 primeros elementos entregados por el modelo (en este caso, los 20 índices con mayor probabilidad en la capa densa).

$$Recall = \frac{\text{relevant items}}{\text{total items}}$$

- MRR@20 (Mean Reciprocal Rank): es el promedio de inversas del ranking de cada label real dentro del vector de 20 "mejores" items entregado por el modelo.

$$\frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

5.2 Modelo original re-implementado en Keras

Para validar nuestra re-implementación del modelo en [1], se siguió el *approach* presentado en [4]: probamos ambas redes sobre MovieLens 20M, considerando los detalles mencionados anteriormente, en particular:

- Se asumió que cada usuario correspondía a una sesión, en donde todas las películas vistas por éste a lo largo de su presencia en la plataforma de *ranking* fueron consideradas como los elementos de dicha sesión.
- No se utilizaron los *ratings* realizados por los usuarios para cada película. Sería interesante, a futuro, integrar esta información de alguna manera en el modelo.
- No sería sensato utilizar Dwell-Time sobre este dataset. Por su naturaleza, no aplica el supuesto de que a más tiempo en un elemento se tenga un mayor interés del usuario por él: si no se utiliza la plataforma en 6 meses, no significa que la última película vista haya sido especialmente interesante.

A continuación se muestra información relevante sobre el dataset:

Table 1: Descripción MovieLens 20M

Número de sesiones	89688
Número de ítems	17554
Largo promedio de sesiones	46.73
Largo máximo de sesiones	219

El dataset MovieLens 20M contiene información que fue obtenida entre Enero del 1995 hasta Abril del 2016. Para los experimentos realizados se dividió el dataset de la siguiente forma:

- Train set: Datos entre Enero del 1995 hasta Marzo del 2013.
- Validation set: Datos entre Abril del 2014 hasta Abril del 2015.
- Test set: Datos entre Abril del 2015 hasta Abril del 2016.

Se entrenaron el modelo original de [1] y nuestra re-implementación durante 9 épocas. Los resultados se presentan en la tabla 2.

Table 2: Resultados implementación de [1]

Modelo	**Recall@20**	**MRR@20**
Baseline: GRU4REC [1]	0.218	0.065
Nuestra implementación	0.243	0.077

Se puede observar que los resultados para ambos modelos son similares, de hecho, nuestra re-implementación ofrece un rendimiento levemente superior al modelo original de [1], con mejoras de un 11% en Recall@20 y un 18% en MRR@20. La hipótesis es que el optimizador utilizado en Keras es ligeramente distinto al utilizado en Theano, y parece funcionar mejor para el problema planteado.

5.3 Dwell Time

Dada la validación anterior, se implementa Dwell-Time de igual forma que en [3] como fase previa de pre-procesamiento de datos. Luego, se utiliza nuestra re- implementación como sistema recomendador.

En este caso, se utiliza un dataset sobre el que podamos hacer el supuesto necesario para la técnica. Para entender fácilmente el efecto de los cambios implementados, optamos por lo mismo que [1] y [3]: el *dataset* RecSys Challenge 2015 (abreviado como RSC15). Este dataset consiste en sesiones anónimas (representadas como una secuencia de items o productos visitados) de un sitio de e-commerce, YouChoose GmbH. Se tienen alrededor de 8.5 millones de sesiones, y un total de 37.500 items distintos. Cada click tiene asociado un timestamp, lo que permite un ordenamiento secuencial de los eventos a nivel de sesión. El dataset ya estaba particionado para entrenamiento, validación y prueba, en donde se aplica el pre-procesamiento únicamente al conjunto de entrenamiento.

Se utiliza como *baseline* el algoritmo Item-KNN debido a su simplicidad y alto rendimiento, relativo a otros métodos básicos analizados en [1]. Además, se contrastan los resultados con el modelo original de [1] sin Dwell Time como *"baseline* recurrente".

Se entrenó la implementación de Dwell Time durante 2 épocas[3], de donde se obtuvieron los siguientes resultados:

Table 3: Resultados implementación de [3]

Modelo	Recall@20	MRR@20
Baseline: Item-KNN	0.507	0.205
GRU4REC [1]	0.607	0.243
Nuestra implementación	0.647	0.275
Dwell Time [3]	0.853	0.636

La implementación de Dwell-Time funciona considerablemente mejor que Item-Knn, con una mejora del 28% en Recall@20 y de un 34% en MRR@20. Por otro lado, se muestra una leve mejora sobre el modelo original [1] sin Dwell Time, lo que sugiere que la información implícita que provee esta técnica sí está siendo útil para que el modelo aprenda de mejor manera. Sin embargo, [3] obtiene resultados considerablemente mejores, con una ganancia adicional de 32% en ambas métricas. No se sabe precisamente a qué se debe esta diferencia significante, pues el pre-procesamiento y modelos son idénticos. Se hipotetiza que entrenar el modelo durante apenas 2 épocas pueda ser la causa.

6 TRABAJO FUTURO

El desarrollo de este proyecto no resultó libre de complicaciones, pero los resultados obtenidos motivan a experimentar y prototipar algunas técnicas (para lo cual utilizar un framework simple ayudará bastante).

En primer lugar, lo que queda pendiente es terminar de limpiar y organizar el código para poder liberarlo a un repositorio público que pueda ser útil en futuras investigaciones en el área.

En segundo lugar, se debe entrenar nuestro modelo con Dwell-Time por más épocas. En [3] se entrena por 10 épocas. Hacer esto nos puede terminar de confirmar (o descartar) la falta de entrenamiento como culpable sobre la diferencia de rendimiento observada.

En tercer lugar, suena interesante explorar el diseño de funciones de pérdida que consideren aprender no sólo a predecir, sino que además de manera conjunta aprendan un buen embedding en paralelo. Lograr esto permite que un modelo con una capa de este tipo no requiera que los pesos sean entregados *a priori*, logrando un entrenamiento más corto y eficiente. En esta línea, [6] es un trabajo que aporta *insights* interesantes.

En cuarto lugar, se debe diagnosticar el problema que presenta nuestro batcher secuencial (que imita el funcionamiento del batcher que propone [2]). Lograr un buen funcionamiento resulta clave para poder experimentar con técnicas básicas de atención sobre el modelo, pues el batcher original que considera sesiones en paralelo imposibilita aplicar dichos mecanismos (o bien, el aplicarlos no sirve de nada, no es realmente atención). Estos mecanismos podrían resultar especialmente útiles sobre *datasets* con sesiones de gran extensión, como por ejemplo MovieLens 20M.

Finalmente, aplicar las sugerencias realizadas por los auditores en la presentacion de posters: incorporar el procesamiento de Dwell-Time al modelo en sí. Es decir, que sea "online" o en demanda. Esto escalaría bien en datasets de un órden de magnitud mayor. Por otro lado, sería posible dar espacio al modelo para que determine por su propia cuenta el valor óptimo para el threshold temporal. Además, se sugirió aplicar un descuento temporal r en el proceso de *boosting*.

7 CONCLUSIONES

En conclusión, este proyecto final cumplió de forma satisfactoria sus objetivos iniciales: se pudo implementar el trabajo presentado en clases [1], y replicar sus resultados, utilizando un framework que permite extender y prototipar sobre éste de manera rápida. Asimismo, se exploró el efecto de una técnica de pre-procesamiento en la calidad de las recomendaciones con éxito, pues se logró mejorar sobre el desempeño de [1], pero no se pudo replicar los resultados obtenidos en el trabajo que inicialmente propone la técnica, [3]. Finalmente, se detectan múltiples oportunidades para refinar el trabajo presentado, y lograr mejores resultados en un problema de recomendación que es bastante relevante en el contexto internacional.

8 REFERENCIAS

(1) Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based Recommendations with Recurrent Neural Networks. CoRR, abs/1511.06939.
(2) Tan, Y.K., Xu, X., & Liu, Y. (2016). Improved Recurrent Neural Networks for Session-based Recommendations. DLRS@RecSys.
(3) Bogina, Veronika and Tsvi Kuflik. "Incorporating Dwell Time in Session-Based Recommendations with Recurrent Neural Networks." RecTemp@RecSys (2017).
(4) Santolaya, D.S. (2017). Using recurrent neural networks to predict customer behavior from interaction data.
(5) Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. EMNLP.

[3]Principalmente por falta de tiempo y extensión: una época se ejecuta en 6 horas.

(6) Wang, J., Virtue, P., & Yu, S.X. (2017). Joint Embedding and Classification for SAR Target Recognition. CoRR, abs/1712.01511.
(7) Hidasi, B., & Karatzoglou, A. (2018). Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. CIKM.