Table Question Answering

Autor
Unai Salaberria Flaño
usalaberria002@ikasle.ehu.eus

Abstract

El entrenamiento sobre estructuras de datos tabulares es una tarea que todavía resulta un desafío para los modelos de lenguaje. Esto se debe a la escasez de este tipos de datos de buena calidad. En (Liu et al., 2021) se propone una nueva forma de afrontar este problema, consiguiendo un modelo capaz de ejecutar sentencias SQL con buen entendimiento de la tabla que analice. El objetivo de este artículo es analizar el trabajo de los autores del modelo TAPEX y probarlo de forma exhaustiva.

1. Introducción

El modelo sobre el que se basa esta práctica fue desarrollado por los autores del paper Tapex: Table Pre-Training via learning a Neural SQL Executor (Liu et al., 2021).

La tarea a la que se enfrenta este modelo es la de responder a preguntas con la información encontrada en una tabla, conocida como 'table question answering' en la literatura. Para ello, tendrá que ser capaz de entender de alguna forma la tabla, y no simplemente leerla y buscar datos.

El objetivo principal de los autores, y con lo que pretenden diferenciarse de otros trabajos sobre el tema, es lograr el preentrenamiento de tablas aprendiendo un ejecutor neuronal SQL sobre un corpus sintético. En otras palabras, pretenden conseguir un modelo que sea capaz de ejecutar queries de SQL para encontrar la información deseada.

2. Otros trabajos

Son dos los modelos a los que los autores hacen referencia a lo largo del trabajo, tanto para mostrar características suyas en las que se han basado como para identificar posibles mejoras en el desarrollo. Estos son TAPAS (Herzig et al., 2020) y TABERT (Yin et al., 2020).

El primero utiliza una aproximación muy parecida a TAPEX, ya que se puede decir que este último es su 'extensión'. TAPAS predice una respuesta tomando unas celdas de la tabla y ejecutando un operador de agregación sobre esta. La diferencia sobre TAPEX es que no ha sido preentrenado con el objetivo de imitar el comportamiento de un ejecutor SQL. Simplemente, aprende a generar respuestas en base a relaciones entre el texto y la tabla, y también entre las propias celdas y columnas de la tabla.

El segundo es algo un poco diferente. TABERT es un modelo basado en BERT el cual aprende sobre las representaciones contextuales de las declaraciones y el esquema de las tablas. Según especifican en (Yin et al., 2020), el modelo ha sido preentrenado sobre 26 millones de tablas y párrafos en inglés.

Pero en general, estos modelos se encuentran con los siguientes problemas:

Para empezar, es necesario un corpus muy grande y de mucha calidad para hacer el preentrenamiento de estos modelos, ya que su comportamiento muestra que cuantos más datos tengan para entrenar (siendo estos de una mínima calidad), mejores resultados obtendrán.

Pero esto es una tarea muy difícil, ya que el corpus se puede obtener de dos formas distintas: 1) tomando tablas de Internet y buscando en el contexto preguntas y respuestas 2) coger tablas y crear preguntas y respuestas de forma manual. Ambas llevan mucho tiempo, y en el primer caso, el corpus no suele ser de mucha calidad.

El otro problema al que se enfrentan estos dos modelos es que tratan las tablas como un formato de texto estructural, y esto requiere de muchos datos para el preentrenamiento, por lo que volvemos de nuevo al problema anterior.

Con esto en mente, el desarrollo del modelo TAPEX busca la forma de realizar el preentrenamiento sin necesidad de un corpus muy grande.

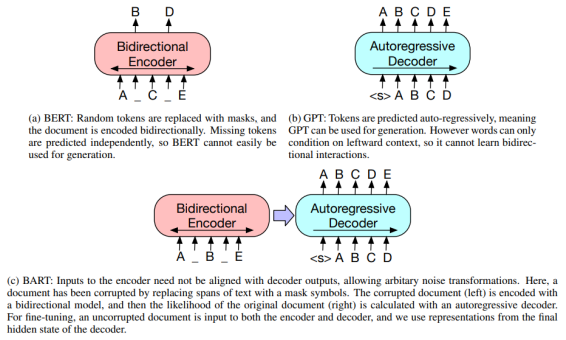
3. El Modelo

3.1. Modelo base

El modelo que se utiliza como base para crear TAPEX es BART (Bidirectional and Auto-Regressive Transformers).

El proceso de entrenamiento de BART es el siguiente: se corrompe texto con cierta probabilidad y luego se aprende a reconstruirlo.

La arquitectura que utiliza es una 'máquina neuronal' basada en un Transformer, y se puede ver como una generalización de BERT, que utiliza codificación bidireccional, y GPT, que utiliza codificación autorregresiva. En la figura 1 se puede ver esta generalización de la que hablo.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.

(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with a mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figura 1: Estructura del modelo BART

En el caso de BART el proceso de entrenamiento es el siguiente: el documento corrupto se pasa al codificador bidireccional, después, el decodificador autorregresivo calcula la probabilidad del documento original. Para hacer fine-tuning el documento corrupto se pasa a ambos, el codificador y el decodificador, y se utilizan las representaciones del último estado oculto del decodificador.

3.2. Cómo funciona TAPEX

Como ya he dicho antes, la estructura del modelo TAPEX está basada en TAPAS, por lo que la forma en la que se introducen los datos y se obtienen los outputs es similar.

Primero, vamos a ver cómo se introducen los datos necesarios para que el modelo haga la predicción. Para codificar la tabla se utilizan embeddings de posición de varios tipos, luego describiré cada uno de ellos. Después, la tabla se aplana convirtiéndola en una secuencia lineal de tokens. El modelo recibirá la pregunta y la tabla juntas como una secuencia de tokens, separadas por un token especial de separación.

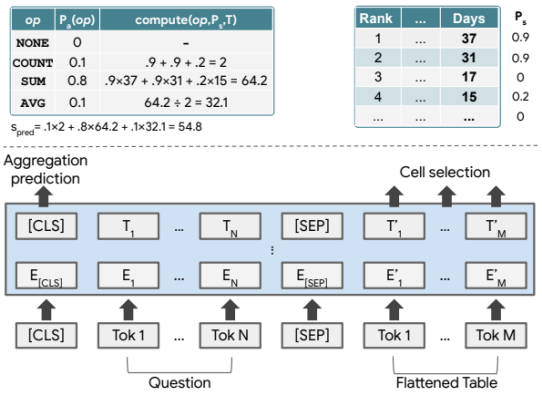


Figura 2: Funcionamiento del modelo TAPEX

Generación de outputs:

Para hacer la predicción, primero se hace la selección de las celdas donde se encuentra la información, después se calcula la probabilidad de cada celda y se realiza un argmax para seleccionar la más probable.

Tipos de embeddings de posición:

- ID de posición: Especifica el índice del token en la secuencia.
- ID de segmento: Será 0 si se trata de una pregunta y 1 si es una tabla.
- ID de fila/columna: Índice de la fila/columna en la que aparece el token.
- ID de ranking: Si los valores de la columna son numéricos, se hace un embedding basado en ranking.
- Respuesta anterior: Se marca si alguno de los tokens actuales fue respuesta en la pregunta anterior, con el objetivo de guardar cierta información que pueda tener correlación.

Selección de celdas:

Según pude leer en (Herzig et al., 2020), el proceso de selección de celdas (dibujado arriba a la derecha en la figura 2), se realiza de la siguiente manera: Las celdas son modeladas como variables independientes de Bernoulli, es decir, cada una de ellas tendrá una probabilidad asociada. El logit de cada celda es la media de los logits de los tokens en esa celda. Entonces, el output de la capa linear final representa la probabilidad de elegir una celda concreta. Al final, se tomará con un argmax la celda o celdas con mayor probabilidad.

Predicción con operador de agregación:

En algunos casos, no es suficiente con que el modelo seleccione una o varias celdas. Algunas de las preguntas requieren algún tipo de razonamiento lógico, por ejemplo, cuando la pregunta requiere que se cuenten celdas, o que se calcule la media de varias.

Es por eso que el output del modelo tiene un operador de agregación opcional. Los operadores disponibles son SUM, COUNT, AVG y NONE, este último para cuando no haga falta utilizar ningún operador. El operador será seleccionado por una capa linear seguida por un softmax.

Proceso de inferencia:

El output final del modelo, que no la predicción final, tendrá un operador de agregación junto con un conjunto de celdas de la tabla. Luego, se toman todas las celdas con probabilidad mayor que 0.5, y se ejecuta el operador sobre ellas. Después de esto, conseguiremos la predicción final del modelo.

4. Proceso de preentrenamiento

Como ya he comentado antes, TAPEX consigue un preentrenamiento eficiente entrenando modelos de lenguaje para que imiten el comportamiento de un ejecutor SQL.

4.1. Tarea de preentrenamiento

TAPEX solamente tiene una tarea de preentrenamiento, hacer ejecuciones SQL. El proceso es similar a cómo se generan las predicciones. Primero, se concatena la pregunta con la tabla que ha sido ya aplanada, después se pasa esto al codificador del modelo. Por último, obtiene el resultado de la ejecución de la consulta a través de un ejecutor SQL (como MySQL), y esto se utiliza como supervisión del decodificador del modelo.

Este proceso se basa en la hipótesis ya anteriormente formulada, que dice que si un modelo de lenguaje puede ser entrenado para ejecutar queries de SQL y conseguir buenos resultados, entonces este entiende el contenido de la tabla.

4.2. Corpus de preentrenamiento

Para obtener buenos resultados, es necesario sintetizar el corpus de preentrenamiento. Hay que tener en cuenta dos factores en este proceso: de dónde se cogen las tablas y cuál va a ser la estrategia de muestreo de queries.

Para lo primero, los autores toman tablas públicas disponibles en la Web. Pero a diferencia de otros trabajos sobre el tema, no necesitan muchas tablas, por lo que sólo toman 1.500 tablas del conjunto de entrenamiento WikiTableQuestions, disponible en HuggingFace.

La estrategia de muestreo de queries que siguen es la de utilizar plantillas de queries de SQL, que están disponibles en el dataset SQUALL. Dada una plantilla, se toman celdas y encabezados de tablas de muestra, y sirven para rellenar las plantillas, creando así queries de SQL completas.

4.3. Resultados con el modelo preentrenado

En el paper de TAPEX (Liu et al., 2022) los autores dicen obtener un acierto del 57.5 %, pero hay que tener en cuenta las circunstancias en las que se ha conseguido esto.

Este resultado se obtiene al ejecutar la versión Large del modelo TAPEX, con un fine-tuning previo sobre el conjunto de datos WikiTableQuestions. En el preentrenamiento para este modelo se han sintetizado cinco millones de pares de queries SQL junto con su resultado de ejecución.

Después, el proceso de preentrenamiento ha tenido 50.000 pasos, ejecutados durante 36 horas con ocho GPUs distintas. El proceso de fine-tuning es de 20.000 pasos.

Con todo esto, quiero decir que es imposible que obtengamos resultados cercanos a ese 57.5 % con una ejecución del modelo solamente preentrenado y en su versión base.

De hecho, el resultado que he obtenido tras 6 horas de ejecución en un cuaderno de Colab es del 18 %. La ejecución la he realizado sobre el conjunto de test de WikiTableQuestions, con 2200 muestras aleatorias de las más de 5000 que tiene. Visto esto, creo que son varias las razones que explican tal diferencia sobre el resultado de los autores:

1. El modelo que he utilizado es la versión base, por lo que, en teoría, obtendré mejores resultados si utilizo TAPEX Large.

2. Los recursos de hardware de los que dispongo son muy inferiores a los que utilizan los autores.

3. El proceso de transformación de las predicciones para tener el mismo formato que los resultados lo he hecho a mano, por lo que puede que haciendo algún cambio más obtenga mejor resultado.

5. Proceso de fine-tuning

Los autores de TAPEX describen su proceso de fine-tuning con el siguiente título 'fine-tuning on downstream tasks'. Esto, de forma literal, significa proceso de fine-tuning para tareas río abajo.

Dicho de esta forma puede que no se entienda demasiado, pero básicamente hace referencia a la estrategia de que un modelo de tipo Transformer realice el fine-tuning sobre otro modelo Transformer preentrenado. Este tipo de tareas 'downstream' requieren razonamiento sobre oraciones en lenguaje natural y tablas, justo el problema que tratamos aquí. Son dos las tareas sobre las que se realiza el fine-tuning: TableQA (respuestas a preguntas sobre una tabla) y TableFV (verificar hechos con tablas).

5.1. Fine-tuning generativo

Dicho esto, los autores proponen realizar el fine-tuning con un enfoque generativo. Se toman las tareas de TableQA y TableFV como de generación de secuencias y utiliza un modelo de lenguaje natural para generar outputs de forma autorregresiva (como hemos visto en el apartado 3).

- Arquitectura: En un principio, se puede utilizar este método para cualquier modelo de lenguaje, pero en este trabajo usan BART. Ya hemos visto antes cómo funciona este modelo.
- Input: Este es idéntico al que recibe el modelo en su fase de preentrenamiento, una concatenación de la pregunta y la tabla aplanada.
- Output: Ya sabemos que el decoder genera la respuesta de forma autorregresiva. En el caso de TableFV, se pasa el mismo input al codificador y decodificador, y se utiliza un clasificador binario sobre el estado oculto del último token del decoder para generar el output.
- Estrategia de fine-tuning: Utilizan dos formas de fine-tuning. La primera, para hacerlo de forma independiente sobre cada tarea; la otra, inspirada por TAPAS (Herzig et al., 2020), primero lo hace sobre tareas parecidas y luego lo continúa en la tarea objetivo.

5.2. Fine-tuning en la práctica

Hasta ahora, hemos visto cómo han hecho los autores de TAPEX para realizar el fine-tuning sobre el conjunto de datos WikiTableQuestions, pero lo mejor es probarlo de primera mano.

Para la primera prueba he tomado la versión vanilla de Tapex Base. El fine-tuning se realiza ejecutando un script que se puede encontrar en el repositorio de HuggingFace del propio modelo. En este script he puesto 2000 pasos como tope (que son unas dos horas utilizando la T4 GPU disponible en Colab), y he quitado la opción de evaluación, ya que la quiero realizar igual que con la versión preentrenada que hemos visto antes.

A pesar de que es posible guardar el modelo en un fichero del Drive, este ocupa demasiado, por lo que lo guardo en el entorno de ejecución para ejecutarlo en el momento. Si quisiese guardar una versión final del modelo fine-tuned simplemente tendría que especificar el directorio en el 'output dir'.

El proceso de evaluación ha sido más simple que el de la versión preentrenada, con el que tuve varios problemas con el tiempo de ejecución disponible con las GPUs de Colab. Para este modelo he realizado una fase de test con 500 muestras, y el resultado ha sido un acierto del 49.4 %. Ya vemos que la mejora ha sido muy notable.

El siguiente experimento que he realizado ha sido ejecutar el script the fine-tuning sobre el modelo TAPEX en su versión Large. El procedimiento ha sido el mismo que el utilizado antes, excepto que he reducido el número de pasos, ya que tarda más en realizarlo.

A pesar de conseguir el modelo fine-tuned, no he podido probarlo sobre el conjunto de test entero porque al hacerlo se supera el límite de la RAM. Simplemente he hecho algunas pruebas sobre conjuntos pequeños de test y una prueba final con un conjunto de 500 muestras. Tras un par de horas de ejecución el resultado ha sido un acierto del 53.6 %.

Viendo los resultados que hemos obtenido tras el fine-tuning de ambos modelos, creo que podemos asegurar que este tiene un gran impacto en el acierto final del modelo. A pesar de todo, creo que no se pueden comparar directamente mis resultados con los de (Liu et al., 2021), ya que deberíamos ejecutar todo el conjunto de test para tener una experimentación realista.

Aquí una tabla con la comparativa de todos los porcentajes de acierto:

Modelo	Acierto
TAPEX BASE preentrenado	18 %
TAPEX BASE fine-tuned	49.4 %
TAPEX LARGE fine-tuned	53.6 %
TAPEX literatura	57.5 %

Cuadro 1: Resultados de cada modelo TAPEX

6. Conclusiones

En este apartado me gustaría hablar sobre algunas de las cosas más interesantes del trabajo, además de comentar ciertas observaciones que he realizado.

Primero de todo, aún no me había topado con la idea de juntar dos modelos como BERT y GPT para aprovechar las mejores características de ambos (Lewis et al., 2019). Creo que es una combinación muy favorable, sobre todo cuando en una misma tarea es necesario mirar el contexto y generar secuencias.

Por otro lado, ha sido mi primera vez trabajando con fine-tuning de modelos, y creo que la forma en la que lo explican en (Liu et al., 2021) es muy accesible para iniciarse. Además, he sido capaz de ponerlo en práctica y ver los buenos resultados que se obtienen.

En cuanto a la idea principal del trabajo, creo que el planteamiento de preentrenar un modelo para convertirlo en un ejecutor SQL es muy ventajoso, ya que este está aprendiendo un comportamiento con el que podrá ejecutar cualquier operador sobre una tabla.

Pero para conseguir esto último, el modelo debe ser capaz de entender completamente la tabla sobre la que trabaja, y creo que es ahí donde todavía hay margen de mejora. Según las pruebas que he ido realizando, TAPEX no entiende del todo algunas de las tablas que maneja. De hecho, sufre cuando la pregunta exige algún tipo de operación agregada compleja.

Dicho esto, y para concluir, creo que TAPEX es una aproximación muy novedosa hacia la tarea de question answering, sobre todo teniendo en cuenta la pequeña cantidad de datos que requiere para el preentrenamiento, en comparación con modelos anteriores. Aún así, ya hemos visto que hay ciertos problemas sobre los que trabajar, por lo que se trata de una tarea con espacio de mejora.

References

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*.