

HPC TOOLS AI: BASELINE

Unai Iborra Albeniz, Gonzalo Silvalde Blanco

iborra.unai@gmail.com, gonzalo.silvalde@udc.es

Resumen

En esta primera práctica de **HPC TOOLS AI**, hemos trabajado sobre **BERT-Base Model**, un modelo desarrollado por Google. Es un *transformer encoder* preentrenado que produce representaciones contextuales de palabras teniendo en cuenta tanto el contexto a la izquierda como a la derecha, por lo tanto, es considerado bidireccional. Para el entrenamiento se ha utilizado como conjunto de datos para el entrenamiento **SQUAD dataset**, un dataset utilizado tradicionalmente para comprensión lectora. Para el *profiling*... ESTO DE LO QUE ENTREGUE UNAI.

1. BERT

BERT (*Bidirectional Encoder Representations from Transformers*) es un transformador bidireccional preentrenado en texto sin etiquetar para predecir tokens enmascarados en una oración y predecir si una oración sigue a otra [1]. La idea principal es que, al enmascarar aleatoriamente algunos tokens, el modelo puede entrenarse en el texto a la izquierda y a la derecha, lo que le proporciona una comprensión más profunda. BERT también es muy versátil porque sus representaciones lingüísticas aprendidas pueden adaptarse a otras tareas de NLP (*Natural Language Processing*) mediante el ajuste de una capa o cabeza adicional.

La arquitectura típica del modelo base de **BERT** consta de 12 bloques de *encoders*, 768 unidades de tamaño oculto y 12 cabezas de atención, totalizando aproximadamente 110 millones de parámetros.

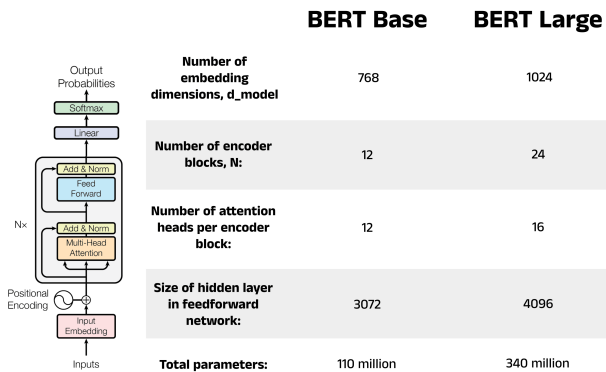


Figura 1: Arquitectura del modelo BERT

El preentrenamiento de BERT se realiza mediante dos tareas principales: **MLM** (*Masked Language Modeling*), donde se enmascara aleatoriamente el 15% de los tokens de entrada y el modelo debe predecirlos; y **NSP** (*Next Sentence Prediction*), donde el modelo aprende a determinar si dos oraciones son consecutivas en el texto original [2]. Este enfoque de preentrenamiento permite

que **BERT** capture relaciones contextuales bidireccionales profundas, superando a modelos anteriores como ELMo o GPT en numerosas tareas de comprensión del lenguaje natural.

En nuestro caso concreto, hemos utilizado **BertForQuestionAnswering** de la librería **transformers** de Hugging Face [3]. Esta clase especializada extiende el modelo BERT base añadiendo una capa de clasificación lineal sobre la última capa oculta del codificador, diseñada específicamente para predecir las posiciones de inicio y fin de la respuesta dentro del contexto proporcionado.

La arquitectura de **BertForQuestionAnswering** genera dos conjuntos de logits: uno para identificar el token de inicio de la respuesta y otro para el token final [4]. Durante el entrenamiento, cuando se proporcionan las etiquetas **start_positions** y **end_positions**, el modelo calcula automáticamente la función de pérdida como la suma de dos pérdidas de entropía cruzada (una para cada predicción). Esta automatización simplifica significativamente el proceso de entrenamiento, ya que no es necesario implementar manualmente el cálculo de la pérdida ni la lógica de backpropagation específica para esta tarea.

Además, hemos configurado el modelo con el optimizador **AdamW** con una tasa de aprendizaje de $3e-5$ y un scheduler lineal con warmup, siguiendo las recomendaciones estándar para el fine-tuning de modelos BERT [5]. El entrenamiento se ha realizado con un tamaño de batch de 8 ejemplos durante 10 épocas, utilizando el subconjunto de entrenamiento de SQuAD previamente preprocesado.

1.1. Tokenización: BertTokenizerFast

Para la tokenización de los datos, hemos empleado BertTokenizerFast, el tokenizador recomendado para BertForQuestionAnswering.

En nuestra implementación, hemos configurado una longitud máxima de 384 tokens (**max_length=384**) [6], un valor estándar para tareas de Question Answering que equilibra la capacidad de procesar contextos extensos con las limitaciones de memoria.

Además en nuestra configuración usamos `stride=128` con `return_overflowing_tokens=True`.

1.2. Dataset: SQuAD

SQuAD (*Stanford Question Answering Dataset*) es uno de los conjuntos de datos más utilizados para entrenar y evaluar modelos de comprensión lectora y respuesta a preguntas [7]. La versión que hemos utilizado, SQuAD v1.1, contiene más de 100.000 pares de pregunta-respuesta basados en artículos de Wikipedia, donde cada respuesta es un segmento de texto extraído directamente del contexto proporcionado.

La estructura de SQuAD es especialmente adecuada para tareas de *extractive question answering*, donde el objetivo es identificar el inicio y el final exactos de la respuesta dentro del contexto. En nuestra implementación, hemos trabajado con un subconjunto de 10.000 ejemplos del conjunto de entrenamiento (`subset_size=10000`) para reducir el tiempo de entrenamiento. El preprocesamiento de estos datos incluye la tokenización conjunta de preguntas y contextos, el cálculo de las posiciones de inicio y fin de las respuestas en términos de tokens, y el manejo de casos donde la respuesta no puede ser localizada en el segmento tokenizado debido al truncamiento.

2. Profiling

Para realizar el *profiling* del entrenamiento se ha utilizado la clase `torch.profiler.profile()`. Ésta permite analizar con detalle las llamadas a función y etapas del modelo, uso de CPU, GPU, memoria, etc.

Se han definido los siguientes tres parámetros para facilitar la configuración del *profiling* en el método `train()`:

1. `profiler=None`,
2. `save_profiler_time_table: bool = False`,
3. `save_tensorboard_metrics: bool = False`.

El parámetro `profiler` permite definir el *profiler* deseado para el entrenamiento según se defina la clase `torch.profiler.profile()`. El parámetro `save_profiler_time_table` permite al entrenamiento guardar en formato tabla los resultados del *profiling*, ordenados según el tiempo tardado en ejecutar CUDA por cada parte de la ejecución del entrenamiento.

El parámetro `save_tensorboard_metrics` permite guardar para una futura visualización con TensorBoard:

1. Los *losses*, *step times* y *learning rates* de cada *step* en formato escalar.
2. El tiempo total, pérdida final, *batch size* y número de épocas.

Para ejecutar fácilmente el servidor de TensorBoard se ha creado el script `/scripts/start_profiling_srv.sh` (configurado para visualizar los resultados de todos los entrenamientos realizados).

Se ha preparado el código para correr con la siguiente configuración de *profiling*: la configuración establecida utiliza un *profiler* que sigue el siguiente *schedule*:

`schedule(wait=2, warmup=100, active=8, repeat=1)`

Esto indica al *profiler* que omita los primeros 2 *steps*, que haga *profiling* de los siguientes 100 pero no utilice ni guarde sus resultados (inicializar el *profiler* causa una bajada de rendimiento significativa según la documentación, por lo que se recomienda utilizar *warmup* para resultados más fiables). Finalmente, se realiza el *profiling* de los 8 siguientes *steps*.

Esta configuración guarda tanto los resultados del *profiling* en formato tabla, como en formato `.json` para visualización mediante *Perfetto*. La decisión de realizar *profiling* en únicamente 8 *steps* ha sido tomada debido a que, en pruebas de *profiling* de todo el entrenamiento, éste finalizaba por falta de memoria al guardar los datos. En los casos de entrenamientos con menos *epochs*, el entrenamiento sí finalizaba, pero los resultados del *profiling* ocupaban decenas de gigabytes.

Aunque el *profiling* no se realiza a lo largo de todo el entrenamiento, los resultados obtenidos son representativos y permiten identificar las secciones más costosas en tiempo y recursos. Además, debido a este *schedule*, los resultados de tiempo total de la tabla de métricas de *profiling* no coincidirán con los resultados de entrenamiento real (el resultado de la tabla es el tiempo total de entrenamiento con *profiling* aplicado).

Se ha explorado la opción de visualizar los resultados del *profiling* mediante TensorBoard, pero no se ha podido realizar dado que esta funcionalidad ha sido deprecada según la documentación de PyTorch[8].

La documentación indica que se debe utilizar *Perfetto* para la visualización del *profiling*.

Referencias

- [1] Hugging Face. *BERT - Hugging Face Documentation*. https://huggingface.co/docs/transformers/model_doc/bert
- [2] Hugging Face. *BERT base model*. <https://huggingface.co/google-bert/bert-base-uncased>
- [3] Hugging Face. *Transformers Library*. https://huggingface.co/docs/transformers/v4.57.1/en/model_doc/bert#transformers.BertForQuestionAnswering
- [4] Hugging Face. *Question Answering - Task Guide*. https://huggingface.co/docs/transformers/tasks/question_answering
- [5] Hugging Face. *Fine-tuning a pretrained model*. <https://huggingface.co/docs/transformers/optimizers>
- [6] Hugging Face Discuss. *Why max_length = 384 by default and not 512?*. <https://discuss.huggingface.co/t/squad-bert-why-max-length-384-by-default-and-not-512/11693>
- [7] Hugging Face. *SQuAD Dataset Card*. <https://huggingface.co/datasets/squad>

[8] Pytorch *Pytorch Documentation*. https://docs.pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html