1. Introducción al Ajuste Fino de Modelos de Lenguaje

El ajuste fino (fine-tuning) es el proceso de tomar un modelo de lenguaje preentrenado y adaptarlo a tareas específicas mediante entrenamiento adicional con datos particulares. Esta técnica permite mejorar el rendimiento del modelo en aplicaciones concretas, como traducción automática, generación de código o análisis de sentimientos.

2. Herramientas y Bibliotecas Recomendadas

Para llevar a cabo el ajuste fino de modelos de lenguaje de manera eficiente y alineada con las prácticas actuales del mercado, se recomiendan las siguientes herramientas:

- **Transformers de Hugging Face**: Biblioteca líder para trabajar con modelos de lenguaje, que proporciona interfaces sencillas para cargar, entrenar y desplegar modelos preentrenados.

- **Datasets de Hugging Face**: Facilita la carga y manipulación de conjuntos de datos para entrenamiento y evaluación de modelos.

- **Accelerate de Hugging Face**: Permite optimizar el entrenamiento distribuido en múltiples GPUs, mejorando la eficiencia y reduciendo los tiempos de entrenamiento.

- **PEFT (Parameter-Efficient Fine-Tuning)**: Biblioteca que implementa técnicas de ajuste fino eficientes en parámetros, como LoRA (Low-Rank Adaptation), permitiendo entrenar modelos grandes con menores recursos computacionales.

- **TRL (Transformer Reinforcement Learning)**: Proporciona herramientas para aplicar técnicas de aprendizaje por refuerzo en modelos de lenguaje, mejorando su capacidad de generación en tareas específicas.

- **BitsAndBytes**: Ofrece soporte para cuantización de modelos, permitiendo reducir el uso de memoria y acelerar el entrenamiento mediante técnicas de precisión mixta.

3. Preparación del Entorno

Antes de comenzar, asegúrese de tener instalado Python y las bibliotecas mencionadas. Puede instalarlas utilizando pip:

```
pip install -U transformers datasets accelerate peft trl bitsandbytes
```

4. Carga del Modelo y Tokenizador

Utilizaremos la biblioteca transformers para cargar el modelo Llama 3.1 8B Instruct y su tokenizador. Dado el tamaño del modelo, es recomendable utilizar técnicas de cuantización para optimizar el uso de memoria, como cargar el modelo en precisión de 4 bits:

```
from transformers import AutoModelForCausalLM, AutoTokenizer,
BitsAndBytesConfig

base_model = "ruta/al/modelo/llama-3.1-8b-instruct"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
)

tokenizer = AutoTokenizer.from_pretrained(base_model)
model = AutoModelForCausalLM.from_pretrained(
    base_model,
    quantization_config=bnb_config,
    device_map="auto",
    torch_dtype=torch.float16,
)
```

5. Preparación de los Datos

Formato de los Archivos de Entrada

Para el ajuste fino, los datos deben estar en un formato estructurado, como JSON Lines (.jsonl), donde cada línea representa una instancia de entrenamiento. Cada instancia puede contener campos como instruction, input y output.

Ejemplo de archivo de entrada (data.jsonl):

```
{"instruction": "Traduce la siguiente oración al francés.",
"input": "Buenos días", "output": "Bonjour"}
{"instruction": "Proporciona un sinónimo para la palabra
'feliz'.", "input": "", "output": "Contento"}
```

Carga de los Datos

Utilice la biblioteca datasets para cargar y procesar el conjunto de datos:

```
from datasets import load_dataset

dataset = load_dataset("json", data_files="ruta/al/data.jsonl")
```

6. Configuración del Entrenamiento

Implementaremos la técnica de Low-Rank Adaptation (LoRA) para un ajuste fino eficiente, utilizando la biblioteca `peft`. A continuación, se detallan los parámetros clave utilizados en la configuración del entrenamiento:

- **`per_device_train_batch_size`**: Número de muestras procesadas por cada dispositivo (GPU/TPU/CPU) en cada paso de entrenamiento. Un valor más alto puede acelerar el entrenamiento, pero requiere más memoria.
- **`per_device_eval_batch_size`**: Número de muestras procesadas por cada dispositivo durante la evaluación.
- **`num_train_epochs`**: Número de veces que el modelo recorrerá el conjunto de datos completo durante el entrenamiento.
- **`learning_rate`**: Tasa a la cual se ajustan los pesos del modelo durante el entrenamiento. Valores más altos pueden conducir a una convergencia más rápida, pero también a la inestabilidad del entrenamiento.
- **`fp16`**: Si se establece en `True`, utiliza precisión de 16 bits para reducir el uso de memoria y acelerar el entrenamiento.
- **`logging_steps`**: Frecuencia (en pasos) con la que se registran las métricas de entrenamiento.
- **`evaluation_strategy`**: Determina cuándo se realiza la evaluación durante el entrenamiento (`"steps"` o `"epoch"`).
- **`save_strategy`**: Determina cuándo se guarda el modelo durante el entrenamiento.
- **`save_steps`**: Número de pasos entre cada guardado del modelo.
- **`warmup_steps`**: Número de pasos de calentamiento durante los cuales la tasa de aprendizaje aumenta linealmente desde cero hasta su valor máximo.
- **`save_total_limit`**: Número máximo de puntos de control del modelo que se guardarán.

LoRA (Adaptación de Bajo Rango) es una técnica diseñada para optimizar el ajuste fino de modelos de lenguaje de gran escala, como Llama 3.1 8B Instruct. Su objetivo principal es reducir la cantidad de parámetros entrenables durante el ajuste fino, lo que disminuye significativamente los recursos computacionales necesarios y facilita la adaptación de estos modelos a tareas específicas.

¿En qué consiste LoRA?

Tradicionalmente, el ajuste fino de modelos de lenguaje implica actualizar todos los parámetros del modelo, lo que puede ser ineficiente y costoso en términos de tiempo y recursos, especialmente cuando se trata de modelos con miles de millones de parámetros. LoRA aborda este desafío introduciendo matrices de bajo rango que se integran en las capas del modelo original. Estas matrices adicionales permiten capturar las adaptaciones necesarias para la nueva tarea sin modificar los pesos preentrenados del modelo base. De esta manera, se preserva el conocimiento general del modelo mientras se ajusta de manera eficiente a tareas específicas.

¿Por qué se utiliza LoRA?

Las principales ventajas de utilizar LoRA en el ajuste fino de modelos de lenguaje incluyen:

- **Eficiencia en el entrenamiento y despliegue**: Al reducir la cantidad de parámetros que necesitan ser entrenados, LoRA disminuye la carga computacional, permitiendo una adaptación más rápida y económica de los modelos.
  DataCamp
- **Preservación de la calidad y velocidad del modelo**: A pesar de la reducción en los parámetros entrenables, LoRA mantiene la calidad y la velocidad de inferencia del modelo original, asegurando que el rendimiento no se vea comprometido.
  DataCamp
- **Menor huella de memoria**: LoRA reduce significativamente los requisitos de memoria GPU para el ajuste fino, lo que hace posible trabajar con modelos grandes incluso en hardware más modesto.
  Brnss Pub Hub
- **Facilidad de adaptación a múltiples tareas**: Las matrices de bajo rango introducidas por LoRA son de tamaño reducido, lo que facilita su carga y descarga para distintas tareas y usuarios, promoviendo la reutilización y personalización eficiente del modelo.
  DataCam

Implementación de LoRA en el Ajuste Fino

Para implementar LoRA en el ajuste fino de un modelo como Llama 3.1 8B Instruct, se pueden seguir los siguientes pasos:

1. **Configuración de LoRA**: Defina los parámetros de LoRA, como el rango (`r`), el factor de escalado (`alpha`), los módulos objetivo y la tasa de abandono (`dropout`).

```
from peft import LoraConfig, get_peft_model,
prepare_model_for_kbit_training

lora_config = LoraConfig(
      r=16,
      lora_alpha=16,
      target_modules=["q_proj", "v_proj"],
      lora_dropout=0.05,
      bias="none",
      task_type="CAUSAL_LM",
)
```

2. **Preparación del Modelo**: Prepare el modelo para el entrenamiento con bits de precisión y aplique la configuración de LoRA.

```
model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, lora_config)
```

7. Entrenamiento del Modelo

Utilice el `SFTTrainer` de la biblioteca `trl` para entrenar el modelo con los datos y la configuración preparados:

```
from trl import SFTTrainer

trainer = SFTTrainer(
      model=model,
      train_dataset=dataset["train"],
      eval_dataset=dataset["validation"],
      peft_config=lora_config,
      dataset_text_field="text",
      max_seq_length=2048,
      tokenizer=tokenizer,
      args=training_args,
)

trainer.train()
```

8. Evaluación y Ajustes Posteriores

Después del entrenamiento, evalúe el rendimiento del modelo en un conjunto de datos de validación para asegurarse de que cumple con los requisitos de la tarea específica. Realice ajustes adicionales en los hiperparámetros o en los datos de entrenamiento si es necesario.

9. Guardado y Despliegue del Modelo

Una vez satisfecho con el rendimiento, guarde el modelo ajustado y considere subirlo a una plataforma como Hugging Face para facilitar su integración en aplicaciones futuras:

```
model.push_to_hub("nombre_del_modelo_ajustado")
tokenizer.push_to_hub("nombre_del_modelo_ajustado")
```

Enlaces de Interés

Para profundizar en el ajuste fino de modelos de lenguaje, puede consultar los siguientes recursos:

- Fine-tuning a un modelo pre-entrenado - Hugging Face
- El Trainer - Hugging Face

.