TL02 transformers

Índice

1 Introducción

transformers: librería de Hugging Face (HF) para usar transformers

Web HF: https://huggingface.co

- **Models:** 1.8M+ con filtros por parámetros, tareas, librerías, etc.
- **Datasets:** 453K+ con filtros por modalidades, tamaño, formato, tareas, librerías, etc.
- **Spaces:** aplicaciones web con filtros por tareas
- **Docs:** Hub & Client Libraries, Deployment & Inference, **Core ML Libraries,** Training & Optimization, etc.

Github transformers: https://github.com/huggingface/transformers

- **Installation:** `pip install transformers`
- **Quickstart:** con `Pipeline` para generación de texto, chat, ASR, etc.
- **Why should I use Transformers?:** para usar modelos SOTA pre-entrenados
- **Why shouldn't I use Transformers?:** no es una librería de propósito general para redes
- **100 projects using Transformers:** Awesome projects built with Transformers
- **Example models:** audio, visión, multimodal y NLP

Github transformers según DeepWiki https://deepwiki.com/huggingface/transformers

Objetivo: introducir HF y transformers

Cuenta HF (opcional): se recomienda crearla

2 Hugging Face

Hub & Client Libraries: hub y librerías cliente asociadas

- **Hub:** **modelos, datasets y spaces gestionados con repos git**
- **Hub Python Library:** librería Python cliente para usar el Hub
- **Huggingface.js:** librerías JavaScript para usar la API HF
- **Tasks:** explorador de demos, modelos y datasets
- **Dataset viewer:** explorador de datasets y API asociada

Ejercicio sobre Tasks: ¿cuál es la tarea más popular (en número de modelos)?

Deployment & Inference: despliegue de modelos e inferencia

- **Inference Providers:** 200k+ modelos alojados por 10+ proveedores de inferencia
- **Inference Endpoints (dedicated):** despliegue de modelos en infraestructura HF
- **Deploying on AWS:** entrenamiento y despliegue de modelos en AWS
- **Text Generation Inference:** toolkit para desplegar y servir LLMs
- **Text Embeddings Inference:** toolkit para desplegar y servir embeddings textuales
- **Microsoft Azure:** integración de modelos y herramientas HF en Azure

Ejercicio sobre Inference Providers: visita el Inference Playground y pregunta a algún LLM

¿En qué semestre se imparte la asignatura Percepción de cuarto curso del grado en ingeniería informática de la UPV?

Core ML Libraries: librerías principales

- **Transformers:** **transformers con PyTorch principalmente**
- **Diffusers:** modelos de difusión en PyTorch
- **Datasets:** acceso y compartición de datasets
- **Transformers.js:** transformers en javascript
- **Tokenizers:** tokenizadores optimizados
- **Evaluate:** evaluación y comparación del rendimiento de modelos
- **timm:** capas, optimizadores y utilidades para modelos de visión
- **Sentence Transformers:** embeddings, recuperación y reránquing

Ejercicio sobre Transformers.js: visita el demo site y prueba algún modelo

Training & Optimization:

- **PEFT:** Parameter-efficient finetuning para LLMs
- **Accelerate:** entrenamiento de modelos con multi-GPU, TPU y precisión mixta
- **Optimum:** optimización de transformers para entrenamiento e inferencia más rápidos
- **AWS Trainium & Inferentia:** entrenamiento e inferencia de transformers/difusores en AWS
- **TRL:** entrenamiento de LMs transformer con aprendizaje por refuerzo
- **Safetensors:** manera segura de almacenar y distribuir pesos de redes
- **Bitsandbytes:** optimización y cuantización de modelos con bitsandbytes
- **Lighteval:** toolkit todo-en-uno para evaluar LLMs con múltiples backends

Ejercicio sobre PEFT: ¿qué modelos y métodos PEFT están disponibles para clasificación de imágenes?

Collaboration & Extras:

- **Gradio:** desarrollo de demos y apps web con unas pocas líneas de Python
- **smolagents:** librería Smol para desarrollar agentes en Python
- **LeRobot:** modelos, datasets y herramientas para robótica en Pytorch
- **AutoTrain:** API e interfaz para entrenamiento de modelos simplificado
- **Chat UI:** frontend de chat basado en HuggingChat
- **Leaderboards:** creación de Leaderboards personalizados
- **Argilla:** herramienta colaborativa para construir datasets de alta calidad
- **Distilabel:** generación de datos sintéticos y feedback IA

Ejercicio sobre gradio: edita el chat básico de gradio Playground para que responda `Sí`, `No` o `No lo sé` al azar

Community:

- **Blog:** muy interesante para estar al día con las últimas novedades
- **Learn:** cursos LLM, Deep RL, CV, Audio, etc.
- **Discord:** chat grupal
- **Forum:** foro muy activo sobre cuestiones diversas
- **Github:** git HF con 334+ repos

Ejercicio sobre el blog: visita el blog y busca algún post reciente que te interese

Soluciones

Tasks: text generation con 295K+ modelos

Inference Providers: `Kimi-K2-Instruct` en `groq` dice:

La asignatura **Percepción** se imparte en **tercer curso, segundo semestre**; no existe una asignatura con ese nombre en cuarto curso del Grado en Ingeniería Informática de la UPV.

Transformers.js: por ejemplo, `codegen-350M-mono` con prompt `def sigmoid(x):` genera:

In [ ]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

PEFT: solo `lora` para los modelos `vit` y `swin`

Gradio:

In [ ]:
```
import random
import gradio as gr

def random_response(message, history):
    return random.choice(["Sí", "No", "No lo sé"])

demo = gr.ChatInterface(random_response, type="messages", autofocus=False)

if __name__ == "__main__":
    demo.launch()
```

Blog: Arc Virtual Cell Challenge: A Primer y web del challenge

3 transformers

Transformers: establece definiciones estandarizadas de modelos para texto, visión, audio, vídeo y multimodal

Conveniencia de la estandarización: una definición estándar facilita el tratamiento del modelo mediante la mayoría de herramientas de entrenamiento (Axolotl, Unsloth, DeepSpeed, FSDP, PyTorch-Lightning, etc.), motores de inferencia (vLLM, SGLang, TGI, etc.) y librerías de modelado asociadas que la aprovechan (llama.cpp, mlx, etc.)

Características principales: para inferencia y entrenamiento

- `Pipeline`: clase para inferencia a partir de una tarea y modelo dados
- `Trainer`: clase para entrenamiento a partir de un modelo, preprocesador y dataset dados
- `generate`: API para la generación de texto con LLMs

Diseño sencillo y fácil de usar: cada modelo se implementa a partir de tres clases principales solamente (configuración, modelo y preprocesador) y puede usarse rápidamente para inferencia o entrenamiento con `Pipeline` o `Trainer`

Conveniencia de uso de modelos preentrenados: cada modelo preentrenado reproduce fielmente el original; su uso compartido facilita el desarrollo de sistemas y ahorra consumo de energía

Aprendizaje: recomendaciones en https://huggingface.co/learn

Instalación: `pip install transformers`

Ejemplo: lectura de un LLM preentrenado reciente con su tokenizador asociado

In [ ]:
```python
from transformers import AutoModelForCausalLM, AutoTokenizer
model_name = "Qwen/Qwen3-4B-Instruct-2507"
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype="auto", device_map="auto")
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

In [ ]:
```python
model
```

Out[ ]:
```
Qwen3ForCausalLM(
  (model): Qwen3Model(
    (embed_tokens): Embedding(151936, 2560)
    (layers): ModuleList(
      (0-35): 36 x Qwen3DecoderLayer(
        (self_attn): Qwen3Attention(
          (q_proj): Linear(in_features=2560, out_features=4096, bias=False)
          (k_proj): Linear(in_features=2560, out_features=1024, bias=False)
          (v_proj): Linear(in_features=2560, out_features=1024, bias=False)
          (o_proj): Linear(in_features=4096, out_features=2560, bias=False)
          (q_norm): Qwen3RMSNorm((128,), eps=1e-06)
          (k_norm): Qwen3RMSNorm((128,), eps=1e-06)
        )
        (mlp): Qwen3MLP(
          (gate_proj): Linear(in_features=2560, out_features=9728, bias=False)
          (up_proj): Linear(in_features=2560, out_features=9728, bias=False)
          (down_proj): Linear(in_features=9728, out_features=2560, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): Qwen3RMSNorm((2560,), eps=1e-06)
        (post_attention_layernorm): Qwen3RMSNorm((2560,), eps=1e-06)
      )
    )
    (norm): Qwen3RMSNorm((2560,), eps=1e-06)
    (rotary_emb): Qwen3RotaryEmbedding()
  )
  (lm_head): Linear(in_features=2560, out_features=151936, bias=False)
)
```

Ejemplo (cont.): inferencia con `generate()` y un `chat_template`

In [ ]:
```python
prompt = '''
¿Cuál es la derivada de la sigmoide en función de la propia sigmoide?
Escribe solo la respuesta final en notación matemática.
'''
messages = [{"role": "user", "content": prompt}]
text = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)
generated_ids = model.generate(**model_inputs, max_new_tokens=16384)
output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist()
output = tokenizer.decode(output_ids, skip_special_tokens=True)
```

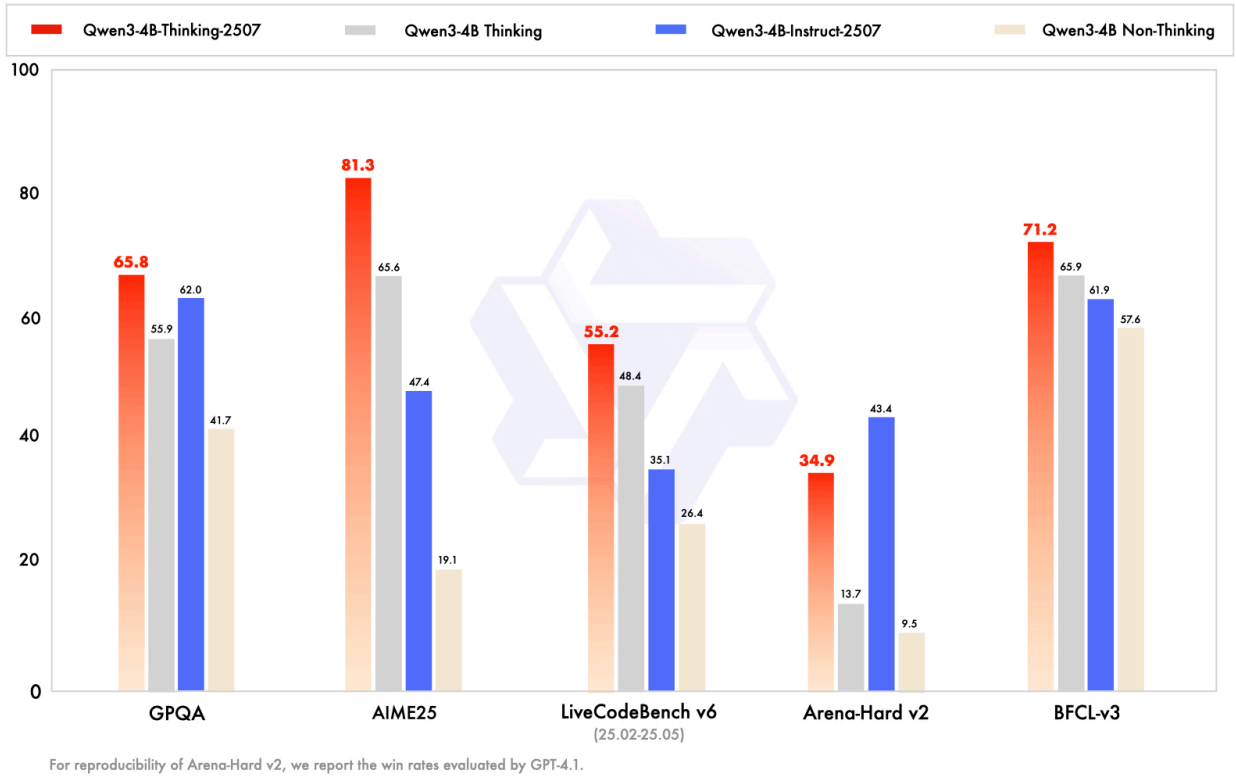
In [ ]:
```python
from IPython.display import display, Math
display(Math(output))
```

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

4 Ejercicio

Bye Qwen3-235B-A22B, hello Qwen3-235B-A22B-2507!: la familia de LLMs Qwen3 de Alibaba, publicada en abril de 2025, se actualizó en julio de 2025 (de ahí el sufijo 2507) para separar LLMs Instruct de LLMs Thinking

Comparación: Qwen3-4B (en modos Non-Thinking y Thinking) vs Qwen3-4B-Instruct-2507 y Qwen3-4B-Thinking-2507



Ejercicio: prueba el modelo `Qwen/Qwen3-4B-Thinking-2507` con el prompt del ejemplo anterior

Solución:

In [ ]:
```
from transformers import AutoModelForCausalLM, AutoTokenizer
model_name = "Qwen/Qwen3-4B-Thinking-2507"
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype="auto", device_map="auto")
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
Loading checkpoint shards:   0%|          | 0/3 [00:00<?, ?it/s]
```

In [ ]:
```
prompt = '''
¿Cuál es la derivada de la sigmoide en función de la propia sigmoide?
Escribe solo la respuesta final en notación matemática.
'''
messages = [{"role": "user", "content": prompt}]
text = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)
generated_ids = model.generate(**model_inputs, max_new_tokens=32768)
output_ids = generated_ids[0][len(model_inputs.input_ids[0]):].tolist()
index = len(output_ids) - output_ids[::-1].index(151668)
thinking_content = tokenizer.decode(output_ids[:index], skip_special_tokens=True).strip("\n")
content = tokenizer.decode(output_ids[index:], skip_special_tokens=True).strip("\n")
```

Razonamiento:

In [ ]:
```
str(thinking_content)[:400]+"..."
```

Out[ ]:
```
"Okay, so I need to find the derivative of the sigmoid function in terms of the sigmoid function itself. Let me reca
ll what the sigmoid function is. I think it's the logistic function, right? The standard sigmoid is σ(x) = 1 / (1 +
e^{-x}). \n\nFirst, I should find the derivative of σ(x) with respect to x. Let me compute that. \n\nSo, dσ/dx = d/d
x [1 / (1 + e^{-x})]. Hmm, using the chain rule here. Let..."
```

Respuesta final:

In [ ]:
```
from IPython.display import display, Markdown
display(Markdown(content))
```

La derivada de la sigmoide en función de la propia sigmoide es $\boxed{\sigma(x)(1-\sigma(x))}$.