Carga y almacenamiento de modelos

Roberto López Castro

Diego Andrade Canosa

Índice

- Introducción
- Estructura de un modelo en Pytorch
- Carga y almacenamiento de modelos completos
- Carga y almacenamiento de modelos por capas
- Carga y almacenamiento de parámetros de un modelo
- Formatos disponibles
- Carga y almacenamiento en GPU
- Carga y almacenamiento en la nube
- torch.hub

citic

Objetivos

- Proporcionar una guía completa sobre carga y almacenamiento de modelos en PyTorch.
- Aprender las diferentes técnicas y opciones disponibles para guardar y cargar modelos en PyTorch.
- Explorar cómo utilizar estas técnicas en diversos escenarios, como modelos completos, por capas y solo parámetros.
- Comprender las consideraciones especiales al trabajar con modelos distribuidos, en GPU y en la nube.

Estructura de un modelo en PyTorch

- Un modelo en PyTorch se compone de dos componentes principales: la arquitectura del modelo y los parámetros del modelo.
  - La arquitectura: estructura y la disposición de las capas y operaciones que componen el modelo.
  - Los parámetros: son los valores que se optimizan durante el entrenamiento y representan los pesos y sesgos de las capas del modelo.
- Durante el proceso de entrenamiento, se ajustan los parámetros del modelo para que se ajusten a los datos de entrenamiento y produzcan predicciones precisas.
- La capacidad de guardar y cargar tanto la arquitectura como los parámetros del modelo es esencial para su reutilización y distribución.

Guardar y cargar el estado del modelo completo

- El método *state_dict()* devuelve un diccionario que mapea los nombres de los módulos y los parámetros a sus respectivos tensores.
- Para guardar el estado del modelo, utilizamos el método *torch.save()* y pasamos el diccionario *state_dict()* junto con la ruta de archivo donde se guardará.
- Para cargar el estado del modelo, utilizamos el método *torch.load()* y pasamos la ruta del archivo que contiene el estado guardado.
- Cargar el estado del modelo nos permite reanudar la capacitación o utilizar el modelo para hacer predicciones sin necesidad de volver a entrenarlo desde cero.

Guardar y cargar el estado del modelo por capas

- En algunos casos, es posible que solo estemos interesados en guardar y cargar el estado de una capa o módulo específico en lugar del modelo completo.
- Para hacer esto, primero accedemos al estado de esa capa específica utilizando su nombre o referencia.
- Podemos obtener los nombres de las capas y módulos del modelo utilizando el método *named_modules()* o *named_parameters()*.
- Una vez que tenemos el estado de la capa deseada, podemos guardarlo utilizando *torch.save()* y cargarlo con *torch.load()* como antes.
- Esto es útil cuando queremos transferir solo una parte del modelo o cuando estamos realizando operaciones específicas en una capa particular.

Guardar y cargar el estado del modelo por capas

- En algur dar y cargar modelo
- Para hac ca utilizanc
- Podemo utilizanc
- Una vez guardarlo utilizando *torch.save()* y cargarlo con *torch.load()* como antes.
- Esto es útil cuando queremos transferir solo una parte del modelo o cuando estamos realizando operaciones específicas en una capa particular.

```
torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            ...
            }, PATH)
```

Guardar y cargar el estado del modelo por capas

- En alg… rdar y carga… model…
- Para h… utiliza… ica
- Poden… utiliza… o
- Una ve… guarda… no antes.
- Esto e… cuand… o o particu…

```
model = torch.nn.Linear(5, 2)
filepath = os.getcwd()
optimizer = optim.SGD(model.parameters(), lr = 0.001, momentum = 0.9)

# create a checkpoint directory to save pytorch model
if not os.path.exists('torch_directory'):
    os.makedirs('torch_directory')

def save_model(model_dictionary):
    checkpoint_directory = 'torch_directory'
    file_path = os.path.join(checkpoint_directory, 'model.pt')
    torch.save(model_dictionary, file_path)

# make a dictionary to save model state and optimizer
model_dictionary = {
            'model_state': model.state_dict(),
            'model_optimizer': optimizer.state_dict()
            }
save_model(model_dictionary)
```

Directory is created. From directory I mean checkpoint directory

This will save torch model in .pt format

Parameters to be saved to model.pt

citic

Guardar y cargar solo los parámetros del modelo

- En algunos casos, puede ser suficiente guardar y cargar solo los parámetros del modelo en lugar del estado completo.
- Para guardar solo los parámetros, utilizamos el método *parameters()* en lugar de *state_dict()*.
- El método *parameters()* nos devuelve una lista de todos los parámetros del modelo.
- Podemos guardar esta lista de parámetros utilizando *torch.save()* y cargarla utilizando *torch.load()*.
- Al cargar solo los parámetros, es importante asegurarse de que la estructura del modelo sea la misma que al guardarlos para evitar errores de incompatibilidad.

Guardar y cargar el modelo en diferentes formatos

- PyTorch ofrece la flexibilidad de guardar y cargar modelos en diferentes formatos de archivo.
- Al guardar un modelo, podemos especificar la extensión del archivo para indicar el formato deseado, como .pt, .pth, .pkl, entre otros.
- Además de los formatos nativos de PyTorch, como .pt y .pth, también es posible utilizar formatos comunes como JSON o HDF5 para almacenar modelos.
- Al cargar un modelo, debemos asegurarnos de utilizar la extensión correcta y especificar el formato correspondiente para garantizar una carga adecuada del modelo guardado.
- La elección del formato de archivo puede depender de la compatibilidad con otras bibliotecas o herramientas, así como de las necesidades y preferencias específicas del proyecto.

Lab 1: 02_load.ipynb

Consideraciones adicionales

- En entornos distribuidos, donde se utilizan múltiples dispositivos o nodos para entrenar modelos, hay consideraciones adicionales:
- Distribución de modelos:
  - Los modelos a menudo se dividen en partes que se almacenan/cargan en diferentes dispositivos/nodos.
  - Es importante un mecanismo de distribución adecuado para dividir y guardar las diferentes partes
- Coherencia de versión:
  - Asegúrate de que todas las instancias del modelo distribuido estén utilizando la misma versión de PyTorch y las mismas dependencias para garantizar la coherencia y evitar incompatibilidades.

Consideraciones adicionales

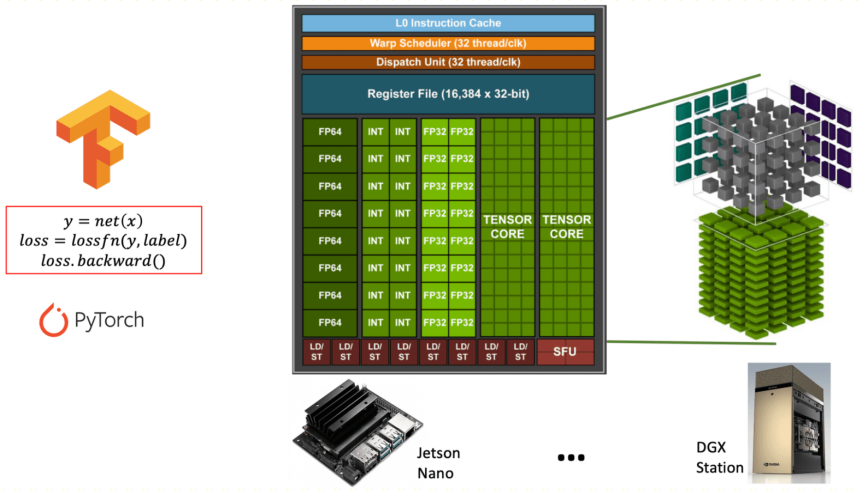
- Sincronización de estados:
  - (Al cargar modelos distribuidos) Sincronizar los estados de los diferentes dispositivos/nodos para asegurarse de que todos tengan los mismos parámetros y listos para la inferencia/entrenamiento.
- Comunicación y transferencia de modelos:
  - En entornos distribuidos, se requiere comunicación entre dispositivos o nodos para transferir partes del modelo o actualizar los parámetros durante el entrenamiento.
  - Utiliza las funcionalidades y protocolos adecuados para la comunicación eficiente y segura de los modelos entre los dispositivos o nodos.

Guardar y cargar modelos en GPU

- Guardar modelos en GPU:
  - Al guardar un modelo que se encuentra en la GPU, asegúrate de utilizar *torch.save()* con el parámetro *map_location* para especificar que el modelo debe guardarse en la GPU.
  - Por ejemplo: *torch.save(model.state_dict(), 'modelo_entrenado.pth')* guardará el modelo en la GPU si está en uso.
- Cargar modelos en GPU:
  - Al cargar un modelo guardado que se encuentra en la GPU, utiliza *torch.load()* con el parámetro *map_location* para cargar el modelo directamente en la GPU.
  - Por ejemplo: *model.load_state_dict(torch.load('modelo_entrenado.pth', map_location='cuda'))* cargará el modelo en la GPU.

Guardar y cargar modelos en GPU

- Mover modelos entre dispositivos:
  - Si deseas mover un modelo desde la CPU a la GPU o viceversa, utiliza los métodos *to()* o *cuda()* para cambiar el dispositivo del modelo según sea necesario.
  - Por ejemplo: *model.to('cuda')* moverá el modelo a la GPU, mientras que model.to('cpu') lo moverá a la CPU.



citic

Guardar y cargar modelos en GPU

- Al cargar y guardar modelos en dispositivos GPU, asegúrate de tener suficiente memoria GPU disponible para alojar el modelo y sus parámetros. -> batch size
- Utilizar dispositivos GPU para cargar y ejecutar modelos puede acelerar significativamente las operaciones, especialmente en modelos grandes y complejos.
- Aprovecha la potencia de la GPU para la carga y ejecución de modelos en PyTorch, lo que te permitirá obtener resultados más rápidos y eficientes en tus tareas de aprendizaje profundo.

Guardar y cargar modelos en la nube

- Almacenamiento en la nube:
  - Servicios de almacenamiento en la nube, como Amazon S3, Google Cloud Storage o Microsoft Azure Blob Storage, permiten guardar modelos entrenados en un repositorio centralizado y seguro en la nube.
  - Estos servicios proporcionan capacidades de almacenamiento escalables y opciones de control de acceso para compartir y proteger los modelos almacenados.
- Plataformas de aprendizaje automático en la nube:
  - Plataformas como Amazon SageMaker, Google Cloud AI Platform o Microsoft Azure Machine Learning ofrecen funcionalidades avanzadas para el entrenamiento, implementación y gestión de modelos en la nube.
  - Estas plataformas suelen incluir opciones integradas para guardar y cargar modelos entrenados, además de facilitar la infraestructura y los recursos necesarios para trabajar con modelos a gran escala.

Guardar y cargar modelos en la nube

- Al utilizar servicios en la nube para guardar y cargar modelos, es importante considerar aspectos como la seguridad, el costo y la escalabilidad de los servicios seleccionados.
- Estas opciones en la nube brindan flexibilidad y accesibilidad, lo que permite compartir y utilizar modelos entrenados de manera eficiente en diferentes entornos y aplicaciones, independientemente de la ubicación física del modelo o de los recursos computacionales necesarios.

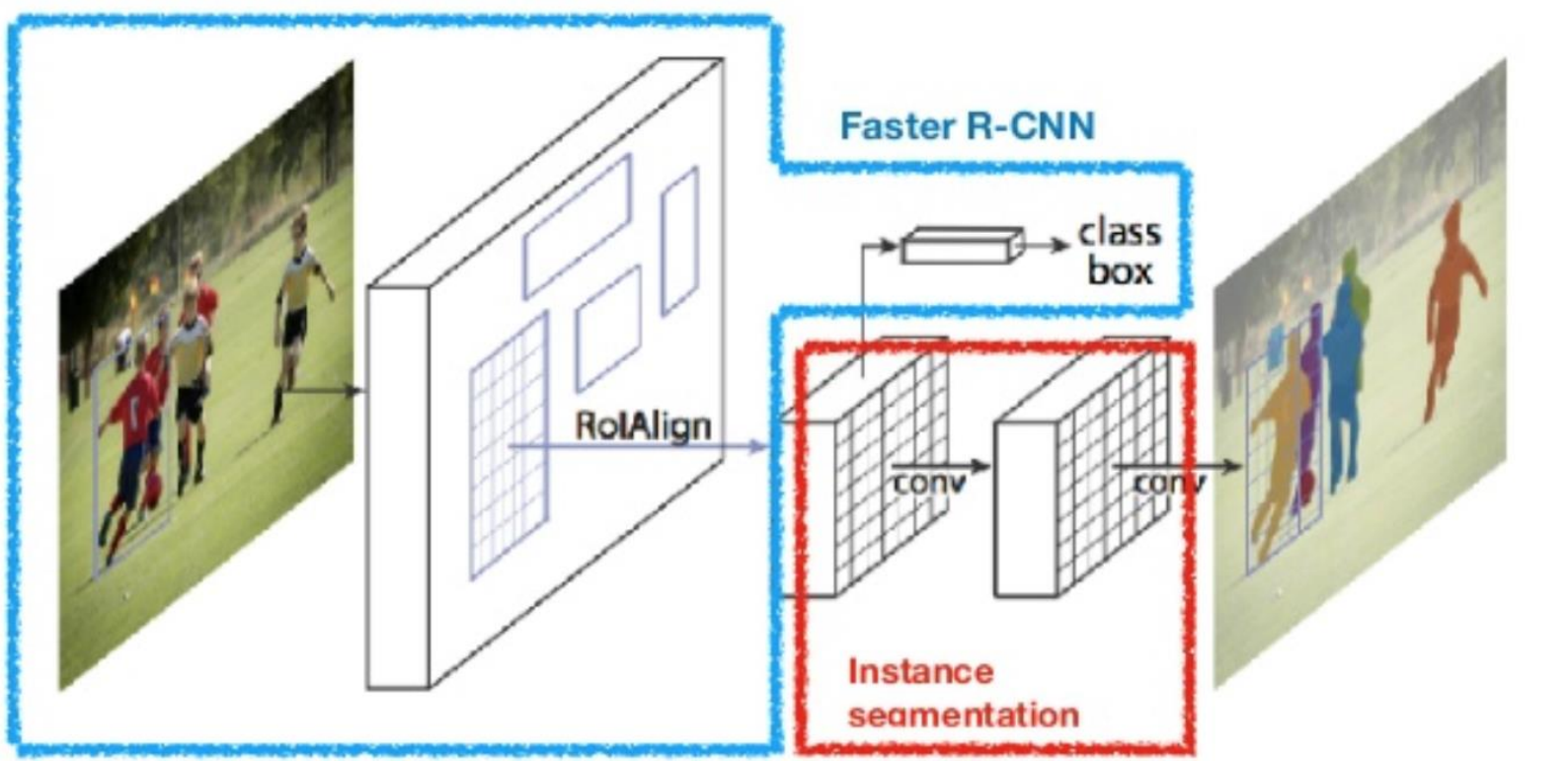
Uso de torch.hub para cargar modelos pre-entrenados

- ¿Qué es torch.hub?
  - torch.hub es una API de PyTorch que permite cargar y utilizar modelos pre-entrenados directamente desde la web.
  - Proporciona acceso a un repositorio centralizado de modelos pre-entrenados en PyTorch, incluyendo modelos populares y de referencia.
- Cómo utilizar torch.hub para cargar modelos pre-entrenados:
  - Utiliza la función torch.hub.load() para cargar un modelo pre-entrenado especificando el nombre del modelo y, opcionalmente, su versión.
  - Por ejemplo: *model = torch.hub.load('pytorch/vision', 'resnet50')* carga el modelo ResNet-50 pre-entrenado desde el repositorio pytorch/vision.

Uso de torch.hub para cargar modelos pre-entrenados

- Beneficios de utilizar torch.hub:
  - Acceso fácil a una variedad de modelos pre-entrenados, incluyendo modelos de visión, procesamiento de lenguaje natural (NLP) y más.
  - Descarga automática de los pesos pre-entrenados y los archivos necesarios para utilizar el modelo.
  - Actualizaciones frecuentes y mantenimiento por parte de la comunidad de PyTorch.
- Personalización y extensión:
  - Puedes utilizar torch.hub para cargar modelos pre-entrenados y luego personalizarlos o extenderlos según tus necesidades.
  - Esto te permite aprovechar los beneficios de los modelos pre-entrenados como punto de partida y adaptarlos a tu problema específico.
- Utilizando torch.hub, puedes acceder rápidamente a modelos pre-entrenados de alta calidad y comenzar a utilizarlos en tus proyectos de aprendizaje profundo sin necesidad de descargar y configurar manualmente los archivos correspondientes.

Uso de torch.hub para cargar modelos pre-entrenados



citic

Cargar y utilizar modelos pre-entrenados en transferencia de aprendizaje

- La transferencia de aprendizaje es una técnica poderosa en el aprendizaje profundo que utiliza modelos pre-entrenados como punto de partida para resolver tareas relacionadas.
- Utiliza torch.hub para cargar un modelo pre-entrenado relacionado con tu tarea o dominio específico.
- Congela los parámetros del modelo pre-entrenado para evitar que se modifiquen durante el entrenamiento.
- Reemplaza la capa de salida del modelo con una nueva capa adaptada al número de clases o a la tarea específica.
- Entrena el modelo con los nuevos datos, ajustando los pesos de la capa de salida mientras se mantienen los pesos pre-entrenados inalterados.

Ejercicio de transferencia de aprendizaje:

- Lab 2: 02_transfer.ipynb

Consideraciones de compatibilidad al cargar modelos

- Diferentes versiones de PyTorch pueden tener incompatibilidades al cargar modelos.
- Verifica que la arquitectura del modelo sea compatible con tu versión de PyTorch.
- Asegúrate de que la versión de CUDA coincida con la versión de PyTorch si usas GPU.
- Mantente actualizado con las últimas versiones y consulta la documentación oficial de PyTorch.
- Compatibilidad entre versiones de PyTorch, modelos y CUDA es clave para cargar y utilizar modelos sin problemas.

Recomendaciones de buenas prácticas

- Documenta la versión y configuración del modelo.
- Guarda y carga solo los parámetros necesarios.
- Verifica la integridad del modelo antes de utilizarlo.
- Mantén un registro de versiones para rastrear cambios.
- Considera el almacenamiento a largo plazo y realiza copias de seguridad.
- Estas prácticas aseguran reproducibilidad, eficiencia y confiabilidad en tus proyectos.