Carnegie Mellon University

Recitation 0.24: Saving and Loading Models

Introduction to Deep Learning
11-785 / 685 / 485

Mengchun Zhang, Nishoak Kosaraju

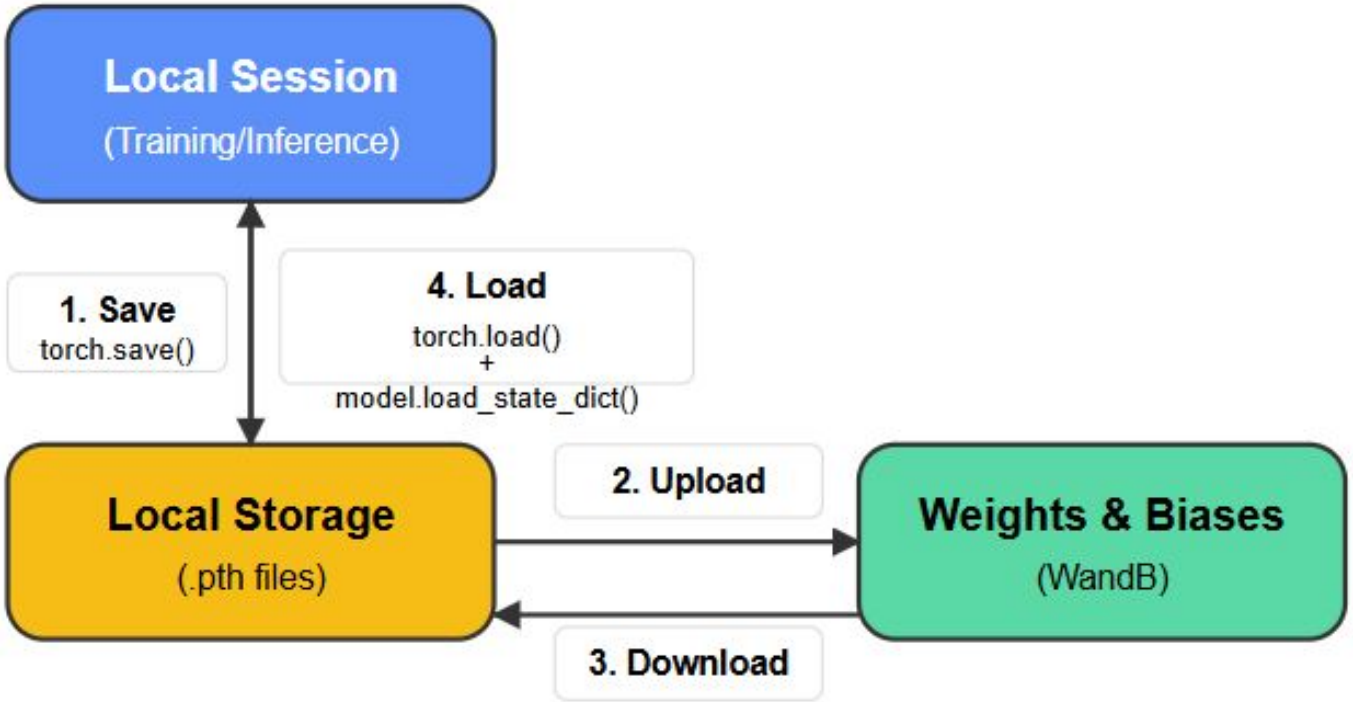
Today's Agenda

- ▶️What is Model Checkpointing?
- 🔄 Flow of a Checkpoint (Save → Store → Load)
- 💾 Saving Locally with torch.save()
- ☁️ Retrieving from Weights & Biases
- 🚀 Loading Locally & Restoring Parameters
- 🤖 Dummy Model Example

What is Checkpointing?

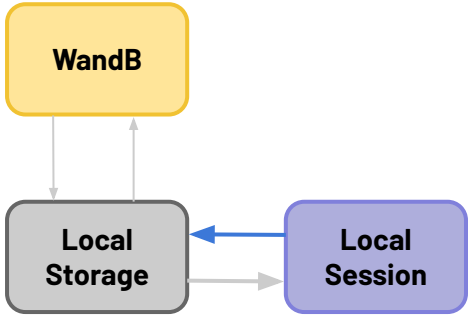
- Training models **takes time...**
- Checkpointing = saving a snapshot of your full training state at a given epoch:
  - Model weights
  - Optimizer & scheduler states
  - Current epoch & metrics
- Why Checkpoint?
  - 🚀Resume after **crashes**
  - 🔄**Rollback** to your **best performing epoch**
  - 🌱Branch experiments from any saved point (try new optimizers/schedulers)

Flow of Model Checkpointing



How to create a Checkpoint

How to Save a model (Locally)



Use torch.save (Uses Python's Pickling to serialize the data)

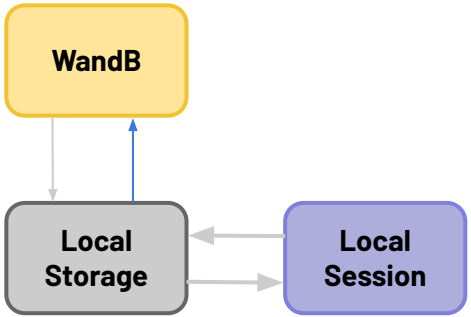
```python
checkpoint_path=f"<run_name>_{epoch}.pth"

# Saving your states locally with torch.save
torch.save({
    'model_state_dict': model.state_dict(),    # saving the model state
    # if isinstance(model, nn.DataParallel) 'model_state_dict': model.module.state_dict()
    'optimizer_state_dict': optimizer.state_dict(),    # saving the optimizer state
    'scheduler_state_dict': scheduler.state_dict(),    # saving the scheduler state
    'epoch': epoch,
    'current_loss': loss
    }, checkpoint_path
)
```

Saves a dictionary of model states and other information

Note: If using nn.DataParallel for training, use model.module.state_dict()

Saves the dictionary into a .pth file

How to Save a Model (to Wandb)



Make sure to begin a run before your training loop

```
# Before the run, you need to have started a run like so....
run = wandb.init(
    project="wandb_quickstart",
    name="<run_name>",
    )

# ...
# ...
# ...
# Within a training loop (or wherever else you want)...

# Option 1:
# create artifacts (keeps track of versioning, and is much more organized to work with between collaborators)
checkpoint_artifact = wandb.Artifact("<run_name>", type="checkpoint") # You can switch type="model if you only want to save a model"

checkpoint_artifact.add_file(checkpoint_path)

run.log_artifact(checkpoint_artifact)

# Option 2:
# directly save the model to wandb
wandb.save(checkpoint_path, base_path=os.path.dirname(checkpoint_path))
```

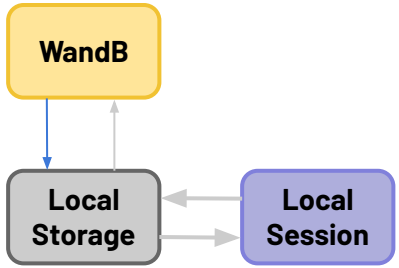
Create the artifact and add the checkpoint file

Upload the artifact to wandb

Option 2: directly save the checkpoint file

How to load a Checkpoint

How to retrieve a checkpoint from Wandb



Obtain the run object if needed

```
# METHOD 1: Download from wandb Artifact
# If you need to re-obtain the run, you can do the following....
api = wandb.Api()
# information can be obtained from the wandb link adddress as follows:
# https://wandb.ai/<USERNAME>/<PROJECT_NAME>/runs/<RUN_ID>?nw=nwuser<USERNAME>
run = api.run("<USERNAME>/<PROJECT_NAME>/<RUN_ID>")

# To retrieve the artifact....
# Get the artifact (choose which version of the model you want)
artifact = run.use_artifact('<run_name>:latest')
# Downloading the artifact
artifact_dir = artifact.download()
# Loading the model dict
checkpoint_dict = torch.load(os.path.join(artifact_dir, '<run_name>'))


# METHOD 2: Download the directly saved file from wandb to Local File
checkpoint_file = wandb.restore('<run_name>', run_path="<USERNAME>/<PROJECT_NAME>/<RUN_ID>").name
checkpoint_dict = torch.load(checkpoint_file)
```

Retrieve and download the artifact

Load the checkpoint with torch.load

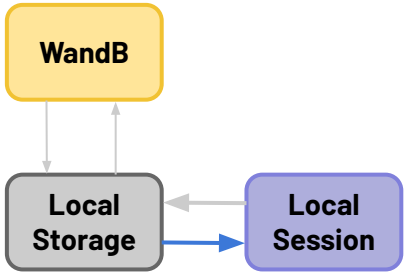
Option 2: restore the file, then load the file

How to organize a model
And load specific parts

How to load a model (locally) and load the parameters

Use torch.load on the locally-saved file if you want

```
# .pth checkpoint file path can also be obtained from a locally saved .pth file.
checkpoint_path = "<checkpoint_file_path>"
checkpoint_dict = torch.load(checkpoint_path)


# Loading model weights
model.load_state_dict(checkpoint_dict['model_state_dict'])
# Loading optimizer state
optimizer.load_state_dict(checkpoint_dict['optimizer_state_dict'])
# Loading the scheduler state
scheduler.load_state_dict(checkpoint_dict['scheduler_state_dict'])
```



Load the checkpoint's model state dict into our model.

Load optimizer and scheduler state dicts

IMPORTANT!!!!!! Your model must have the same parameter names and dimensions!!!!
Your optimizer/scheduler must be the same type of optimizer and scheduler as checkpoint's if you load them
Pytorch will simply match the keys from the checkpoint (i.e. layer/parameter names) to what it can find in your model

Our Dummy Model

Organize in submodules

```python
# A simple submodule
class DummySubmodule(nn.Module):
    def __init__(self):
        super(DummySubmodule, self).__init__()
        self.layer = nn.Linear(in_features = 32, out_features = 32)

    def forward(self, x):
        return self.layer(x)

# A simple network
class DummyNetwork(nn.Module):

    def __init__(self):
        super(DummyNetwork, self).__init__()

        self.lower_layer = nn.Sequential(
            nn.Linear(in_features = 32, out_features = 64),
            nn.ReLU(),
            nn.Linear(in_features = 32, out_features = 64),
            nn.ReLU(),
        )
        self.upper_layer = nn.Sequential(
            nn.Linear(in_features = 64, out_features = 32),
            nn.ReLU(),
        )

        self.module1 = DummySubmodule()

    def forward(self, x):
        res = self.lower_layer(x)
        res = self.upper_layer(res)
        res = self.submodule(res)
        return res
```

```python
# Print model's state_dict
print("Model's state_dict:")
for param_tensor in model.state_dict():
    print(param_tensor, "\t", model.state_dict()[param_tensor].size())
```

```
Model's state_dict:
lower_layer.0.weight     torch.Size([64, 32])
lower_layer.0.bias       torch.Size([64])
lower_layer.2.weight     torch.Size([64, 32])
lower_layer.2.bias       torch.Size([64])
upper_layer.0.weight     torch.Size([32, 64])
upper_layer.0.bias       torch.Size([32])
module1.layer.weight     torch.Size([32, 32])
module1.layer.bias       torch.Size([32])
```

Keys

Values

Organize by class names and Sequential

Loading the specific weights

Filter out the weights you want into a dictionary

```
# If you want to load specific parts of your model (in our case, we can load just the lower layers or just the upper layers)
specific_weights = { # Creates dictionary of only desired weights
    key: value
    for key, value in checkpoint_dict['model_state_dict'].items()
    if 'lower_layer' in key
}

model.load_state_dict(specific_weights, strict=False)
```

Load your weights like normal

IMPORTANT!!!!!! For the parameters you load, your model needs to have the same name and dimensions.