Checkpointing in ML Code

Dominik Viehhauser

Checkpoints

- Saving model weights during training

Checkpoints

- Saving model weights during training
- Save optimizer state if applicable (e.g. Adam)

Checkpoints

- Saving model weights during training
- Save optimizer state if applicable (e.g. Adam)
- General benefits:

Checkpoints

- Saving model weights during training
- Save optimizer state if applicable (e.g. Adam)
- General benefits:
  - Training can be resumed in case of failures

Checkpoints

- Saving model weights during training
- Save optimizer state if applicable (e.g. Adam)
- General benefits:
  - Training can be resumed in case of failures
  - Intermediate results can be compared and used (e.g. early stopping)

Checkpoints

- Saving model weights during training
- Save optimizer state if applicable (e.g. Adam)
- General benefits:
  - Training can be resumed in case of failures
  - Intermediate results can be compared and used (e.g. early stopping)
  - Be aware of storage requirements -> Avoid too frequent checkpointing

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days
  - Employ checkpointing and reduce max job time

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days
  - Employ checkpointing and reduce max job time
  - Splits a single long running job into multiple shorter ones with dependencies

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days
  - Employ checkpointing and reduce max job time
  - Splits a single long running job into multiple shorter ones with dependencies
  - Improves scheduling behavior of SLURM

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days
  - Employ checkpointing and reduce max job time
  - Splits a single long running job into multiple shorter ones with dependencies
  - Improves scheduling behavior of SLURM
    - Faster response time for updates/maintenance on the nodes

Benefits on the Cluster

- Increased resistance to failures: hardware, software, user error
- Runtime limit of most jobs: up to 5 days
  - Employ checkpointing and reduce max job time
  - Splits a single long running job into multiple shorter ones with dependencies
  - Improves scheduling behavior of SLURM
    - Faster response time for updates/maintenance on the nodes
    - Improve fairness of job scheduling

Checkpointing in PyTorch

Checkpointing in PyTorch

- PyTorch support saving and restoring model weights

```python
import torch

model = ...

torch.save(model.state_dict(), filename)

model.load_state_dict(torch.load(filename))
```

Checkpointing in PyTorch

- PyTorch support saving and restoring model weights
- Multiple approaches possible

```python
import torch

model = ...

torch.save(model.state_dict(), filename)

model.load_state_dict(torch.load(filename))
```

Checkpointing in PyTorch

- Store information about current epoch in filename

```python
import torch

model = ...

if start_epoch > 0:
    resume_epoch = start_epoch - 1
    model.load_state_dict(torch.load("epoch-{resume_epoch}.pth"))

for epoch in range(start_epoch, n_epochs):
    ...
    torch.save(model.state_dict(), "epoch-{epoch}.pth")
```

Checkpointing in PyTorch

- Store information about current epoch in filename
- Also, possible to store multiple/all checkpoints in a single file

```python
import torch

model = ...

if start_epoch > 0:
    resume_epoch = start_epoch - 1
    model.load_state_dict(torch.load("epoch-{resume_epoch}.pth"))

for epoch in range(start_epoch, n_epochs):
    ...
    torch.save(model.state_dict(), "epoch-{epoch}.pth")
```

Checkpointing in PyTorch

- Store information about current epoch in filename
- Also, possible to store multiple/all checkpoints in a single file
  - Easier automatic restart of last checkpoint

```python
import torch

model = ...

if start_epoch > 0:
    resume_epoch = start_epoch - 1
    model.load_state_dict(torch.load("epoch-{resume_epoch}.pth"))

for epoch in range(start_epoch, n_epochs):
    ...
    torch.save(model.state_dict(), "epoch-{epoch}.pth")
```

Checkpointing in PyTorch

- Store information about current epoch in filename
- Also, possible to store multiple/all checkpoints in a single file
  - Easier automatic restart of last checkpoint
- Including a start epoch allows resuming training from a specific checkpoint

```python
import torch

model = ...

if start_epoch > 0:
    resume_epoch = start_epoch - 1
    model.load_state_dict(torch.load("epoch-{resume_epoch}.pth"))

for epoch in range(start_epoch, n_epochs):
    ...
    torch.save(model.state_dict(), "epoch-{epoch}.pth")
```

Checkpointing in PyTorch

- Some optimizer also have states, e.g. Adam

```python
import torch

model = ...
optimizer = ...

if start_epoch > 0:
    resume_epoch = start_epoch - 1
    model.load_state_dict(torch.load("epoch-{resume_epoch}.pth")['model'])
    optimizer.load_state_dict(torch.load("epoch-{resume_epoch}.pth")['optimizer'])

for epoch in range(start_epoch, n_epochs):
    ...
    torch.save({
        'optimizer': optimizer.state_dict(),
        'model': model.state_dict(),
    }, "epoch-{epoch}.pth")
```

Links and sources

- Checkpointing in PyTorch `https://pytorch.org/tutorials/beginner/saving_loading_models.html`
- Checkpointing in Tensorflow `https://www.tensorflow.org/guide/checkpoint`

Thank you for your attention!