Deep Learning & AI News #6

Interesting Things Related to Deep Learning

Mar 6th, 2021
PyTorch 1.8 Release, including Compiler and Distributed Training updates, and New Mobile Tutorials

https://pytorch.org/blog/pytorch-1.8-released/

https://github.com/pytorch/pytorch/releases/tag/v1.8.0

• AMD GPU support via ROCm (binaries available directly from the installer menu)

• Now possible to fit large models onto GPUs w/o external libraries: pipeline and model parallelism

• Determinants & eigenvalues via torch.linalg w/o switching to NumPy
PyTorch adds binaries for AMD GPU support

<table>
<thead>
<tr>
<th>PyTorch Build</th>
<th>Stable (1.8.0)</th>
<th>Preview (Nightly)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your OS</td>
<td>Linux</td>
<td>Mac</td>
</tr>
<tr>
<td>Package</td>
<td>Conda</td>
<td>Pip</td>
</tr>
<tr>
<td>Language</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Compute Platform</td>
<td>CUDA 10.2</td>
<td>CUDA 11.1</td>
</tr>
<tr>
<td>Run this Command:</td>
<td></td>
<td>pip install torch -f <a href="https://download.pytorch.org/whl/rocm4.0.1/torch_stable.html">https://download.pytorch.org/whl/rocm4.0.1/torch_stable.html</a></td>
</tr>
</tbody>
</table>

https://pytorch.org  (Currently only for Linux though)
The following packages will be downloaded:

<table>
<thead>
<tr>
<th>package</th>
<th>build</th>
</tr>
</thead>
<tbody>
<tr>
<td>cudatoolkit-11.1</td>
<td>h6406543_8</td>
</tr>
<tr>
<td>pytorch-1.8.0</td>
<td>py3.8_cuda11.1_cudnn8.0.5.0</td>
</tr>
<tr>
<td>torchaudio-0.8.0</td>
<td>py38</td>
</tr>
</tbody>
</table>

Total: 2.48 GB

The following packages will be UPDATED:

```
cudatoolkit          11.0.3-h15472ef_8    --> 11.1.1-h6406543_8
pytorch               1.7.1-py3.8_cuda11.0.221_cudnn8.0.5.0    --> 1.8.0-py3.8_cuda11.1_cudnn8.0.5.0
torchaudio           0.7.2-py38    --> 0.8.0-py38
```

[Proceed ([y]/n)? y

<table>
<thead>
<tr>
<th>Downloading and Extracting Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>pytorch-1.8.0</td>
</tr>
</tbody>
</table>
Distributed Training

- [Beta] Pipeline Parallelism
- [Beta] DDP Communication Hook
- ...
- (Prototype) ZeroRedundancyOptimizer
- (Prototype) CUDA-support in RPC using TensorPipe
Model Parallelism using multiple GPUs

Typically for large models which don’t fit on a single GPU, model parallelism is employed where certain parts of the model are placed on different GPUs. Although, if this is done naively for sequential models, the training process suffers from GPU under utilization since only one GPU is active at one time as shown in the figure below:

The figure represents a model with 4 layers placed on 4 different GPUs (vertical axis). The horizontal axis represents training this model through time demonstrating that only 1 GPU is utilized at a time (image source).

Pipelined Execution

To alleviate this problem, pipeline parallelism splits the input minibatch into multiple microbatches and pipelines the execution of these microbatches across multiple GPUs. This is outlined in the figure below:

The figure represents a model with 4 layers placed on 4 different GPUs (vertical axis). The horizontal axis represents training this model through time demonstrating that the GPUs are utilized much more efficiently. However, there still exists a bubble (as demonstrated in the figure) where certain GPUs are not utilized. (image source).
3) VGG16 with Pipeline Parallelism

Below we first define the blocks we are going to wrap into the model:

```python
block_1 = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=3,
        out_channels=64,
        kernel_size=(3, 3),
        stride=(1, 1),
        padding=1),
    torch.nn.ReLU(),
    torch.nn.Conv2d(in_channels=64,
        out_channels=64,
        kernel_size=(3, 3),
        stride=(1, 1),
        padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=(2, 2),
        stride=(2, 2))
)

block_2 = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=64,
        out_channels=128,
        kernel_size=(3, 3),
        stride=(1, 1),
        padding=1),
    torch.nn.ReLU(),
    torch.nn.Conv2d(in_channels=128,
        out_channels=128,
        kernel_size=(3, 3),
        stride=(1, 1),
        padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=(2, 2),
        stride=(2, 2))
)
```

2. The chunks refer to the `microbatches`, for more details, see [https://pytorch.org/docs/1.8.0/pipeline.html?highlight=microbatches](https://pytorch.org/docs/1.8.0/pipeline.html?highlight=microbatches)

```python
from torch.distributed.pipeline.sync import Pipe

block1 = block_1.cuda(0)
block2 = block_2.cuda(0)
block3 = block_3.cuda(2)
block4 = block_4.cuda(2)
block4 = block_5.cuda(3)
block4 = classifier.cuda(0)

model_parallel = torch.nn.Sequential(
    block_1, block_2, block_3, block_4, block_5, classifier)
model_parallel = Pipe(model_parallel, chunks=8)
optimizer = torch.optim.Adam(model_parallel.parameters(), lr=learning_rate)
```
for epoch in range(num_epochs):
    model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):
        features = features.to(device)
        targets = targets.to(device)

        # FORWARD AND BACK PROP
        logits = model(features)
        if isinstance(logits, torch.distributed.rpc.api.RRef):
            logits = logits.local_value()
        loss = loss_fn(logits, targets)
        optimizer.zero_grad()
        loss.backward()

        # UPDATE MODEL PARAMETERS
        optimizer.step()

        # LOGGING
        log_dict['train_loss_per_batch'].append(loss.item())

        if not batch_idx % logging_interval:
            print('Epoch: %03d/%03d | Batch %04d/%04d | Loss: %.4f' % (epoch+1, num_epochs, batch_idx, len(train_loader), loss))

        if not skip_epoch_stats:
            model.eval()
The work generalizes what other malware investigators have demonstrated as promising convolutional neural networks originally developed to solve image problems but applied to a new abstract domain in pixel bytes from executable files.

We converted the CSV format [16] to greyscale images using the intermediate NetPBM text format (PGM) to create ASCII-raw images, then the ImageMagick [25] command-line tools for compressing the image to viewable JPEG files.

Figure 1 Virus-MNIST showing 10 classes. The “0” class represents non-malicious examples. The other 9 virus families were clustered using a K-means method to match with the standard MNIST format and multi-class solutions.
Simple considerations for simple people building fancy neural networks


Victor Sanh

https://huggingface.co/blog/simple-considerations

1) Put aside machine learning and simply focus on your data:

• Are the labels balanced?
• Are there gold-labels that you do not agree with?
• How were the data obtained? What are the possible sources of noise in this process?
• Are there any preprocessing steps that seem natural (tokenization, URL or hashtag removing, etc.)?
• How diverse are the examples?
• What rule-based algorithm would perform decently on this problem?
Simple considerations for simple people building fancy neural networks


Victor Sanh
https://huggingface.co/blog/simple-considerations

2) Start as simple as possible to get a sense of the difficulty of your task and how well standard baselines would perform (e.g., use a logistic regression baseline)

- How would a random predictor perform (especially in classification problems)? Dataset can be unbalanced...
- What would the loss look like for a random predictor?
- What is (are) the best metric(s) to measure progress on my task?
- What are the limits of this metric? If it’s perfect, what can I conclude? What can’t I conclude?
- What is missing in “simple approaches” to reach a perfect score?
- Are there architectures in my neural network toolbox that would be good to model the inductive bias of the data?
3) **Model debugging**

Try to overfit a small batch of examples (16 for instance) and get 0-loss. If not possible, there may be a bug.

- You forgot to call `model.eval()` in evaluation mode (in PyTorch) or `model.zero_grad()` to clean the gradients
- Something went wrong in the pre-processing of the inputs
- The loss got wrong arguments (for instance passing probabilities when it expects logits)
- Initialization doesn’t break the symmetry (usually happens when you initialize a whole matrix with a single constant value)
- Some parameters are never called during the forward pass (and thus receive no gradients)
- The learning rate is taking funky values like 0 all the time
- Your inputs are being truncated in a suboptimal way

**Plot loss curves**

![Loss Curves](image1)

![Accuracy Curves](image2)
4. 💡 Tune but don’t tune blindly

I generally stick with a random grid search as it turns out to be fairly effective in practice. Some people report successes using fancy hyperparameter tuning methods such as Bayesian optimization but in my experience, random over a reasonably manually defined grid search is still a tough-to-beat baseline.

Compare a couple of runs with different hyperparameters to get an idea of which hyperparameters have the highest impact.

Favor (as most as possible) a deep understanding of each component of your neural network instead of blindly (not to say magically) tweak the architecture.

https://huggingface.co/blog/simple-considerations
Generative Adversarial Transformers

Drew A. Hudson, C. Lawrence Zitnick

https://arxiv.org/abs/2103.01209
https://github.com/dorarad/gansformer

![Image of sample images generated by the GANsformer, along with a visualization of the model attention maps.](image)

**Figure 1.** Sample images generated by the GANsformer, along with a visualization of the model attention maps.

<table>
<thead>
<tr>
<th>Model</th>
<th>CLEVR</th>
<th>LSUN-Bedroom</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FID ↓</td>
<td>IS ↑</td>
</tr>
<tr>
<td>GAN</td>
<td>25.0244</td>
<td>2.1719</td>
</tr>
<tr>
<td>k-GAN</td>
<td>28.2900</td>
<td>2.2097</td>
</tr>
<tr>
<td>SAGAN</td>
<td>26.0433</td>
<td>2.1742</td>
</tr>
<tr>
<td>StyleGAN2</td>
<td>16.0534</td>
<td>2.1472</td>
</tr>
<tr>
<td>VQGAN</td>
<td>32.6031</td>
<td>2.0324</td>
</tr>
<tr>
<td>GANsformer₅</td>
<td>10.2585</td>
<td>2.4555</td>
</tr>
<tr>
<td>GANsformer₇</td>
<td>9.1679</td>
<td>2.3654</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>FFHQ</th>
<th>Cityscapes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FID ↓</td>
<td>IS ↑</td>
</tr>
<tr>
<td>GAN</td>
<td>13.1844</td>
<td>4.2966</td>
</tr>
<tr>
<td>k-GAN</td>
<td>61.1426</td>
<td>3.9990</td>
</tr>
<tr>
<td>SAGAN</td>
<td>16.2069</td>
<td>4.2570</td>
</tr>
<tr>
<td>StyleGAN2</td>
<td>10.8309</td>
<td>4.3294</td>
</tr>
<tr>
<td>VQGAN</td>
<td>63.1165</td>
<td>2.2306</td>
</tr>
<tr>
<td>GANsformer₅</td>
<td>13.2861</td>
<td>4.4591</td>
</tr>
<tr>
<td>GANsformer₇</td>
<td>12.8478</td>
<td>4.4079</td>
</tr>
</tbody>
</table>

[Submitted on 1 Mar 2021 (v1), last revised 2 Mar 2021 (this version, v2)]