Machine Learning Overview: CPU vs GPU vs Multi-GPUs Performance

Sonal Rami

23/11/2023

Machine learning tools

Software tools and libraries	Hardware
• TensorFlow • Keras • Scikit-learn • Pandas • NumPy • Jupyter Notebooks • Python	**CPUs**: Essential for general-purpose tasks **GPUs**: Provide significant speedup for matrix calculations and ML computations, and deep learning tasks **TPUs** (Tensor Processing Units)

- Single GPU can significantly speed up machine learning tasks compared to CPU
- Multi-GPU provides enhanced capabilities for handling very large datasets and complex models

TensorFlow for Single GPU

1. **Install Prerequisites**:
   Ensure you have a compatible NVIDIA GPU
   Install NVIDIA drivers, CUDA Toolkit, and cuDNN
   (https://www.tensorflow.org/install/source#gpu)

2. **Install TensorFlow with GPU Support**:
   Use pip to install TensorFlow GPU version: pip install tensorflow-gpu
   (https://www.tensorflow.org/install/pip#macos)

3. **Verify GPU Access**:

```
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

TensorFlow for Multi-GPU

1. **Hardware Setup**:

   Ensure the machine has multiple NVIDIA GPUs installed (check nvidia-smi)

2. **Software Setup**:

   Similar to the single GPU setup, ensure all drivers, CUDA, and cuDNN are installed, and then install TensorFlow GPU.

3. **Enable Multi-GPU Support**:

   Use tf.distribute.Strategy, TensorFlow’s standard way to distribute computations across multiple devices.

4. **MirroredStrategy for Multi-GPU**:

   ```
   strategy = tf.distribute.MirroredStrategy()
   ```

5. **Wrap Model Building and Compilation**:

   ```
   with strategy.scope():
       model = ...……….# Build your model here
       model.compile(...)
   ```

6. **Batch Size Adjustment**:

   Increase batch size accordingly

7. **Train the Model**:

   Train your model as usual. TensorFlow handles the distribution of computations.

Tips for Multi-GPU Training

- Data Loading: Make sure the data/ batch_size is efficient enough to supply to all GPUs
- Performance Tuning: Monitor the usage of GPUs (using nvidia-smi) to ensure balanced utilization
- Debugging: Start with a single GPU with shallow model
- while single GPU setups in TensorFlow are relatively straightforward

Here are the steps for running Jupyter Notebook and a Python script on a remote GPU server:

Generate SSH Key Pair: - ssh-keygen -b 2048 -t rsa

- Choose a path to save your SSH key pair and set a password when prompted.
- Install the private SSH key on your local computer

Connecting to the GPU Server and Running Jupyter Notebook:

1. SSH into the Remote Server (Local):

```
- ssh -i path/to/your/private/key user@remote_server_ip
```

2. Create a Screen Session (Remote):

```
- screen -S sessionname
```

3. Set GPU Visibility (Remote):

```
- export CUDA_VISIBLE_DEVICES=2,3
```

4. Start Jupyter Notebook (Remote):

```
- jupyter notebook --no-browser --port 1234
```

5. Detach from the Screen Session (Remote): - Press Ctrl + a followed by d to detach from the screen session

6. Reattach to the Screen Session (Local):

```
- screen -S sessionname
```

7. Set Up Port Forwarding (Local):

```
- ssh -NL 1234:localhost:1234 user@remote_server_ip
```

8. Detach from the Screen Session (Local):

- Press Ctrl + a followed by d to detach from the screen session

Thank you!!!

Questions?

sonal.rami@awi.de