Part 1: Compression Benchmark

Input: Dumped EMB data
Output: Compression Ratio, De/Compression Throughput

Step 1. Do Quantization

To simulate lossy compression, please first apply quantization. Run `python quantization.py EMB_file_path decay_stage`, to generate quantization code of inputs. Modify `EMB_file_path` as the embedding data directory. Modify `decay_stage` as the decay stop point. Function `build_error_bound()` define the build strategy of table-wise and iteration-wise adjustment, `step-wise decay` by default.

Change `intType` variable to choose use `int8` or `int16` as quantization code datatype.

Step 1.1(Optional) Data Padding

To simulate compression on larger batch_size, run `python padding_files.py` to padding EMB data with its own copy. Modify `copy_times` as multiple of original data.

Step 2. Choose Lossless Encoder

There are four encoder we apply, LZ4 encoder, ANS encoder, GPULZ encoder, and Huffman encoder. Run `python lossless_encoder.py` to generate compression ratio, compression throughput and decompression throughput. For more details about executable binary usage, please refer to documents of above repos and `EXECUTABLES` variable in `lossless_encoder.py`. Before

To install and use LZ4 and ANS encoder, refer to nvcomp.

Usage example `benchmark_ans_chunked -f {filename}`.

To install GPULZ encoder, refer to gpulz.

Usage example `gpulz -i {fliename}`.

To install Huffman encoder, refer to cusz. After compiling, the execution binary `bin_hf` is under `example` folder.

Usage example `bin_hf {filename} x y z booklen`.

Step 3. Parse Log

To extract the log file, run `python huffman_parser.py`, `python nvcomp_parser.py`, and `python gpulz_parser.py` for different lossless encoders' log. These script will read log file as input and print compression ratio, compression throughput, and decompression throughput. Modify `file_path` as input log path.

Step 4. Speed-up Calculation

To calculate the speed-up, run `python speedup_calculation.py logFilePath bandwidth` to calculate the speed-up of lossless encoders. NOTE: `bandwidth` should consider the time cost of quantization and dequantization.

Step 5. Visualization

TBD.

Part 2: Compression in Training

Method 1: Compression Simulation

You can use a quantization function to mimic the lossy compression.

```
def quantize_and_dequantize_data(x, eb):
    # Ensure calculations are performed on the same device as x (GPU in
this case)
    # eb = torch.tensor(eb, device=x.device)
    # Quantization
    eb = eb * (x.max()-x.min()).item()
    x.data = (x.data / (2*eb)).round() * (2*eb)  # Directly modify the
tensor data
```

Usage example:

Single GPU

```
ly = self.apply_emb(lS_o, lS_i, self.emb_l, self.v_W_l)
decay_func = str(os.environ.get("DECAY_FUNC", "linear"))
eb_conf = stage_check(iter, decay_func)
if enable_compress and not is_test:
    # do quantization and dequantization to simulate compression
    for i in range(len(ly)):
        quantize_and_dequantize_data(ly[i], eb_conf * error_bound[i])
```

Multiple GPU

```
for q in range(ext_dist.my_size):
    start_idx = q * quarter_length
    end_idx = (q + 1) * quarter_length if (q < ext_dist.my_size) else
len(ly[i])
    # Directly pass the slice of tensor
    quantize_and_dequantize_data(ly[i][start_idx:end_idx], eb_conf *
error_bound[i])
```

Method 2: Python binding

You can also use a compressor's python binding to compressed the data. For example, SZ's binding usage.

```
from pysz import SZ

def sz_comp_decomp(data, r_eb, data_shape, data_type):
    lib_extention = "so" if platform.system() == 'Linux' else "dylib"
    sz_path = os.environ.get("SZ_PATH",
"/N/u/haofeng/BigRed200/SZ3_build/lib64/")
    sz = SZ(sz_path+"libSZ3c.{}".format(lib_extention))
    a_eb = (data.max()-data.min()) * r_eb
    data_cmpr, data_ratio = sz.compress(data, 0, a_eb, 0, 0)
    data_dcmp = sz.decompress(data_cmpr, data_shape, data_type)
    return (data_dcmp.astype(data_type), data_ratio)

for i in range(len(ly)):
    new_ly, ly_i_ratio = sz_comp_decomp(data=ly_data[i], r_eb=eb_conf *
error_bound[i], data_shape=ly_data[i].shape, data_type=np.float32)
    ly[i].data = torch.from_numpy(new_ly).data.to(ly_devices[i])
    record_cr(ly_i_ratio, i)
```