Repository Overview: Protein Structure to Image and Hierarchical VQ-VAE Training

Repository Overview

This repository contains two major parts:

1. **Protein Structure to Image**
   A Python script (`pdb_to_png_distogram.py`) for parsing protein structures (`*.pdb` or `*.cif`) and converting them into PNG "distogram" images (plus associated FASTA files). These images capture backbone distances and angular features.
2. **Hierarchical VQ-VAE Training**
   A hierarchical VQ-VAE model implemented with PyTorch and Transformer blocks (`model.py`, `train.py`, and `dataloader.py`). This model is designed to learn quantized latent representations of the generated protein images.

Below is an overview of the file structure and detailed explanations of both the implementation and the underlying mathematical concepts.

1 Protein Structure to Distogram Image

File: `pdb_to_png_distogram.py`

Overview

- Reads protein structures (`*.pdb` or `*.cif`) using Biotite (or a related library).
- Extracts backbone atoms **N**, **CA**, and **C** (and estimates **CB**).
- Computes distances, dihedral angles, and planar angles between residues, forming a 6D representation.
- Normalizes and merges these channels into a 3-channel image (RGB). Each pixel (i, j) corresponds to the backbone geometry between residue i and residue j.
- Saves the result as a PNG image and also exports the protein sequence in a FASTA file.
- Bins the resulting images according to the protein length (number of residues) to keep image sizes manageable.
- Uses multiprocessing to parallelize processing and automatically deletes original `*.pdb` or `*.cif` files after successful conversion.

Key Sections

Configuration & Mappings

- A dictionary `non_standard_to_standard` is used to map nonstandard residues to standard residue names.
- Another dictionary `three_to_one_letter` maps three-letter codes (e.g., `VAL`) to one-letter codes (e.g., `V`).
- The code ensures multi-chain or multi-model structures are skipped to reduce complexity.

Backbone Extraction & 6D Representation

2.1 Coordinate and Geometry Extraction For each residue, we collect coordinates of the backbone atoms: `N`, `CA`, `C`, and we also estimate `CB`.

Approximating C_β:

The script uses a geometric approach to approximate the position of C_β if it is not explicitly present. The approximate formula is

$$C_\beta \approx -0.5827\,\vec{a} \;+\; 0.5680\,\vec{b} \;-\; 0.5407\,\vec{c} \;+\; C_\alpha,$$

where

$$\vec{b} = C_\alpha - N, \quad \vec{c} = C - C_\alpha, \quad \vec{a} = \vec{b} \times \vec{c}.$$

This is a known approximation in structural biology for generating a placeholder side-chain direction.

2.2 Distance and Angles Each pair of residues (i, j) is described by:

- **Distance between C_{β_i} and C_{β_j}:**

$$d_{ij} = \| C_{\beta_j} - C_{\beta_i} \|.$$

- **Dihedral angles ω and θ:**

  Using the standard 4-point dihedral formula (with vectors N, CA, C, C_β). If we define `get_dihedrals(a, b, c, d)`, then we compute:

$$\begin{aligned}
b_0 &= -(b - a), \\
b_1 &= c - b, \quad \text{(normalized so that } b_1 \leftarrow \frac{b_1}{\|b_1\|}\text{)}, \\
b_2 &= d - c, \\
v &= b_0 - (b_0 \cdot b_1)\, b_1, \\
w &= b_2 - (b_2 \cdot b_1)\, b_1, \\
x &= v \cdot w, \\
y &= (b_1 \times v) \cdot w, \\
\text{dihedral} &= \arctan 2\,(y,\, x).
\end{aligned}$$

- **Planar angle ϕ:**

  This angle is computed using three points $(C_{\alpha_i},\ C_{\beta_i},\ C_{\beta_j})$ to measure the angle $\angle_{i_{Cb}, i_{Ca}, j_{Cb}}$.

These angles and distances are stored in separate 2D arrays of shape (L, L) if the protein has L residues. The script normalizes these values (e.g., dividing distances by a maximum range and scaling angles by π) and merges some channels to limit the final image shape.

Normalization & Image Generation

- **Distances** are scaled to the range $[-1, 1]$ based on a maximum value $d_{\max}$ (e.g., 80.0 Å).
- **Angles** are scaled or offset so that, for example, $\omega \in [-\pi, \pi]$ becomes ω/π.
- The channels are merged to form a $3 \times L \times L$ array, which is saved as a PNG using the `PIL.Image` module.

FASTA Output

For each structure, a corresponding FASTA file is generated to store the sequence of the protein for reference.

Binning by Residue Count

- Proteins are grouped into directories such as `bin_40_64/` or `bin_65_128/` to limit the size of the 2D images.
- For example, a 65-residue protein leads to a $3 \times 65 \times 65$ image that is stored in `bin_65_128/`.

Parallel Processing & Deletion

- The script uses `multiprocessing.Pool` for parallelism.
- After successfully generating the PNG and FASTA files, the original `*.pdb` or `*.cif` file is deleted to free up space.

2 Dataloader & Infinite Streaming

File: `dataloader.py`

Purpose

- Provides **Iterable Datasets** that stream large volumes of images without loading them all at once.
- Uses chunked or streaming shuffling to feed data in random order.
- Handles train/validation splits in a **deterministic** way based on the MD5 hash of the file path.
- Supports multi-processing and Distributed Data Parallel (DDP) to shard data among workers and ranks.

Key Mechanics

IterableImageDataset

- Walks through a single "bin" folder of PNG images.
- Maintains a shuffle buffer to randomize the order.
- Applies standard PyTorch transforms (e.g., `T.ToTensor()`, normalization).
- Partitions data into "train" or "val" subsets by hashing each filename and comparing the hash value against a specified `valid_frac`.

IterableImageDatasetUni

- Similar to `IterableImageDataset` but loops over multiple bin folders (e.g., `bin_40_64/`, `bin_65_128/`, etc.).
- Optionally skips certain bins if desired.

Batch Collation & Padding

- The function `collate_with_padding` ensures each batch is zero-padded to a dimension that is a multiple of a defined `patch_size` so that the subsequent model can operate on uniformly sized inputs.
- This is especially important when training Transformers that assume consistent input dimensions.

Infinite Loader

- The function `cycle(dl)` yields data from a DataLoader in an endless loop.
- This mechanism is useful for large-scale or indefinite training cycles.

3 Hierarchical VQ-VAE & Transformer Blocks

Files:

- `model.py` — Contains the core hierarchical VQ-VAE, Transformer blocks, quantization modules, etc.
- `train.py` — The training script orchestrating gradient updates, logging, checkpointing, and other training logistics.

3.1 Transformer Blocks & Rotary Embedding

Multi-Head Attention with Rotary Embedding

- The code uses standard multi-head self-attention, partitioning the embedding dimension `n_embd` into `n_head` heads.
- **Queries** Q, **Keys** K, and **Values** V each have a dimension given by

$$\text{latent\_dim} = n_{\text{head}} \times \left(\frac{\text{latent\_dim}}{n_{\text{head}}} \right).$$

- The attention scores are computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^\top}{\sqrt{d_{\text{head}}}}\right)V,$$

where

$$d_{\text{head}} = \frac{\text{latent\_dim}}{n_{\text{head}}}.$$

- **Rotary Embedding:**
The module `rotary_embedding_torch` is used to incorporate relative positional information by rotating the Q and K vectors in a manner that depends on position indices. This provides a more continuous version of positional encoding capable of handling 2D sequences elegantly.

MLP Feedforward

- A two-layer feedforward block with hidden dimension `mlp_hidden_dim` (typically $2 \times n_{\text{embd}}$), utilizing a GELU activation function.

Residual & LayerNorm

- Standard Transformer architecture with two residual connections per block:
  - One residual branch around the multi-head self-attention sub-layer.
  - One residual branch around the MLP sub-layer.
- Each residual branch is preceded by Layer Normalization.

3.2 Vision Transformer Encoders / Decoders

Rather than splitting images into fixed-size patches (as in a traditional Vision Transformer), the code uses convolution-based downsampling (or upsampling) to reduce (or restore) resolution by factors of 2 multiple times.

VitEncoder

- Implements a **DownsampleStack** that repeatedly halves the resolution using `Conv2d` with `stride=2`. For example, if `downscale_factor` is 2, the convolution is applied once; if it is 4, it may be applied twice, etc.
- The resulting feature map is flattened into a shape (B, $H \times W$, C), where B is the batch size.
- The flattened features are processed through several Transformer Blocks (self-attention).
- Finally, the sequence is reshaped back to (B, C, H, W).

VitDecoder

- Takes features from the codebook (or a concatenation of multiple code levels).
- Applies a 1×1 convolution to project the features to the base dimension `n_embd`.
- Processes the projected features through several Transformer Blocks.

- Uses an **UpsampleStack** with `ConvTranspose2d` (with `stride=2`) repeatedly to upscale the feature maps until the original resolution is reached.

3.3 Vector Quantization (VQ) and Codebook

The `Quantize` module uses a codebook

$$\mathbf{E} \in \mathbb{R}^{\text{codebook\_dim} \times \text{codebook\_size}}$$

to quantize the feature maps. The main idea is as follows:

1. **Projection:**
   Project the input tensor of shape (B, in_channels, H, W) to a tensor of shape (B, codebook_dim, H, W).
2. **Flattening:**
   Flatten each spatial location so that the tensor becomes a collection of vectors.
3. **Nearest Code Selection:**
   Compute the squared Euclidean distance from each vector to every codebook entry and use `argmin` (or equivalently, `argmax` of the negative distance) to select the nearest code.
4. **Quantization:**
   Replace each feature vector with its corresponding nearest codebook embedding.
5. **EMA Updates:**
   Update the codebook entries using an Exponential Moving Average (EMA) to keep them stable (refer to Oord et al., *Neural Discrete Representation Learning*).
6. **Commitment Loss:**
   Enforce that the encoder output remains close to the discrete codes by computing the loss:

   $$L_{\text{commit}} = \left\| \text{sg}[z_q] - z_e \right\|^2,$$

   where $\text{sg}(\cdot)$ is the stop-gradient operator, z_q is the quantized (nearest) code, and z_e is the continuous encoder output.

3.4 Hierarchical Approach

- The model is structured into multiple levels $\ell = 0, 1, 2, \ldots$, with each level compressing the image at a successively coarser scale.
- The top-level code has the smallest spatial resolution (e.g., $\frac{1}{8}$ or $\frac{1}{16}$ of the original image).
- During decoding, the top-level code is first decoded into a coarse representation. The next level then uses a combination of the corresponding encoder output at that level **plus** the upsampled coarse-level decoded output, followed by codebook quantization. This process is repeated until the finest scale is reached.

3.5 Loss Function

Reconstruction Loss

$$L_{\text{recon}} = \|\hat{x} - x\|^2,$$

where x is the original image and $\hat{x}$ is the final decoded (reconstructed) image.

VQ Commitment Loss

The commitment losses from all levels are summed:

$$L_{\text{vq}} = \sum_{\ell} \left\| \text{sg}[q_\ell] - e_\ell \right\|^2.$$

Total Loss

$$L = L_{\text{recon}} + \beta\, L_{\text{vq}},$$

where β is a hyperparameter (default value 0.25).

3.6 Training Script (train.py)

- **DistributedDataParallel (DDP):**
  The training script uses DDP for multi-GPU scaling.
- **Mixed Precision:**
  Implemented with `autocast` (using `float16` or `bfloat16`) to improve training efficiency.
- **WandB Logging:**
  Used for logging training curves and saving reconstruction sample images.
- **Checkpointing:**
  The script saves the model's `state_dict`, optimizer state, and iteration count to facilitate resuming training.

4 Back-Conversion from Distogram Images to 3D Coordinates

The generated 2D distogram images are not arbitrary pictures but precise representations of the protein structure backbone. Each pixel (i, j) in a distogram image encodes the geometric relationship (e.g., the distance between C_{β_i} and C_{β_j}) derived from the protein backbone. This representation is robust enough to allow one to reconstruct the original 3D coordinates by inverting the process using classical Multidimensional Scaling (MDS).

The Forward Mapping: Backbone to Distogram

- **Backbone Extraction:**
  The backbone atoms N, C_α, and C are extracted and the position of C_β is approximated using

  $$C_\beta \approx -0.5827\,\vec{a} \;+\; 0.5680\,\vec{b} \;-\; 0.5407\,\vec{c} \;+\; C_\alpha,$$

  with $\vec{b} = C_\alpha - N$, $\vec{c} = C - C_\alpha$ and $\vec{a} = \vec{b} \times \vec{c}$.
- **Pairwise Distance Calculation:**
  The pairwise distance between residues is computed as:

  $$d_{ij} = \| C_{\beta_j} - C_{\beta_i} \|.$$

  These distances, along with calculated dihedral and planar angles, are normalized and merged into a 3-channel tensor, which is then mapped to pixel values.

The Inverse Mapping: From Distogram to 3D Coordinates using Classical MDS

- **Distance Recovery:**
  The lower triangle of the first channel of the tensor holds the normalized distances. Inverting the normalization:
  $$d_{ij} = \left(\frac{d_{\text{norm}} + 1}{2}\right) d_{\max},$$
  recovers the original inter-residue distances.

- **Classical MDS:**
  With the full symmetric distance matrix $D \in \mathbb{R}^{L \times L}$ at hand, classical MDS reconstructs the 3D coordinates by the following steps:

  1. **Double-Centering:** Compute the matrix
     $$B = -\frac{1}{2} J D^2 J, \quad \text{with} \quad J = I - \frac{1}{L}\mathbf{1}\mathbf{1}^\top.$$

  2. **Eigen-Decomposition:**
     Solve $B = V \Lambda V^\top$ to obtain eigenvalues Λ and eigenvectors V.

  3. **Coordinate Reconstruction:**
     Taking the top 3 eigenvalues $\lambda_1, \lambda_2, \lambda_3$ (setting negative ones to zero for numerical stability) and corresponding eigenvectors, the 3D coordinates are given by:
     $$X = V_3 \,\text{diag}(\sqrt{\lambda_1},\ \sqrt{\lambda_2},\ \sqrt{\lambda_3}),$$
     where V_3 contains the eigenvectors corresponding to the largest eigenvalues.

- **Procrustes Alignment:**
  Finally, if required, a Procrustes analysis is used to align the reconstructed coordinates X with the original C_β positions, ensuring that the inversion is exact (yielding an RMSD of 0, up to numerical precision).