Mini-project 2 in Advanced Machine Learning (02460)

by YuLiang (s250771), AdrianLopez (s232101) QingwenZeng (s232892),& Abrahim (s193473)

Part A: pullback geodesics We trained a VAE with 3 ensemble decoders, and then computed the geodesics for 25 fixed test point pairs using either one fixed decoder or all three decoders. We parameterized the true geodesics using a piecewise linear curve with T=256 segments and verified that the energy of each curve had essentially converged . Since both settings share the same encoders, they also share the same latent space. As shown in the figure 1,

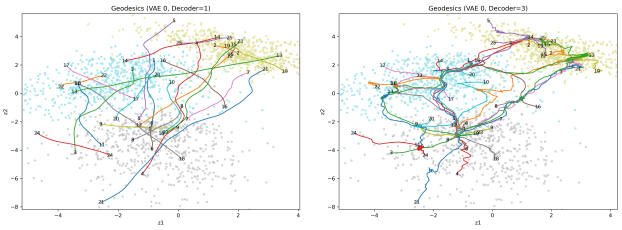

Figure 1: Geodesics between 25 randomly sampled latent variables. The left panel shows results using a standard VAE("SVAE"), while the right panel shows a VAE with 3 ensemble decoders("EVAE").The numbers on the figure indicate the index of the test point pair at the start and end of the geodesics in both plots.

Comparision between EVAE SVAE First, let's follow a pair of test points—take the 3rd point, for example. You'll notice the green line shows that EVAE takes a path that, at each step, tries to stay in regions with potential data points. So it makes several turns, doing its best to avoid both inter-class and intra-class gaps. Compared to that, the SVAE only bends slightly toward the data points near the boundary between the two classes in the middle, while at the beginning and end—within intra-class regions—it mostly moves straight. Other points exhibit similar trends.Ultimately, the EVAE geodesic paths overlap heavily on datapoints, resulting in the blank island areas (both within-class and inter-class) being enclosed one by one. In contrast, the SVAE geodesic paths show little overlap, forming rough inter-class blank islands and almost no intra-class ones.

Reliability of SVAE Next, we compared the reliability of SVAE (see figure 3). The method: for a trained EVAE, we computed the geodesics using only a single decoder, repeating the experiment three times with a different fixed decoder each time, while the encoder is shared and unchanged, which allows us to directly observe variations in geodesics within the same latent space. We found that the more a geodesic passes through inter-class blank regions, the more the trajectories differ—for example, pairs 4, 7, and 10—indicating that different decoders produce significantly different decoded values in those areas. On the other hand, Imagine a hypothetical experiment where an EVAE is trained with 10 decoders, and 5 decoders are selected each time to compute geodesics. Regardless of which 5 are chosen, the resulting trajectories would likely follow similar data-supported paths in the same latent space.**Theoretical Analysis**

Recall $\mathcal{E}(c) \approx \sum_{i=0}^{N} \mathbb{E}_{l,k} \left\| f_l(c(t_i)) - f_k(c(t_{i+1})) \right\|^2$,where f_l and f_k denote decoder ensemble members drawn uniformly. Because the VAE does not model regions of the latent space without data support well, it may fill in those areas with approximated values interpolated from nearby points.If T is large, adjacent curve points are very close, and the energy differences can be very small. As a result, within data clusters, the SVAE tends to take relatively straight paths. In contrast, in the blank regions at class boundaries, there may be a noticeable transition in the decoded values, causing the path to bend slightly. However, for EVAE, each decoder may fill these blank gaps with different values((maybe higher or lower who knows)—whether the gaps are within or between classes. So even if two points in a blank region are close to each other, the cost of a straight connection becomes too high due to our sampling method (randomly choosing two decoders for each pair). As a result, EVAE prefers to stick to the data points(more certain) and avoid all blank areas as much as possible.

Part B: Ensemble VAE geometry In this part, we set T=64, trained 10 independent EVAEs (each with 3 decoders), computed the COV for each of the 10 test points using 1, 2, and 3 decoders respectively , followed by averaging the results.

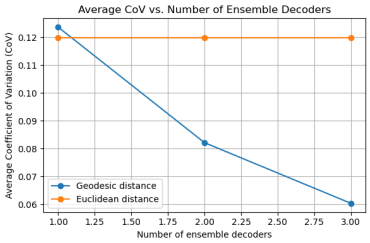

Figure 2: The coefficient of variation across multiple training runs for different number of decoders.

As shown in Fig.2 First, since Euclidean distance does not involve decoder-based geodesic computation, the constant COV reflects the diversity of latent spaces across the 10 independent VAEs. Second, as theoretically analyzed earlier, increasing the number of decoders used in computation leads to greater variation in decoded values within blank regions, causing the path to stay closer to data-supported areas. Meanwhile, the decoded values near data regions remain relatively consistent across different decoders. Based on these two reasons, regardless of how the latent space varies,, the outputs from different VAEs will therefore not differ significantly , resulting in a COV that is lower than that of the Euclidean distance across VAEs. As for the single-decoder computation, the abnormal 0.004 difference: It's possible that individual single decoders are still slightly away from the optimal curve, especially for points that are far away, and the optimization hasn't been fully completed 100%. For a few specific test points, the loss of certain decoders keeps decreasing teeny tiny and extremely slowly no matter how small the learning rate is set, leading to subtle deviations. .

Code snippets

1. Parametric Curve Using the Piecewise Linear Curve Method from Page 68

```
class CubicCurve(nn.Module):  # Although named CubicCurve, this is actually piecewise linear
__init__(self, c0, c1) self.T = 256  self.register_buffer("c0 and c1"  end point non-trainable
intermediate_points =  c0 + (c1 - c0) * (i + 1) / self.T  # Initialize curve as straight line
c_t = nn.Parameter(intermediate_points)  # Make intermediate points trainable
register_buffer("t_grid", torch.linspace(0, 1, self.T + 1)) #t0...tT
def forward(self, t): Compute the value of the piecewise linear curve at position t
idx = torch.searchsorted(t_grid, t, right=True) - 1  # given t get its left segment index
t0, t1 = t_grid[idx],t_grid[idx + 1]  alpha = (t - t0) / (t1 - t0)
control points= cat([c0 c_t, c1]) # linearly interpolated between two control points
ct0, ct1 = control_points[idx], control_points[idx + 1]
        return (1 - alpha.unsqueeze(-1)) * ct0 + alpha.unsqueeze(-1) * ct1
```

2.compute energy $\mathcal{E}[\gamma] \approx \sum_{t=0}^{T-1} \mathbb{E}_{\theta,\theta' \sim q(\theta)q(\theta)} \left[\left\| f_\theta(\gamma(t+\frac{1}{T})) - f_{\theta'}(\gamma(t/T)) \right\|^2 \right]$

```
def compute_curve_energy(curve, decoders, T, fixed_indices=None, ):
 # During each optimization step, the form of objective function must remain unchanged (since gradient descent can only optimize a fixed form of  function), so the index set for randomly sampling from three decoders for each time segment T needs to be fixed in advance
    for i in range(T):
    t0 = torch.tensor([i / T])t1 = torch.tensor([(i + 1) / T])
    x0 = curve(t0)   x1 = curve(t1)
    idx1, idx2 = fixed_indices[i]  #[(idx1_t0, idx2_t0), (idx1_t1, idx2_t1), ...]
    y0 = decoders[idx1](x0).mean y1 = decoders[idx2](x1).mean#compute geodesics under the pull-back metric associated with  mean of the Gaussian decoder.
    energy = torch.norm(y1 - y0, p=2) **2
    total_energy += energy return total_energy#Here we use single sample Monte carlo,so we can simply return total enegry without writing another for loop
```

3.compute geodesics(Adam+Learning rate shcedule+early stopping

```
#Because the distances of different test points vary, the optimization difficulty and required steps differ accordingly. Therefore, we use early stopping and also `ReduceLROnPlateau` scheduling to decrease the learning rate, facilitating toward the stable minumum
def optimize_geodesic(c0, c1, decoders, T, steps, lr):
    curve = CubicCurve(c0, c1).to(device)
    optimizer = torch.optim.Adam(curve.parameters(), lr=lr)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
optimizer,mode='min',factor=0.7,patience=50,threshold=1e-3,
verbose=True,min_lr=1e-4 #Once the loss drops below 1e-3 within 50 steps, the learning rate is reduced to 70% of its previous value)
 fixed_indices = [(torch.randint(0, len(decoders), (1,), ).item(),torch.randint(0, len(decoders), (1,)).item()) for _ in range(T)]# the above mentioned fixed random indicies is generated inside the optimization function but not inside the optimization loop
 with tqdm(range(steps)) as pbar for step in pbar:
optimizer.zero_grad()  energy = compute_curve_energy(curve, decoders, T=T, fixed_indices)
energy.backward() optimizer.step()  #minimize enegry
only if step >= 500: consider early stopping. if within 100 steps the enegry differences is less than 1e-4,early stopping
    return curve, energy_log#After we get the optimized curve
    curve_obj = curves_list[i] t_vals = torch.linspace(0, 1, 256)
     gamma = curve_obj(t_vals) ax.plot(gamma[:, 0], gamma[:, 1]#then we get the geodesics
```

4.caculate distances and average COV

```
def compute_all_geodesic_distances(models_dict, test_image_pairs
# distances == [decoder_num - 1][pair_idx][vae_idx]
for number_of_decoders in range(1, max_decoder_num + 1):
   for pair_idx, (x1, x2) in enumerate(test_image_pairs):
         for m in range(num_vaes):
#first seed for choosing 1-2 decoders should not depend pair_idx,because Because in essence, we are no longer comparing the diversity of 10 VAEs  decoders, but effectively that of 30, It breaks the control of variables.
seed_dec = hash(("decoder_select", m, number_of_decoders)) % (2**32) random.seed(seed_dec)
    selected_decoders = random.sample(list(model.decoders), number_of_decoders)
#the second seed for optimizing should depend on pairs_index to ensure within optmiztation, for different points, same vae using 3 decoders to calculate enegry , we can  better approximate energy(Monte Carlo Caculation)
seed_fixed = hash(("fixed_indices", m, pair_idx, number_of_decoders)) % (2**32) torch.manual_seed(seed_fixed)   curve, energy_log = optimize_geodesic()  then return these
energy = energy_log[-1]  final energy(minimum)
Using the square root of energy value to approximate the distance
z1,z2 = model.encoder(x1).base_dist.loc euclidean = torch.norm(z1 - z2, p=2). #  Eucledian
    for decoder_idx, all_pairs in enumerate(distances): #for num of decoder
        for vae_dists in all_pairs:
            mean = np.mean(d)  std = np.std(d)    # Std across 10 VAEs
            cov = std / mean   # Compute cov_ij for the current test point
        avg_cov = np.mean(covs) # Average cov_ij over 10 test points for that decoder number
```

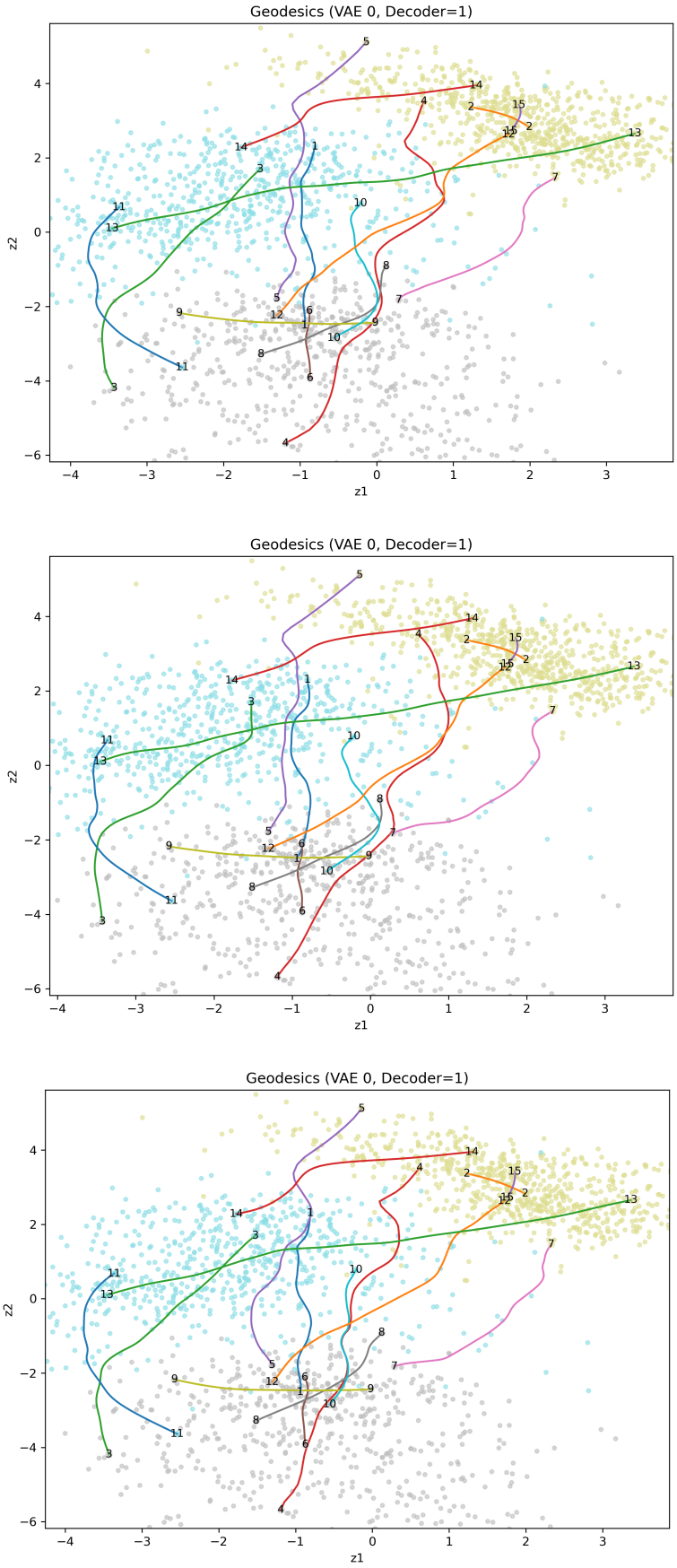

Figure 3: Comparison of SVAE geodesics using different fixed decoders: decoder 0 (top), decoder 2 (middle), and decoder 3 (bottom). One common encoder. T=64,num of pairs=15

References