Mini-project 2 in Advanced Machine Learning (02460)

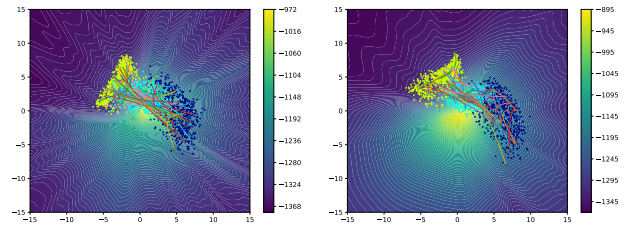
N. Holm (s204503), M. Harborg (s204138), A. Fiehn (s204125) & O. Elmgreen (s204070) - May 29, 2025

Part A: Fisher-Rao geodesics

We consider a subset of MNIST (LeCun et al., 1998), specifically the classes 0, 1, and 2. First, a VAE is trained on this data with a std. Gaussian prior, $p(\mathbf{z})$, and a cont. Bernoulli likelihood, $p(\mathbf{x}|\mathbf{z})$, as introduced in Loaiza-Ganem and Cunningham. 50 point pairs are sampled at random and parameterized polynomial curves of order 3 are used to approximate the distance between the points along the data manifold. We can then compute the geodesics by minimizing the curve energy w.r.t. curve parameters, as measured by the Fisher-Rao metric, since we consider our data manifold to be in the probability distribution space.

The Fisher-Rao curve energy is approximated as $\mathcal{E}(c) \approx \sum_{i=0}^{N} \mathrm{KL}(f(c(t_i)) \,||\, f(c(t_{i+1})))$, where f is a PDF, here of a cont. Bernoulli parametrized by our decoder.

The resulting 50 geodesics approximations can be seen in Figure 1a. We see that the curves do not always seem to follow the most dense areas when drawn from the starting point to the ending point.



(a) One decoder (b) 10 decoder ensemble members

Figure 1: Geodesics between randomly sampled latent variable pairs - these pairs are the same for both models. The background color shows the average entropy[1] of the decoder output distribution at that latent coordinate. 1a shows results using one standard VAE, and 1b shows an ensemble decoder VAE with 10 ensemble members.

To get an idea of whether the computed geodesics are reliable across different training runs, we have trained 5 different VAEs and computed the geodesics for the same 6 data point pairs across these 5 VAEs. We can then compute the Coefficient of Variation (i.e. $\frac{\sigma}{\mu}$, denoted CV) across the curve energies of these geodesics, in order to quantify the variability between training runs.

	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$	$\mathbf{c}_6$
μ	1.490	11.654	30.076	132.009	37.713	29.252
σ	0.466	4.229	4.942	25.491	16.172	6.472
CV	0.313	0.363	0.164	0.193	0.429	0.221

(a) Summary statistics for the approximated energy of the same 6 curves given 5 different single decoder VAEs.

	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$	$\mathbf{c}_6$
μ	42.982	46.022	48.273	48.449	46.634	105.881
σ	9.383	10.042	7.596	2.383	9.197	24.381
CV	0.218	0.218	0.157	0.049	0.197	0.230

(b) Summary statistics for the approximated energy of the same 6 curves given 5 different VAEs with size 10 ensembles of decoders.

Figure 2: Comparative summary statistics for the approximated energy of the same curves (plotted in Figure 4) given by different configurations of VAEs.

These results indicate that the energy is very dependent on the specific VAE configuration, along with the specifics of the energy optimization process itself. This inherent variability leads us to the second part of the project where we used an ensemble of 10 decoders in an attempt to minimize this. Figure 2b shows that ensemble VAEs have less variability in energies across runs, although the latent spaces may still rotate freely relative to each other (see Figure 4).

[1] Negative values are linked to the continuous Bernoulli distribution. We provide entropy_test.py as verification of the correctness of `torch.distributions`, which we have used.

Part B: Ensemble VAE geometry

In the second part of the exercise, we trained a VAE with an ensemble of 10 decoders and estimated the energy by calculating the expected KL divergence between consecutive points decoded by different decoders, approximating the expectation via Monte Carlo estimation, i.e.
$\mathcal{E}(c) \approx \sum_{i=0}^{N} \frac{1}{M} \sum_{m=1}^{M} \mathrm{KL}\left[f_l^{(m)}(c(t_i)) \,||\, f_k^{(m)}(c(t_{i+1}))\right]$.
The resulting geodesics can be seen in Figure 1b. As opposed to those computed with a single decoder VAE, we see that the geodesics computed from the ensemble seems move more consistently along areas with high data density. We now attempt to quantify the impact of using an ensemble by using 1,2,...,10 of these decoders and computing the proximity of geodesics to latent points defined as: $\textbf{proximity} = \max_t \min_n \|c(t) - x_n\|$. The results in Figure 1, 2, and 3 all show that geodesics align better with the latent space using a decoder ensemble compared to single VAEs. This analysis using proximity can however be misleading as we use the same encoder for all of the ensembles, which is trained together with the ensemble of the 10 decoders.

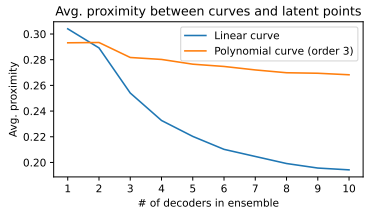


Figure 3: The average proximity between geodesics and latent variables as a function of the number of ensemble members. We consider 10 curves, for both p.w.-linear (blue) and deg. 3 poly. (orange).

Part C: Impact of initalization

To evaluate the impact of curve initialization on finding geodesics, we use the abstract density metric $\mathbf{G}_\mathbf{x} = \frac{1}{p(\mathbf{x})+\epsilon}$ (Hauberg, 2024), with $\epsilon = 10^{-4}$. $\mathbf{G}_\mathbf{x}$ is inversely proportional to the data density, given as a mixture of normals $p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{N}(x|x_n, \sigma^2 I)$. By first optimizing our curve using a density metric, we "hot start" the process, as opposed to the "cold start" of straight lines for piecewise linear - or random ($\sim \mathcal{N}(0,1)$) coefficients for polynomials. In Figure 5, the density metric curves bend towards the Fisher-Rao curves, indicating a lower initial curve energy using hot start. This is supported in Figure 6 where we also often see faster convergence but similar final loss, except for Figure 6b due to the density metric resulting in a linear curve.

Code snippets

In an attempt to fit the most important code snippets in one page, some of them have been shortened a bit.

Fisher-Rao energy approximations. `decoders[0]` is our VAE decoder and `cd` are all the curve points.

```
f = decoders[0]
d1 = f(cd[:-1]) # start point dists
d2 = f(cd[1:]) # end point dists
energy = torch.sum(KL(d1, d2)) # compute total energy
```

Code for computing geodesics. The following code snippet shows a stripped down version of our polynomial curve parametrization and energy optimization formulation. This formulation can be used with both of our curve types as they inherit from `nn.Module`, and it is compatible with both the L-BFGS and ADAM optimizers.

```
class PolyCurve(nn.Module): # stripped down class to fit - see code hand-in for working version
    def __init__(self, start: torch.Tensor, end: torch.Tensor, S: int, K: int):
        # define start and end points for the polynomial approximation
        self.c0 = nn.Parameter(start, requires_grad=False)
        self.c1 = nn.Parameter(end, requires_grad=False)
        # random init coeffs for 1st,...,(K-1)'th order polynomial term for each p latent dims
        self.W1 = nn.Parameter(torch.randn((self.K-1, self.p)), requires_grad=True)
        ts = torch.linspace(0.,1.,self.S+1).unsqueeze(1)
        self.T = torch.concatenate([ts**k for k in range(1,self.K+1)],dim=1) # cache
        self.lin_term = ((1. - ts)@self.c0 + ts@self.c1) # cache
    @property
    def W(self):
        w_iK = -torch.sum(self.W1,dim=0).unsqueeze(0)
        return torch.concatenate([self.W1, w_iK],dim=0) # W
    @property
    def curve(self):
        T = self.T
        poly_term = T @ self.W
        return self.lin_term + poly_term # c_tilde
...
def training_closure():
    optimizer.zero_grad()
    energy = approximate_fisher_rao_energy(curve, decoders, mc_samples=energy_mc_samples)
    energy.backward()
    return energy # optimizer.step() is then called manually for ADAM, or internally by LBFGS
```

Model average Fisher-Rao curve energy. `decoders` is a list of all the decoders, `cd` contains the curve points, `mc_samples` is the number of samples used to estimate the expectation. ***Note***: This implementation uses a batched formulation to turbocharge execution time 🚀, making it somewhat convoluted and harder to read.

```
# precompute probs for each decoder, i.e. outputs of their internal CNNs.
probs = torch.cat([torch.sigmoid(fi.decoder_net(cd)).unsqueeze(0) for fi in decoders],dim=0)
# method to randomly sample a pair of decoder (indices) without replacement
generate_curve_pair = lambda: torch.randperm(len(decoders))[:2].unsqueeze(0)
# pregenerate index pairs - M samples for each N curve segments
mc_pairs = torch.cat([torch.cat([generate_curve_pair()
                                 for _ in range(mc_samples)],dim=0).unsqueeze(-1) # for M samples
                                 for _ in range(len(cd-1))],dim=-1) # for N curve points
energies = torch.zeros(len(cd)-1)
batch_indices = torch.arange(len(cd)) # used for indexing
for i,mcp in enumerate(mc_pairs):
    probs1, probs2 = probs[mcp[0], batch_indices], probs[mcp[1], batch_indices]
    d1 = td.Independent(td.ContinuousBernoulli(probs=probs1),3)
    d2 = td.Independent(td.ContinuousBernoulli(probs=probs2),3)
    energies[i] = torch.sum(KL(d1,d2))
energy = torch.mean(energies)
```

Proximity of curves. First, for a given curve parametrization (`pwlin` or `poly`) we have calculated geodesics for each of the 1-10 decoder ensembles, which are then loaded into `curves`. The proximity is then computed (via the code handout) between each of these curves and the latent points. This yields a list of proximities, which we take the average of.

```
proxs = [[proximity(cps, latents) for cps in cs] for cs in curves] # Calc. proximities per curve
avg_proxs = [torch.tensor(prox).mean() for prox in proxs] # average over proximities for each curve
```

Initialization with density metric. Code from `project2.py` main plot for computing geodesic curves without and with initializing using a density metric.

```
curve_coldstart = create_curve(z0,z1).to(device)
energy_cold = train_fisher_rao(curve_coldstart, decoders, optim_type='adam', energy_mc_samples=4)
metric = DensityMetric(data=latents, device=device) # Hotstart init curves w. density metric
curve = create_curve(z0,z1).to(device)
train_curve(curve, metric, steps=5_000, lr=1e-2, device)
energy = train_fisher_rao(curve, decoders, optim_type='adam', energy_mc_samples=4)
```

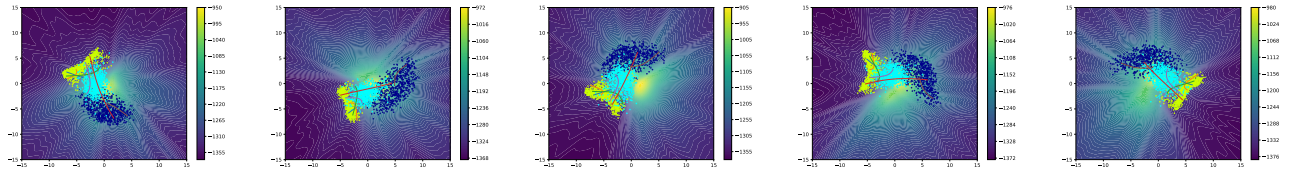
References

S. Hauberg. *Differential geometry for generative modeling*. 2024.

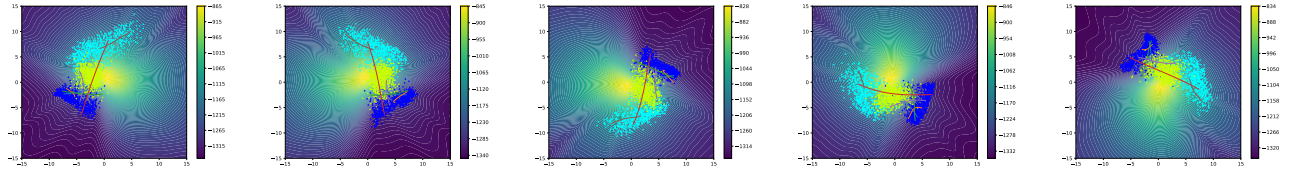
Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/.

G. Loaiza-Ganem and J. P. Cunningham. The continuous bernoulli: fixing a pervasive error in variational autoencoders. *2019*. URL https://github.com/cunningham-lab/cb.

Appendix



(a) VAE - single decoder 1 **(b)** VAE - single decoder 2 **(c)** VAE - single decoder 3 **(d)** VAE - single decoder 4 **(e)** VAE - single decoder 5



(f) VAE - ensemble (10) decoder 1 **(g)** VAE - ensemble (10) decoder 2 **(h)** VAE - ensemble (10) decoder 3 **(i)** VAE - ensemble (10) decoder 4 **(j)** VAE - ensemble (10) decoder 5

Figure 4: Plots of the 6 computed geodesics for each of the different models trained to create Figure 2.

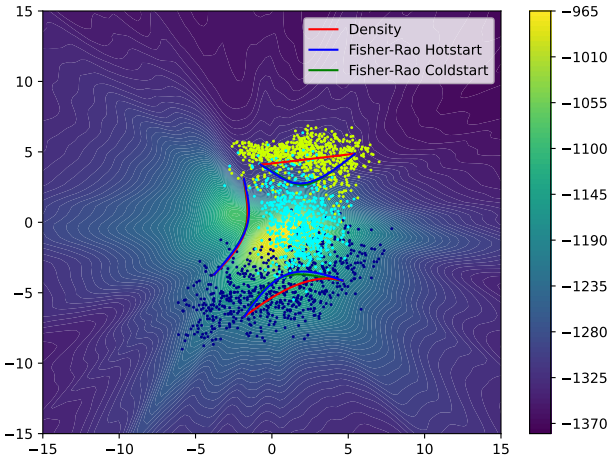
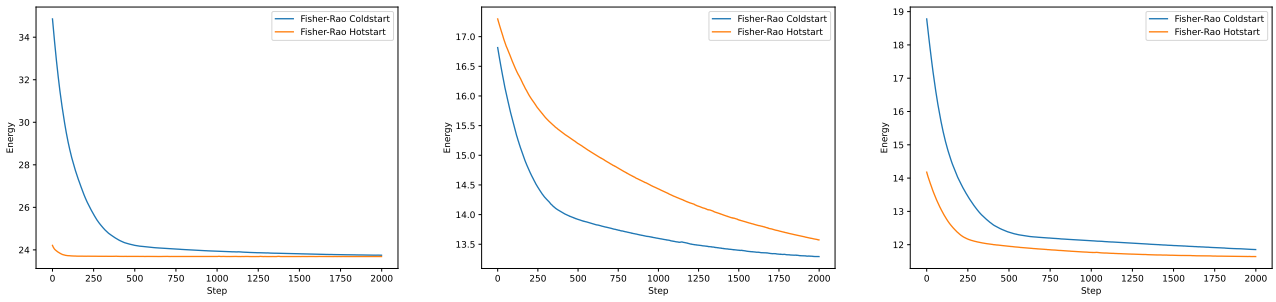


Figure 5: Selected 3rd degree polynomial curves coldstarted using fisher-rao metric (green), compared to hotstarted fisher-rao (blue) using the density metric (red).



(a) Curve 1 (left curve on Figure 5) **(b)** Curve 2 (top curve on Figure 5) **(c)** Curve 3 (bottom curve on Figure 5)

Figure 6: Comparison plots of the energy over training steps for each polycurve on the Figure 5 using Fisher-Rao coldstart or hotstart with the density metric.