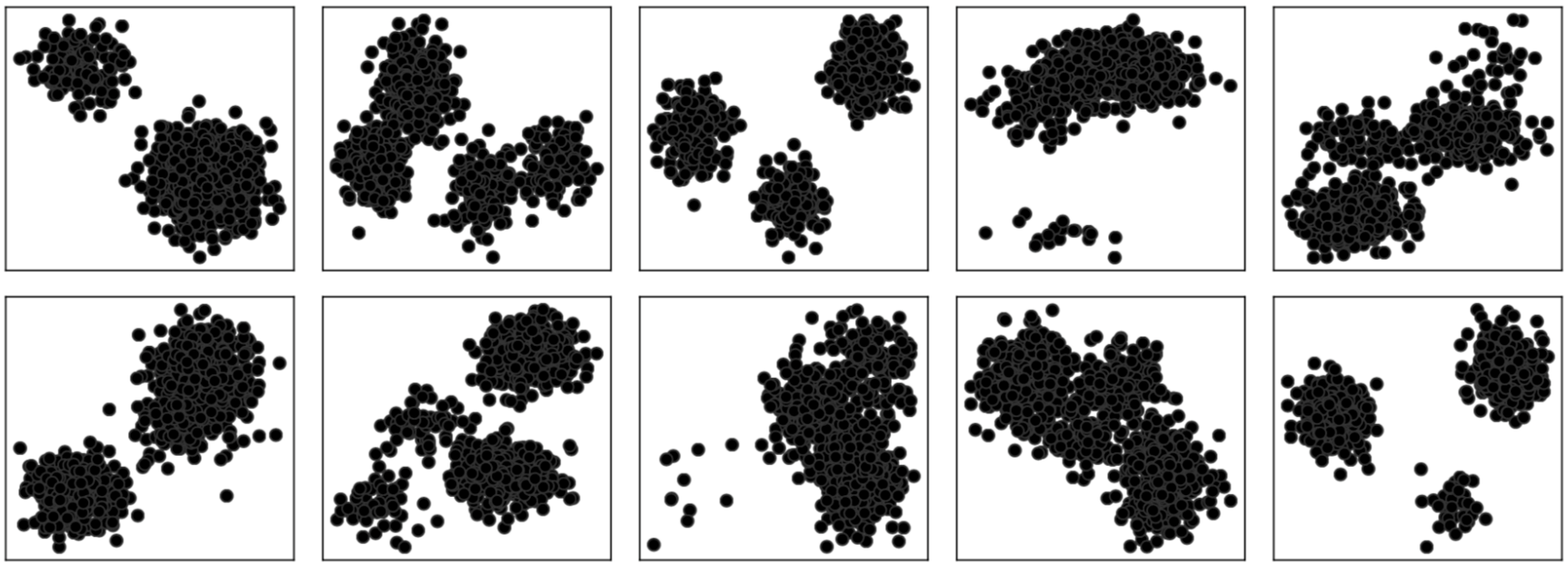
Towards Deep Amortized Clustering

Juho Lee[1,2], Yoonho Lee[1] and Yee Whye Teh[3,4]
[1] AITRICS, [2] KAIST,
[3] University of Oxford, [4] DeepMind

AITRICS

Amortized clustering

- Assume we have a batch of datasets to cluster.
- Should we run an iterative algorithm for each dataset independently until convergence?

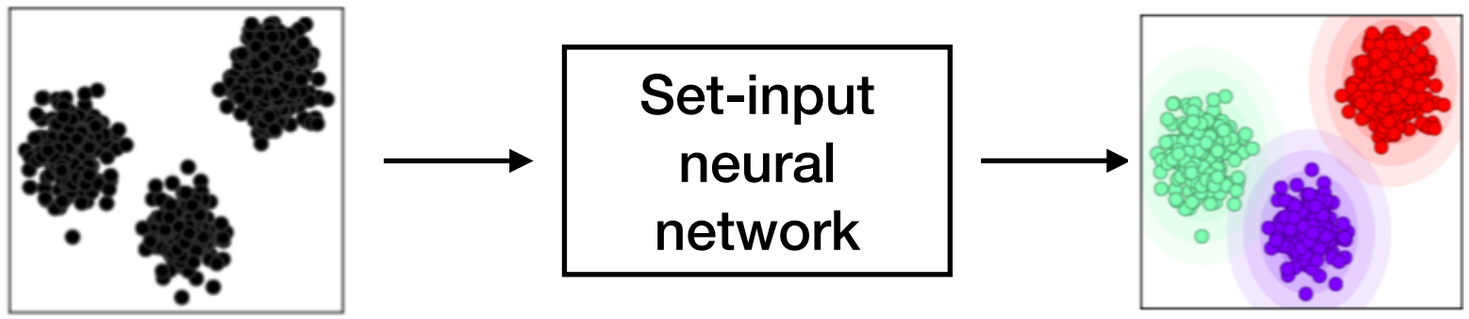


Amortized clustering

- Amortization: reuse the previous inference results to accelerate the inference of related problems. [Gershman and Goodman, 2014]

- Amortized clustering: reuse the previous inference (clustering) results to accelerate the inference (clustering) of new datasets.

- Deep amortized clustering: construct a deep neural network that takes a dataset X and produces the clustering results. Reuse this network to efficiently cluster (possibly in only a few forward passes) new datasets.



Clustering with Mixture of Gaussians

$$p(X; \pi, \mu, \sigma) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i; \mu_k, \mathrm{diag}(\sigma_k^2)) \,.$$

$$\theta^* = \arg\max_{\theta} \; \log p(X; \theta)$$

$$p(x_i \in \textbf{cluster } k) = \frac{\pi_k \mathcal{N}(x_i; \mu_k, \mathrm{diag}(\sigma_k^2))}{\sum_\ell \pi_\ell \mathcal{N}(x_i; \mu_\ell, \mathrm{diag}(\sigma_\ell^2))} \,.$$

- Expectation-Maximization (EM) algorithm to maximize the log-likelihood.

Amortized clustering for MoG

- Construct a neural network $f(\cdot\,;\phi)$ taking sets and outputs the maximum-likelihood parameter θ under mixture of Gaussian assumption.

- Train the network by maximizing the expected likelihood over datasets.

$$\mathscr{L}(\phi) = \mathbb{E}_{p(X)}[\log p(X; \theta = f(X; \phi))]\,.$$

- At each iteration,
  - Sample a batch of datasets $X_1, \ldots, X_B$ from $p(X)$.
  - Update the ϕ with the stochastic gradient $\frac{1}{B}\sum_{b=1}^{B} \nabla_\phi \log p(X_b; \theta = f(X_b; \phi))\,.$

Set transformer [Lee et al., 2019a]

- Permutation invariant neural network based on self-attention.
- Composed of a variant of transformer layer [Vaswani et al., 2017] that performs a permutation equivariant operation on sets.
- Basic building blocks
  - Multihead attention block (MAB): residual connection + multihead dot-product attention followed by a feed-forward layer,

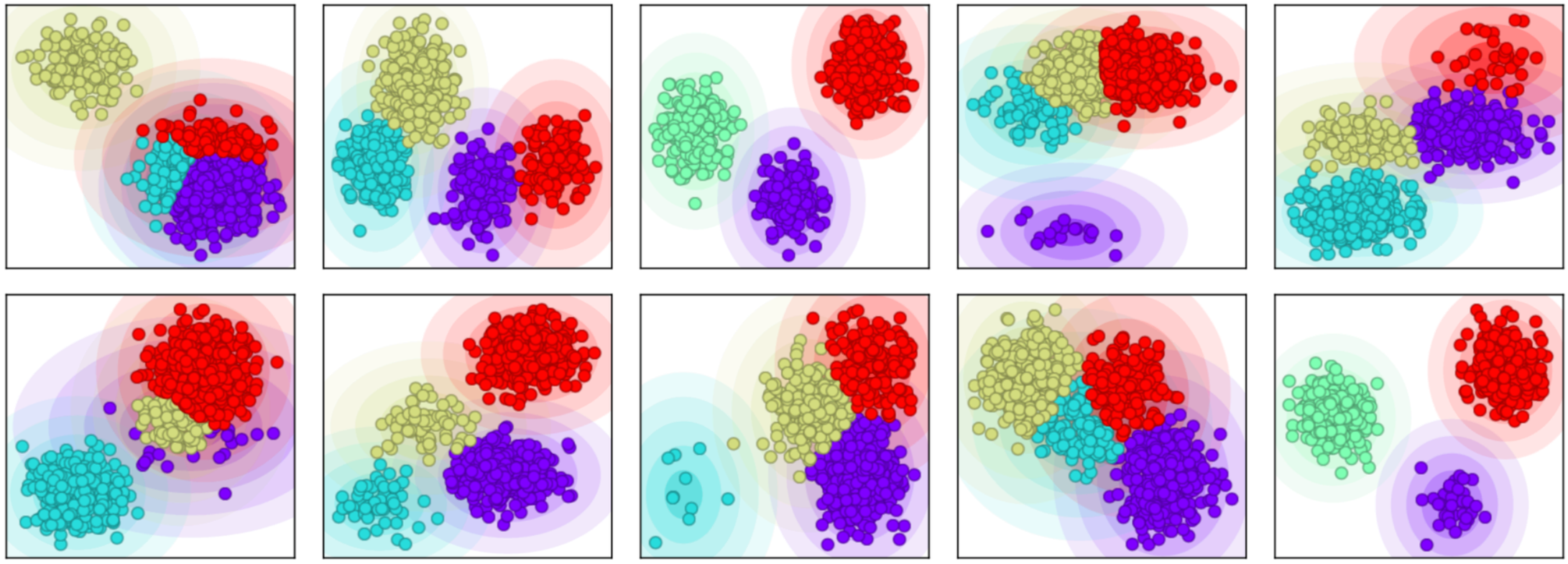
$$\mathrm{MAB}(X, Y) = \mathrm{FFN}(X + \mathrm{Att}(X, Y)) .$$

  - Self attention block (SAB): MAB applied in self-attention way,

$$\mathrm{SAB}(X) = \mathrm{MAB}(X, X) .$$

Solving Amortized clustering

- Set transformer solves amortized clustering accurately, and for this self-attention in both encoder and decoder is crucial.



Deep amortized clustering (DAC)

- Without assuming anything about the number of clusters, given a dataset distribution $p(X, y)$ that generates a dataset X and corresponding cluster label y, learn a neural network $f(\cdot\,; \phi)$ that can amortize the clustering of novel datasets by a few forward passes.

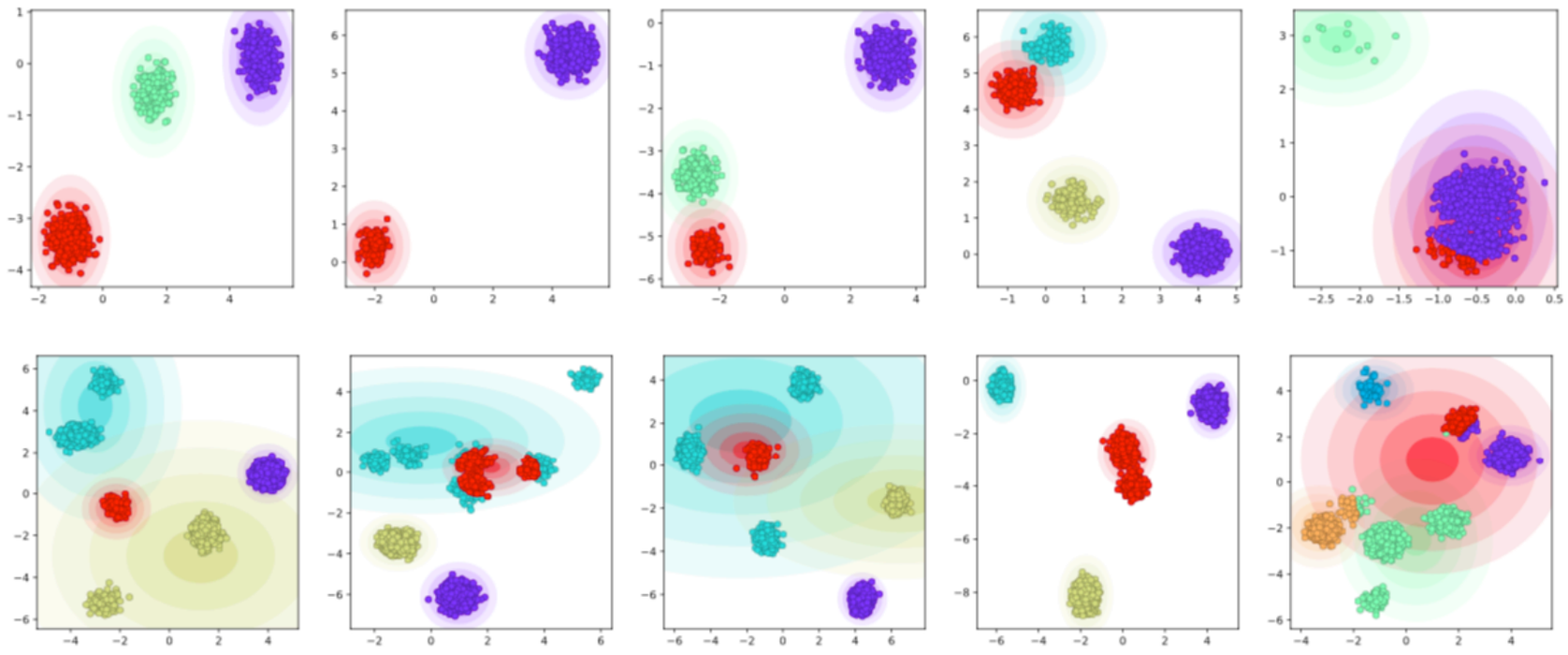
- For an easy scenario, we may assume a simple parametric distribution (e.g., Gaussians) to define a cluster, but in principle we do not assume anything about the cluster shape, and let the algorithm learn from the datasets.

Handling variable numbers of clusters

- Simple set-transformer decoder won’t work because it assumed fixed-dimensional outputs.

- A naive way is to construct a RNN-like decoder that iteratively produce cluster parameters and decides when to stop, but this fails to generalize.

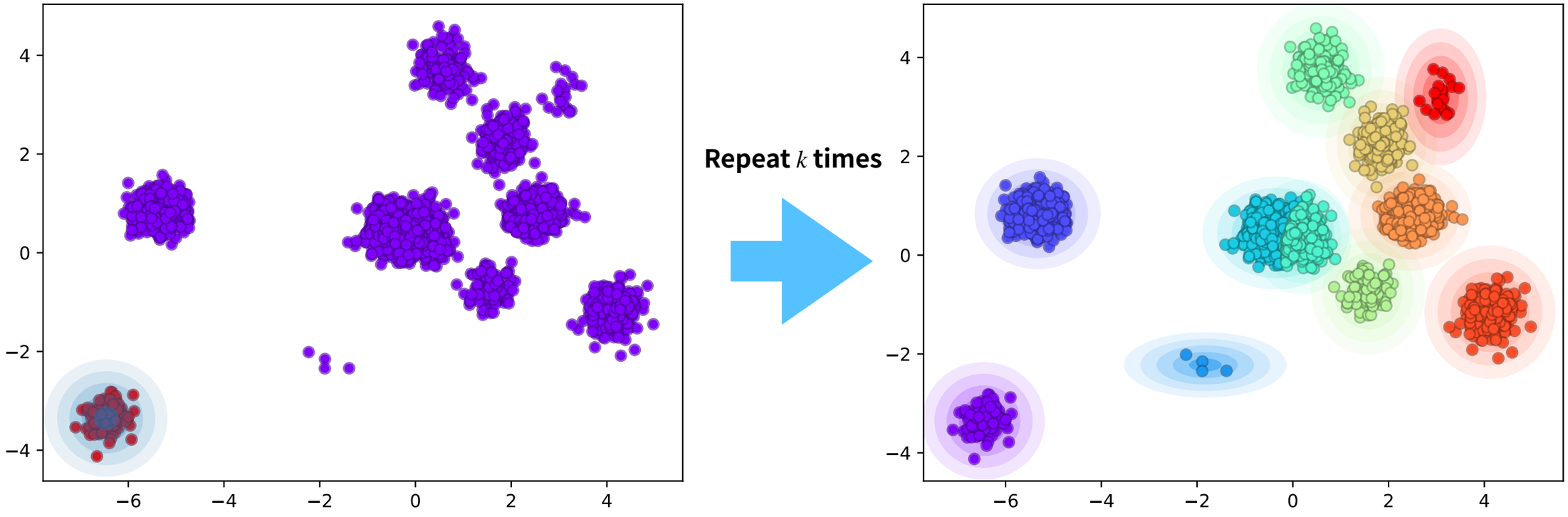
The RNN-like set transformer trained with datasets having {1,…4} clusters works well for new datasets having {1,…,4} clusters (top row), but completely fails for datasets having >4 clusters (bottom row).



Filtering

- Instead of trying to find the whole clusters at once, learn to isolate one cluster at a time.

- “Filter out" a cluster at each iteration, and repeat until no data point remains.



Training the filtering network

- Construct a filtering neural network $f(\,\cdot\,;\phi)$ taking a set X and produces,
  - a single parameter θ to describe a cluster,
  - a membership probability vector $\mathfrak{m} \in [0,1]^{n_X}$ denotes to what extent that each data point in X belongs to the cluster described by θ.

- An example architecture:

$$
\begin{aligned}
\textbf{encode data:} \quad & H_X = \mathrm{ISAB}_L(X), \\
\textbf{decode cluster:} \quad & H_\theta = \mathrm{PMA}_1(H_X), && \theta = \mathrm{rFF}(H_\theta), \\
\textbf{decode mask:} \quad & H_\mathfrak{m} = \mathrm{ISAB}_{L'}(\mathrm{MAB}(H_X, H_\theta)), && \mathfrak{m} = \mathrm{sigmoid}(\mathrm{rFF}(H_\mathfrak{m})).
\end{aligned}
$$

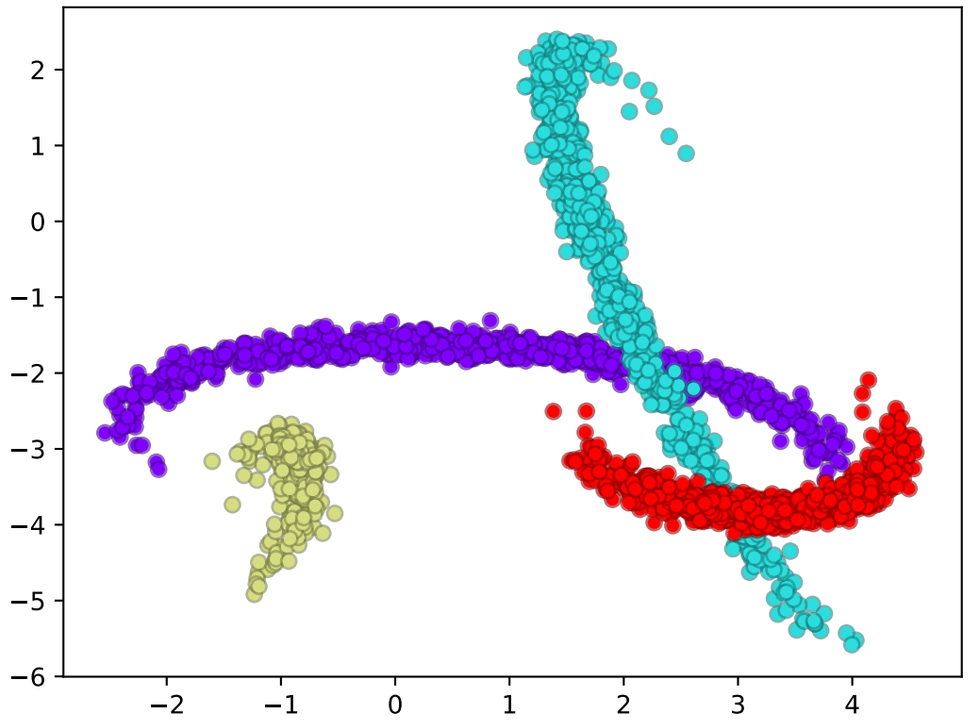
Training the filtering network

- The loss function to train the filtering network is,

$$\mathscr{L}(X, y, \mathfrak{m}, \theta) = \min_{j=1,\ldots,k_X} \left(\frac{1}{n_X} \sum_{i=1}^{n_X} \mathrm{BCE}(\mathfrak{m}_i, 1_{[y_i=j]}) - \frac{1}{n_{X|j}} \sum_{i|y_i=j} \log p(x_i; \theta) \right),$$

where n_X is the number of data points in X, k_X is the number of clusters in X, $n_{X|j}$ is the number of data points belong to the jth cluster, BCE is the binary cross entropy.

Beyond MoG - Filtering complex clusters



- Don’t assume particular cluster densities, but learn them from data as well.
  - Neural density estimators conditioned on cluster contexts.
  - Maximize the lower bound on the likelihood using variational distributions.

Beyond MoG - Filtering complex clusters

- Masked autoregressive flows (MAF) [Papamakarios et al., 2017] conditioned on cluster context parameter θ.

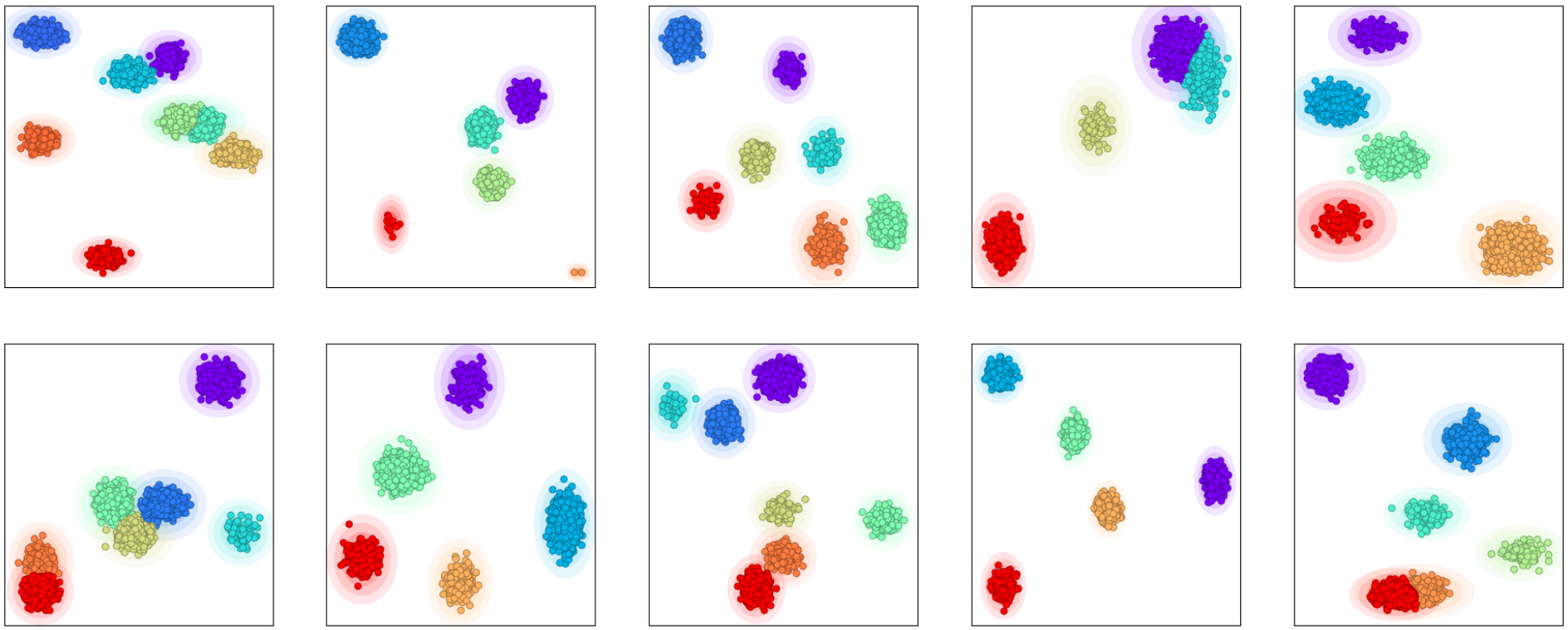
$$p(x;\theta) = \prod_{\ell=1}^{d} \mathcal{N}(u_\ell; 0,1)\sigma(x_{1:\ell-1};\theta), \quad u_\ell = \frac{x_\ell - \mu(x_{1:\ell-1};\theta)}{\sigma(x_{1:\ell-1};\theta)}.$$

- Computing lower bound on the likelihood conditioned on the cluster context parameter θ.

$$p(x;\theta) \geq \sum_{x \in X} \mathbb{E}_{q(z|x;\theta)} \left[\log \frac{p(x \mid z;\theta)p(z;\theta)}{q(z \mid x;\theta)} \right]$$

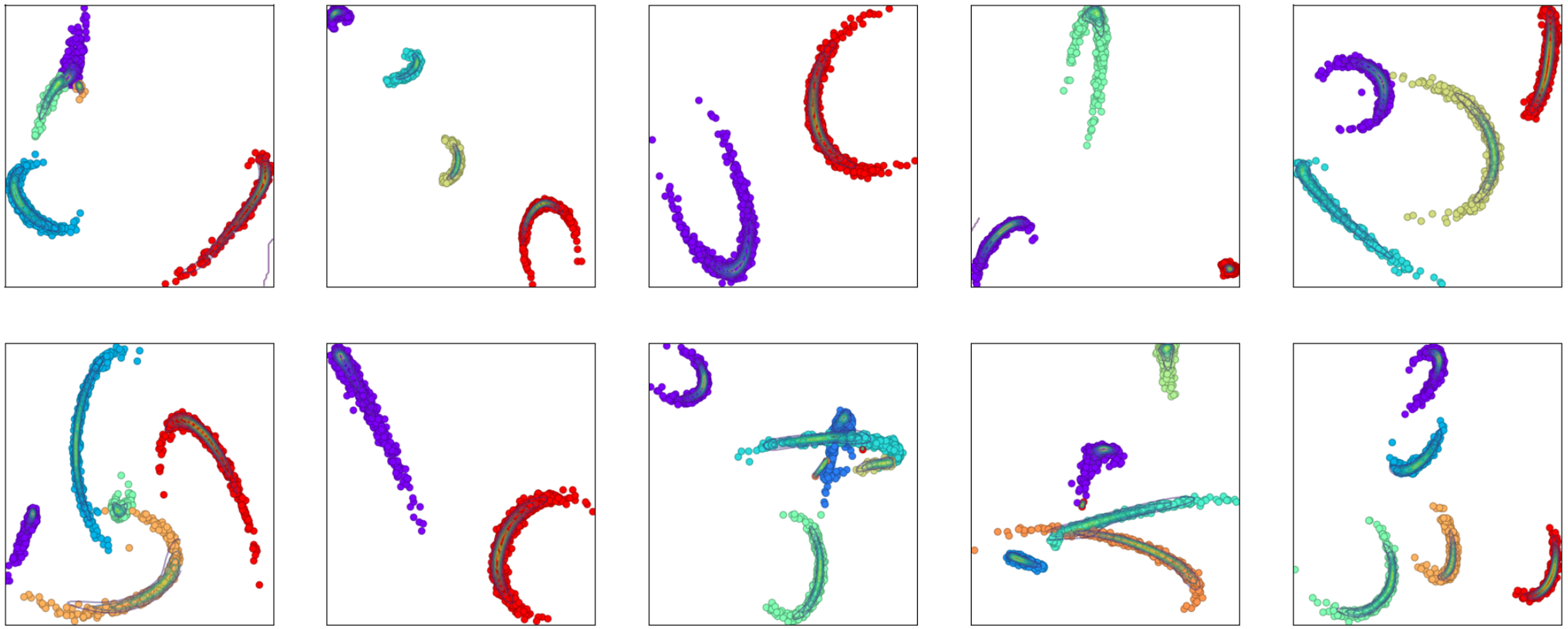
DAC for MoG

- Trained with random datasets having {1,…,4} clusters and tested with datasets having arbitrary number of clusters.



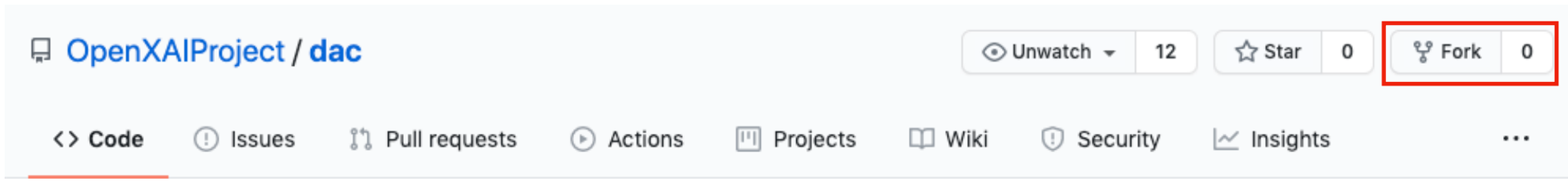
DAC for Warped MoG

- Trained with MAF as a density estimator.



Setting up the code

- Fork OpenXAIProject/dac into your account.



- Clone (or download) into your local machine

```
git clone https://github.com/OpenXAIProject/dac
```

Setting up the code

- Open `utils/paths.py` and set the ROOT directory to work with.

```
ROOT = ##### set your ROOT directory ######

results_path = os.path.join(ROOT, 'dac', 'results')
benchmarks_path = os.path.join(ROOT, 'dac', 'benchmarks')
datasets_path = os.path.join(ROOT, 'datasets')
```

- Install required packages (Python >= 3.5.0 required)

```
pip install -r requirements.txt
```

Training

- To train a DAC model for mixture of Gaussians data,

```
python -m scripts.run \
            —model models/mog.py \
            —run_name (experiment id) \
            —gpu (gpu number) \
            —loss_type (min or anc)
```

- To train a DAC model for mixture of warped Gaussians data,

```
python -m scripts.run \
            —model models/warped.py \
            —run_name (experiment id) \
            —gpu (gpu number) \
            —loss_type (min or anc)
```

Testing on clustering benchmark

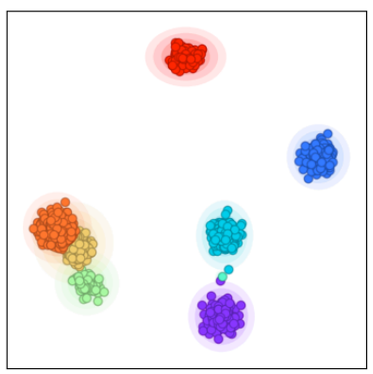
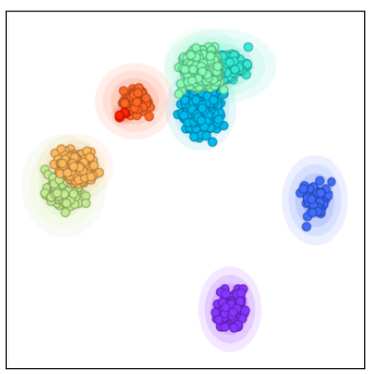
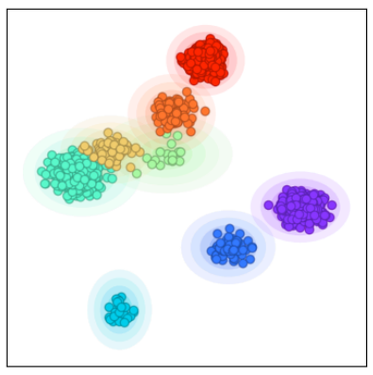
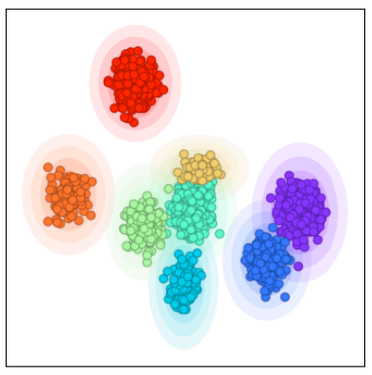
- To test the trained model,

```
python -m scripts.test_cluster \
        —model models/(mog.py or warped.py)\
        —run_name (experiment id) \
        —gpu (gpu number) \
        —loss_type (min or anc)
```

```
john@lang:~/torch/dac$ python -m scripts.test_cluster --model models/mog.py --gpu 2
Namespace(B=100, K=4, N=1000, clustering_benchmark=None, filename='test_cluster.log', filtering_benchm
ark=None, gpu='2', loss_type='min', lr=0.0005, max_iter=50, model='models/mog.py', num_steps=20000, ru
n_name='trial', vB=10, vK=4, vN=1000)
100%|██████████████████████████████████████████████████████████| 100/100 [00:16<00:00,  6.04it/s]
INFO:mog_trial:ll -1.5460, oracle -1.5245, ARI 0.9707, NMI 0.9745, k-MAE 0.2340, et 0.0132, (16.571 se
cs)
john@lang:~/torch/dac$
```

Visualizing the clustering results

- To visualize the clustering results obtained by trained DAC models,

```
python -m scripts.vis_cluster \
                —model models/(mog.py or warped.py)\
                —run_name (experiment id) \
                —gpu (gpu number) \
                —loss_type (min or anc) \
                —vB (number of datasets) \
                —vN (max number of data per dataset) \
                —vK (max number of clusters)
```

References

[Gershman and Goodman, 2014] Gershman, S. and Goodman, N. D. Amortized inference in probabilistic reasoning. Proceedings of Annual Conference of the Cognitive Science Society, 2014.

[Ritchie et al., 2016] Ritchie, D. and Horsfall, P. and Goodman, N. D. Deep amortized inference for probabilistic programs. arXiv:1610.05735, 2016.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. ICLR, 2014.

[Zaheer et al., 2017] Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. NIPS, 2017.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. NeurIPS, 2017.

[Lee et al., 2019a] Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., and Teh, Y. W. Set transformer: a framework for attention-based permutation-invariant neural networks. ICML, 2019.

[Lee et al., 2019b] Lee, J. Lee Y., and Teh, Y. W. Deep amortized clustering. arXiv:1909.13433, 2019.

[Papamakarios et al., 2017] Papamakarios, G. and Pavlakou, T. and Murray, I. Masked autoregressive flow. NeurIPS, 2017.