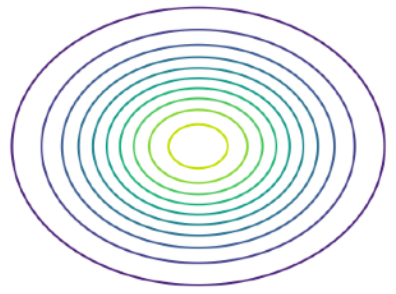
Diffusion Models

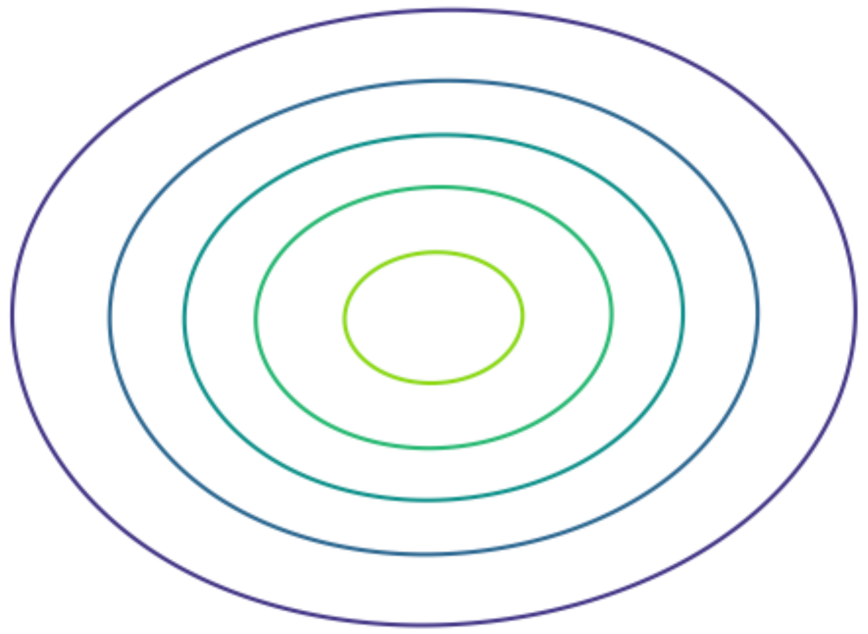
Handley Group Meetings

David Yallup - 19/03

Modelling distributions



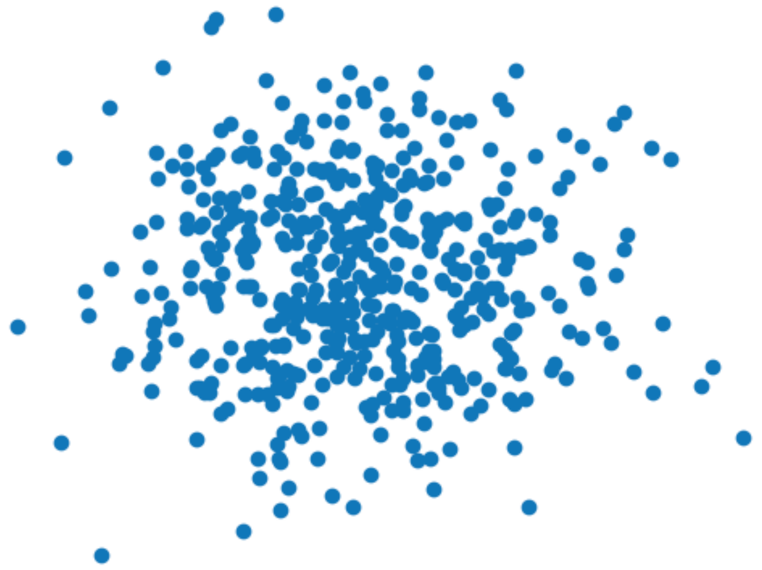
$$\frac{1}{2\pi\sigma}e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



Distribution

Can be represented by

.rvs
PolyChord.run
emcee.sample



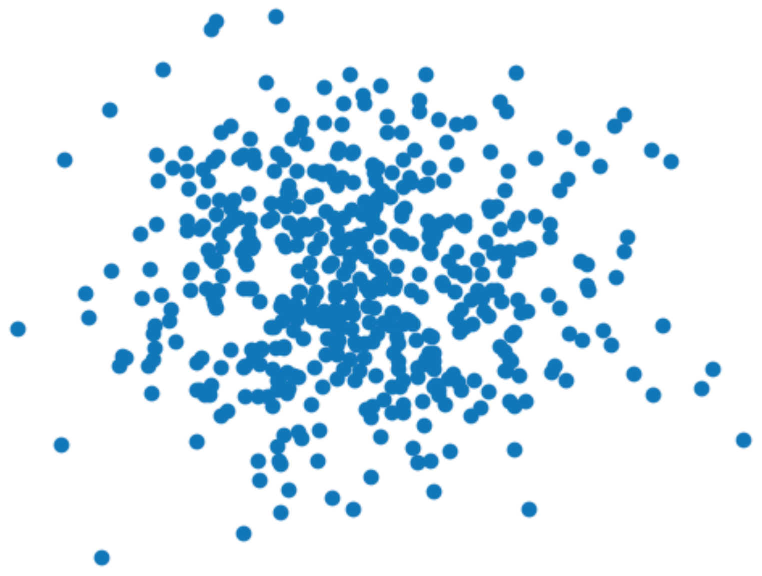
Samples

Implicitly defines

GP.fit
KDE.fit
NormalisingFlow

Modelling distributions

If You have the full loop you can “Emulate”.
If you only have Samples, the ability to bestow them with a distribution is useful!

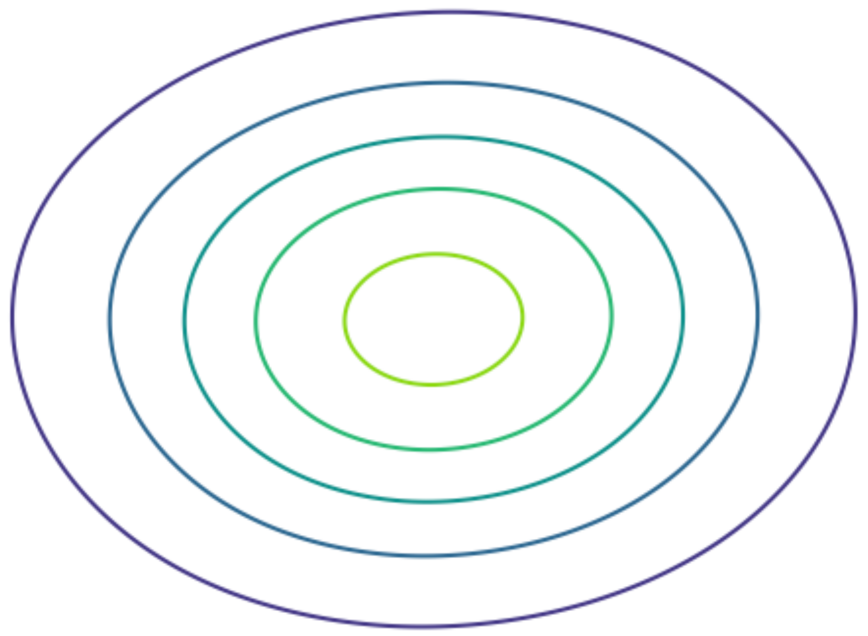


Distribution

Samples

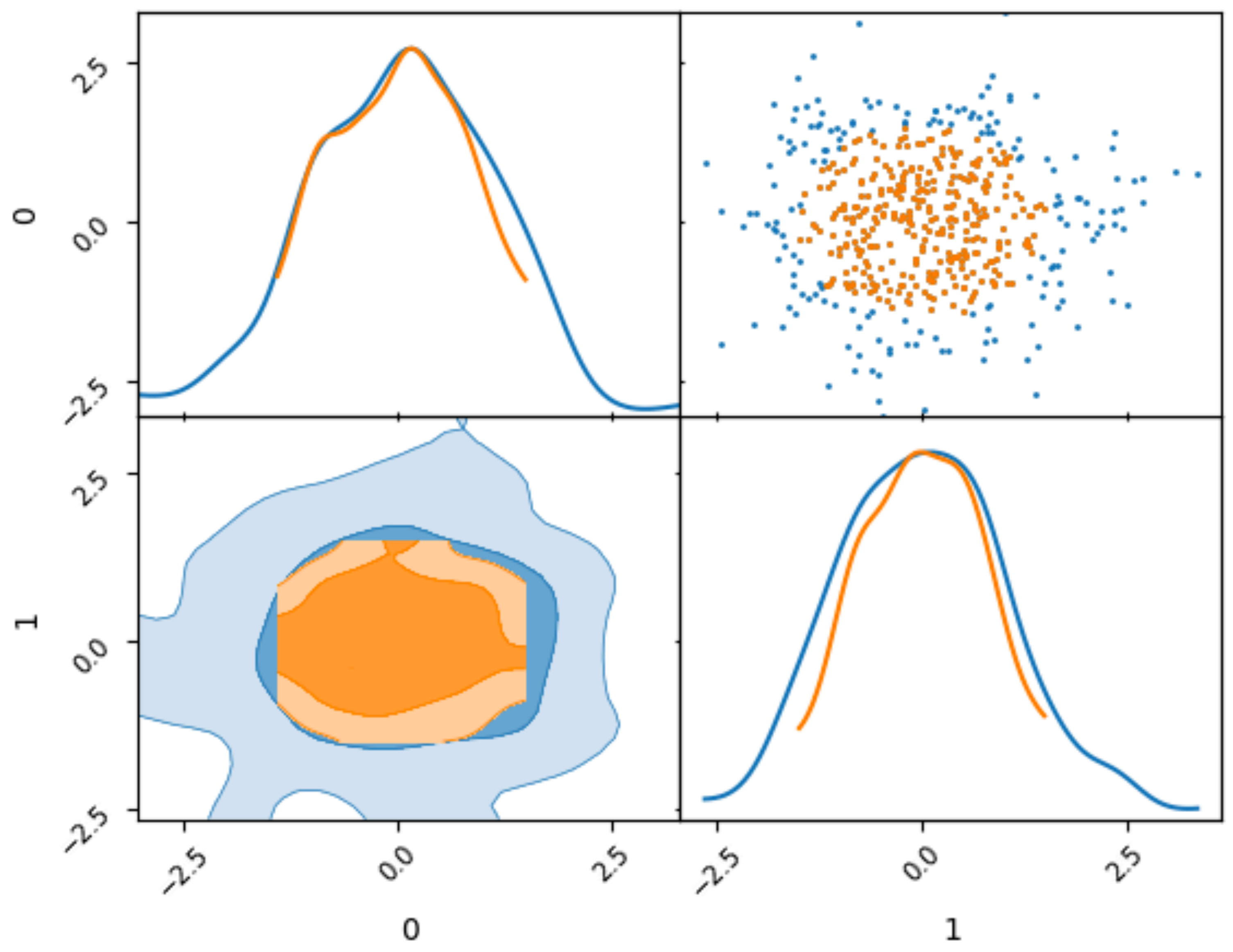
Implicitly defines

GP.fit
KDE.fit
NormalisingFlow



Nested Sampling with distributions

“Rejection sampling”

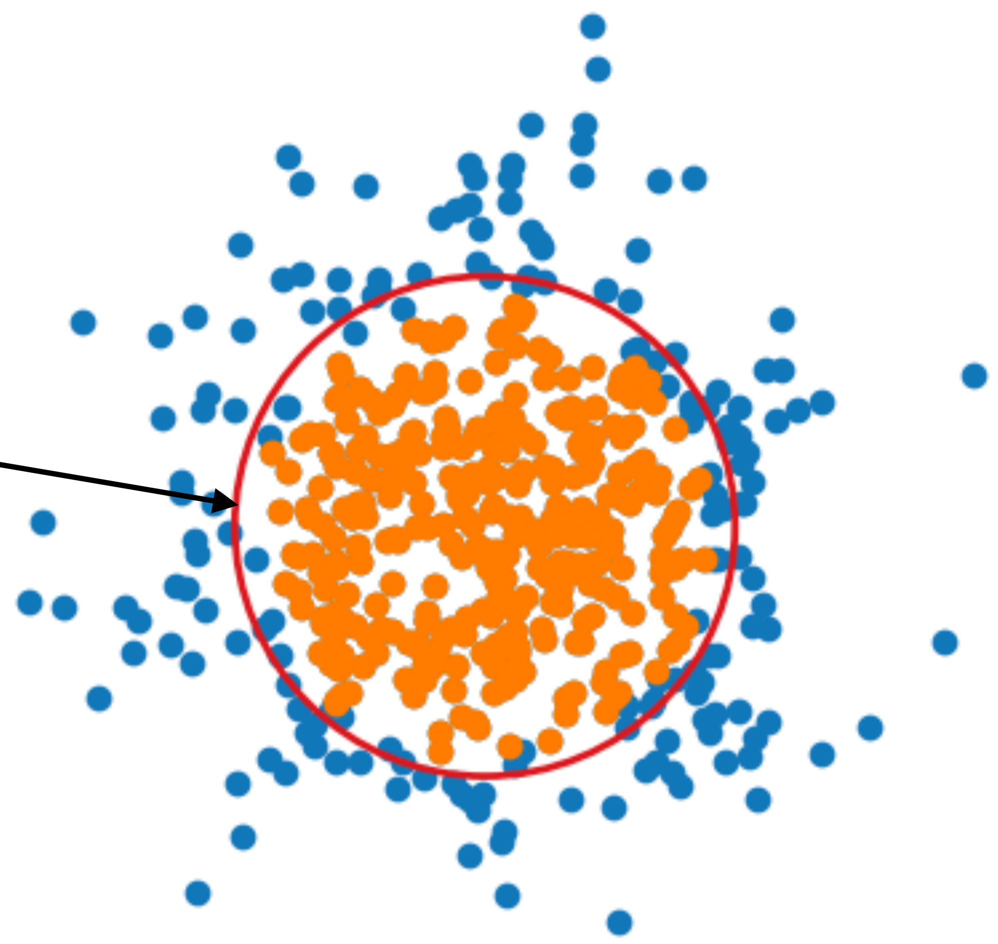


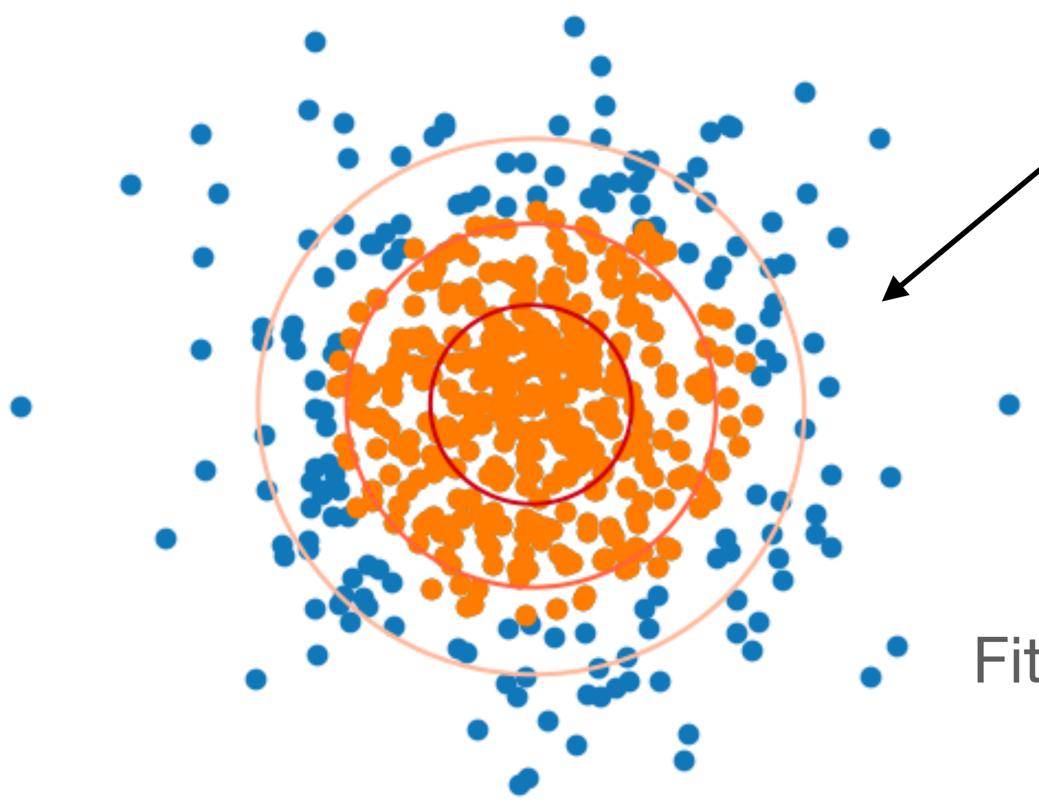
- Blue Samples from a known Distribution
- Orange Samples from likelihood constrained region

Orange implicitly defines a distribution we want more of

How can we “bestow” these points with a distribution?

Fit an ellipse: MultiNest



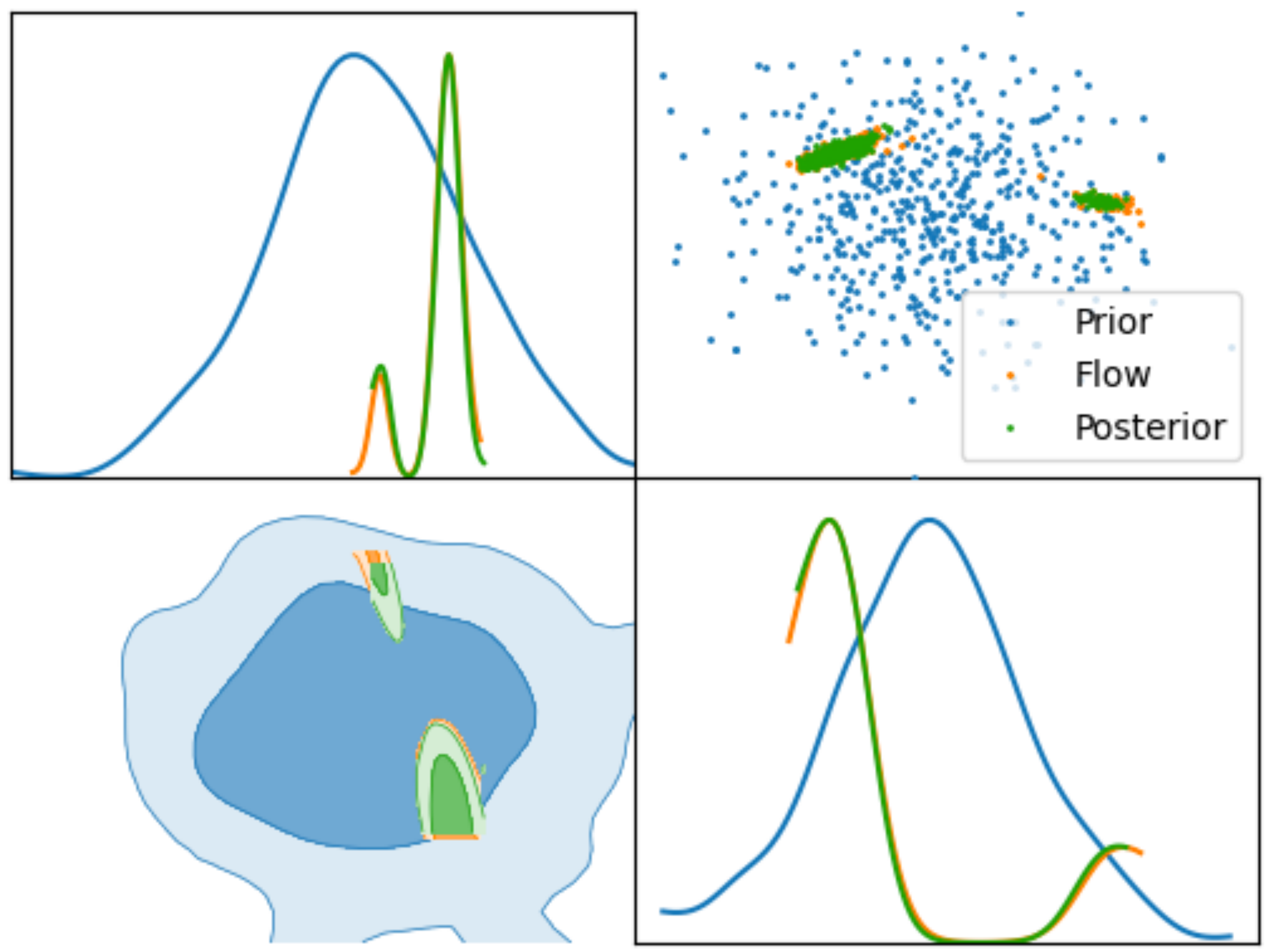


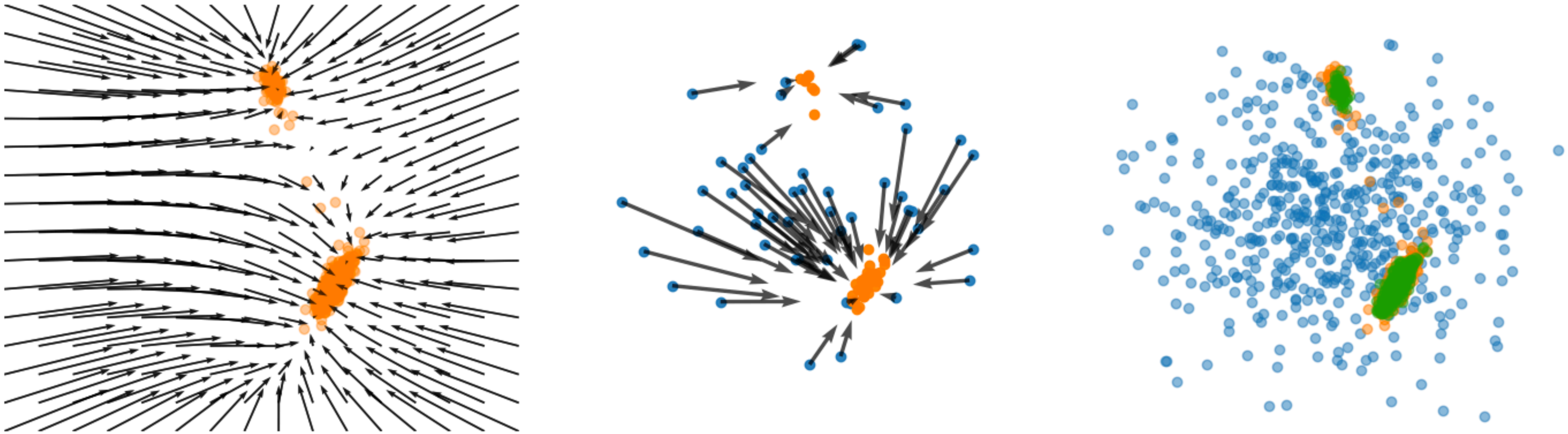
Fit a neural density: nessai etc.

Diffusion Models

The current vogue in density estimation

Optimal Transport, Diffusive processes, Continuous flows, Score based generative models, flow matching etc.etc. All the same thing!

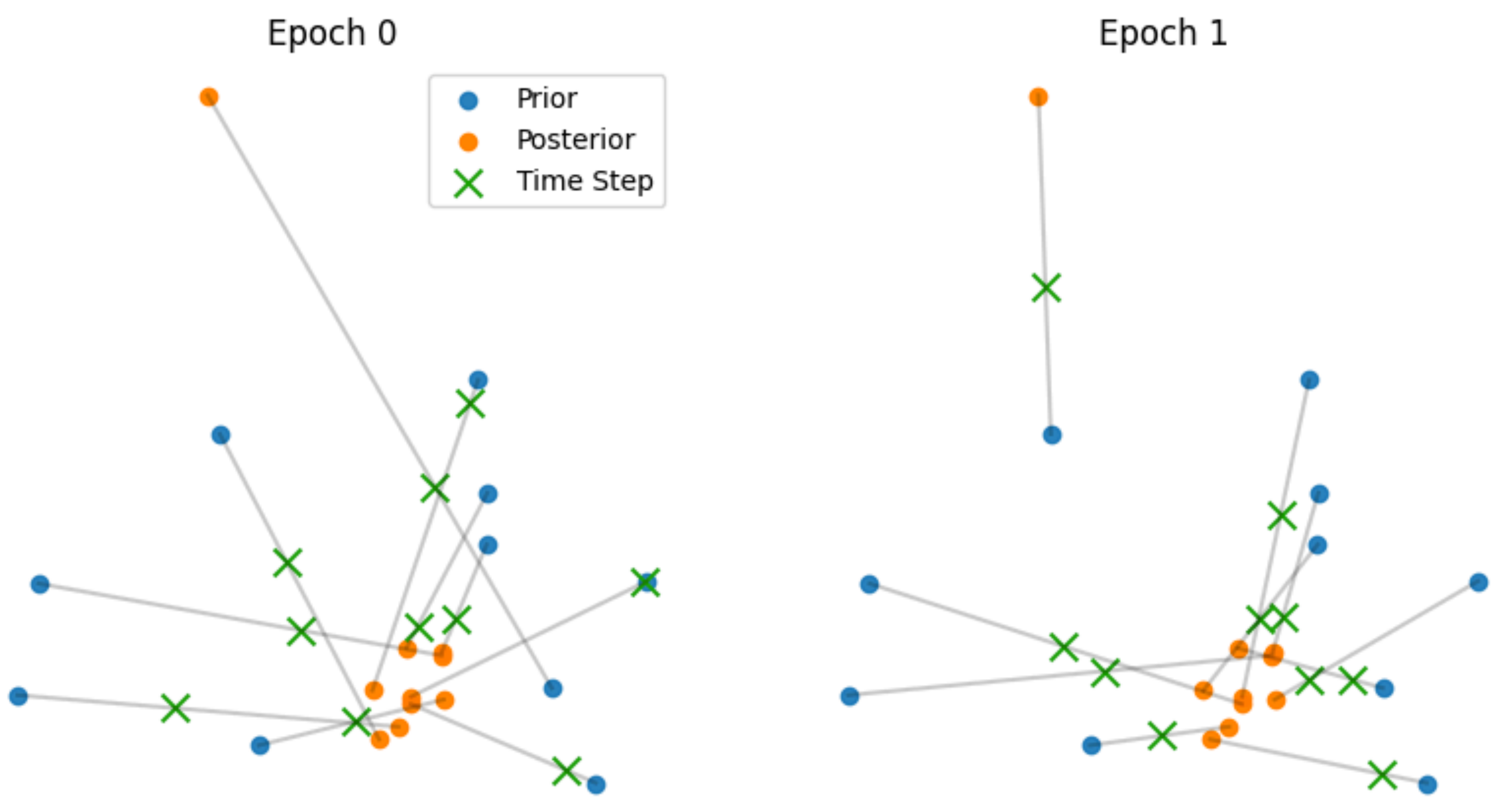




Neural Networks Learn gradient fields (Score) that map one distribution to another

How it works

[2302.00482] Flow matching, incredibly simple objective!

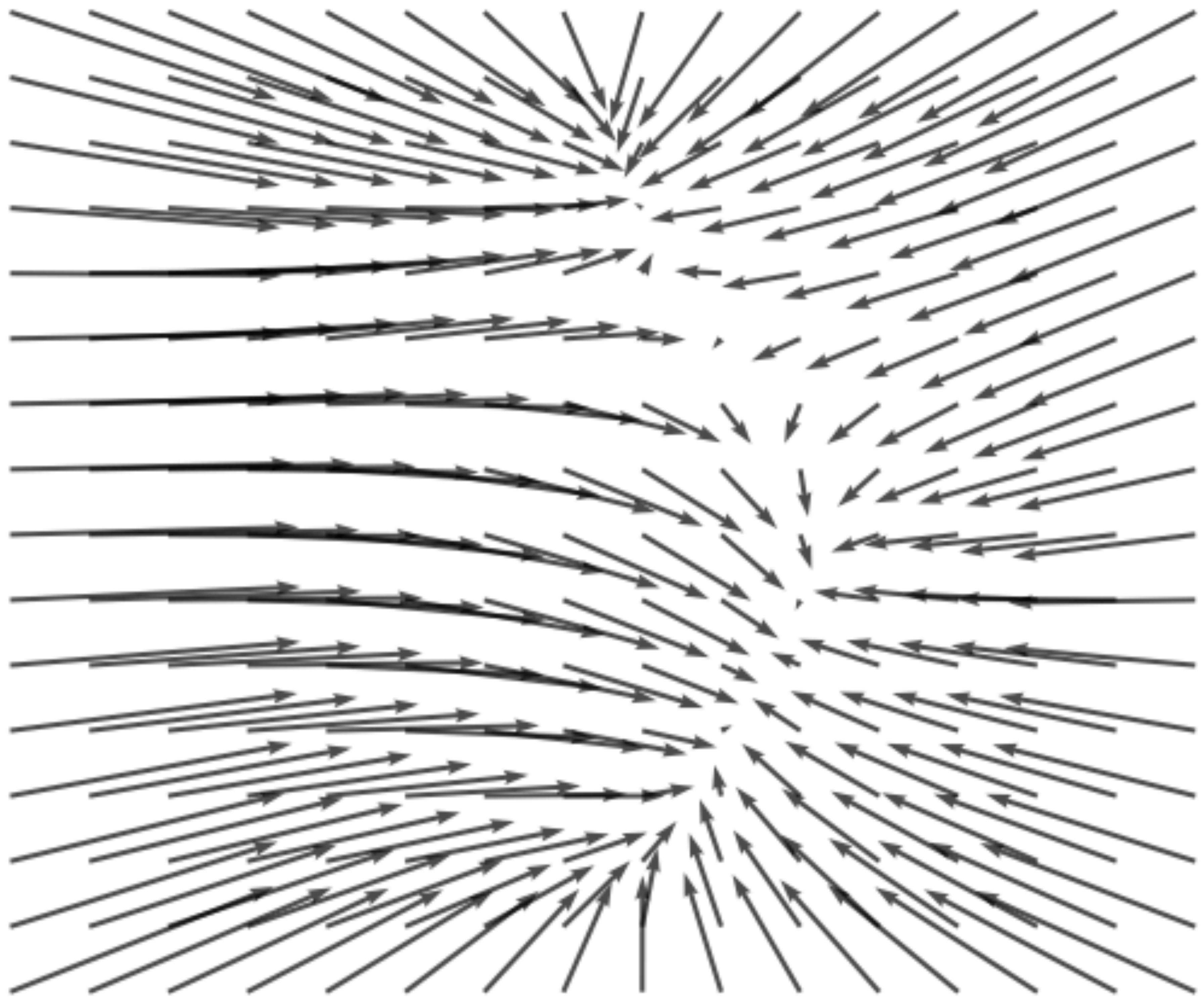



```python
@partial(jit, static_argnums=[0])
def loss(self, params, batch, batch_prior, batch_stats, rng):
    """Loss function for training the CFM score.

    Args:
        params (jnp.ndarray): Parameters of the model.
        batch (jnp.ndarray): Target batch.
        batch_prior (jnp.ndarray): Prior batch.
        batch_stats (Any): Batch statistics (batchnorm running totals).
        rng: Jax Random number generator key.

    """
    # sigma_noise = 1e-3
    rng, step_rng = random.split(rng)
    N_batch = batch.shape[0]

    t = random.uniform(step_rng, (N_batch, 1))
    noise = random.normal(step_rng, (N_batch, self.ndims))
    psi_0 = t * batch + (1 - t) * batch_prior + self.noise * noise

    output, updates = self.state.apply_fn(
        {"params": params, "batch_stats": batch_stats},
        psi_0,
        t,
        train=True,
        mutable=["batch_stats"],
    )
    psi = batch_prior - batch
    loss = jnp.mean((output - psi) ** 2)
    return loss, updates
```

- Pair up a random set of prior and target samples
- Generate a random time step for each pair, $t \leftarrow [0,1]$
- Simple MSE regression on the prior-target vectors and a neural network learning the vector field at the time step

By shuffling and simulating a new time step we build up coverage of the whole space and trace out all paths

What to do with a learned vector field



Diffusion:

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t \sqrt{1-\beta_t} x_{t-1}, \beta_t I)$$

$$q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})$$

Solve a SDE, easy to simulate solutions from
[yang-song.net/blog/2021/score]

Continuous Flows:

$$\frac{dy}{dt} = \nabla_\theta(t, y(t)), \quad y(0) = y_0$$

Solve an ODE, slightly harder but nicer properties for science
[2202.02435]

Jacobians are an interesting part of this

Often for scientific applications need this