Leveraging Data-Assimilation ideas and tools in Machine Learning

Peter Jan van Leeuwen

Department of Atmospheric Science

Colorado State University

Deep Learning

- Formulate a loss-function using input-output pairs (Z_i, X_i) as training data:

$$J(w) = \sum_{i-1}^{M} (X_i - f(w, Z_i))^2$$

- Find *w* such that *J(w)* is minimized.
- But, …, where does this come from? Why do we minimize a square?
- (And is it in any way **Intelligent, or is it only Artificial Intelligent?)**

- From a data-assimilation perspective, this looks very much like 3DVar and 4DVar, but in which prior information is ignored, and uncertainty quantification is not present.
- Can we be more consistent in our ML methods?

Can ML be formulated as a Bayesian Inference Problem?

- Yes!
- Advantages of a Bayesian Inference formulation:
  - Uncertainty quantification becomes an integral part of the ML, which means it becomes **useful for real applications**.
  - Extra (physical?) constraints can be formulated as part of the prior, and uncertainties are automatically included
  - Merging of data assimilation and ML becomes natural

- So, let's reformulate ML as a proper Bayesian Inference problem.

Intermezzo: probability calculus

A few equations:

$$p(a, b) = p(a|b)p(b)$$

$$p(a, b|c) = p(a|b, c)p(b|c)$$

$$\int p(a, b)\ db = p(a)$$

$$\int p(a, b|c)\ db = p(a|c)$$

$$p(a|c) = \int p(a, b|c)\ db$$

What are the sources of uncertainty in ML?

ML finds a relation between input *z* and output *x*, based on training and testing data:

$$x = f(z, \theta_{tr}, \theta_{te}) \qquad \theta_{tr} = (X_{train}, Z_{train})$$

However, the predicted value of *x* is uncertain, we need the **probability density function:**

$$p(x|z, \theta_{tr}, \theta_{te})$$

The pdf of *x* arises from uncertainty in 5 variables:

- input and output training data
- input and output testing data
- learned weights
- function *f(..)*
- new input *z*

Typically, most are ignored and/or included incorrectly…

But where are the ML weights?

Start from the probability density description:

$$p(x|z,\theta_{tr},\theta_{te}) = \int p(x,w|z,\theta_{tr},\theta_{te})\, dw$$

$$= \int p(x|w,z,\theta_{tr},\theta_{te})p(w|z,\theta_{tr},\theta_{te})\, dw =$$

$$= \int p(x|w,z,\theta_{te})p(w|\theta_{tr})\, dw$$

This is how to include the ML weights in the problem.

How to include uncertainty in the training data?

- We can consider the training data θ_{tr} as a random draw from a distribution $p(\theta_{tr})$, centered around the true training data without error θ_{tr}^{t}.
- This true training data set is not known, but it will be quite useful. Let us consider all possible true training data sets that would lead to us drawing θ_{tr} .

$$p(x|z,\theta_{tr},\theta_{te}) = \int p(x,\theta_{tr}^{t}|z,\theta_{tr},\theta_{te})\, d\theta_{tr}^{t}$$

$$= \int p(x|\theta_{tr}^{t},z,\theta_{tr},\theta_{te})p(\theta_{tr}^{t}|z,\theta_{tr},\theta_{te})\, d\theta_{tr}^{t} =$$

$$= \int p(x|\theta_{tr}^{t},z,\theta_{te})p(\theta_{tr}^{t}|\theta_{tr})\, d\theta_{tr}^{t}$$

- Note that $p(\theta_{tr}^{t}|\theta_{tr})$ is exactly the uncertainty pdf for the training data set.
- In a similar way, we can bring in uncertainty in new input and testing data.

Taking all uncertainty sources into account

The **fundamental equation for deep learning**:

$$p(x|z,\theta_{tr},\theta_{te}) = \iiiint p(x|w,z^t,\theta_{te}^t)p(w|\theta_{tr})p(\theta_{tr}^t|\theta_{tr})p(\theta_{te}^t|\theta_{te})p(z^t|z)\; d\theta_{tr}^t d\theta_{te}^t dw dz^t$$

(5) (4) (3) (2) (1)

in which $\theta_{tr} = (X_{train}, Z_{train})$

The 5 uncertainty sources:

(1) = uncertainty in new input data
(2) = uncertainty in testing data
(3) = uncertainty in training data
(4) = uncertainty in the weights
(5) = uncertainty in f(..) (determined from testing data)

This is what each ML application should try to approximate!!!

(Same equation for linear regression!)

The hard part: uncertainty in the weights

We start with a prior pdf for the weights *p(w)* : typically uniform *U[-1,1]*

The weights are trained with training (and validation) data

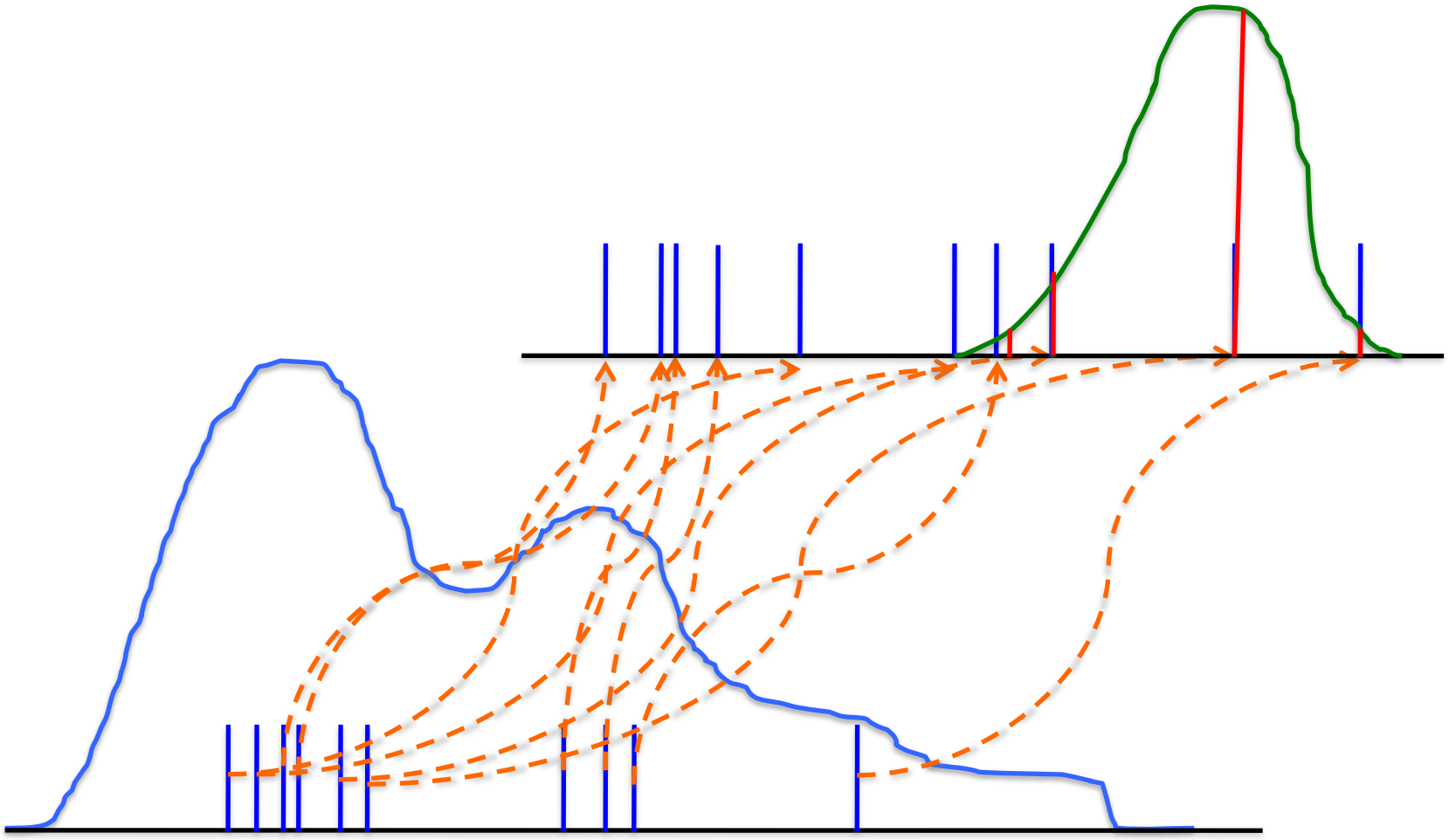
$$\theta_{tr} = (X, Z)$$

in which *Z* and *X* contain the input-output training data pairs. These training data can be seen as the observations that bring us from prior to posterior.

The posterior pdf for the weights after training with data, using Bayes Theorem:

$$p(w|\theta_{tr}) = \frac{p(\theta_{tr}|w)}{p(\theta_{tr})} p(w)$$

Intermezzo: Particle Filters I



$$p(w) = \frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i)$$

$$\overline{w} = \int wp(w)dw =$$

$$= \int w\frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i)\,dw =$$

$$= \frac{1}{N}\sum_{i=1}^{N} w_i$$

Intermezzo: Particle Filters II

Remember that, in a particle filter, we draw samples from the prior pdf as:

$$p(w) = \frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i)$$

For the standard particle filter we then write the posterior pdf of the weight vectors given the training data (observations):

$$p(w|\theta_{tr}) = \frac{p(\theta_{tr}|w)}{p(\theta_{tr})}p(w) = \frac{p(\theta_{tr}|w)}{p(\theta_{tr})}\frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i)$$

$$= \frac{1}{p(\theta_{tr})}\frac{1}{N}\sum_{i=1}^{N}p(\theta_{tr}|w_i)\delta(w - w_i)$$

$$= \sum_{i=1}^{N}\alpha_i\delta(w - w_i) \quad \text{with importance weights} \quad \alpha_i = \frac{p(\theta_{tr}|w_i)}{\sum_{j=1}^{N}p(\theta_{tr}|w_j)}$$

Intermezzo: Particle Filters III

Now, assume we will draw the samples not directly from the prior, but from a so-called **proposal density**

$$q(w|\theta_{tr}) = \frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i^q)$$

To use this in Bayes Theorem, simply multiply and divide by $q(w|\theta_{tr})$:

$$p(w|\theta_{tr}) = \frac{p(\theta_{tr}|w)}{p(\theta_{tr})}p(w)\frac{q(w|\theta_{tr})}{q(w|\theta_{tr})}$$

If we now use the expression for $q(w|\theta_{tr})$ in Bayes Theorem, we find:

$$p(w|\theta_{tr}) = \sum_{i=1}^{N}\beta_i\delta(w - w_i)$$

with importance weights $$\beta_i = \frac{p(\theta_{tr}|w_i)}{\sum_{j=1}^{N}p(\theta_{tr}|w_j)}\frac{p(w_i)}{q(w_i|\theta_i)}$$

The ML connection with particle filters

In ML methods that 'include uncertainty' we start with samples w_i from the prior pdf, such that

$$p(w) = \frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i)$$

We recognize this as … **a particle filter**

Then these particles w_i are modified by the training, which we can think of as applying a **proposal density** $q(w|\theta_{tr})$, hence Bayes Theorem becomes:

$$p(w|\theta_{tr}) = \frac{p(\theta_{tr}|w)}{p(\theta_{tr})}\frac{p(w)}{q(w|\theta_{tr})}q(w|\theta_{tr})$$

and the result of the training is that the particles are drawn from $q(w|\theta_{tr})$:

$$q(w|\theta_{tr}) = \frac{1}{N}\sum_{i=1}^{N}\delta(w - w_i^q)$$

The importance weights

Using this in Bayes theorem leads to:

$$p(w|\theta_{tr}) = \sum_{i=1}^{N} \frac{1}{N} \frac{p(\theta_{tr}|w_i^q)}{p(\theta_{tr})} \frac{p(w_i^q)}{q(w_i^q|\theta_{tr})} \delta(w - w_i^q)$$

Hence, each weight vector in ML obtains an **importance weight** of:

$$\beta_i^q = \frac{1}{N} \frac{p(\theta_{tr}|w_i^q)}{p(\theta_{tr})} \frac{p(w_i^q)}{q(w_i^q|\theta_{tr})}$$

such that:

$$p(w|\theta_{tr}) = \sum_{i=1}^{N} \beta_i^q \; \delta(w - w_i^q)$$

Incorrect posterior weights pdf used in ML

A common mistake in ML is to assume that the trained weights are iid samples, so one takes:

$$\beta_i^q = \frac{1}{N}$$

(**see standard ML methods such as Bagging, MC-dropout, Deep Ensembles,…**).

However, each ML weight should be weighted by its **likelihood values** and **proposal densities** from the transformation due to training. This is all standard statistics from over 50 years ago, called importance sampling, as in a particle filter.

If this importance weighting is done properly, **all these ensemble ML methods are degenerate…** Can we do better?

Yes, see data-assimilation literature.

Connection with the loss function used in ML

The loss function in ML comes from the likelihood:

$$p(\theta_{tr}|w) = p(X_{tr}|w, Z_{tr})p(Z_{tr})$$

in which Z_{tr} is the input training data and X_{tr} is the output training data.

If we now assume that the error in the NN is independent, Gaussian distributed and the errors in the input and output data are negligible, we find:

$$p(\theta_{tr}|w) = A \exp\left[-\frac{1}{2}\sum_{i=1}^{N_{tr}} \frac{(X_i - f(w, Z_i))^2}{\sigma^2}\right]$$

The maximum of this pdf corresponds to the minimum of the loss function.

Think again about all the other uncertainty sources that are ignored…

The fundamental equation for deep learning:

$$p(x|z,\theta_{tr},\theta_{te}) = \iiiint p(x|w,z^t,\theta_{te}^t)p(w|\theta_{tr})p(\theta_{tr}^t|\theta_{tr})p(\theta_{te}^t|\theta_{te})p(z^t|z)\; d\theta_{tr}^t d\theta_{te}^t dw dz^t$$

(5) (4) (3) (2) (1)

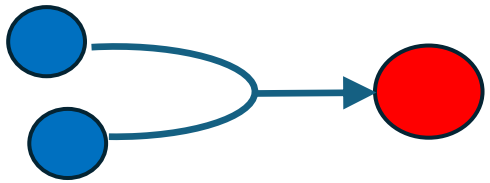
in which

(1) = uncertainty in new input data
(2) = uncertainty in testing data
(3) = uncertainty in training data
(4) = uncertainty in the weights
(5) = uncertainty in f(..) (determined from testing data)

How can we use this in practice?

- Even though many pdf are typically Gaussian, $p(w|\theta_{tr}^t)$ and $p(x|w,z^t,\theta_{te}^t)$ are not!
- A practical way is to use ensembles
- (With a so-called equal-weight particle filter for $p(w|\theta_{tr}^t)$)

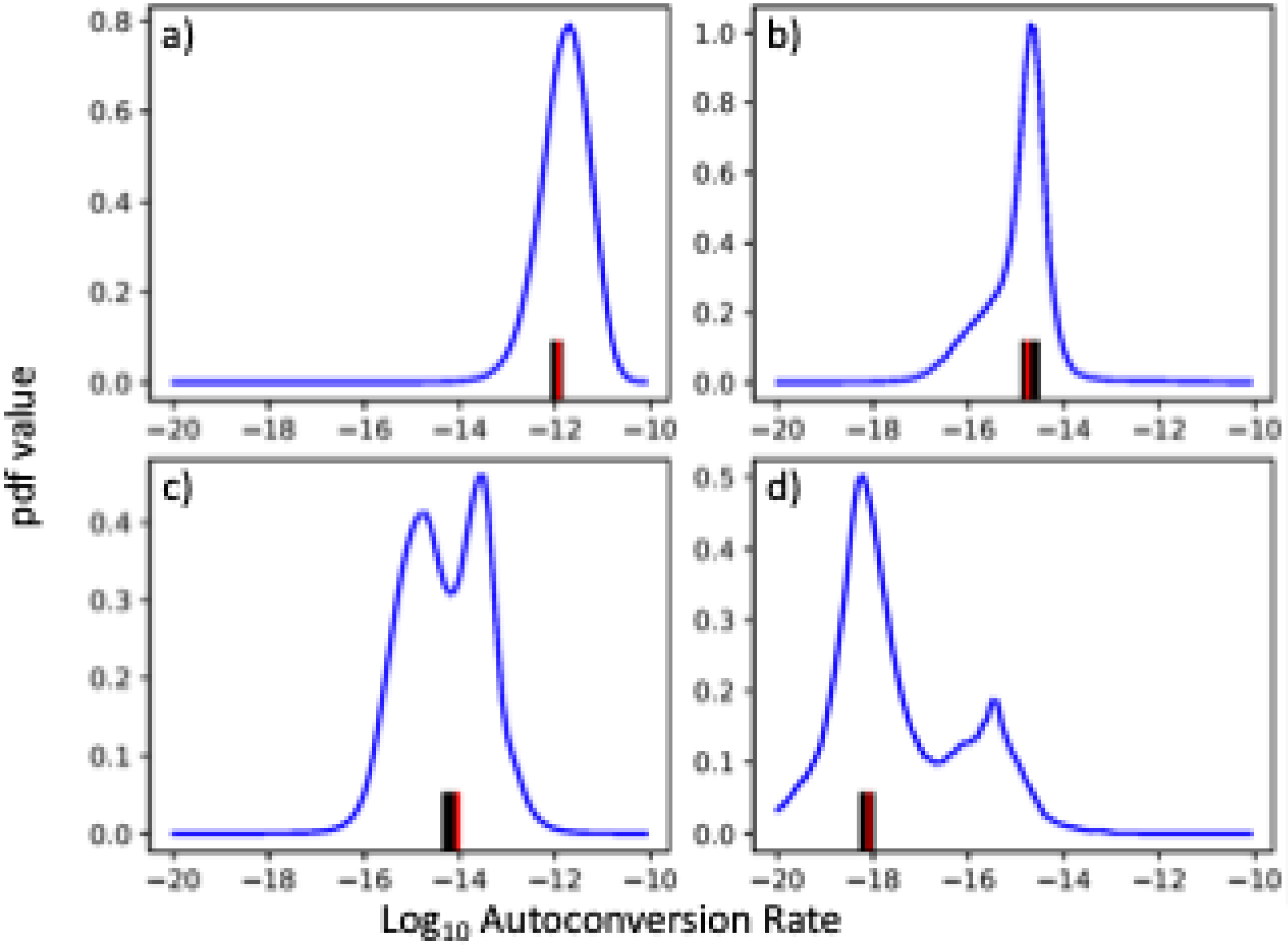
Example: autoconversion rate in stratocumulus clouds



two cloud drops form a rain drop

Examples of autoconversion rate pdf for 4 different input vectors

- Total uncertainty pdfs
- From costfunction minimum
- 20 Bagging output samples (without proper importance weighting…)



Data-assimilation for machine learning DA

Bayes Theorem:

$$p(x|y) = \frac{p(y|x)}{p(y)} p(x)$$

Transforms the prior pdf into the posterior pdf.

1) For linear and weakly nonlinear problems, this leads to EnKF, 3DVar and 4DVar, and hybrids.

2) In the nonlinear data-assimilation problem, we typically work with samples.
   Reformulate the data assimilation as:
   How to move samples from the prior to samples from the posterior?
   And how to do this fast and accurately? (or optimally?)

A whole suite the methods:

- Entropic optimal transport
  - Deterministic mapping that minimizes the (quadratic) cost of moving probability mass from prior to posterior. Entropy regularization allows the Sinkhorn algorithm
- Schroedinger bridge
  - Finds the stochastic map that is closest to Brownian motion in KL sense
- ‘Physical nudging’
  - Deterministic. ‘Nudging’ and synchronization are special cases
- Langevin-based Monte-Carlo
  - Particle Flow filters/smoothers: One chain for all particles, leads to interacting particles
  - ML: Generative diffusion: one chain for each member, *but how to bring in the likelihood*?
- ML: Stochastic interpolants generalizes Optimal transport, Schroedinger bridge, and generative diffusion. Relies on samples from prior and posterior.

Langevin-based Monte-Carlo and generative diffusions

The Langevin equation is known to generate samples from **any pdf *p(x)***, starting from samples from any other pdf.

It can be generalized as

$$dx = (D + Q) \cdot \nabla \log p(x) dt + \nabla \cdot (D + Q) dt + \sqrt{2} D^{1/2} dW$$

for any positive semi-definite matrix D and antisymmetric matrix Q.

This methodology is used in generative diffusion ML models as follows:

Choose $D = \sigma^2(t) I$ and $Q = 0$ and use a standard Gaussian as target pdf, such that

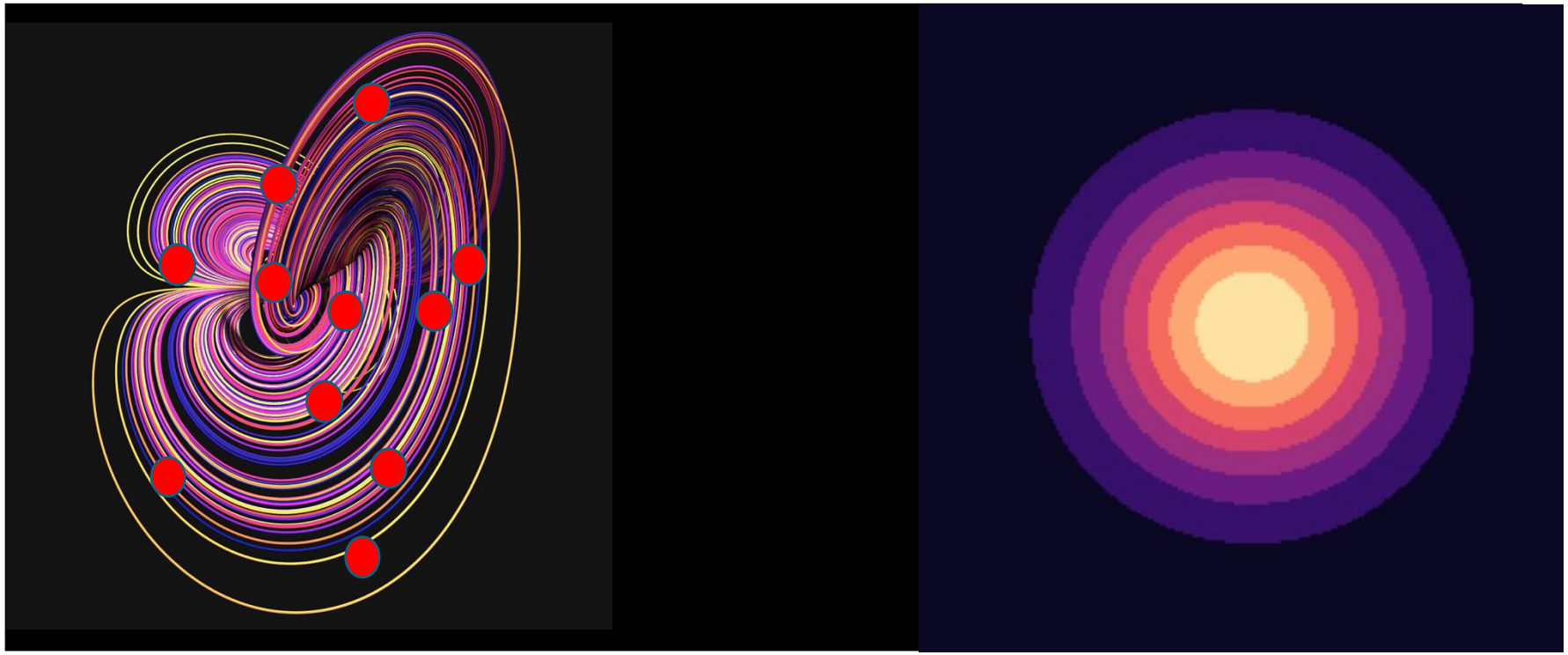
$$\nabla \log p_G(x) = -x$$

Hence, simply add random white noise at every pseudo timestep or iteration.

Machine learning: Diffusion-based generative methods

Step 1: Transform prior ensemble members to samples from standard Gaussian

Prior pdf
$p(x)$



Multivariate standard Gaussian

The transformation is done by adding Gaussian noise to the samples in an iterative way, and add a forcing towards the standard Gaussian.

Langevin dynamics to move prior samples to Gaussian samples

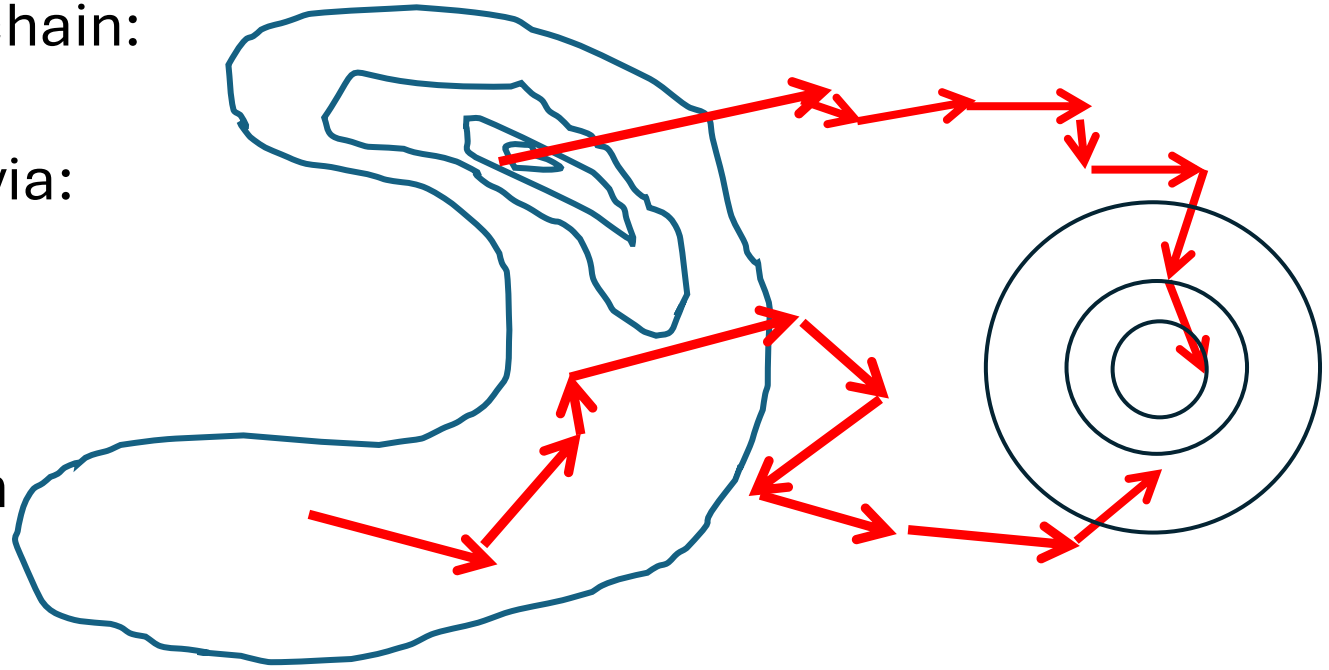
For each particle we will have a Markov chain:

The MC evolves forward in pseudo time via:

$$dx^t = -bx^t dt + \sigma dW$$

(Remember that for a standard Gaussian

$$\nabla_x \log p(x) = -x \qquad)$$

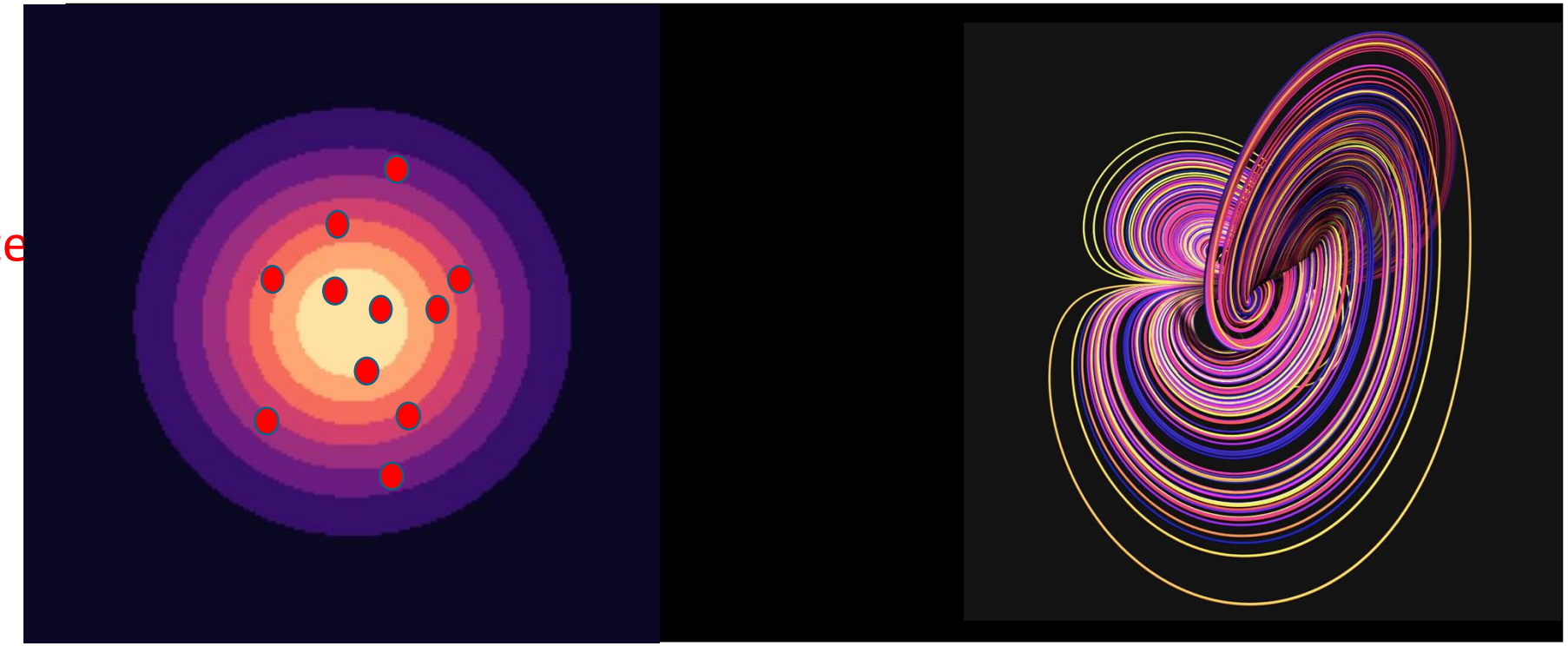


Since the SPDE is linear **the transition densities** $p(x^t|x^{t-1})$ **are all Gaussian!**

Machine learning: Diffusion-based generative modeling

Step 2: Transform standard Gaussian samples to prior samples

Multivariate standard Gaussian



Prior pdf

$p(x)$

This transformation is done by adding noise and **adding a forcing towards the prior.**
Learn this forcing towards the prior **or use analytical form**.

Use ML or use analytical form…

For each particle we will have a Markov chain:

$$dx^t = -bx^t dt + \sigma dW$$

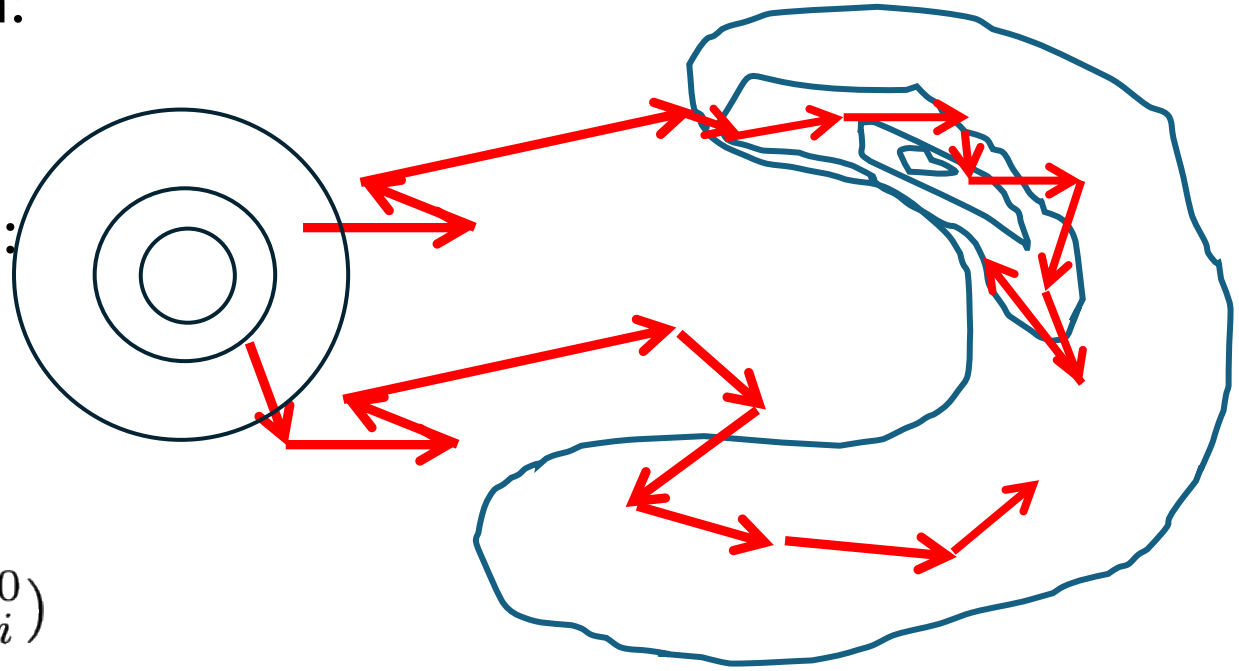
The MC evolves backward in pseudo time via:

$$dx^t = bx^t dt - \sigma^2 \nabla \log p^t(x^t) dt + \sigma dW$$

in which:

$$p^t(x^t) = \int p(x^t|x^0)p(x^0)\, dx^0 = \sum_{i=1}^{N} p(x^t|x_i^0)$$

and each $p(x^t|x_i^0)$ is Gaussian!
Hence, no need to learn it, fast to calculate directly.



But what about data assimilation?

Diffusion-based data assimilation

Remember, to generate samples from the prior we start from Gaussian samples and use

$$dx^t = bx^t dt - \sigma^2 \nabla \log p^t(x^t) dt + \sigma dW$$

We can also generate samples from the posterior by using instead:

$$dx^t = bx^t dt - \sigma^2 \nabla \log p^t(x^t|y) dt + \sigma dW$$

Since $\nabla \log p^t(x^t|y) = \nabla \log p^t(x^t) + \nabla \log p(y|x^t)$ we can write this as:

$$dx^t = bx^t dt - \sigma^2 \nabla \log p^t(x^t) dt - \sigma^2 \nabla \log p(y|x^t) dt + \sigma dW$$

but what is $p(y|x^t)$? We only know the likelihood for the physical state: $p(y|x) = p(y|x^0)$

Diffusion-based data assimilation

Add likelihood forcing $\nabla \log p(y|x)$ to the evolution equation, which will lead to posterior?

$$dx^t = bx^t dt - \sigma^2 \nabla \log p^t(x^t) dt - \sigma^2 h^t \nabla \log p(y|x^t) dt + \sigma dW$$

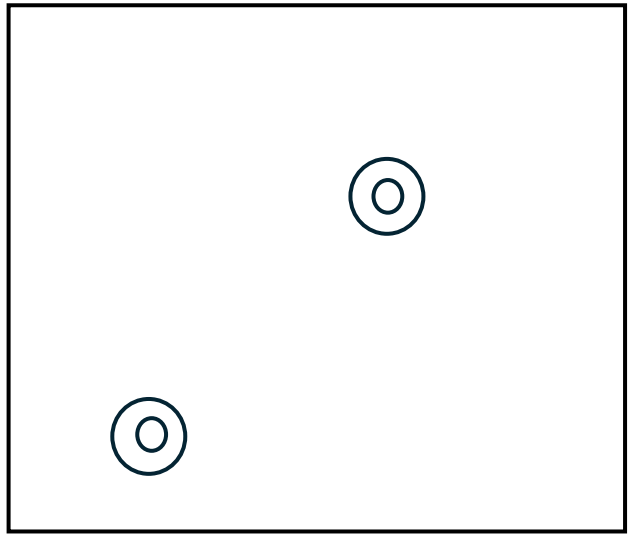
Prior pdf
$p(x)$

Posterior pdf
$p(x|y)$

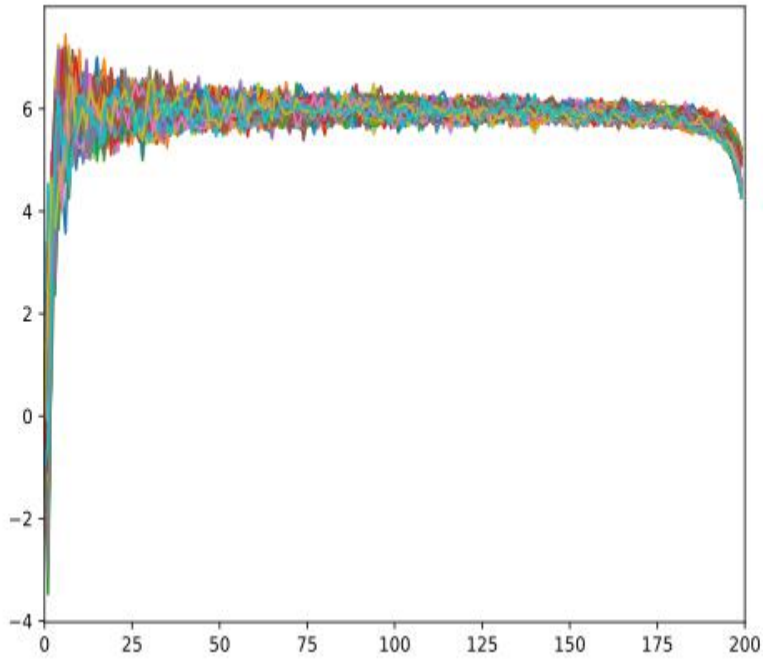
However, there is no principled way to weight the prior and likelihood forcings, yet…

Example: backward flow with extra likelihood term

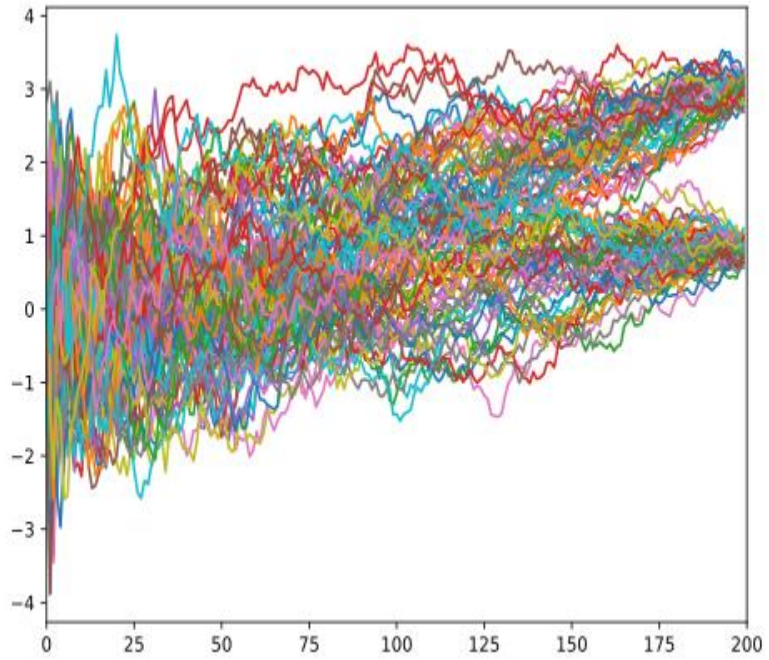
Observe first component, y=6, standard deviation 0.1. Use 100 particles.



Bimodal prior pdf



Observed component



Unobserved component

Backward flow from standard Gaussian to ‘posterior pdf’. Two issues:

- likelihood and prior forcing not ‘well balanced’, and
- no communication between state components

Indeed, this is not correct!

How to find the likelihood term ?

1) The likelihood follows the Kolmogorov backward equation:

$$\frac{\partial p(y|x^t)}{\partial t} = -\nabla_x \cdot \left[M(x^t)p(y|x^t)\right] - \frac{1}{2}\nabla_x \cdot \left[D \cdot \nabla p(y|x^t)\right]$$

with final condition $p(y|x^0)$.

When *M* is nonlinear, this equation must be solved numerically, expensive!

Use approximations (many available; this is a very old problem in DA).

2) Use Chapman-Kolmogorov:

$$p(y|x^t) = \int p(y, x^0|x^t)\, dx^0 = \int p(y|x^0, x^t)p(x^0|x^t)\, dx^0$$
$$= \int p(y|x^0)p(x^0|x^t)\, dx^0$$

and use approximations, e.g., ensemble representation for $p(x^0|x^t)$ (similar to EnKS).

Using ensembles to solve the likelihood term

Specifically, assume the joint pdf to be Gaussian. This leads to:

$$dx_t = -bx_t dt - \nabla_x \log p(x_t) dt + Force\, dt + \sqrt{2} D^{1/2} dW_t$$

in which:

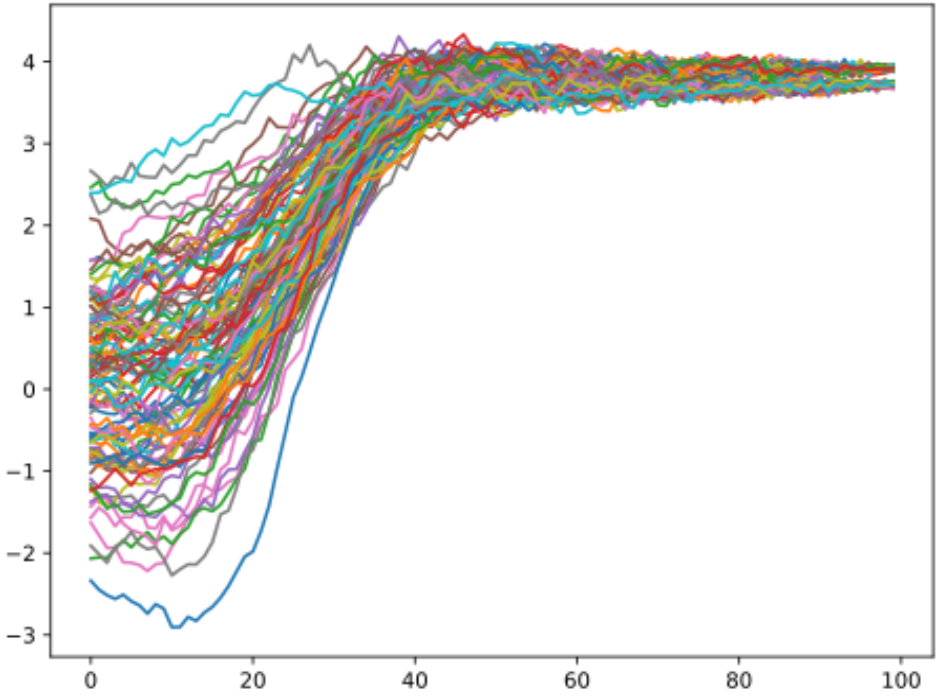
$$Force = -DC_{tt}^{-1} C_{tT} (C_{TT} - C_{Tt} C_{tt}^{-1} C_{tT})^{-1} \sum_{i=1}^{N} w_i (x_i^0 - \overline{x^0} + C_{Tt} C_{tt}^{-1} (x^t - \overline{x^t}))$$

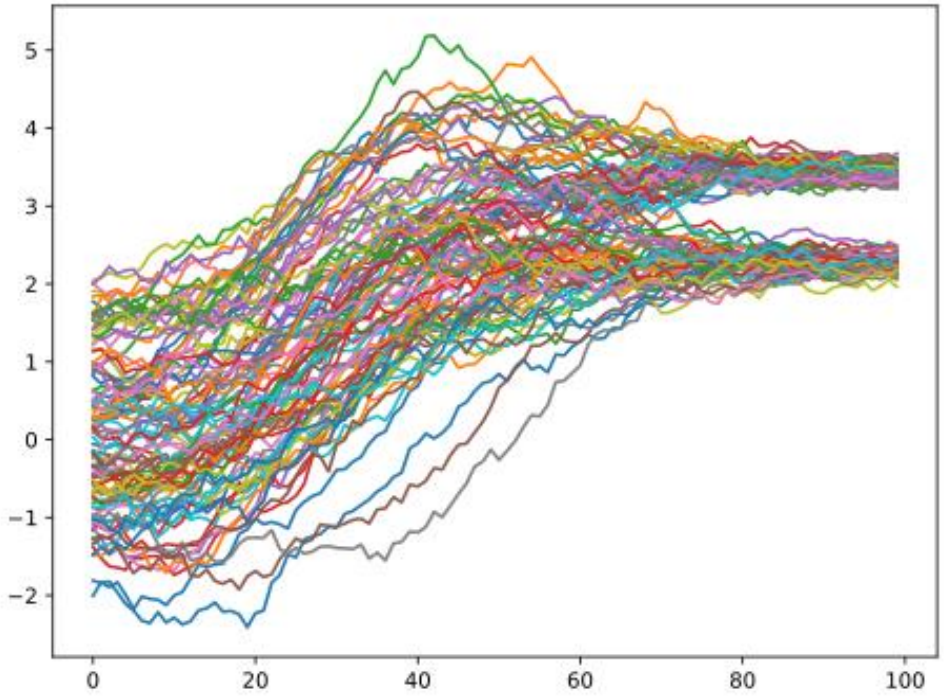
with $w_i \propto p(y|x_i^0)$

We can now use this equation for the evolution of particles in diffusion method, in which x_i^0 are the original prior particles!

Example: backward flow with extra likelihood term

Prior: 2-dimensional bimodal Gaussian with means (3,3) and (0.8,0.8), standard deviations 0.1.
Observe first component, y=4, standard deviation 0.1. Use 100 particles.





The correct answer!

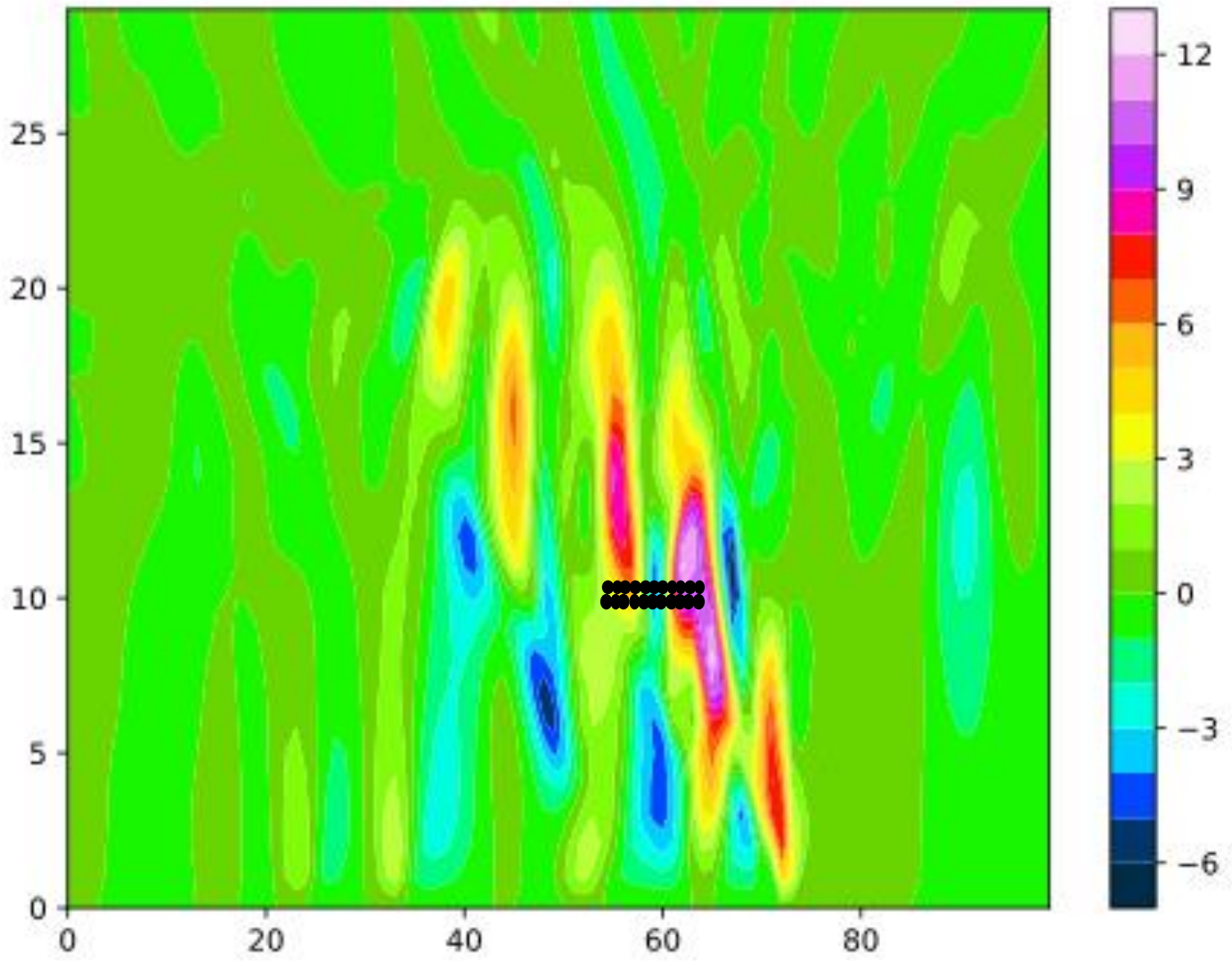
(Note we used time stepping ideas from stochastic interpolants.)

Example: Squall line

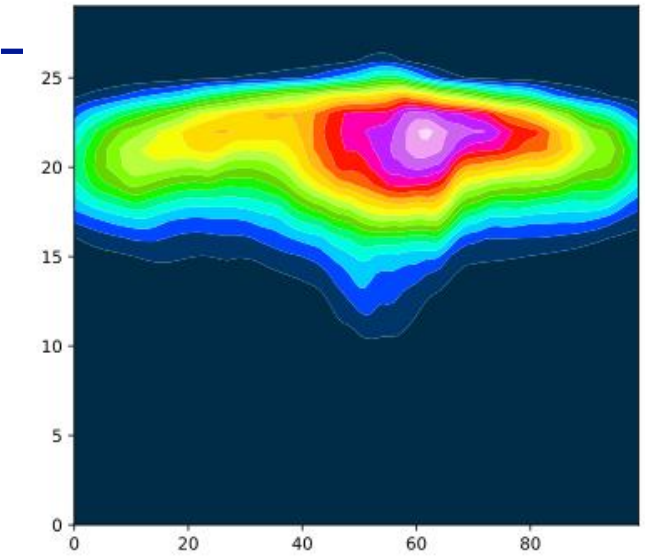
Application on squall-line case, using Cloud Model 1 (CM1)

- dx = 1 km, dz = 500 m
- Observe w at 20 grid points
- 50 particles
- Obs error standard deviation 0.2
- Extremely nonlinear!

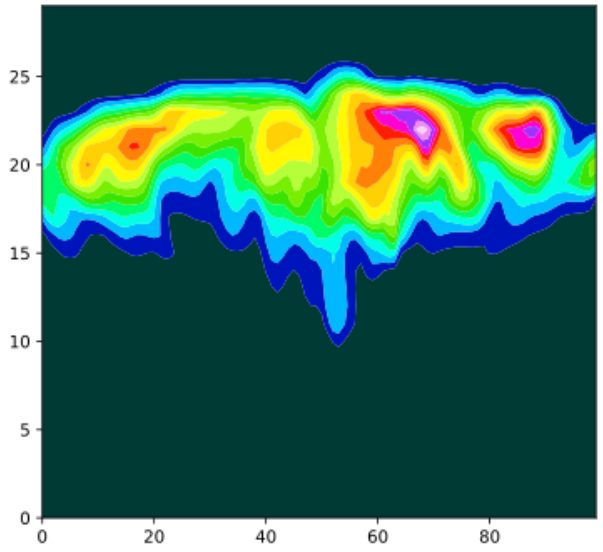
Vertical velocity.
Black dots are observation locations.



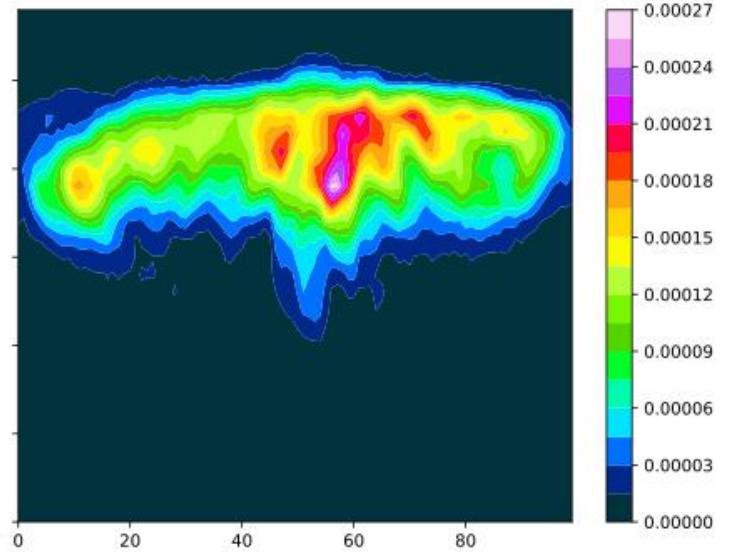
Results on ice and snow content



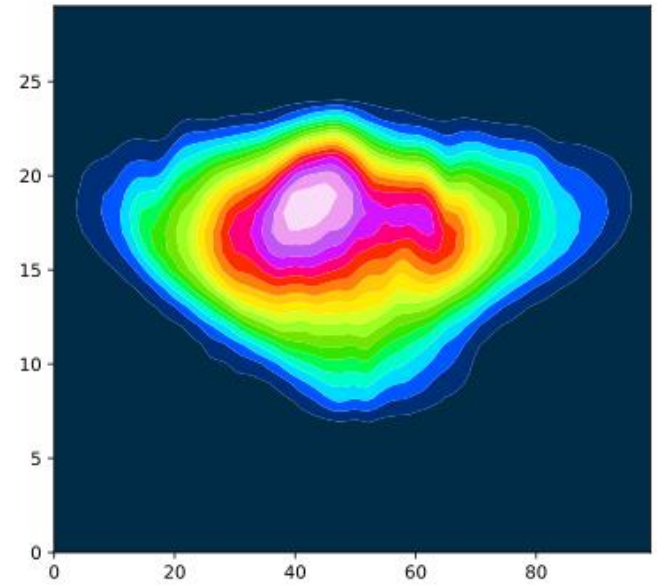
Prior mean

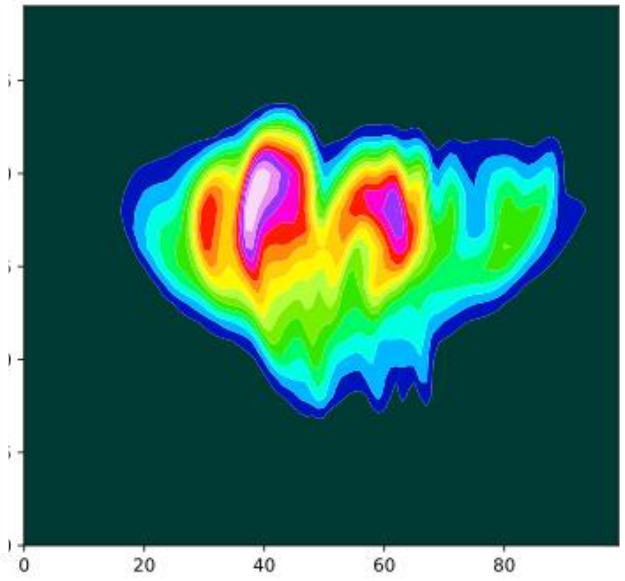


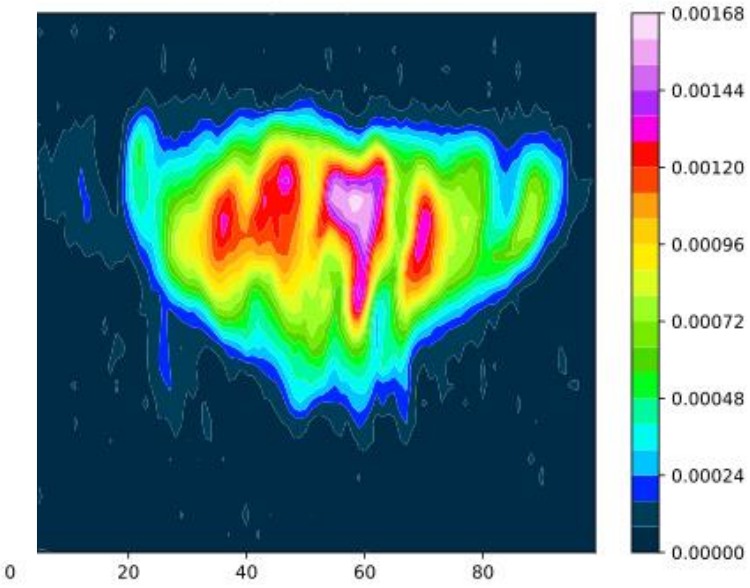
True state



posterior state







Conclusions

- Uncertainty quantification is essential for ML to be fully accepted by science and society, and makes it more robust to outliers.
- An equation can be derived for the **full uncertainty pdf of an output**, based on all uncertainties in the system: the **fundamental equation for machine learning.**
- **Techniques from data assimilation** (or Bayesian Inference) can be used to find efficient ways to approximate this overall expression.
- Standard UQ methods for ML are incomplete or incorrect (Bagging, MC-Dropout, Deep Ensembles, Conformal Prediction, Evidential DL, quantile regression,…).
- Nonlinear data assimilation can be formulated as: How to generate independent samples from the posterior pdf. Generative ML methods use this idea.
  - Generative diffusion methods include likelihood in ad-hoc manner
  - Combining diffusive generative methods with **techniques from DA** largely solves this.
- The fields are merging!