ENM 540: Data-driven modeling and probabilistic scientific computing

Summary

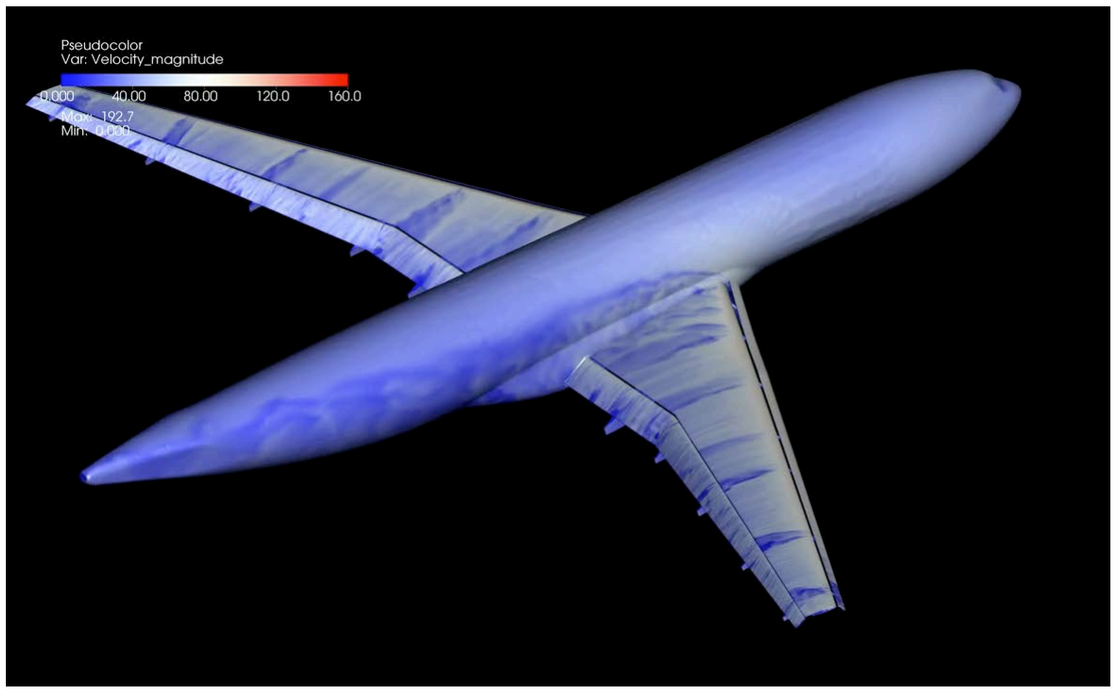
Paris Perdikaris

April 19, 2018

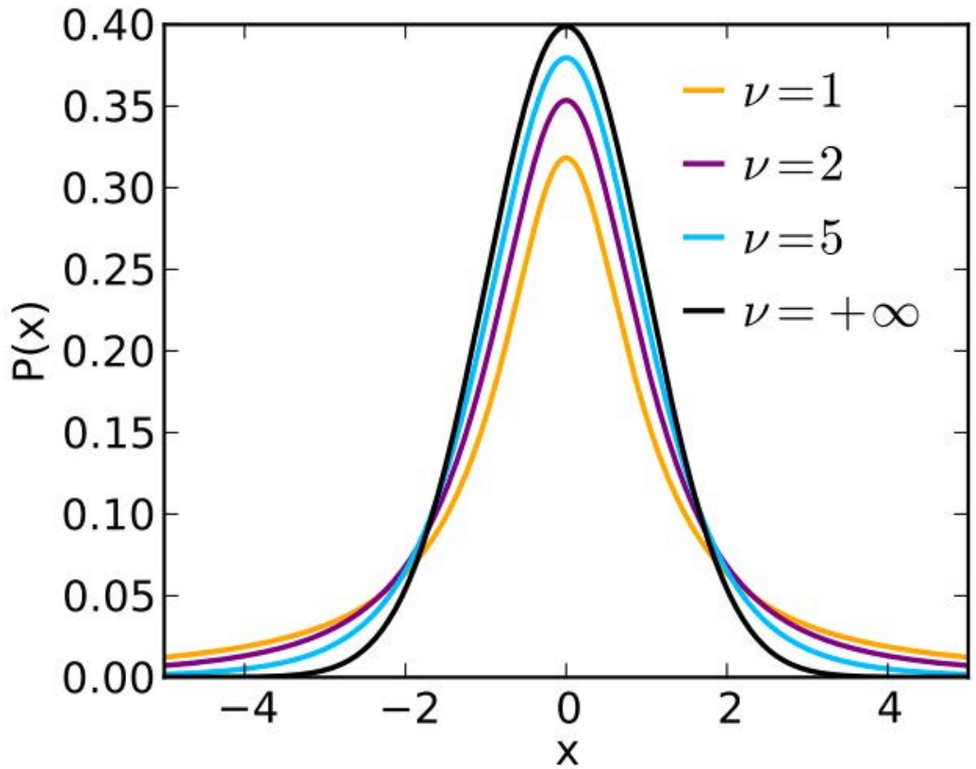
Data-driven modeling

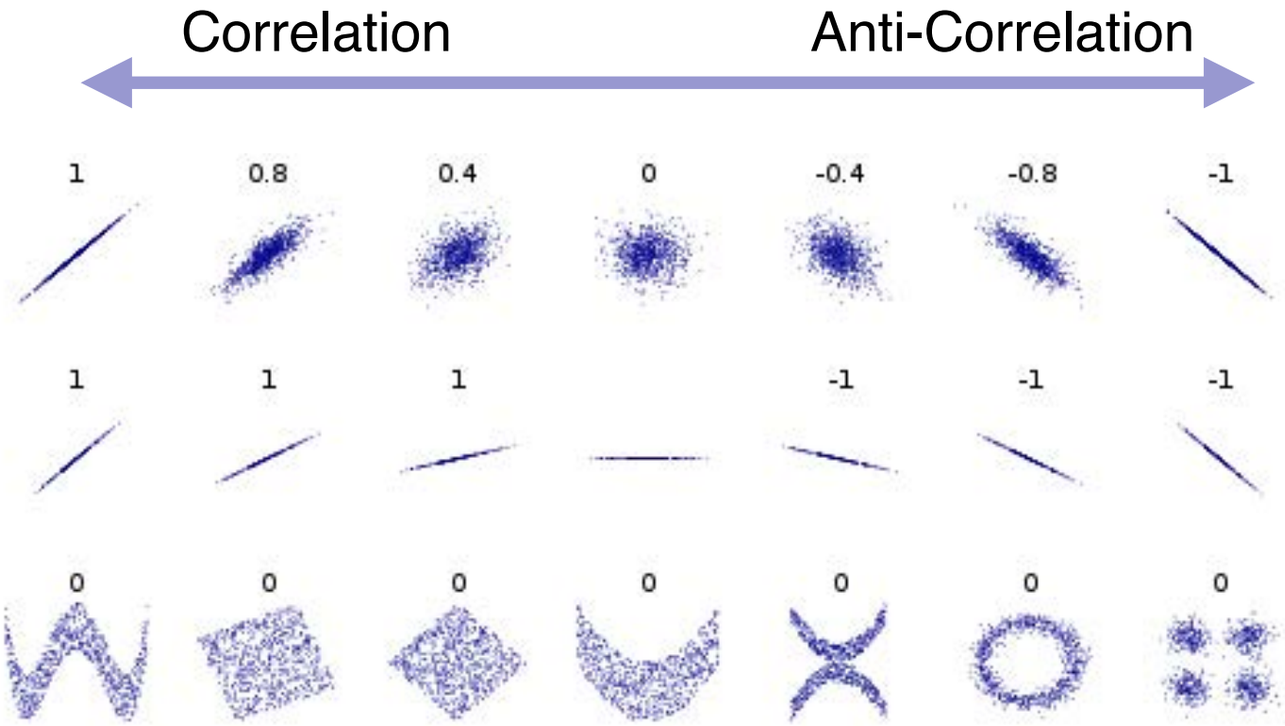
Machine learning

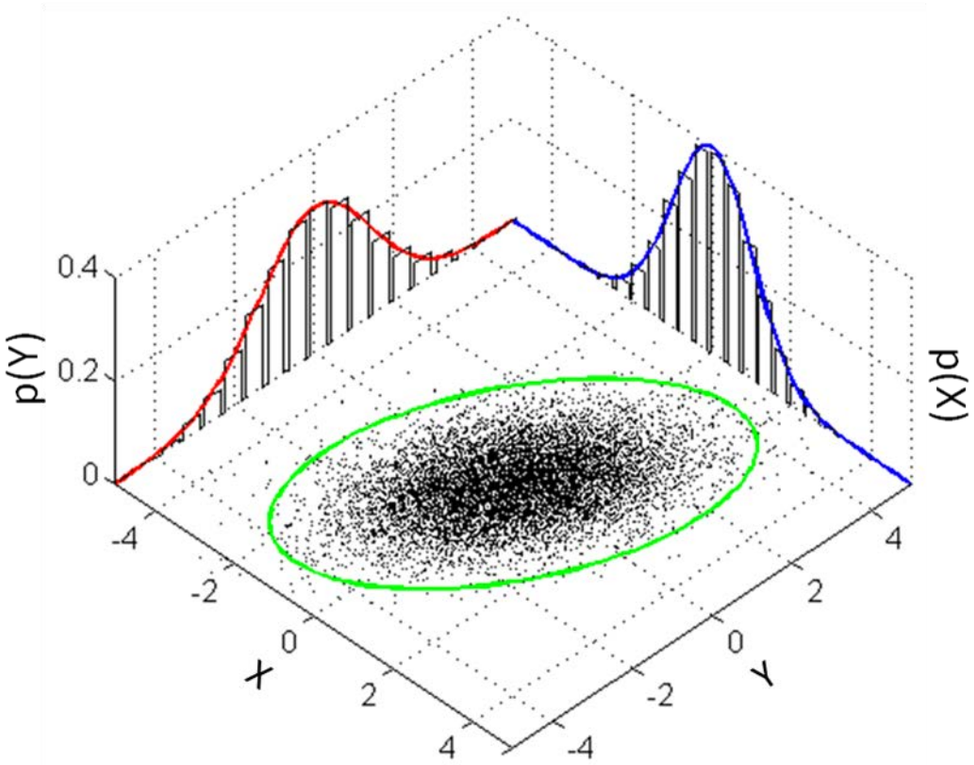
CSE

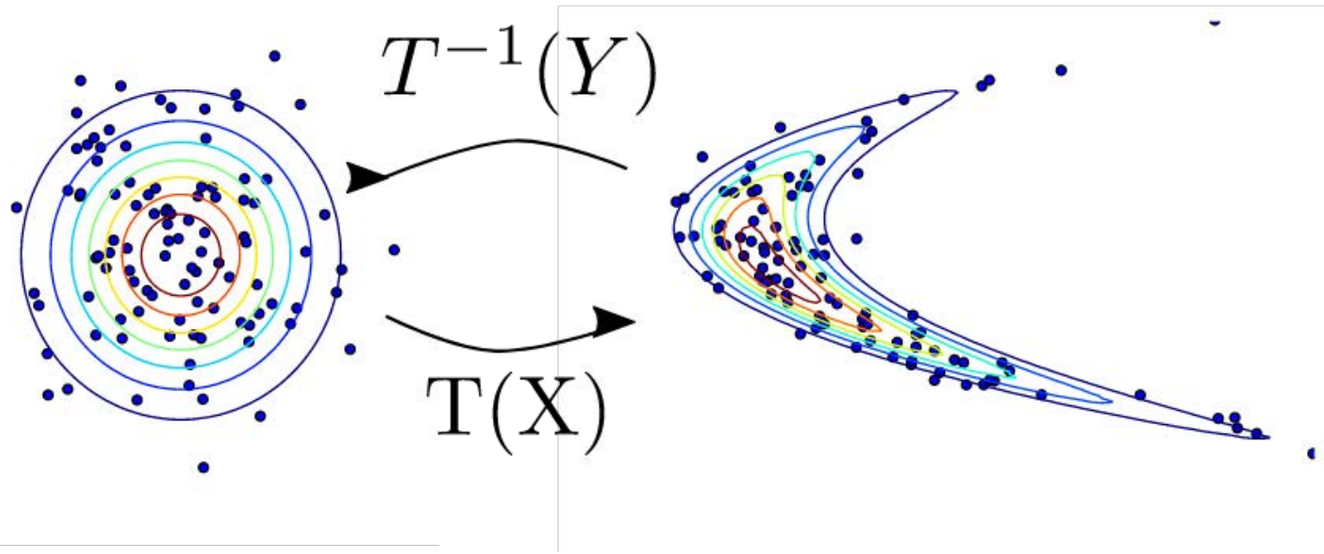


A probabilistic perspective









A probabilistic perspective

Classical models can be rigorously formulated from a probabilistic standpoint!

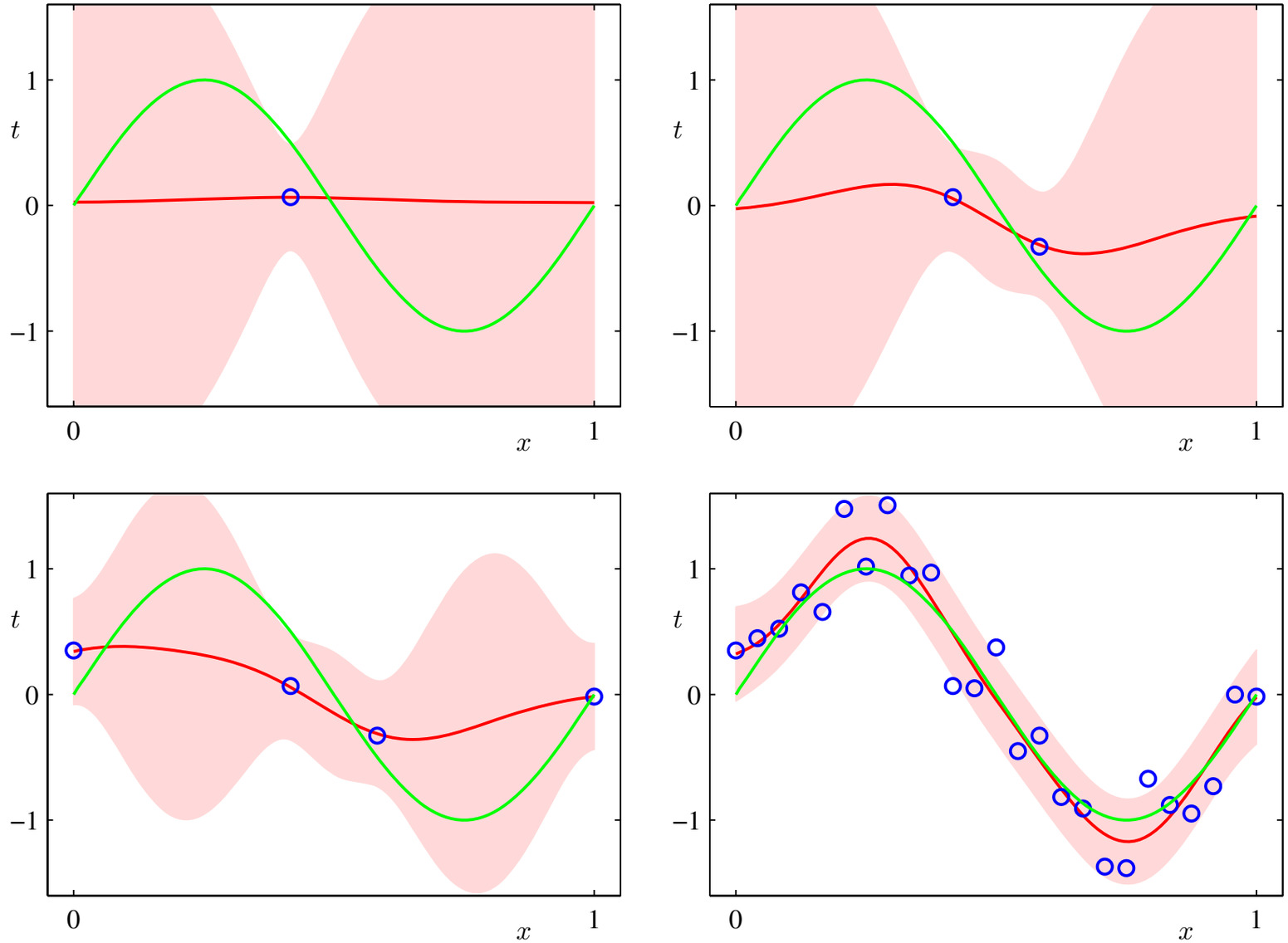


Figure 3.8 Examples of the predictive distribution (3.58) for a model consisting of 9 Gaussian basis functions of the form (3.4) using the synthetic sinusoidal data set of Section 1.1. See the text for a detailed discussion.

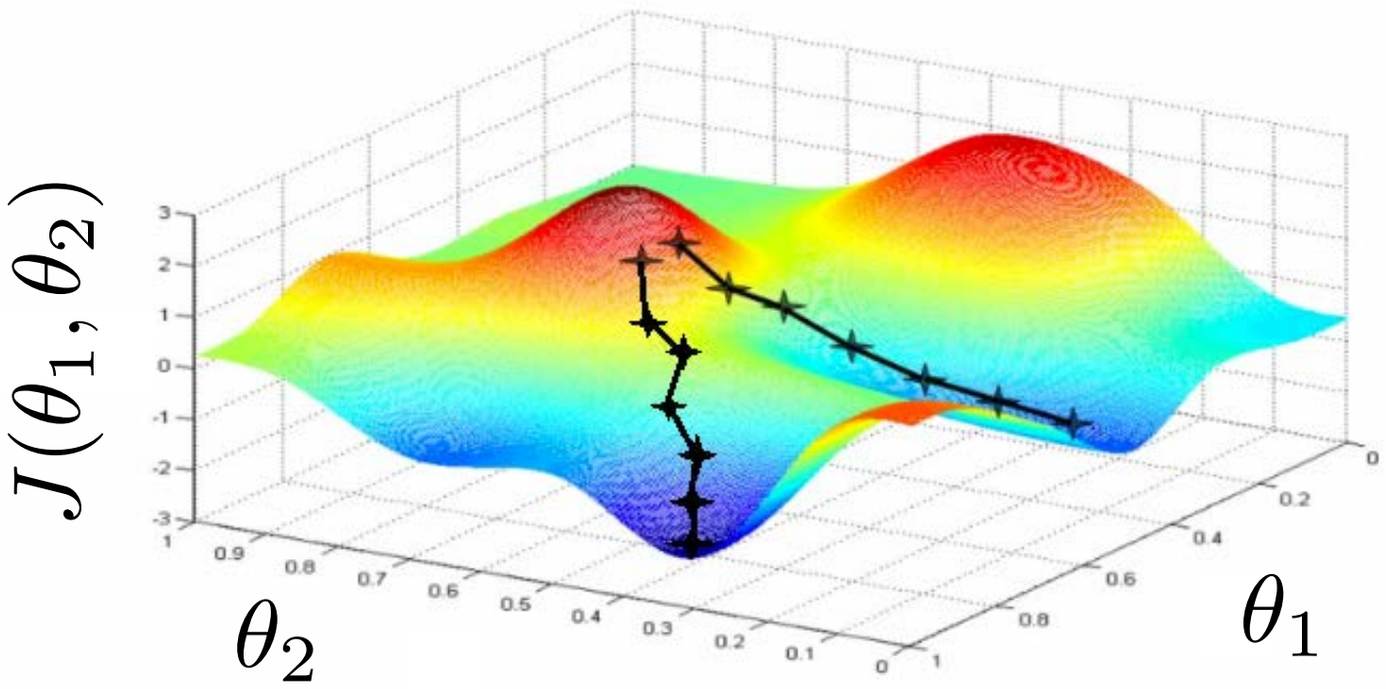
$$\boldsymbol{y} = w^T \phi(\boldsymbol{x}) + \epsilon$$

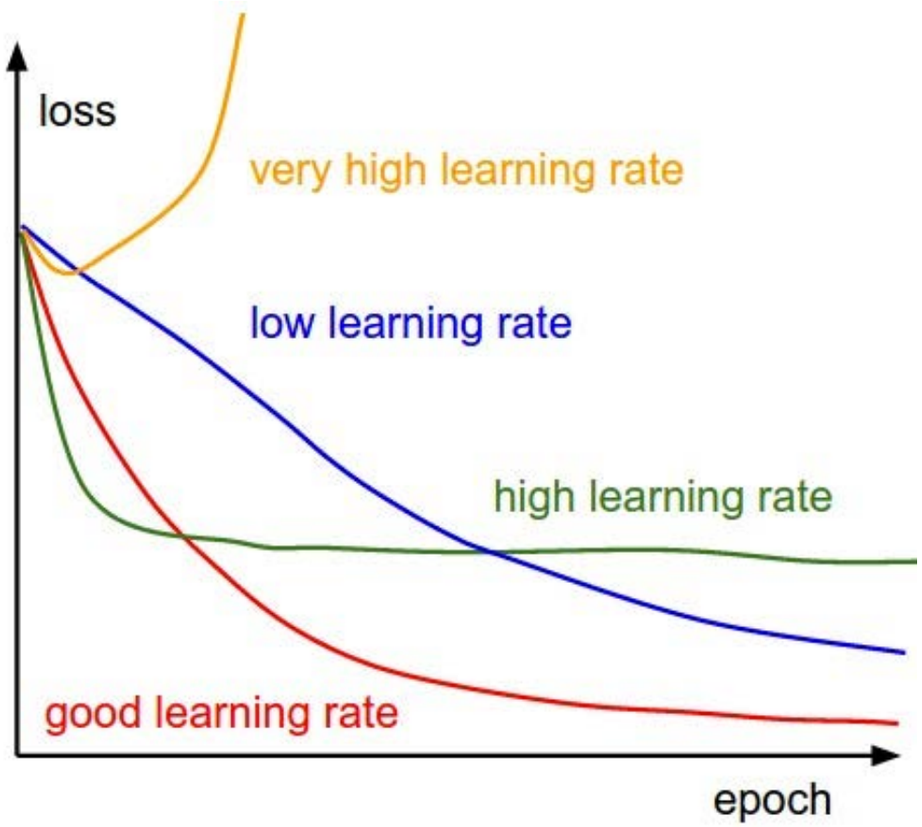
Optimization

Modern statistics and ML pose great challenges on optimization and define new exciting directions for research!

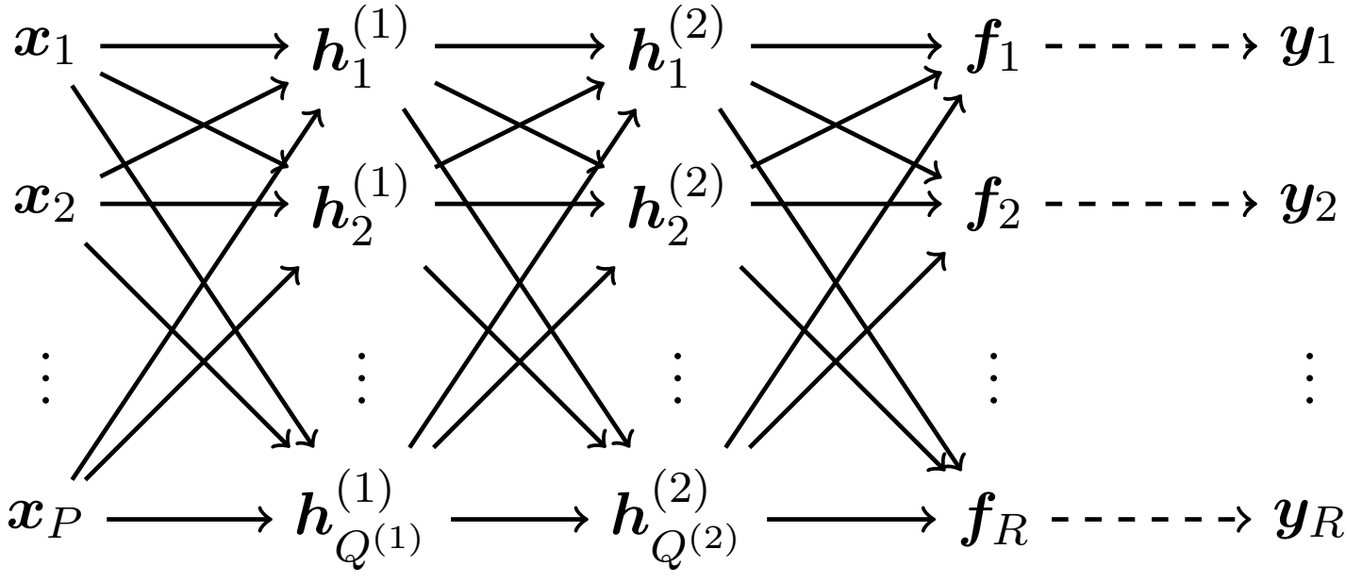
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} J(\boldsymbol{\theta})$$

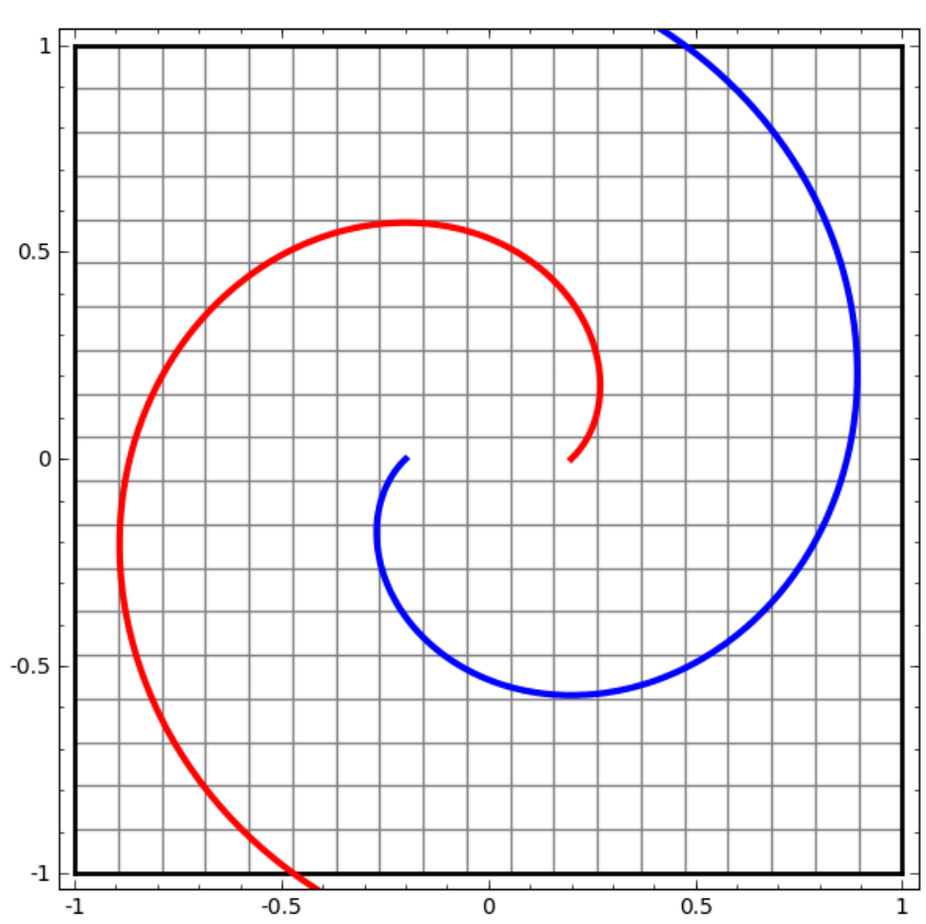
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$





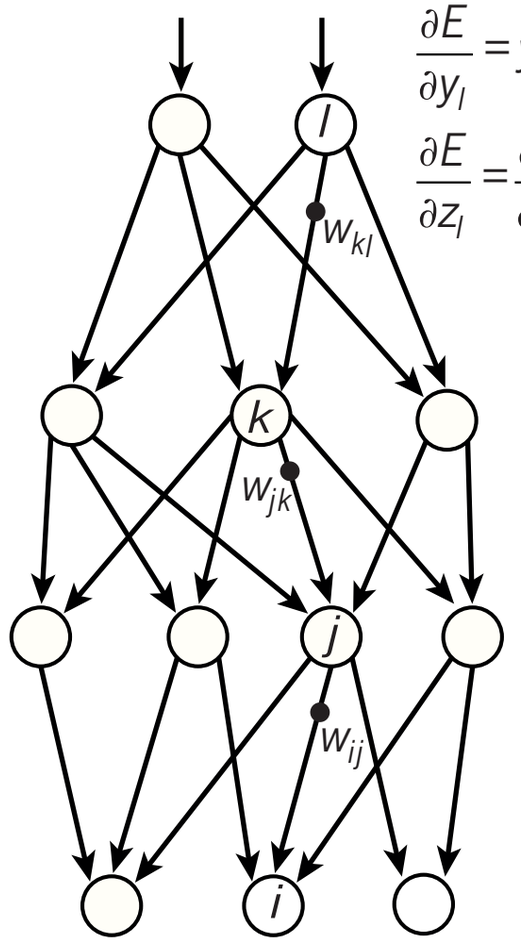
Neural networks





$$\frac{\partial E}{\partial y_k} = \sum_{l \,\varepsilon\, \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \,\varepsilon\, \text{H2}} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

Automatic differentiation

The chain rule, forward and reverse accumulation [edit]

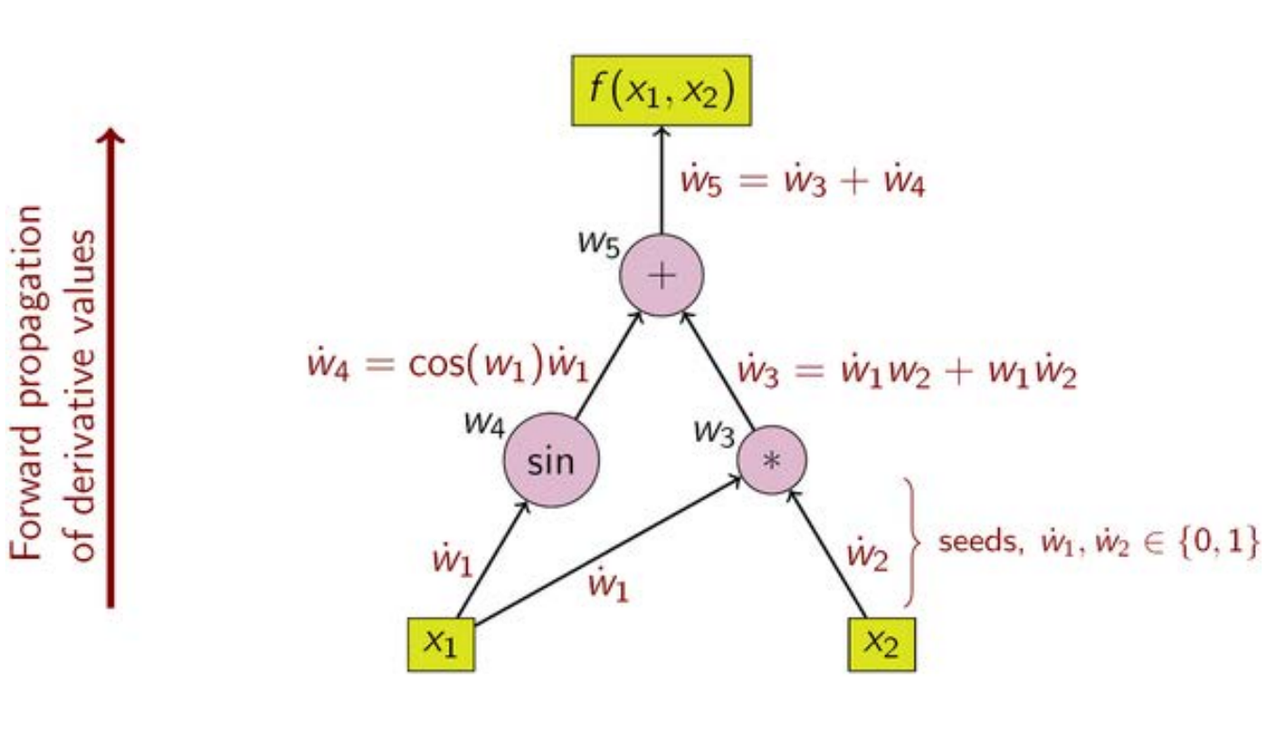
Fundamental to AD is the decomposition of differentials provided by the chain rule. For the simple composition $y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$ the chain rule gives

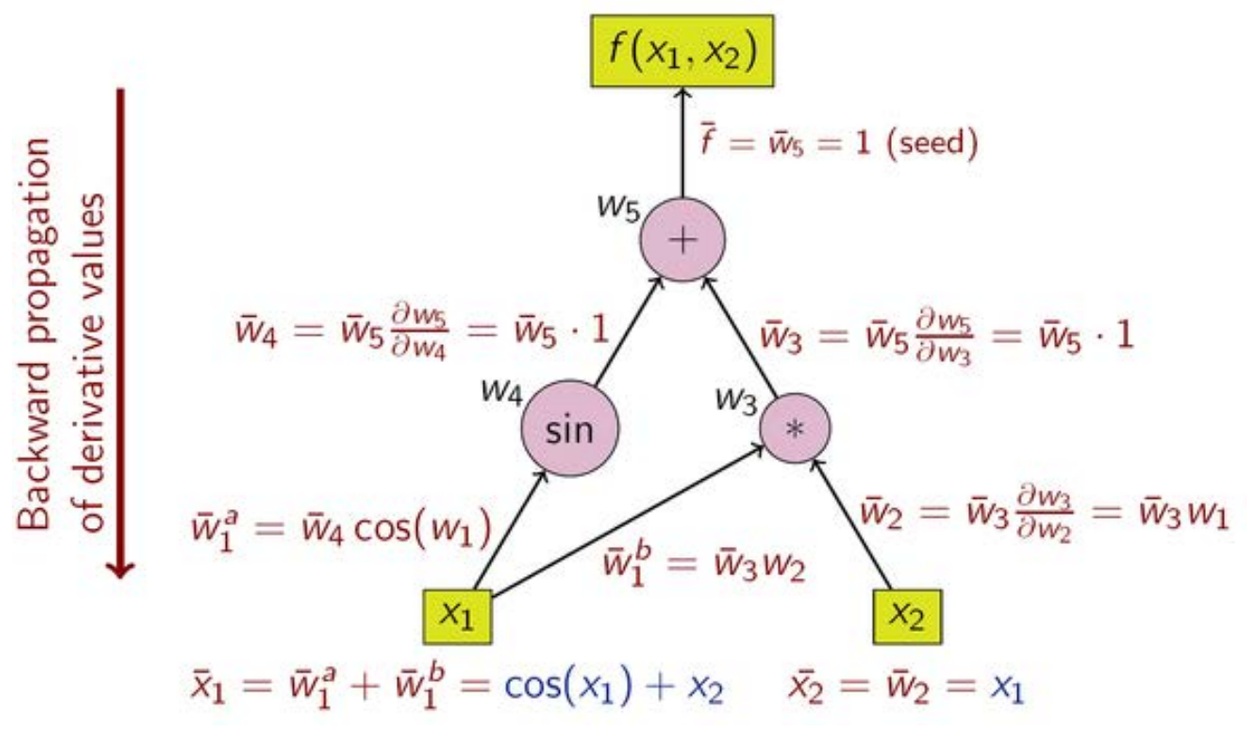
$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx}$$

Usually, two distinct modes of AD are presented, **forward accumulation** (or **forward mode**) and **reverse accumulation** (or **reverse mode**). Forward accumulation specifies that one traverses the chain rule from inside to outside (that is, first compute dw_1/dx and then dw_2/dx and at last dy/dx), while reverse accumulation has the traversal from outside to inside (first compute dy/dw_2 and then dy/dw_1 and at last dy/dx). More succinctly,

1. **forward accumulation** computes the recursive relation: $\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}}\frac{dw_{i-1}}{dx}$ with $w_3 = y$, and,
2. **reverse accumulation** computes the recursive relation: $\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}}\frac{dw_{i+1}}{dw_i}$ with $w_0 = x$.

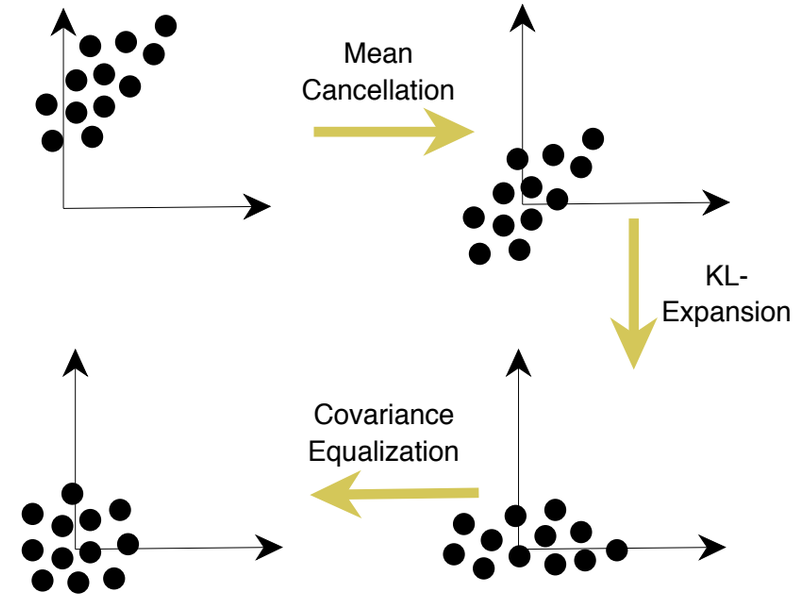
Example: $z = f(x_1, x_2) = x_1 x_2 + \sin x_1$





Practical tips

- Data normalization
- Stochastic optimization
- Network initialization
- Batch normalization
- Dropout



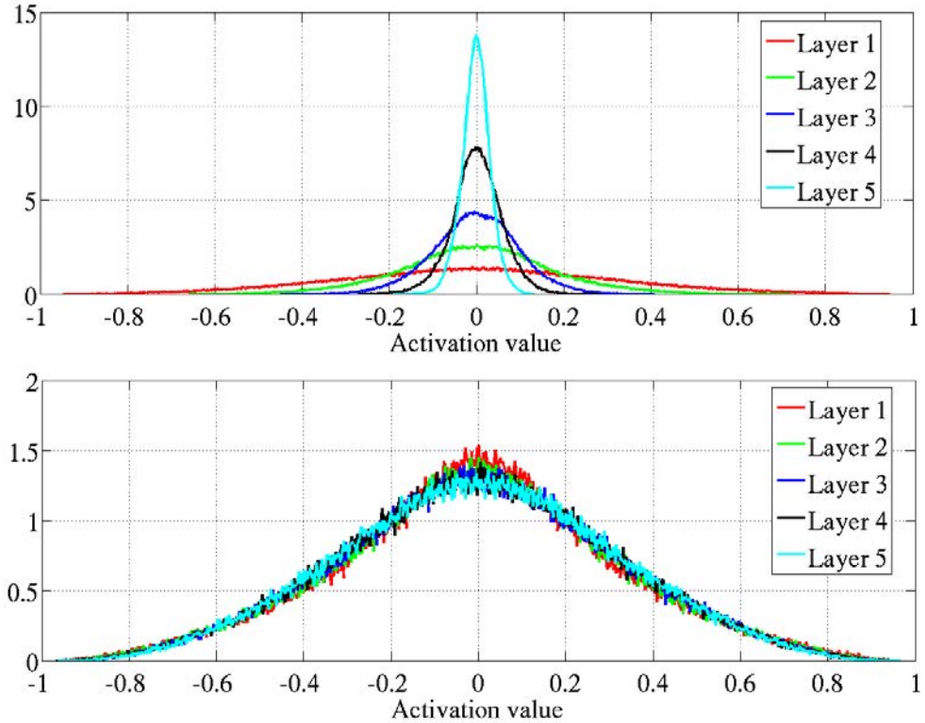


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

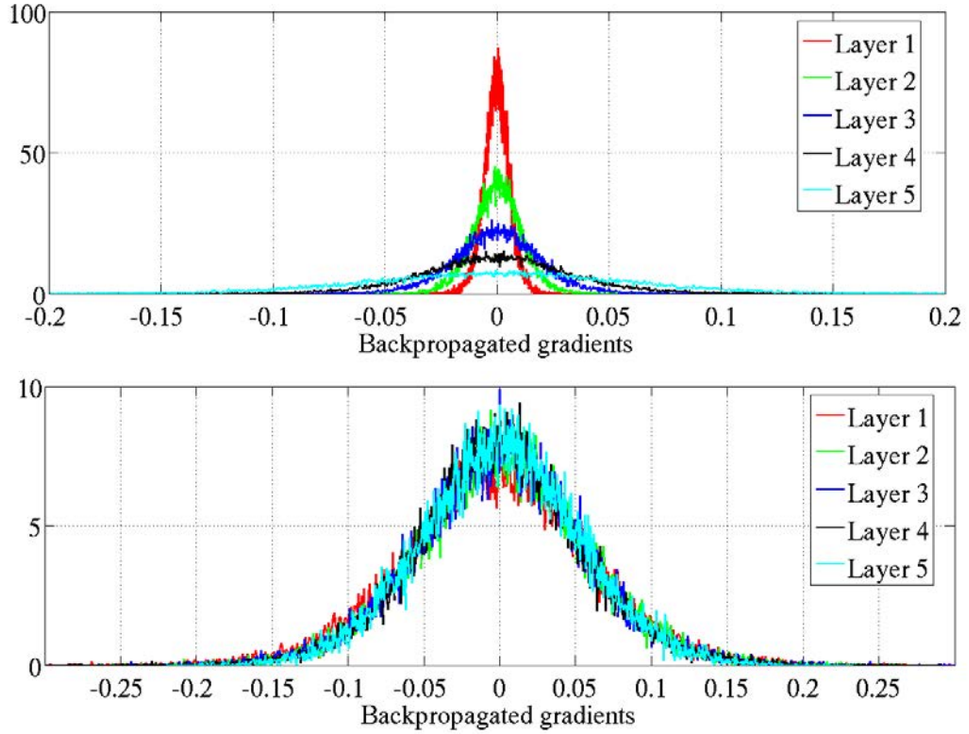


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Applications in scientific computing

Example: Constraining the outputs of NNs to satisfy differential equations

$$
\begin{aligned}
&u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\
&u(0, x) = -\sin(\pi x), \\
&u(t, -1) = u(t, 1) = 0.
\end{aligned} \tag{3}
$$

Let us define $f(t, x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

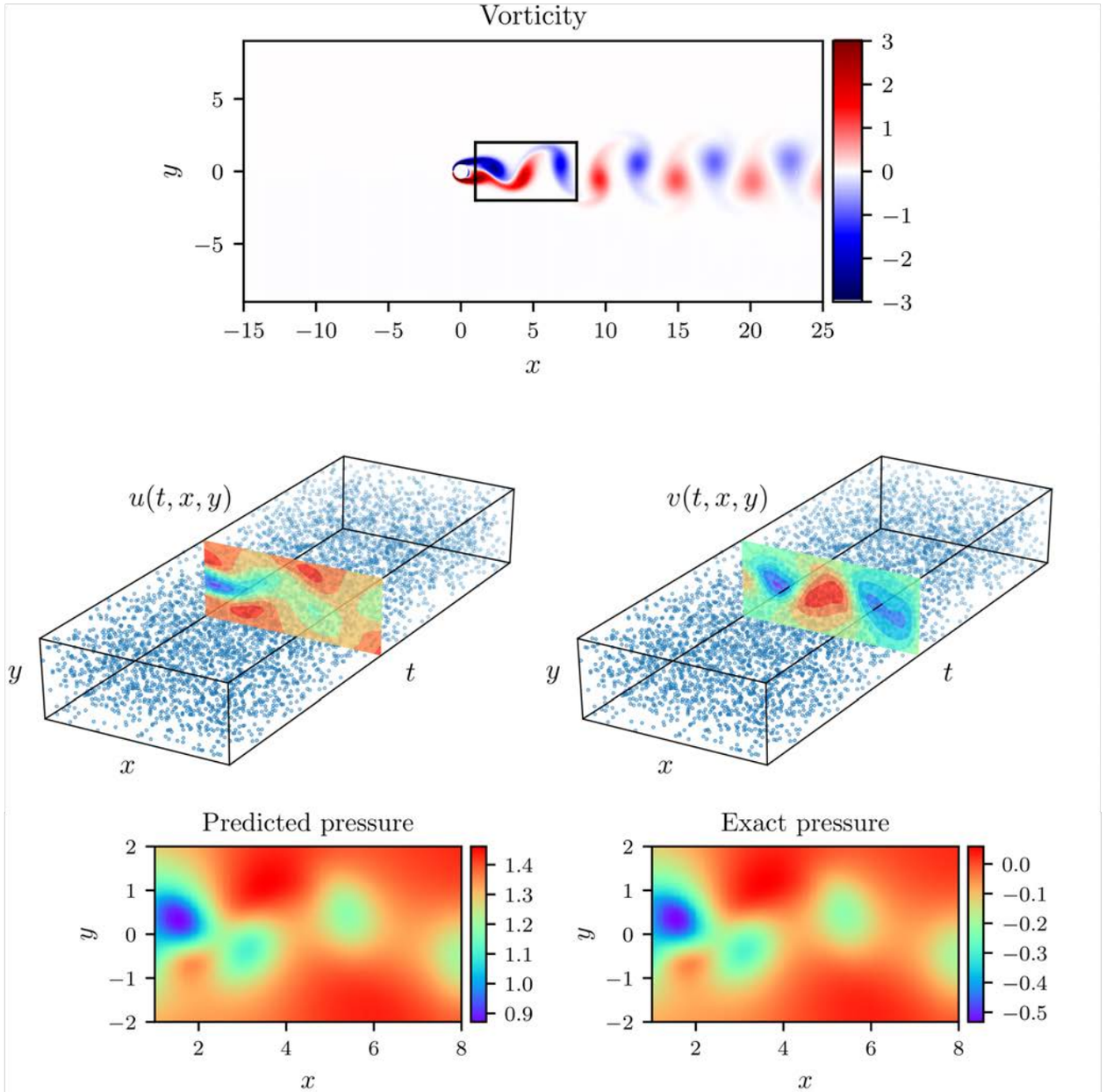
```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the *physics informed neural network* $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

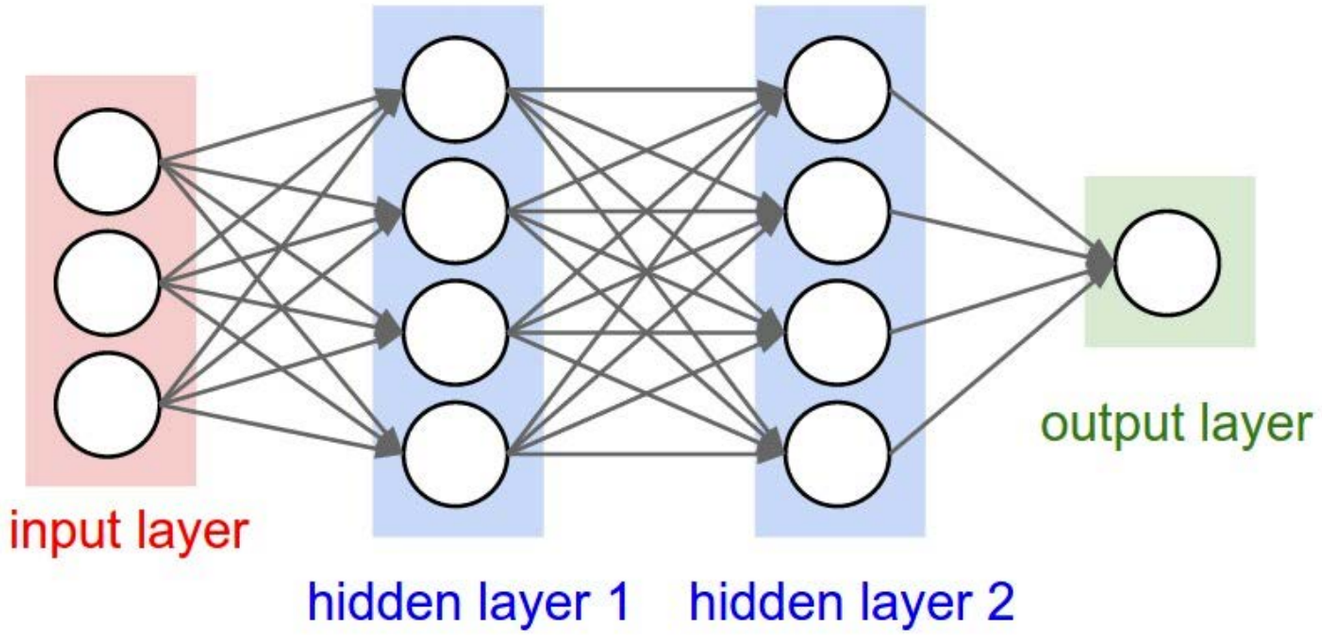
Applications in scientific computing

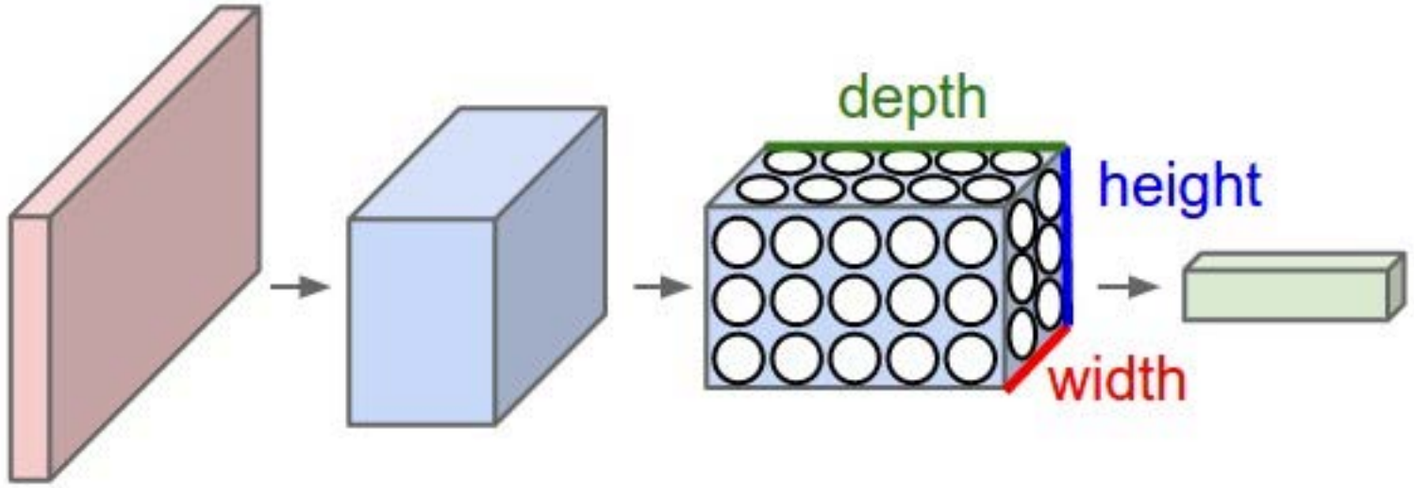
Example: Discovering new physical models and calibrating parameters from data



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Convolutional neural networks

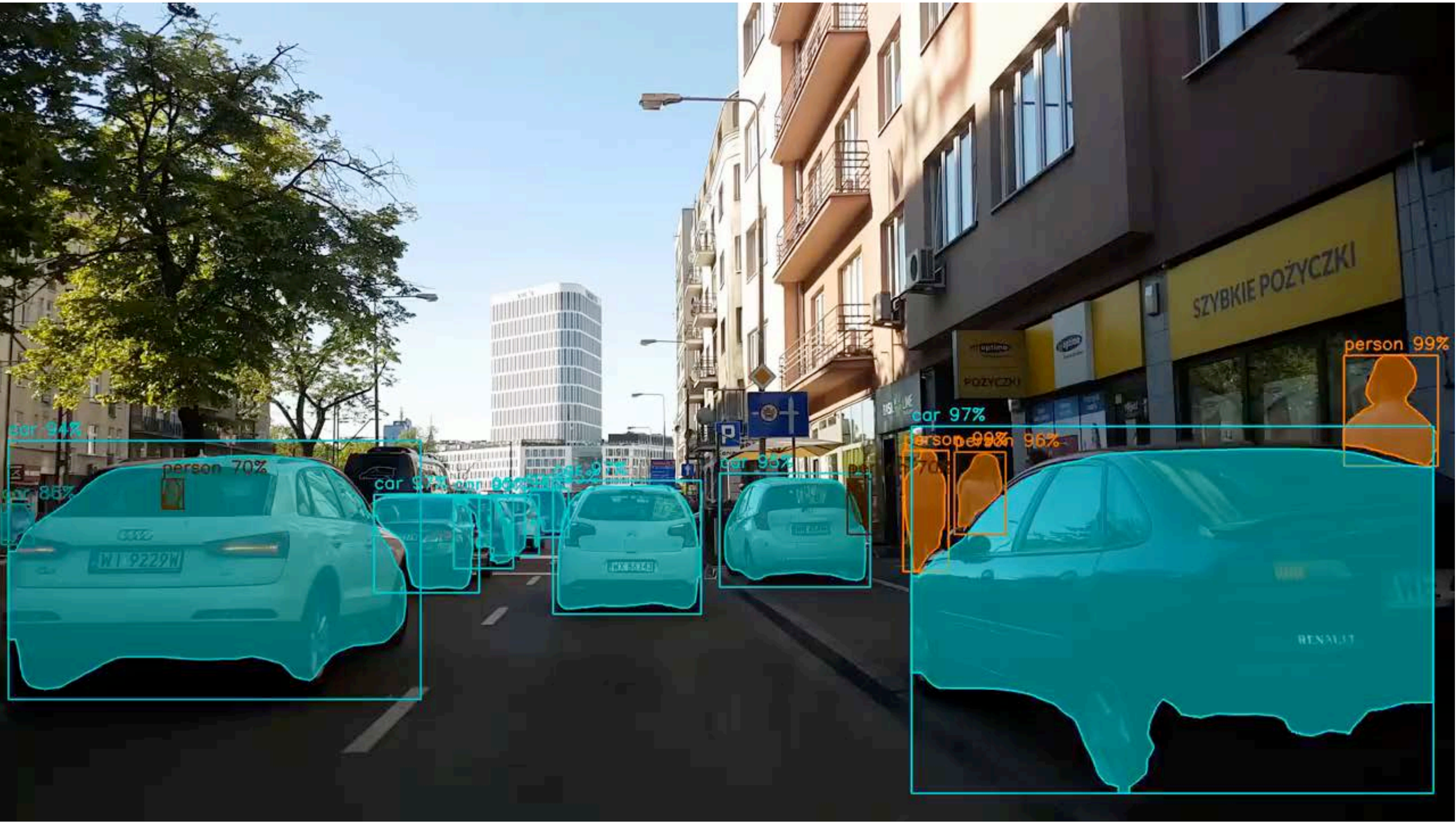




Top: A regular 3-layer Neural Network.
Bottom: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

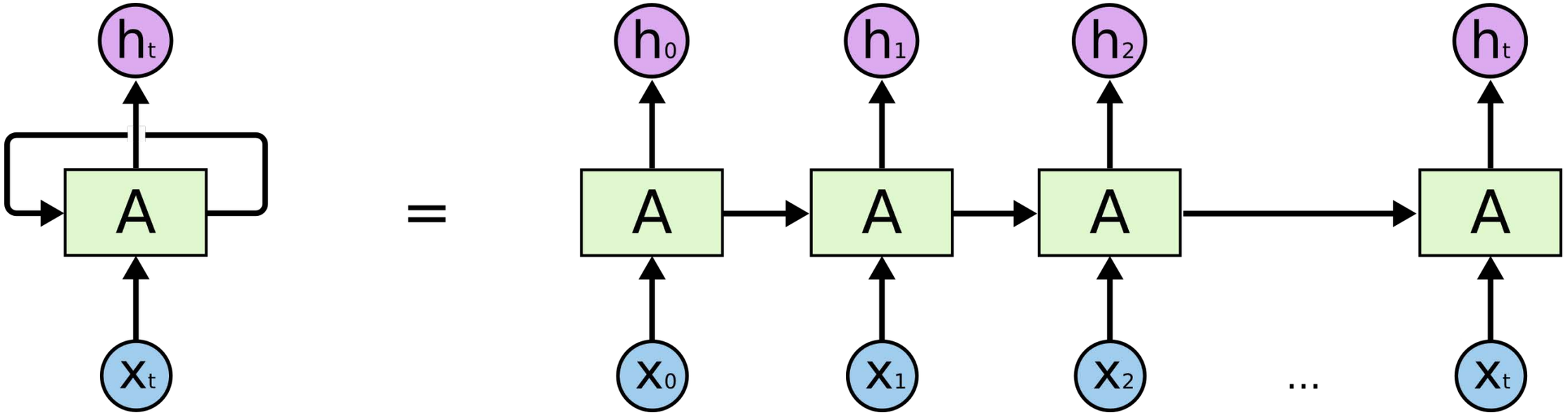
http://cs231n.github.io/convolutional-networks/

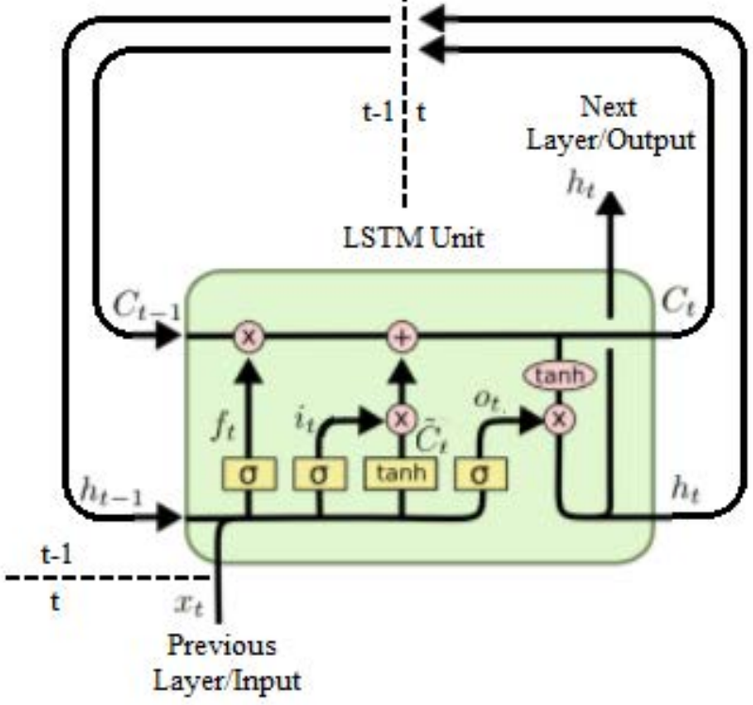
Convolutional neural networks



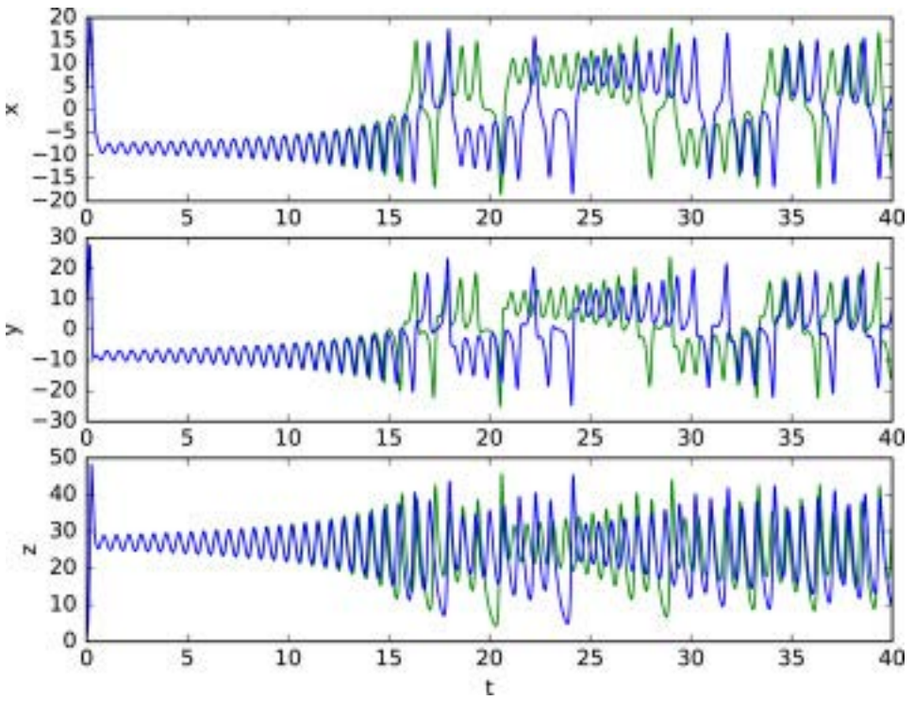
He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017, October). Mask r-cnn. 2017 IEEE International Conference on Computer Vision (ICCV).

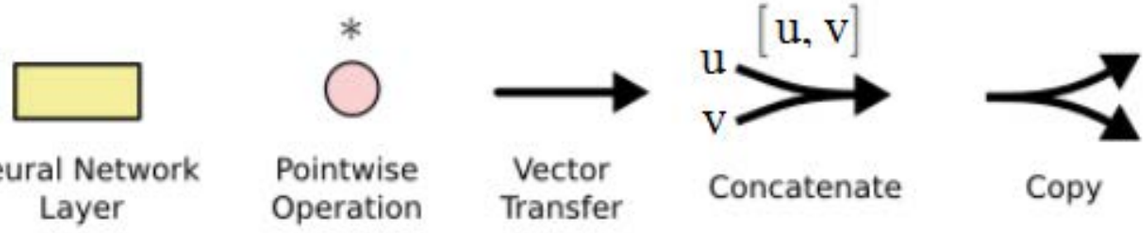
Recurrent neural networks





$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

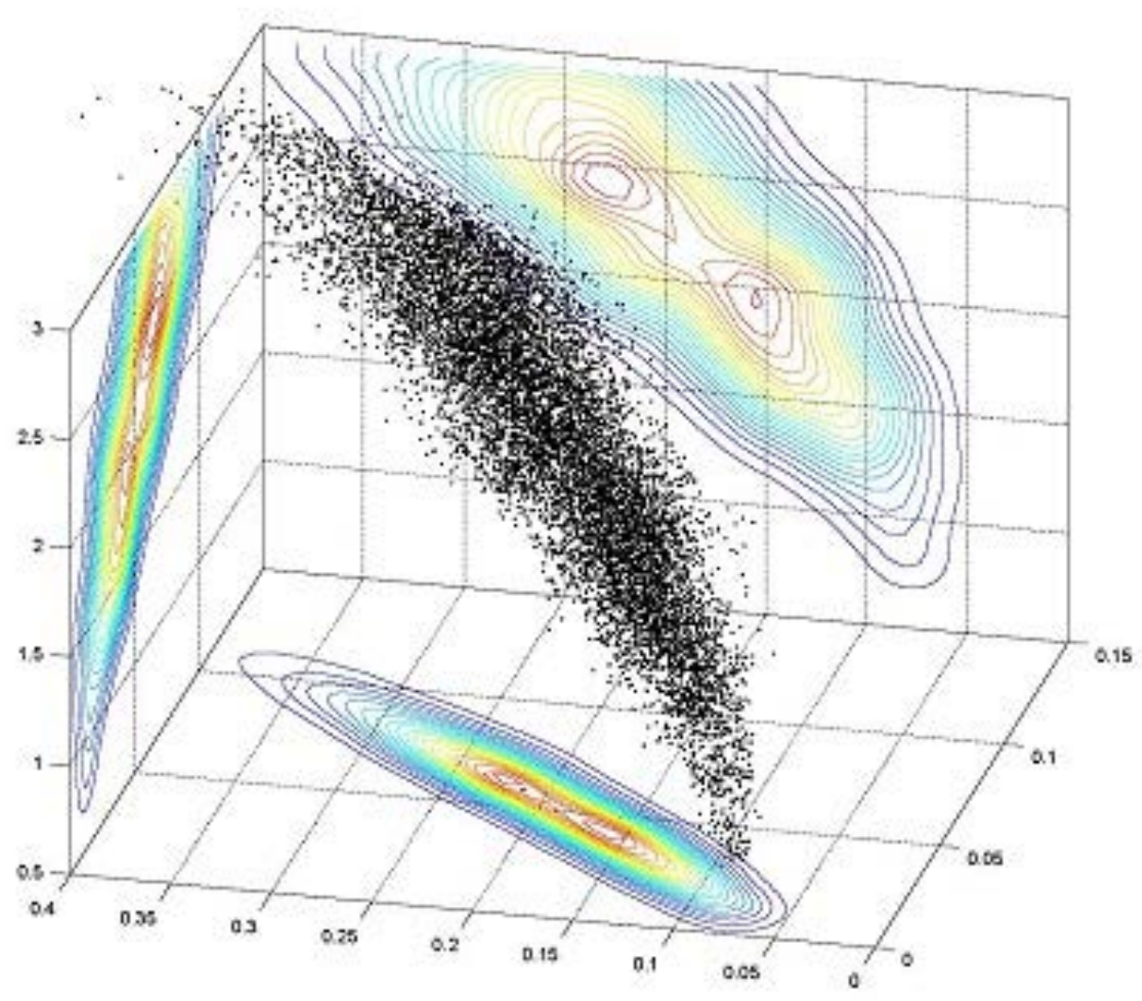




Approximate inference: Sampling methods

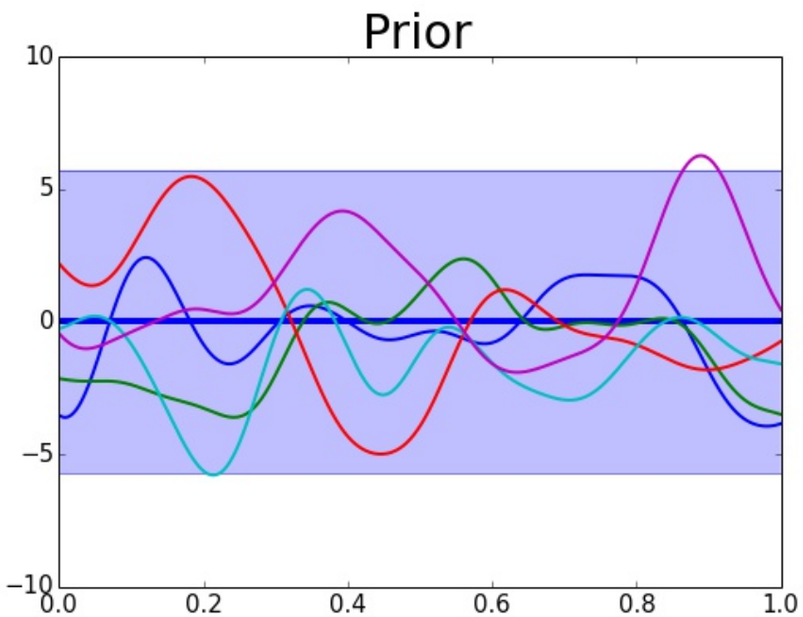
$$\mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i),$$

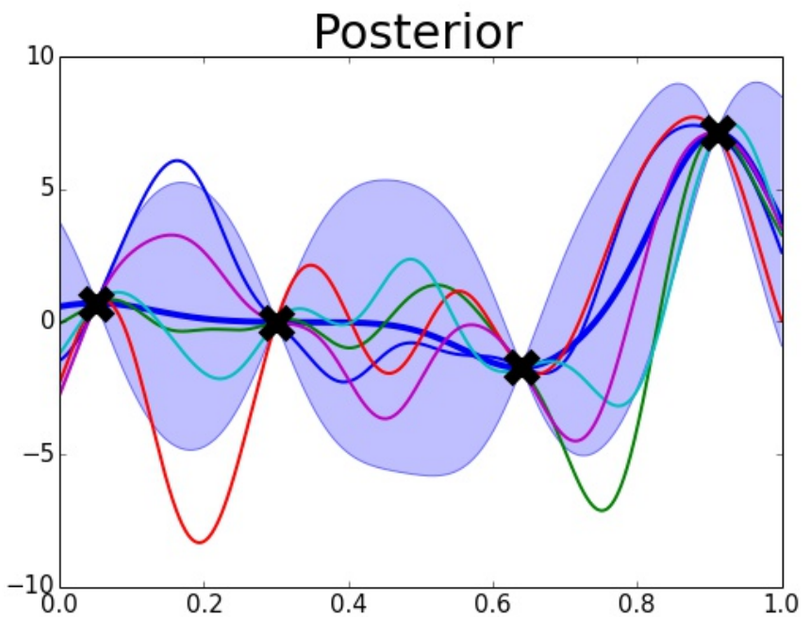
where x_i are drawn iid from $p(x)$



Gaussian process regression

$$y = f(\boldsymbol{x}) + \epsilon \qquad f \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}))$$





Training via maximizing the marginal likelihood

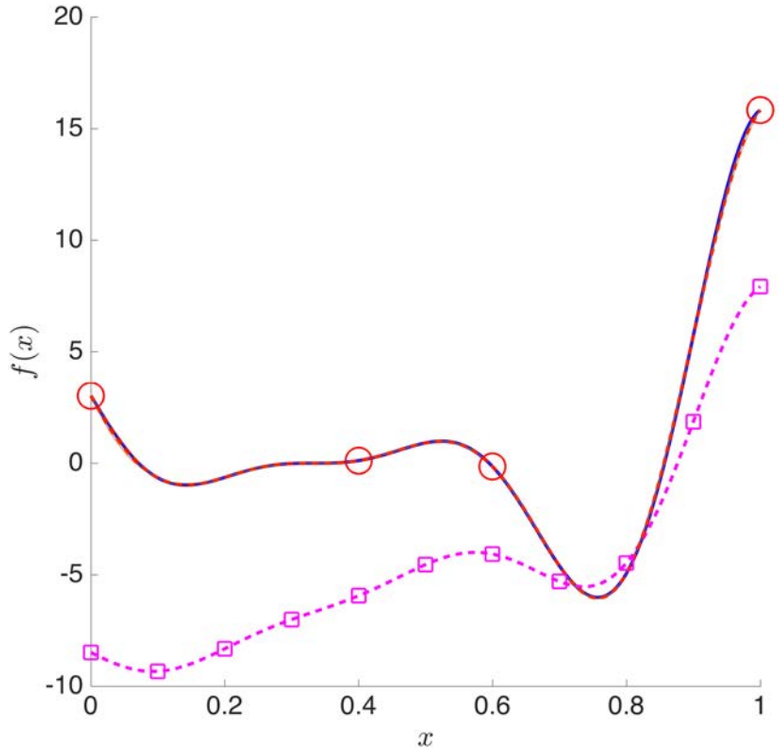
$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\log|\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}| - \frac{1}{2}\boldsymbol{y}^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{y} - \frac{N}{2}\log 2\pi$$

Prediction via conditioning on available data

$$p(f_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}_*) = \mathcal{N}(f_*|\mu_*, \sigma_*^2),$$
$$\mu_*(\boldsymbol{x}_*) = \boldsymbol{k}_{*N}(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{y},$$
$$\sigma_*^2(\boldsymbol{x}_*) = \boldsymbol{k}_{**} - \boldsymbol{k}_{*N}(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{k}_{N*},$$

Demo code: https://github.com/PredictiveIntelligenceLab/GPTutorial

Multi-fidelity regression



Multi-fidelity observations:

$$\boldsymbol{y}_L = f_L(\boldsymbol{x}_L) + \boldsymbol{\epsilon}_L$$
$$\boldsymbol{y}_H = f_H(\boldsymbol{x}_H) + \boldsymbol{\epsilon}_H$$

Probabilistic model:

$$f_H(\boldsymbol{x}) = \rho f_L(\boldsymbol{x}) + \delta(\boldsymbol{x})$$
$$f_L(\boldsymbol{x}) \sim \mathcal{GP}(0, k_L(\boldsymbol{x}, \boldsymbol{x}'; \theta_L))$$
$$\delta(\boldsymbol{x}) \sim \mathcal{GP}(0, k_H(\boldsymbol{x}, \boldsymbol{x}'; \theta_H))$$
$$\epsilon_L \sim \mathcal{N}(0, \sigma_{\epsilon_L}^2 \boldsymbol{I})$$
$$\epsilon_H \sim \mathcal{N}(0, \sigma_{\epsilon_H}^2 \boldsymbol{I})$$

Training:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_L \\ \boldsymbol{y}_H \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} k_L(\boldsymbol{x}_L, \boldsymbol{x}'_L; \theta_L) + \sigma_{\epsilon_L}^2 \boldsymbol{I} & \rho k_L(\boldsymbol{x}_L, \boldsymbol{x}'_H; \theta_L) \\ \rho k_L(\boldsymbol{x}_H, \boldsymbol{x}'_L; \theta_L) & \rho^2 k_L(\boldsymbol{x}_H, \boldsymbol{x}'_H; \theta_L) + k_H(\boldsymbol{x}_H, \boldsymbol{x}'_H; \theta_H) + \sigma_{\epsilon_H}^2 \boldsymbol{I} \end{bmatrix} \right)$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_L \\ \boldsymbol{x}_H \end{bmatrix} \qquad -\log p(\boldsymbol{y}|\boldsymbol{X}, \theta_L, \theta_H, \rho, \sigma_{\epsilon_L}^2, \sigma_{\epsilon_H}^2) = \frac{1}{2}\log|\boldsymbol{K}| + \frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1}\boldsymbol{y} - \frac{N_L + N_H}{2}\log 2\pi$$

Prediction:

$$p(f(\boldsymbol{x}^*)|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}^*) \sim \mathcal{N}(f(\boldsymbol{x}^*)|\mu(\boldsymbol{x}^*), \sigma^2(\boldsymbol{x}^*))$$
$$\mu(\boldsymbol{x}^*) = \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{X})\boldsymbol{K}^{-1}\boldsymbol{y}$$
$$\sigma(\boldsymbol{x}^*) = \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{X})\boldsymbol{K}^{-1}\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}^*)$$

M.C Kennedy, and A. O'Hagan. *Predicting the output from a complex computer code when fast approximations are available,* 2000.

Demo code: https://github.com/PredictiveIntelligenceLab/GPTutorial

Bayesian optimization

Goal: Estimate the global minimum of a function: $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d} g(\mathbf{x})$ (potentially intractable)

Setup: g(x) is a black-box and expensive to evaluate objective function, noisy observations, no gradients.

Idea: Approximate g(x) using a GP surrogate: $y = f(\mathbf{x}) + \epsilon, \quad f \sim \mathcal{GP}\left(f|0, k(\mathbf{x}, \mathbf{x}'; \theta)\right)$

Utilize the posterior to guide a sequential or parallel sampling policy by optimizing a chosen expected utility function

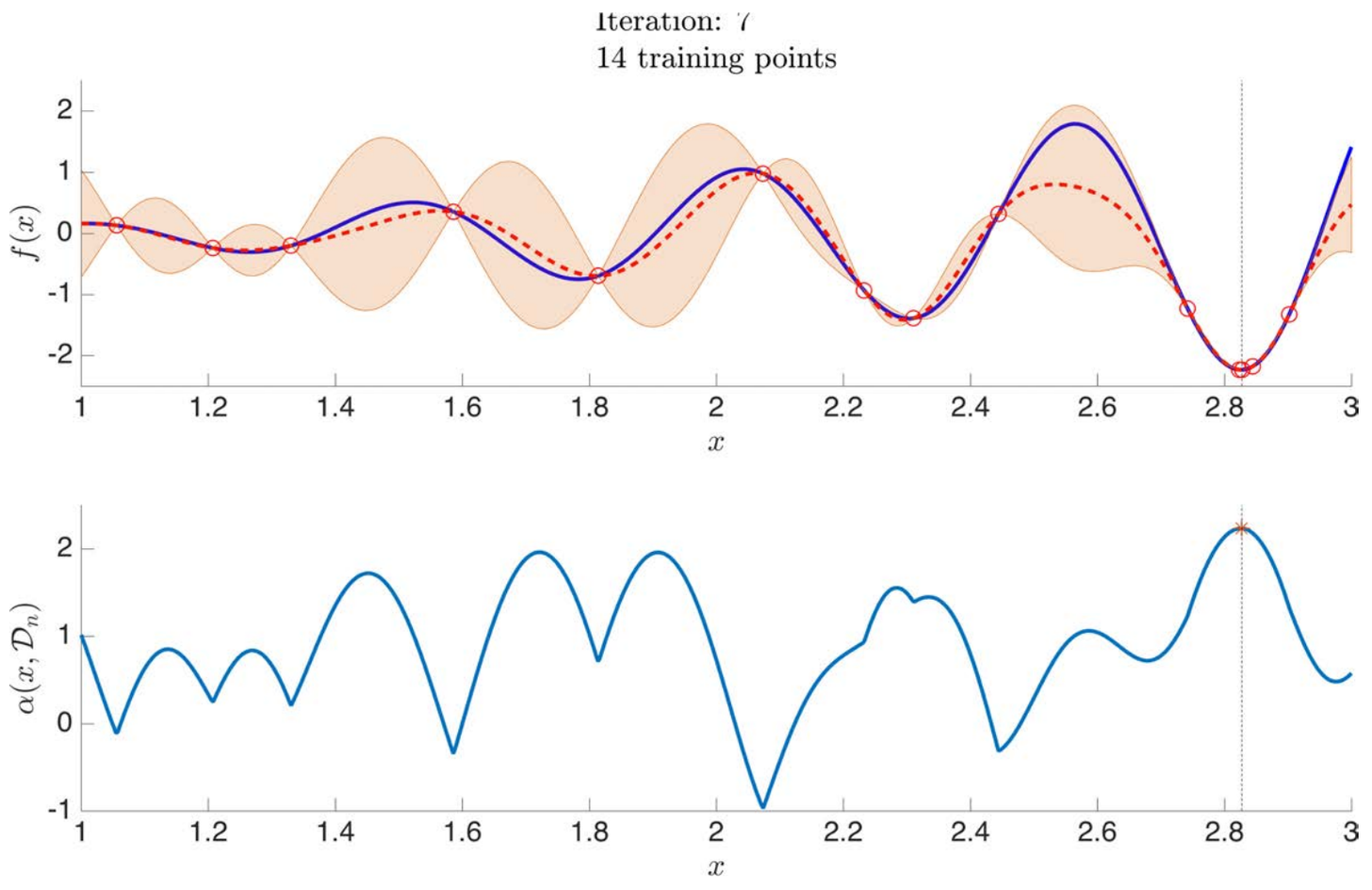
$$\alpha(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_\theta \mathbb{E}_{v\,|\,\mathbf{x},\theta}[U(\mathbf{x}, v, \theta)]$$

The optimization problem is transformed to:

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}} \alpha(\mathbf{x}; \mathcal{D}_n)$$

Remark:

Acquisition functions aim to balance the trade-off between exploration and exploitation.



e.g. sample at the locations that minimize the lower super-quintile risk confidence bound

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

A new paradigm in scientific computing

$$\mathcal{L}_x u(x) = f(x)$$

$$u(x) \sim \mathcal{GP}(0, g(x, x'; \theta)) \quad \longrightarrow \quad f(x) \sim \mathcal{GP}(0, k(x, x'; \theta))$$

$$k(x, x'; \theta) = \mathcal{L}_x \mathcal{L}_{x'} g(x, x'; \theta)$$

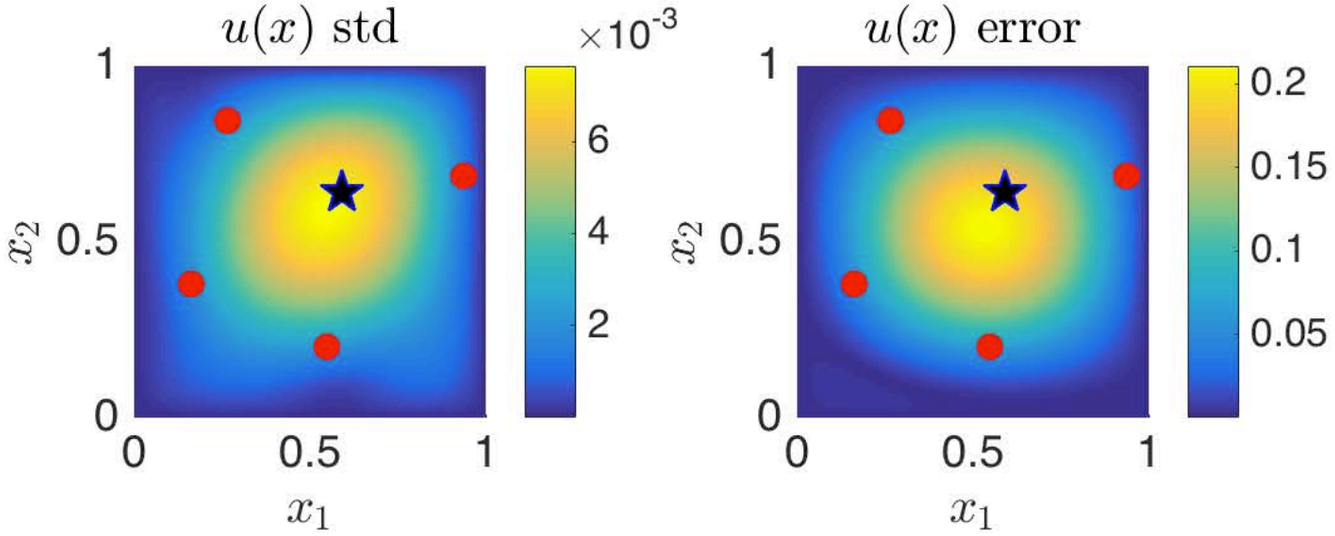
$$-\log p(\boldsymbol{y}|\phi, \theta, \sigma_{n_u}^2, \sigma_{n_f}^2) = \frac{1}{2}\log|\boldsymbol{K}| + \frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1}\boldsymbol{y} + \frac{N}{2}\log 2\pi\,,$$

where $\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_u \\ \boldsymbol{y}_f \end{bmatrix}$, $p(\boldsymbol{y}|\phi, \theta, \sigma_{n_u}^2, \sigma_{n_f}^2) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$, and $\boldsymbol{K}$ is given by

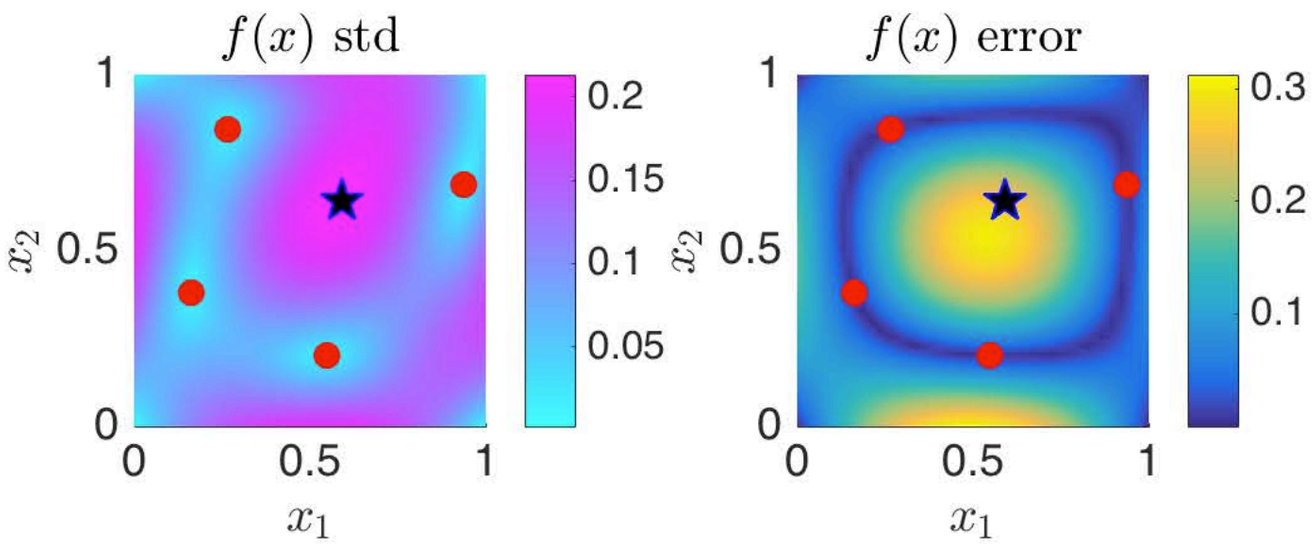
$$\boldsymbol{K} = \begin{bmatrix} k_{uu}(\boldsymbol{X}_u, \boldsymbol{X}_u; \theta) + \sigma_{n_u}^2 \boldsymbol{I}_{n_u} & k_{uf}(\boldsymbol{X}_u, \boldsymbol{X}_f; \theta, \phi) \\ k_{fu}(\boldsymbol{X}_f, \boldsymbol{X}_u; \theta, \phi) & k_{ff}(\boldsymbol{X}_f, \boldsymbol{X}_f; \theta, \phi) + \sigma_{n_f}^2 \boldsymbol{I}_{n_f} \end{bmatrix}.$$

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. Journal of Computational Physics, 348, 683-693.

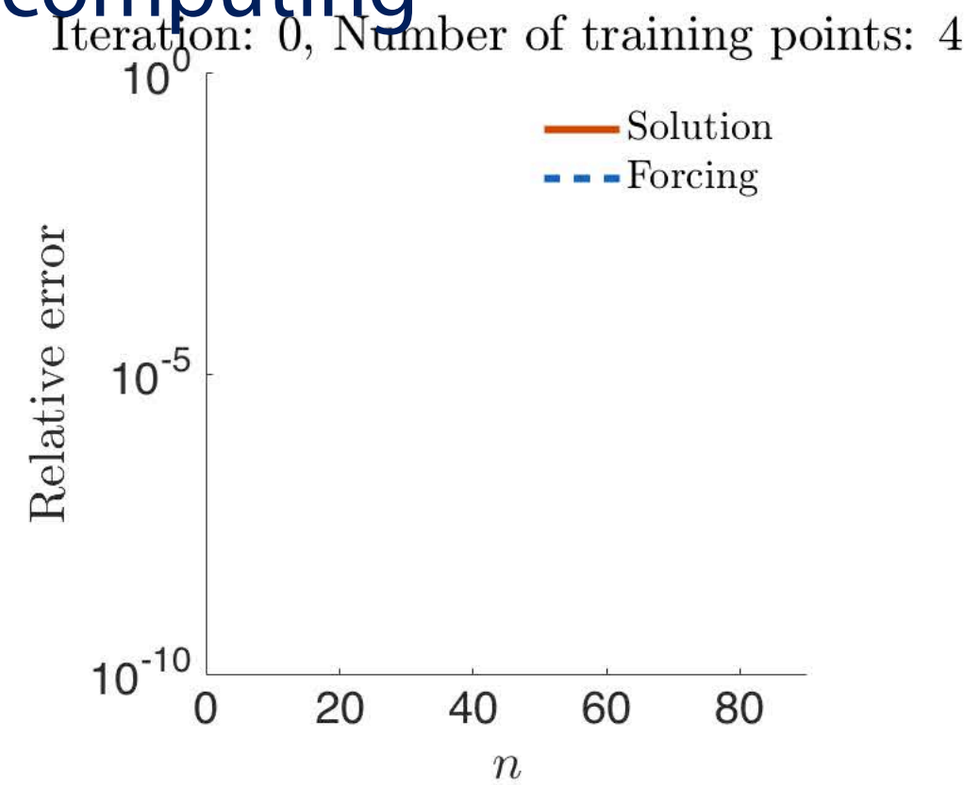
A new paradigm in scientific computing

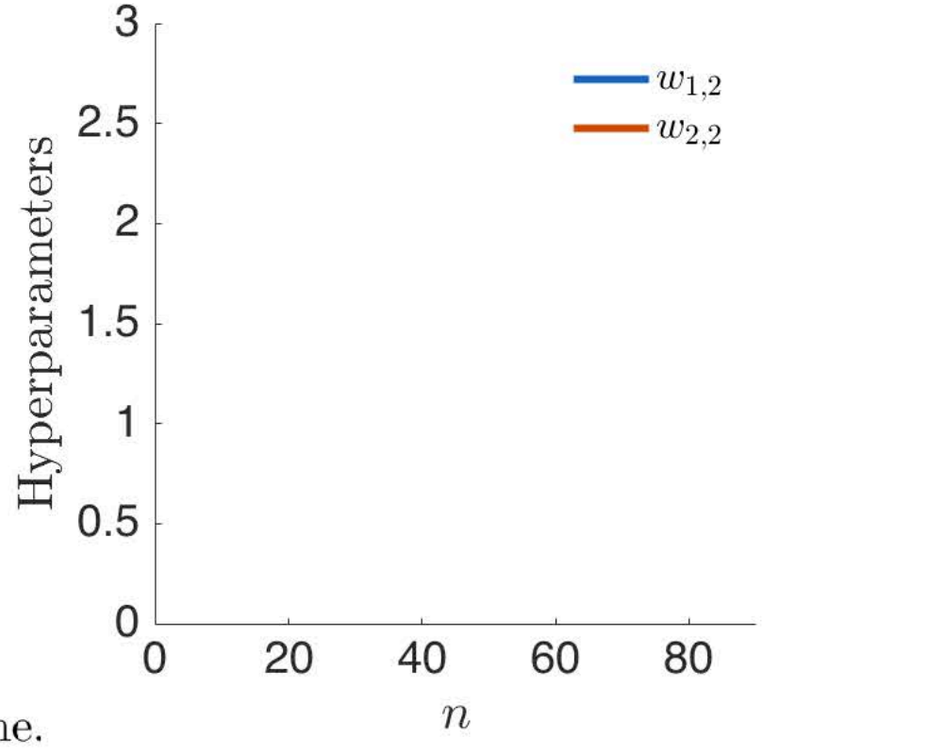


$$\frac{\partial^2 u}{\partial {x_1}^2} + \frac{\partial^2 u}{\partial {x_2}^2} = f(x_1, x_2)$$



- denotes the training data actively collected by the scheme.
- ★ denotes the next sampling point suggested by the active learning scheme.



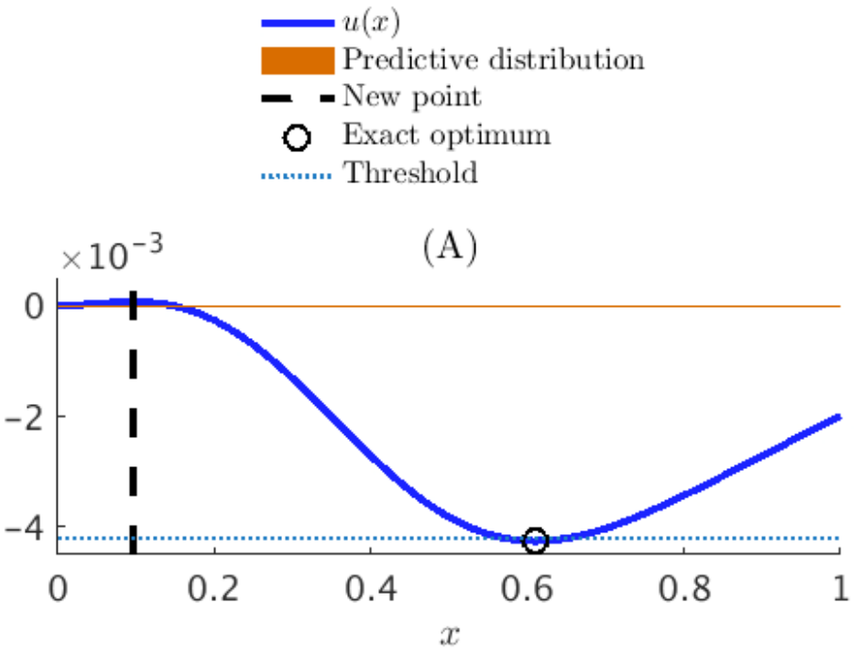


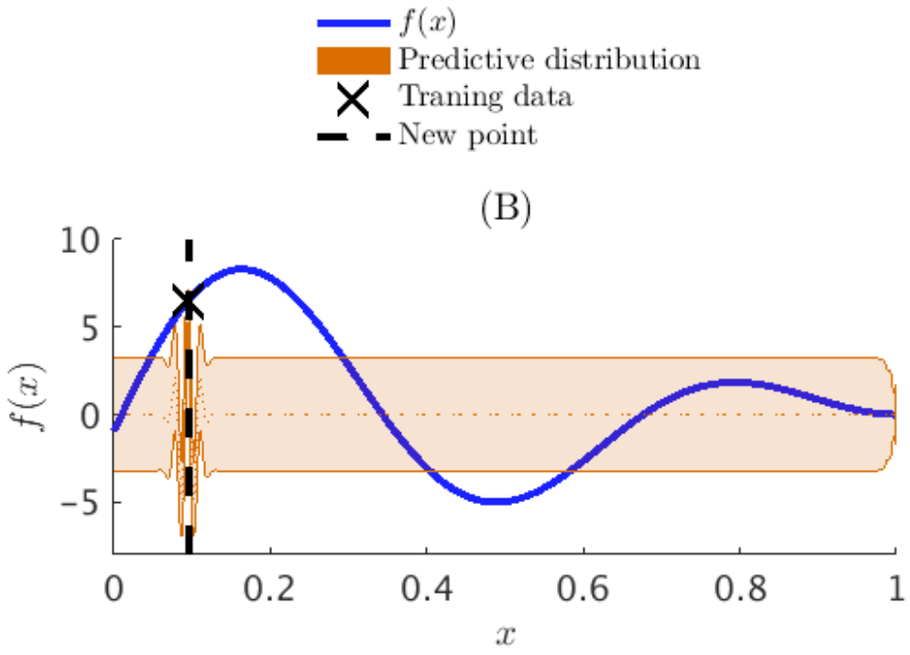
M. Raissi,., P. Perdikaris, and G.E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, http://128.84.21.199/abs/1607.04805, 2016

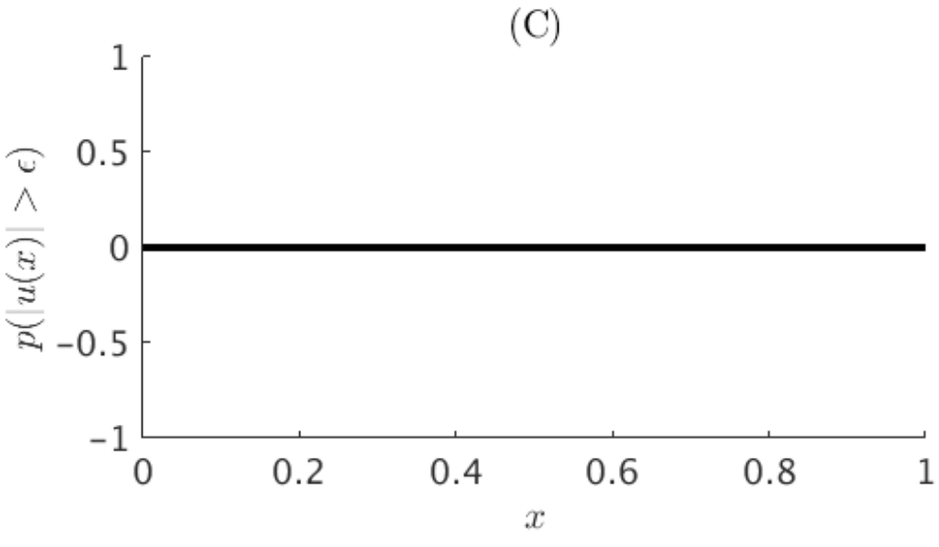
A new paradigm in scientific computing

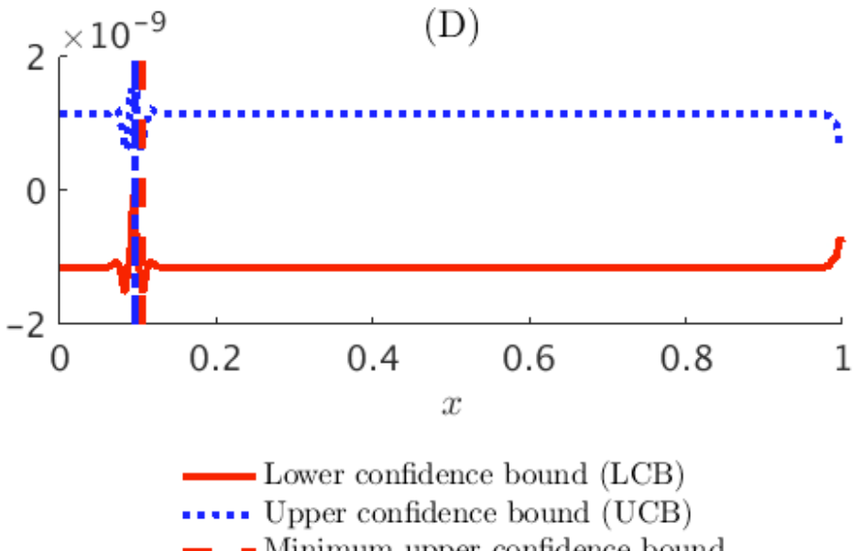
$$\mathcal{L}_x u(x) := \frac{d^4}{dx^4} u(x) = f(x)$$
$$u(0) = u'(0) = 0$$
$$u''(1) = 0$$
$$u'''(1) = f(1)$$

1. Given noisy observations of the loading $f(x)$, solve for the displacement $u(x)$.
2. Find the maximum displacement $|u(x)|$.
3. Given the threshold ϵ, find the probability of failure.









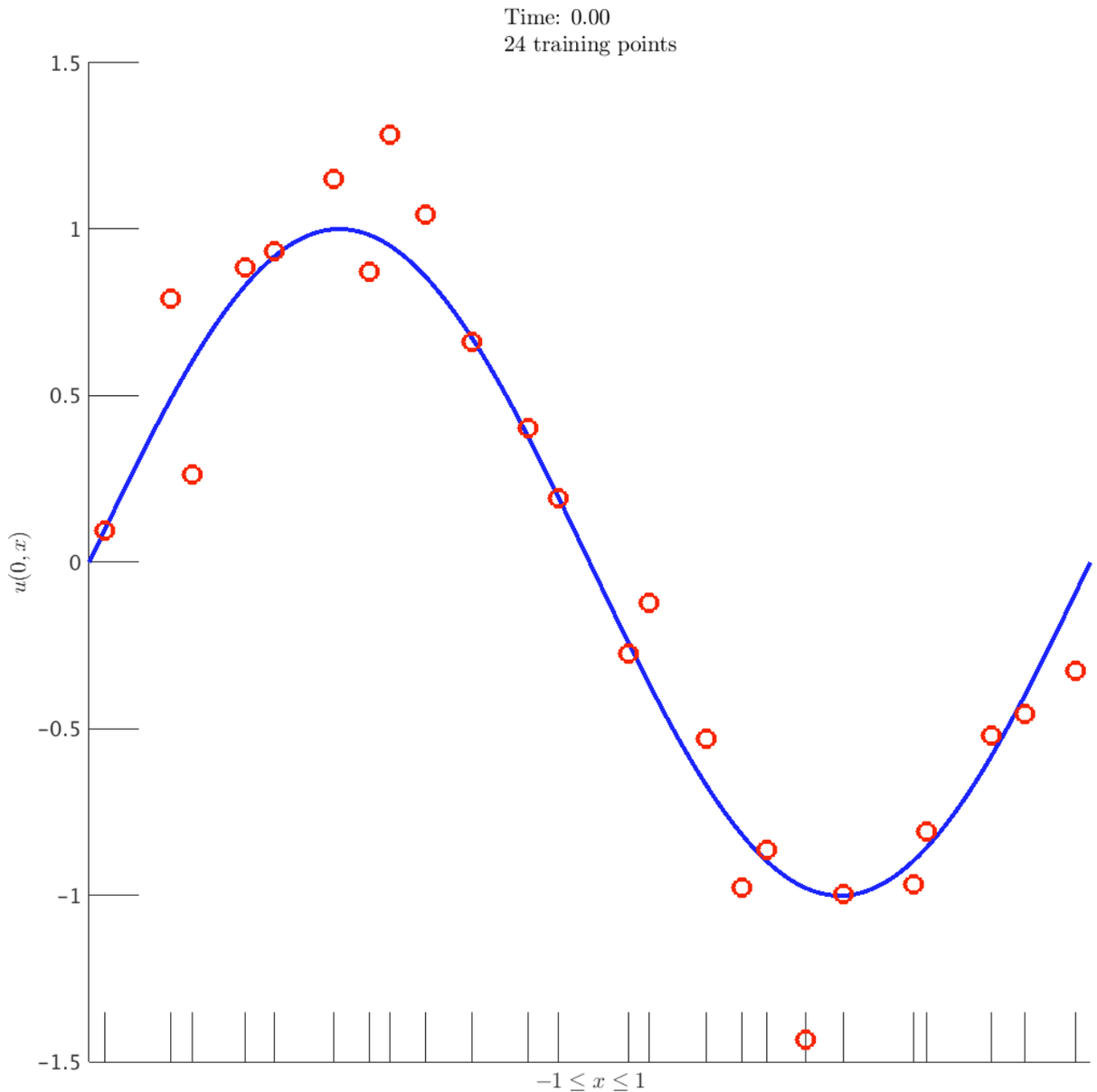
A new paradigm in scientific computing

Example: 1D viscous Burgers —> *The equation, along with the choice of a time-stepping scheme define a GP prior!*

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \in [0, 1],$$
$$u(0, x) = u^0(x)$$
$$u(t, 0) = u(t, 1) = 0. \quad \nu = \pi/100$$

$$+ \quad \mathcal{L}_x u^{n+1}(x) := u^{n+1}(x) + \Delta t \; u^n(x) \; \frac{du^{n+1}(x)}{dx} - \Delta t \; \nu \frac{d^2 u^{n+1}(x)}{dx^2} = u^n(x)$$

e.g., Backward Euler time-stepping



Remarks:

- Despite starting from a stationary prior:

  $$u^{n+1,n+1}(x) \sim \mathcal{GP}(0, k^{n+1,n+1}(x, x'; \theta))$$

  the structure of the of the kernel:

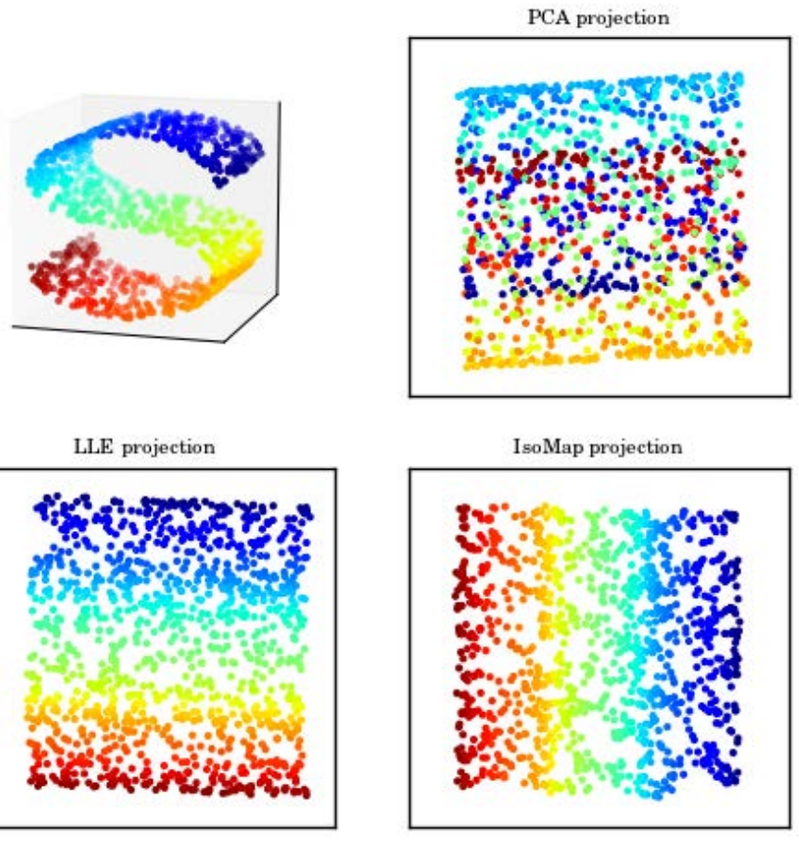
  $$k^{n,n}(x, x'; \theta) = \mathcal{L}_x \mathcal{L}_{x'} k^{n+1,n+1}(x, x'; \theta)$$

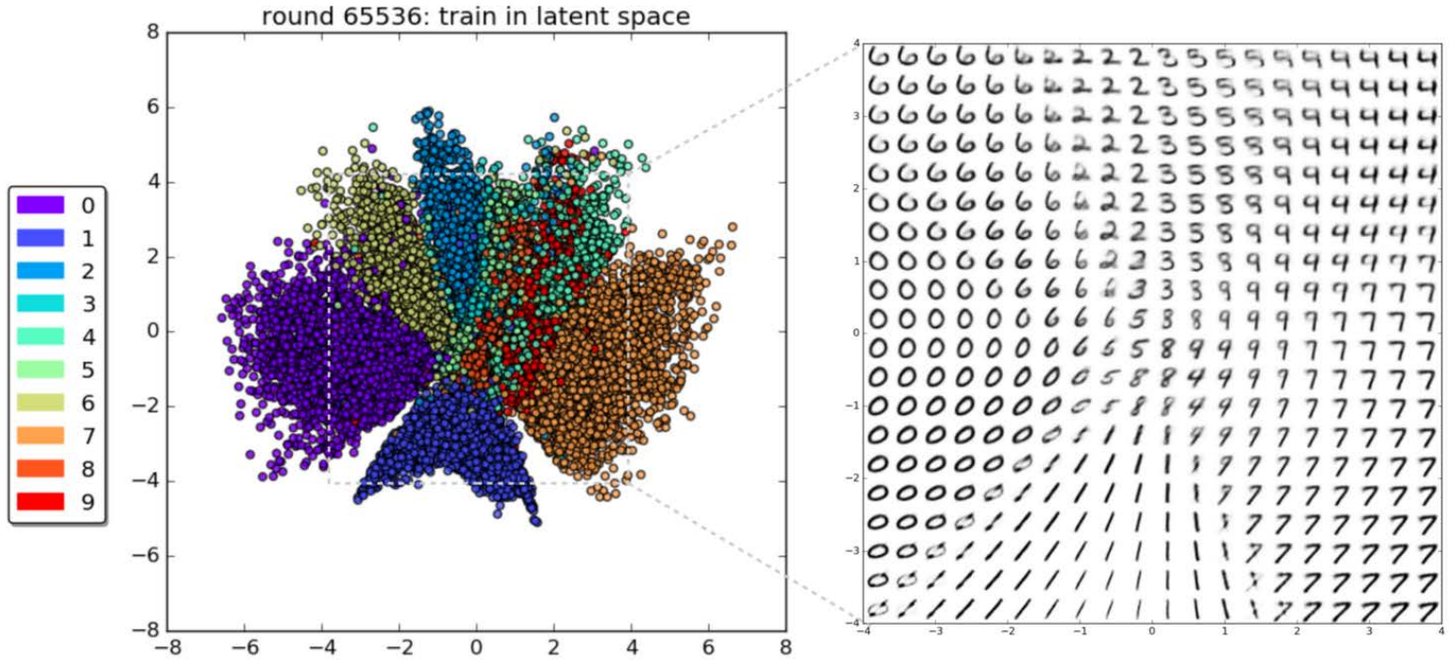
  can generate discontinuous solutions!
- This is a general approach applicable to any nonlinear equation and any time-stepping scheme.
- We only need to derive the kernel $k^{n,n}(x, x'; \theta)$
- Under this setup, fully implicit time-stepping schemes have the same complexity as their explicit counterparts. Hence, one can obtain highly accurate and stable schemes at no extra cost.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Numerical Gaussian Processes for Time-Dependent and Nonlinear Partial Differential Equations. SIAM Journal on Scientific Computing, 40(1), A172-A198.

Unsupervised learning

- Density estimation
- Learning to draw samples from a distribution
- Learning to denoise samples from a distribution
- Find a low-dimensional manifold that the data lies near
- Cluster the data into groups of related examples





A classic unsupervised task:

Find the "best" representation of the data.

*By "best" we can mean different things, but generally speaking we are looking for a representation that preserves as much information about x as possible while obeying some penalty or constraint aimed at keeping the representation simpler or more accessible than x itself.

PCA and probabilistic PCA

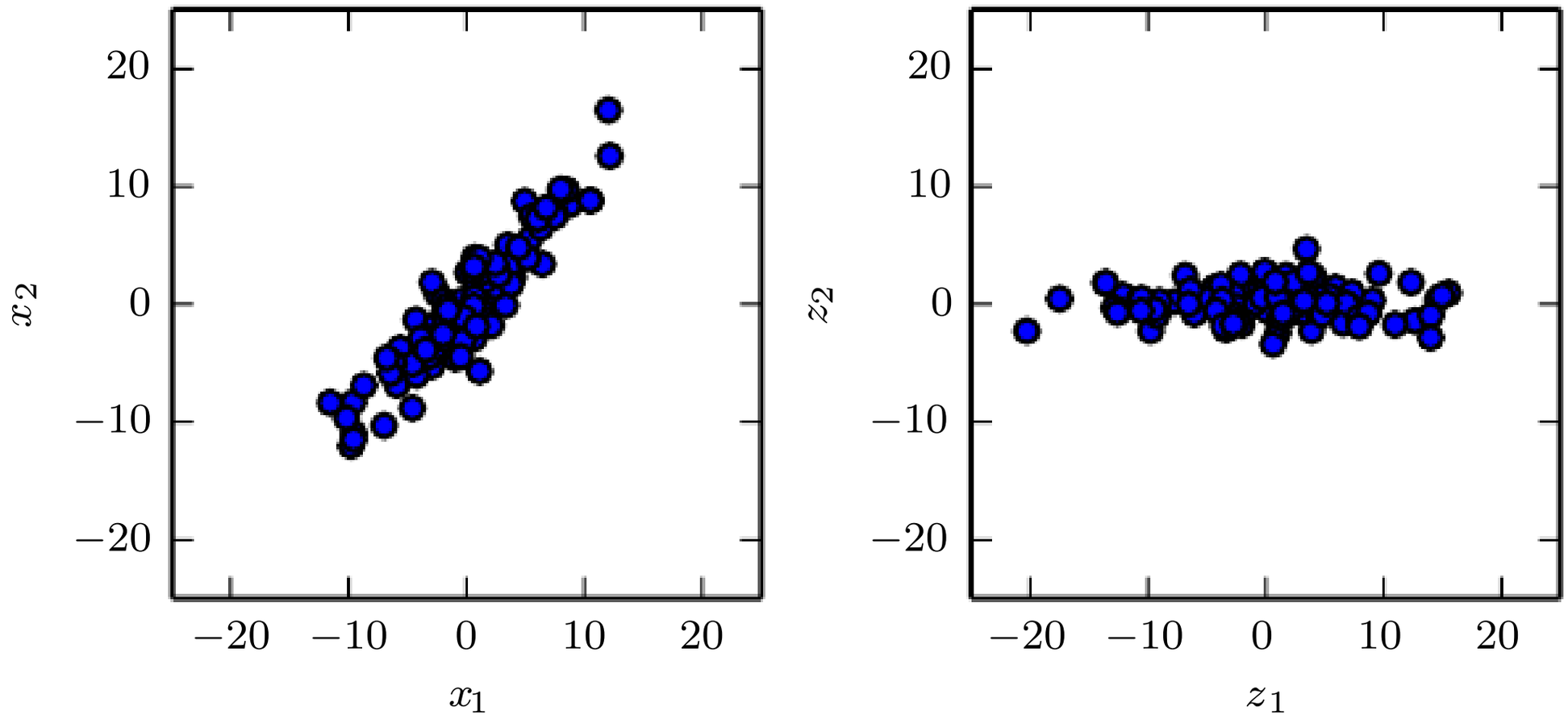


Figure 5.8: PCA learns a linear projection that aligns the direction of greatest variance with the axes of the new space. *(Left)*The original data consists of samples of $\boldsymbol{x}$. In this space, the variance might occur along directions that are not axis-aligned. *(Right)*The transformed data $\boldsymbol{z} = \boldsymbol{x}^\top \boldsymbol{W}$ now varies most along the axis z_1. The direction of second most variance is now along z_2.

Tricks of the trade

Jensen's inequality, importance sampling and variational bounds:

$$
\begin{aligned}
-\log p(x) &= -\log \int p(x,y)dy \\
&= -\log \int q(y|x)\frac{p(y,x)}{q(y|x)}dy \\
&\leq -\int q(y|x)\log\frac{p(y,x)}{q(y|x)}dy
\end{aligned}
$$

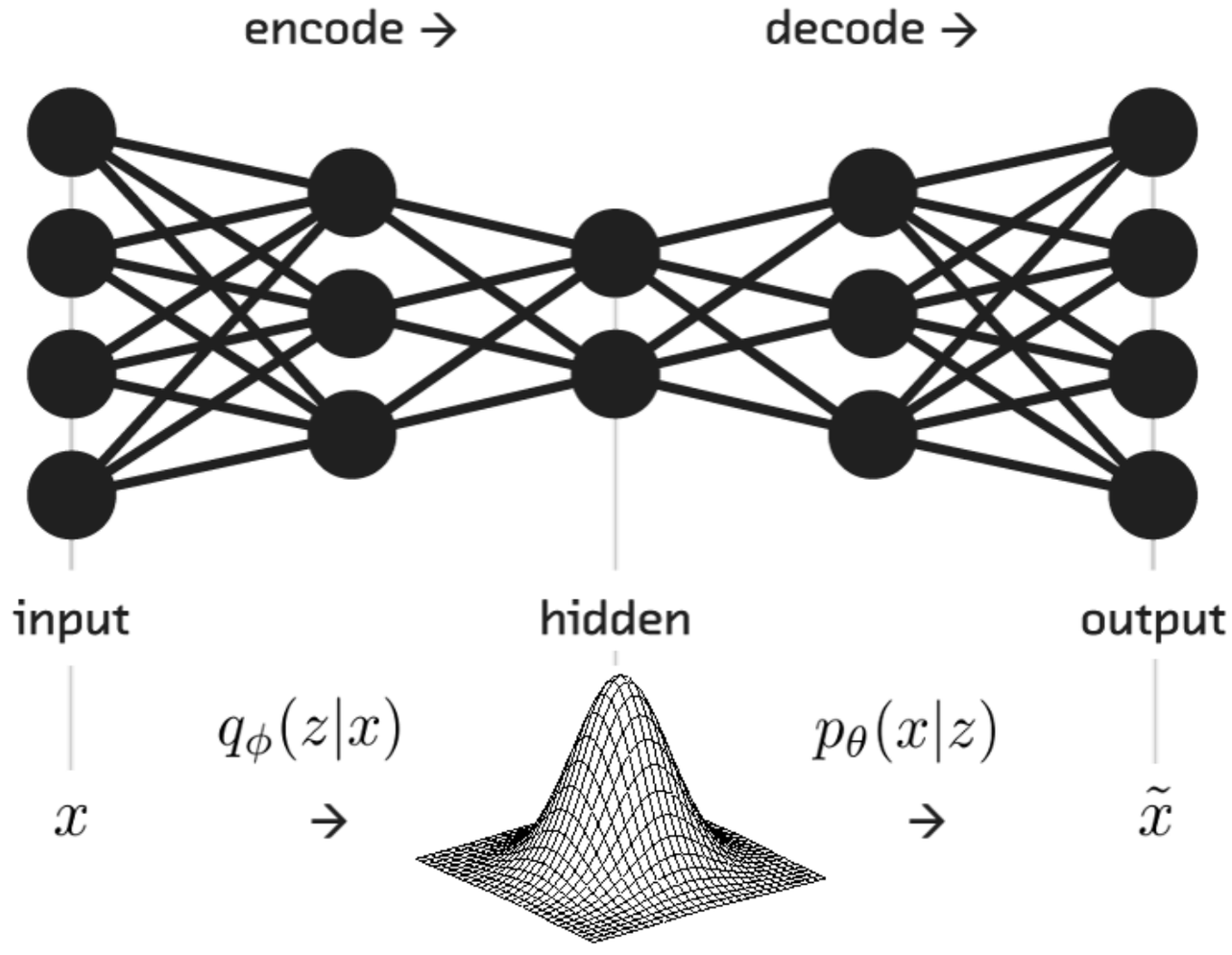
Unbiased gradients through the re-parametrization trick

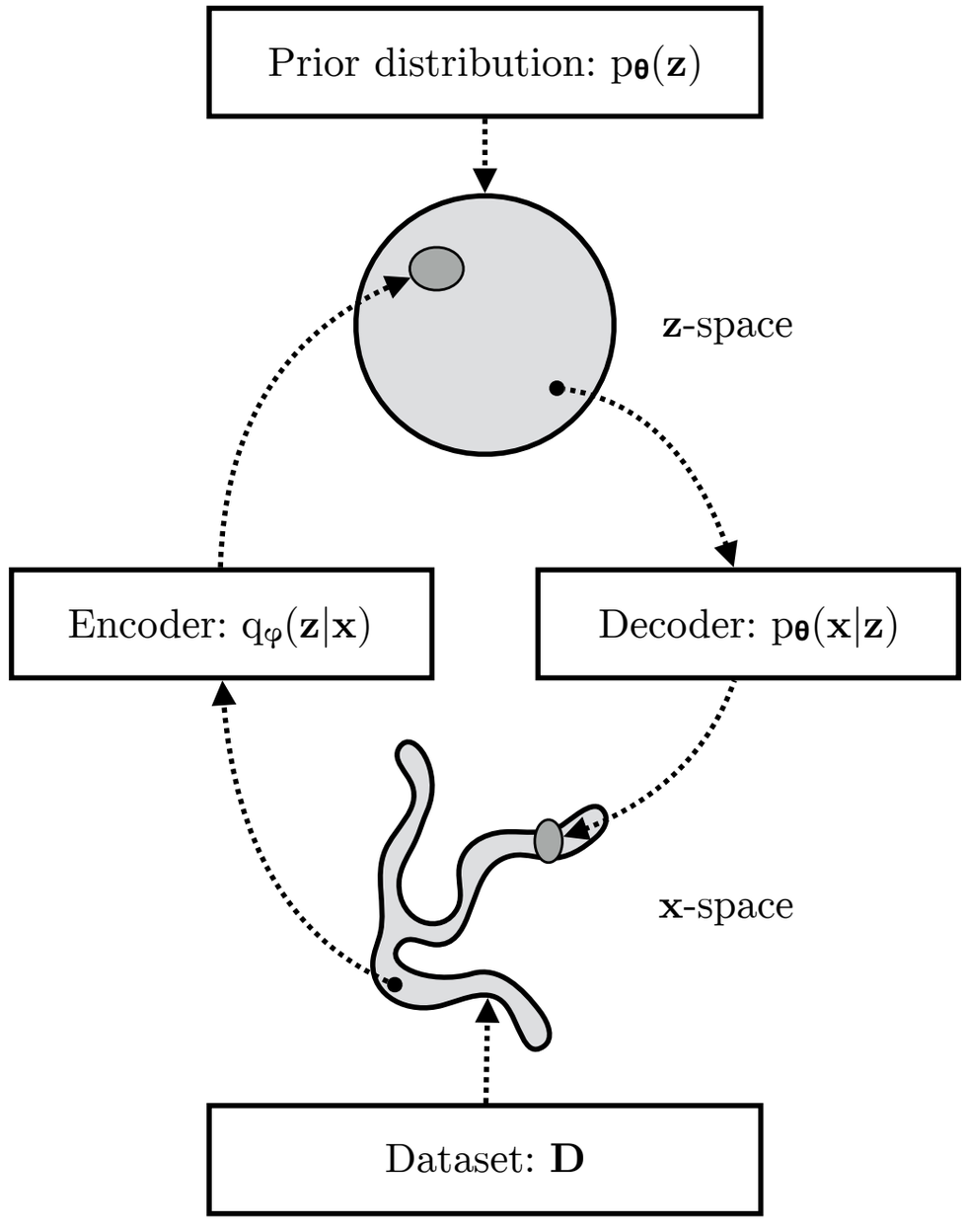
$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta)f(z)dz \\
&= \nabla_\theta \int p(\epsilon)f(z)d\epsilon = \nabla_\theta \int p(\epsilon)f(g(\epsilon,\theta))d\epsilon \\
&= \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(g(\epsilon,\theta))] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]
\end{aligned}
$$

Density ratio estimation from samples using binary classifiers

$$
\begin{aligned}
r(x) &= \frac{\rho(x)}{q(x)} = \frac{p(x|y=+1)}{p(x|y=-1)} \\
&= \frac{p(y=+1|x)p(x)}{p(y=+1)} \Big/ \frac{p(y=-1|x)p(x)}{p(y=-1)} \\
&= \frac{p(y=+1|x)}{p(y=-1|x)} = \frac{p(y=+1|x)}{1-p(y=+1|x)} = \frac{S(x)}{1-S(x)}
\end{aligned}
$$

Variational auto-encoders

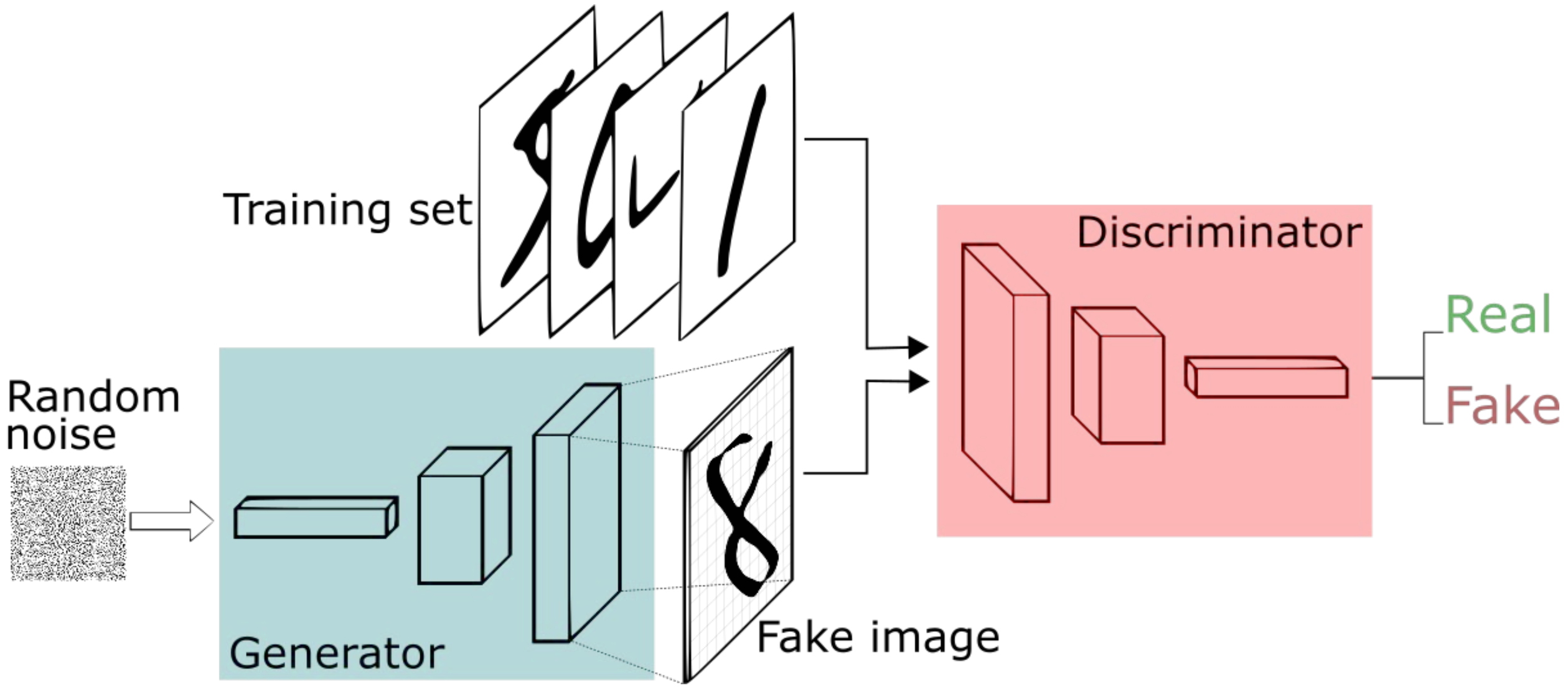




A VAE learns stochastic mappings between the observed **x**-space, whose empirical distribution $q_D(\mathbf{x})$ is typically complicated, and a latent **z**-space, whose distribution can be relatively simple (such as spherical, as in this figure). The generative model learns a joint distribution $p_\theta(\mathbf{x},\mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$, factorized into a prior distribution over latent space, $p_\theta(\mathbf{z})$, and a stochastic decoder $p_\theta(\mathbf{x}|\mathbf{z})$. The stochastic encoder $q_\varphi(\mathbf{z}|\mathbf{x})$, also called inference model, approximates the true but intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$ of the generative model.

Kingma, D. P. (2017). Variational inference & deep learning: A new synthesis. (PhD Thesis)

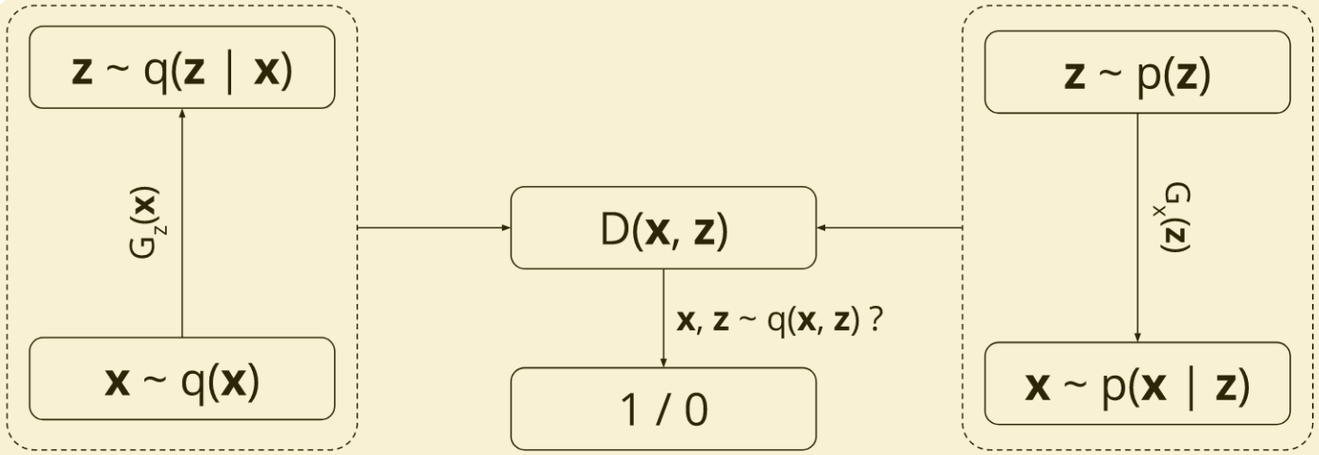
Generative adversarial networks



$$\min_G \max_D V(D,G) = \mathbb{E}_{q(\mathbf{x})}[\log(D(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

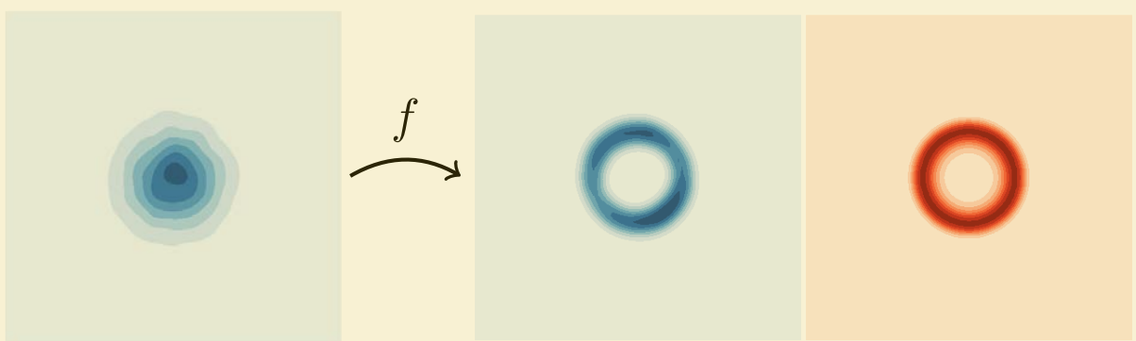
Convergence of variational and adversarial learning



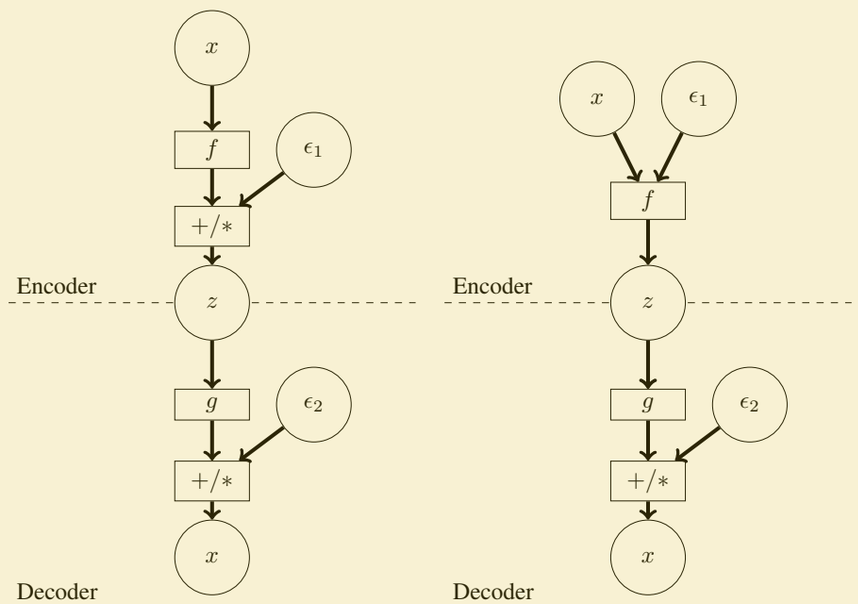
Adversarially learned inference

$$\min_G \max_D V(D,G) = \mathbb{E}_{q(\mathbf{x})}[\log(D(\mathbf{x}, G_z(\mathbf{x})))] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(G_x(\mathbf{z}), \mathbf{z}))]$$

Instead of using approximating distributions of a given pre-defined form (e.g. Gaussian) we can implicitly parametrize them using deep neural networks.



Adversarial Variational Bayes



(a) Standard VAE (b) Our model