Neural ODE solvers

AI Tech, 22.05.2022

Michał Stypułkowski

University of Wrocław

Source: link

Introduction

Differential equations

“Since Newton, mankind has come to realize that the **laws of physics** are always expressed in the language of **differential equations**.”

- Steven Strogatz

Differential equations - overview

1. Ordinary differential equations (ODEs):
    - Equation with a function of **one independent variable**.
    - Often describing **change of function in time**.
    - Some of ODEs have closed-form solutions.

Differential equations - overview

1. Ordinary differential equations (ODEs):
   - Equation with a function of **one independent variable**.
   - Often describing **change of function in time**.
   - Some of ODEs have closed-form solutions.

2. Partial differential equations (PDEs):
   - Equation with a function of **at least two independent variables**.
   - Usually no analytical solutions.

Differential equations - examples

1. Ordinary differential equations (ODEs):
   - Pandemic model.
   - Predator-prey equations.
   - Newton’s law of cooling/heating.

2. Partial differential equations (PDEs):
   - Diffusion process.
   - Fluid dynamics.
   - Black–Scholes equation.

Differential equations - examples

1. Ordinary differential equations (ODEs):
   - Pandemic model. ← FUN!
   - Predator-prey equations.
   - Newton’s law of cooling/heating.

2. Partial differential equations (PDEs):
   - Diffusion process.
   - Fluid dynamics.
   - Black–Scholes equation.

Differential equations - examples

1. Ordinary differential equations (ODEs):
   - Pandemic model.
   - Predator-prey equations.
   - Newton’s law of cooling/heating.

2. Partial differential equations (PDEs):
   - Diffusion process.
   - Fluid dynamics.
   - Black–Scholes equation.

FUN!

MORE FUN!
But way more difficult :(

Numerical ODE solvers

Initial value problem

We face an **initial value problem** of the form:

$$x'(t) = f(t, x)$$
$$x(t_0) = x_0$$

We will often call function f a **dynamic function**.

Taylor series

Using the Taylor series of a real infinitely differentiable function x at point $t + h$ we can get:

$$x(t+h) = x(t) + hx'(t) + \frac{1}{2!}h^2x''(t) + \frac{1}{3!}h^3x'''(t) + \frac{1}{4!}h^4x^{(4)}(t) + \dots$$

Taylor series

Using the Taylor series of a real infinitely differentiable function x at point $t+h$ we can get:

$$x(t+h) = x(t) + hx'(t) + \frac{1}{2!}h^2x''(t) + \frac{1}{3!}h^3x'''(t) + \frac{1}{4!}h^4x^{(4)}(t) + \dots$$

In other words, when we know the function x and its value at time t, we can calculate its value **small step in the future** using **infinite** number of derivatives!

Taylor series

Using the Taylor series of a real infinitely differentiable function x at point $t+h$ we can get:

$$x(t+h) = x(t) + hx'(t) + \frac{1}{2!}h^2x''(t) + \frac{1}{3!}h^3x'''(t) + \frac{1}{4!}h^4x^{(4)}(t) + \dots$$

In other words, when we know the function x and its value at time t, we can calculate its value **small step in the future** using **infinite** number of derivatives!

LET US APPROXIMATE!

Euler method

The simplest (I order) method based on the Taylor series.

We can approximate the value of a $x(t+h)$:

$$x(t+h) \approx x(t) + hx'(t) = x(t) + hf(t,x)$$

Euler method

The simplest (I order) method based on the Taylor series.

We can approximate the value of a $x(t+h)$:

$$x(t+h) \approx x(t) + hx'(t) = x(t) + hf(t,x)$$

In practice, we use the following formula to update x_n:

$$x_{n+1} = x_n + hf(t, x_n)$$

Runge-Kutta of order 4 (RK4)

This method is based on the Taylor series up to 4th derivative. The formula is:

$$x(t+h) \approx x(t) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)$$

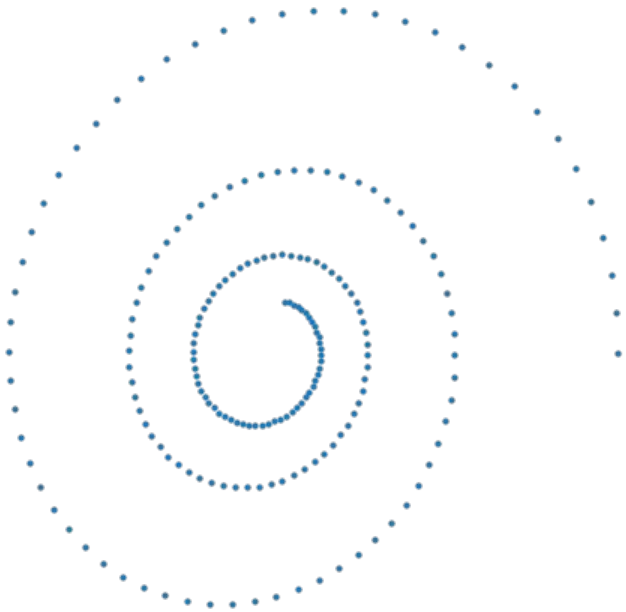
where

$$\begin{aligned}
F_1 &= hf(t, x), \\
F_2 &= hf(t + \frac{1}{2}h, x + \frac{1}{2}F_1), \\
F_3 &= hf(t + \frac{1}{2}h, x + \frac{1}{2}F_2), \\
F_4 &= hf(t + h, x + F_3).
\end{aligned}$$

Unknown dynamic function

Let us now change the perspective.

Suppose we **do not know** the dynamic function but instead we are **given data points** from some trajectory.

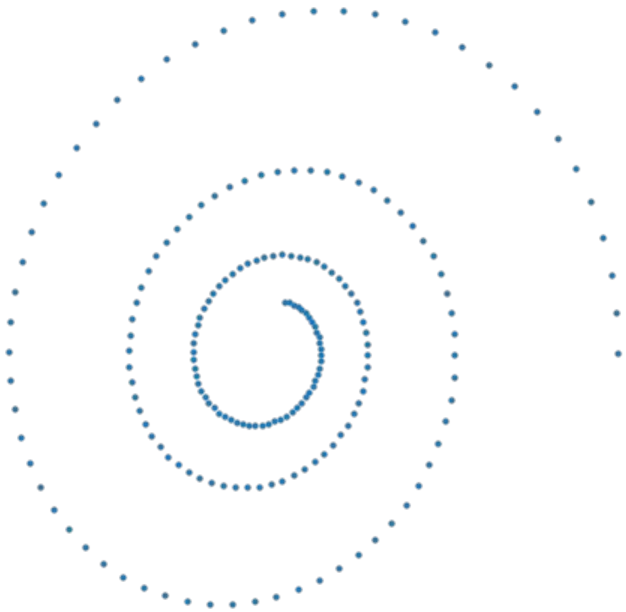


Unknown dynamic function

Let us now change the perspective.

Suppose we **do not know** the dynamic function but instead we are **given data points** from some trajectory.

We want to find a dynamic function that fits the data points best.

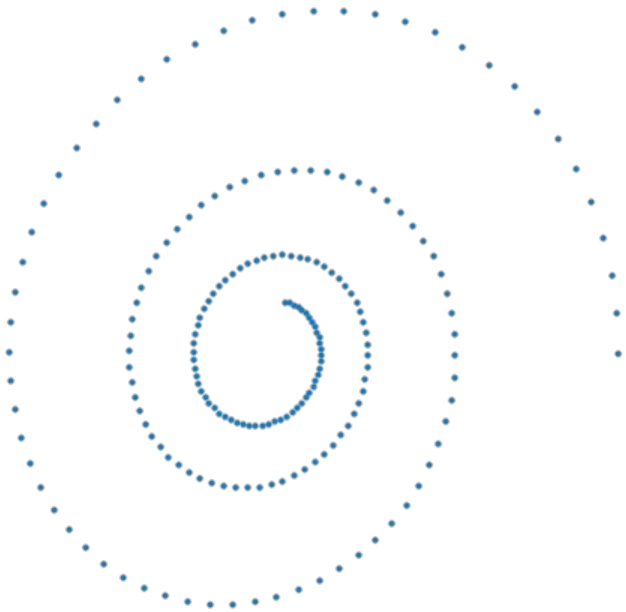


Unknown dynamic function

Let us now change the perspective.

Suppose we **do not know** the dynamic function but instead we are **given data points** from some trajectory.

We want to find a dynamic function that fits the data points best.



PARAMETRIZE IT!

Neural network as an ODE

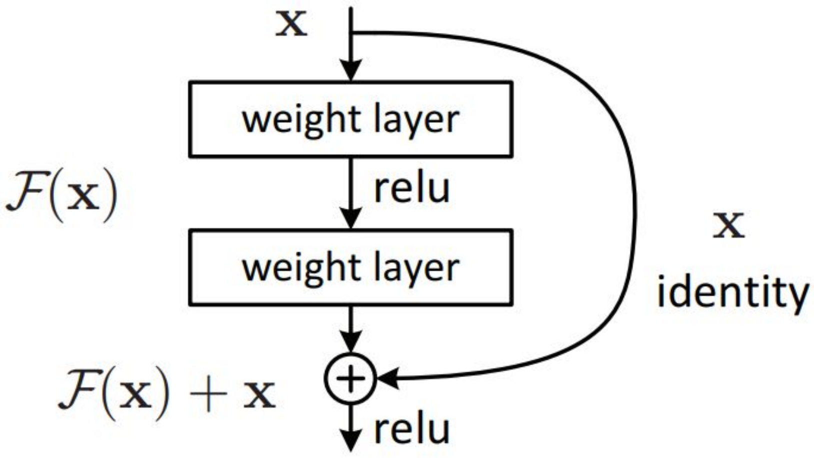
Sequence of transformations

Models such as:

- Residual networks
- RNNs
- Normalizing flows

use a combination of transformations:

$$x_{t+1} = x_t + f(x_t, \theta_t)$$



Source: He, Kaiming, et al., 2016

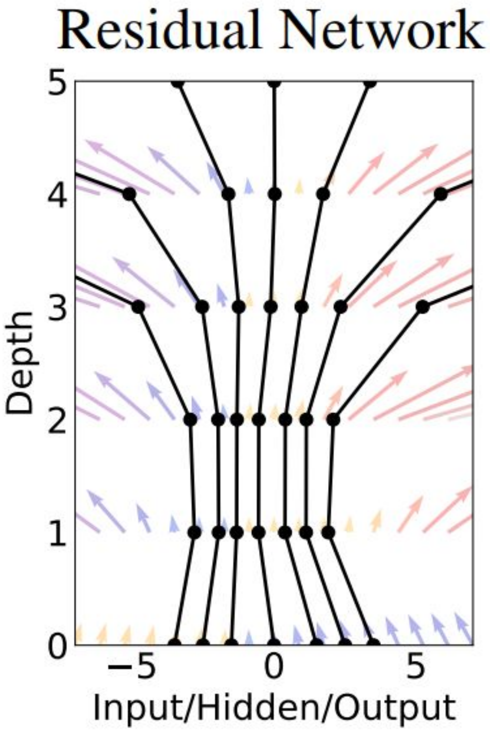
Sequence of transformations

Models such as:

- Residual networks
- RNNs
- Normalizing flows

use a combination of transformations:

$$x_{t+1} = x_t + f(x_t, \theta_t)$$



Source: Chen, Ricky TQ, et al., 2018

Sequence of transformations

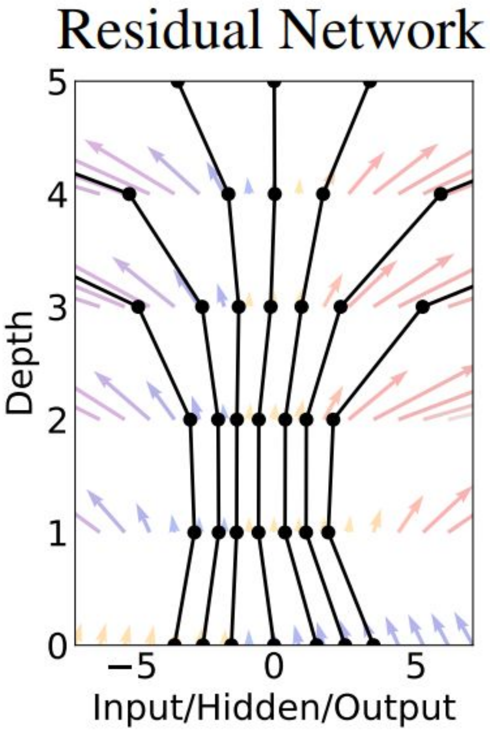
Models such as:

- Residual networks
- RNNs
- Normalizing flows

use a combination of transformations:

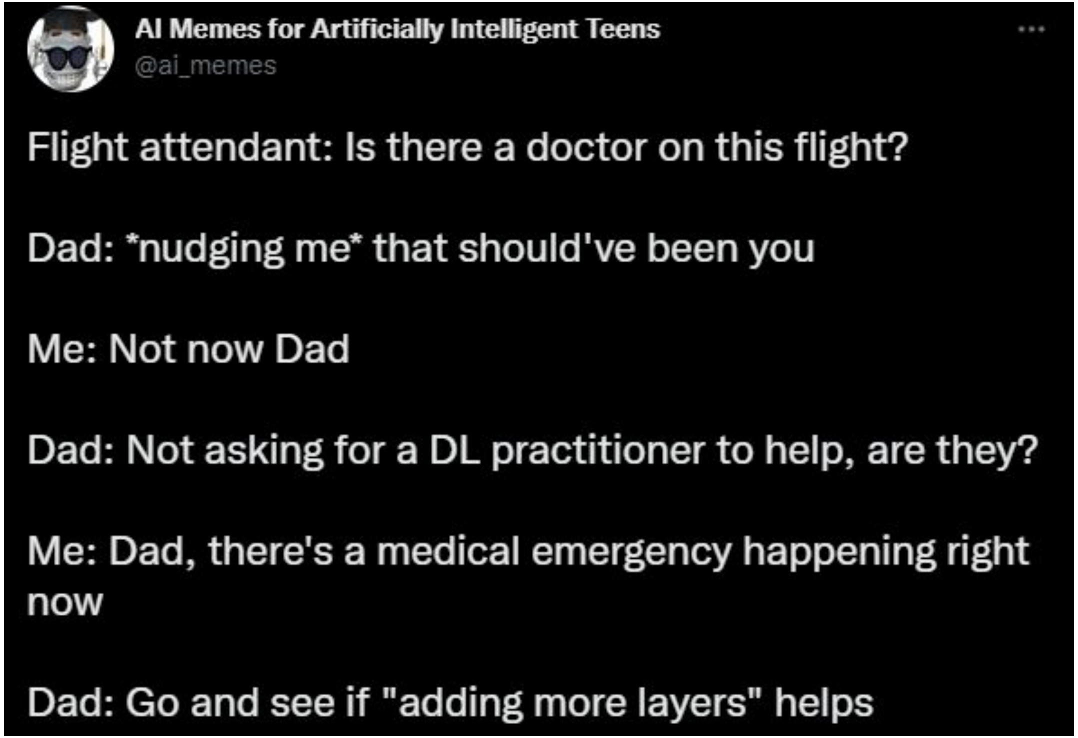
$$x_{t+1} = x_t + f(x_t, \theta_t)$$

This can be seen as an **Euler discretization**!



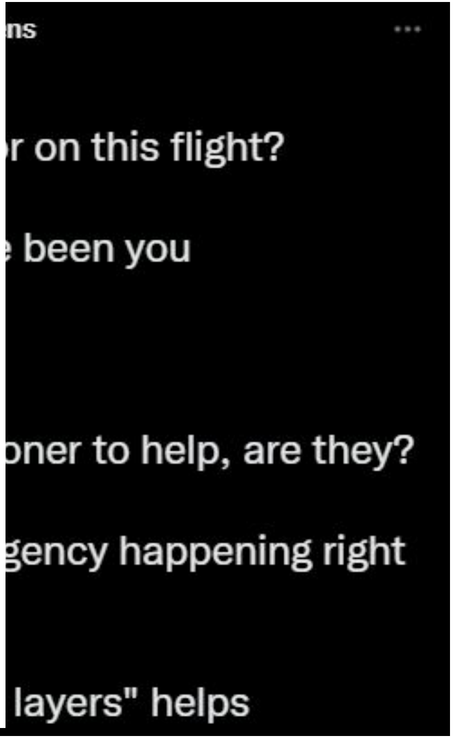
Source: Chen, Ricky TQ, et al., 2018

More layers…



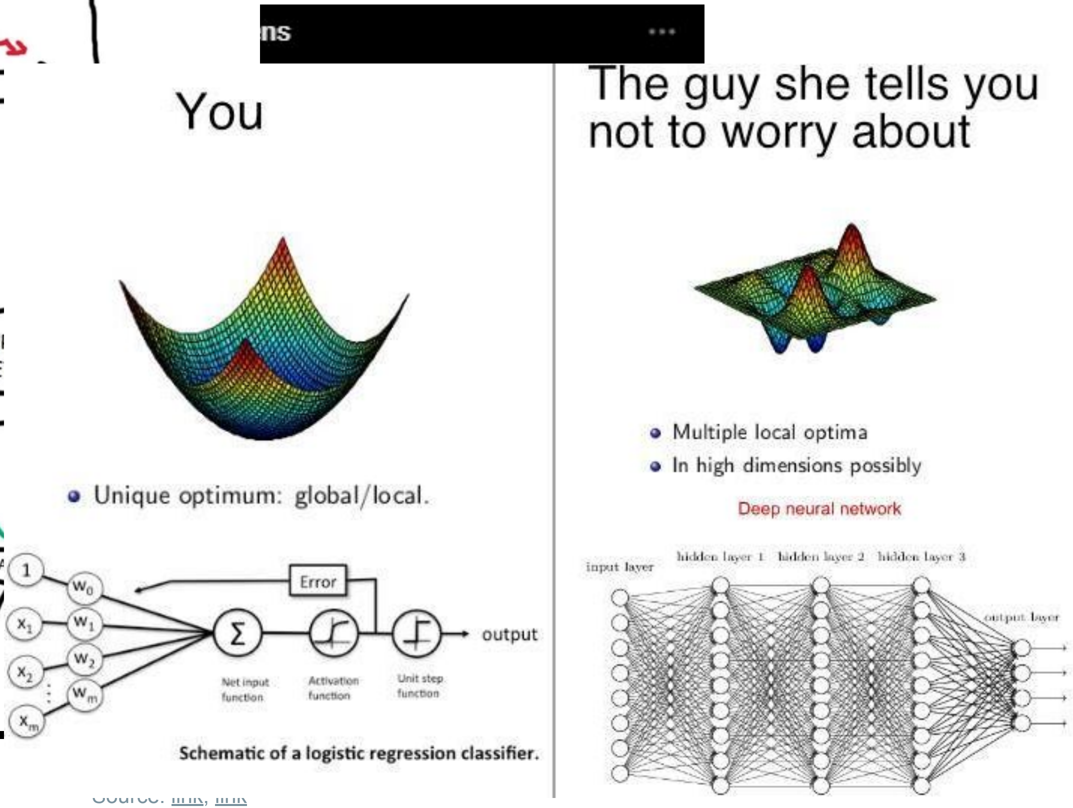
Source: link, link

Mor



Source: [link](), [link]()

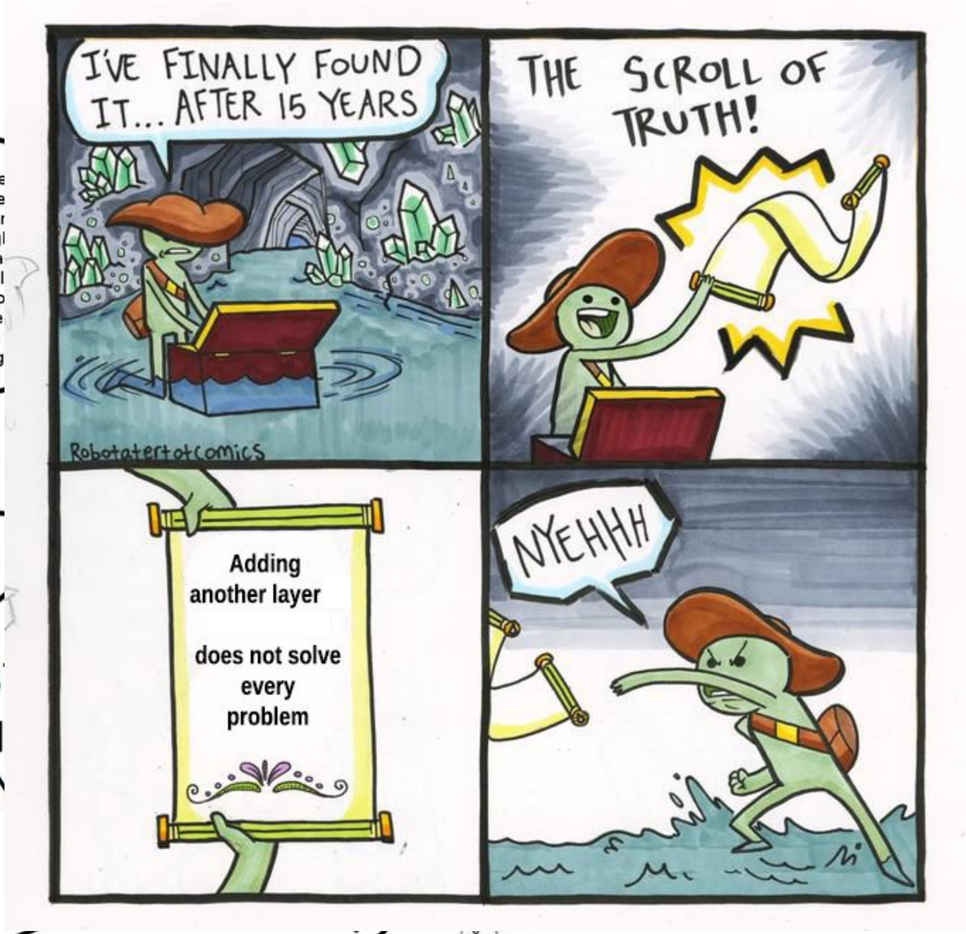
Mor

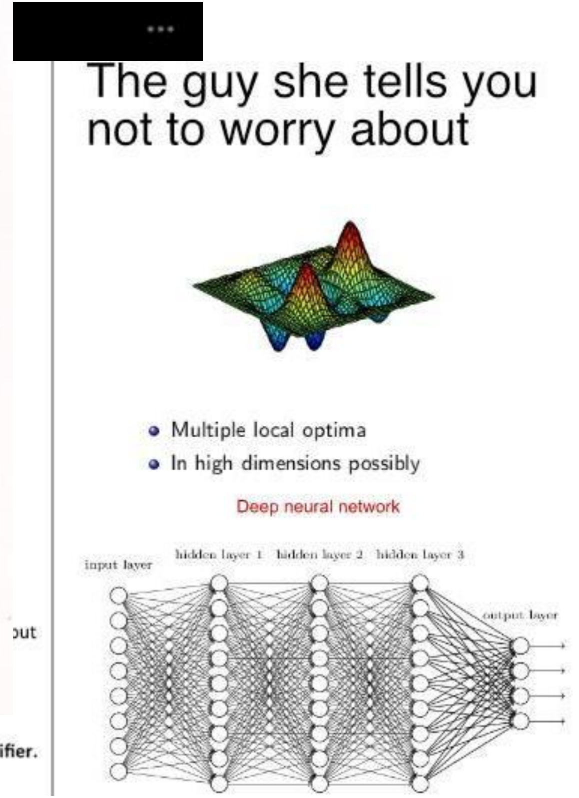


Source: link, link

Mor

Schematic of a logistic regression classifier.

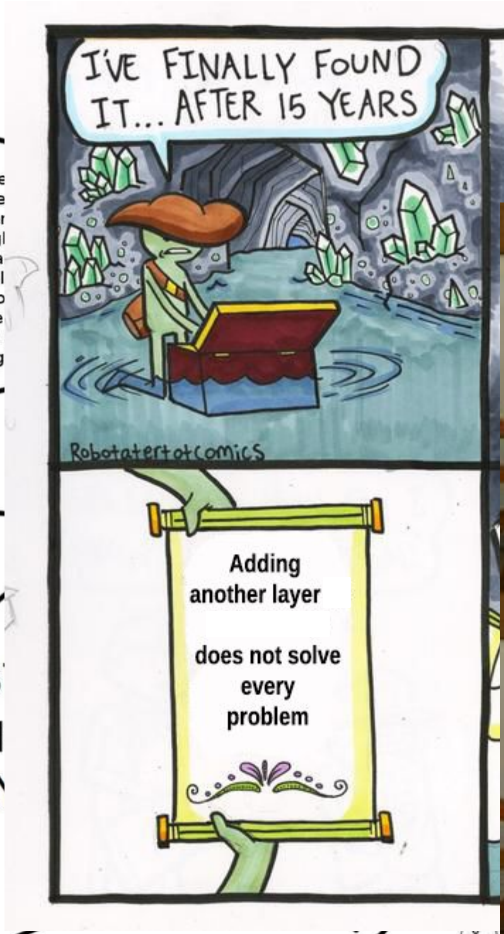




Source: link, link

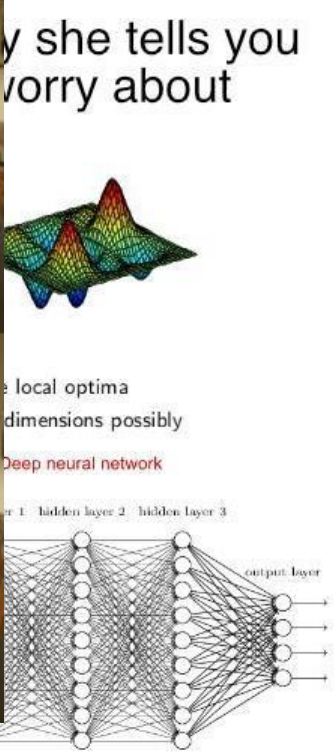
Mor

Gentleme
overgene
VC-Dimen
is too high
experts a
structural
Rework o
make the
unbiased
soft marg
S
M
LA



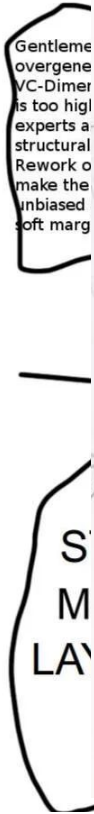
I'VE FINALLY FOUND IT... AFTER 15 YEARS
THE SCROLL OF TRUTH!
Robotatertotcomics
Adding another layer
does not solve every problem

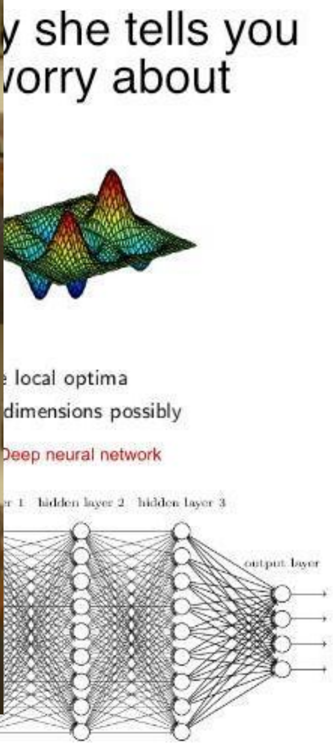
*Statistician
*Data scientist
*variance bounds
Kernel size, VC dimension
*add more layers



y she tells you
vorry about
local optima
dimensions possibly
Deep neural network
hidden layer 2 hidden layer 3
output layer

Mor





Source: link, link

Continuous latent dynamics

If we add **infinite number of layers** and take **infinitely small steps**, we end up with a continuous dynamics on the hidden states:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

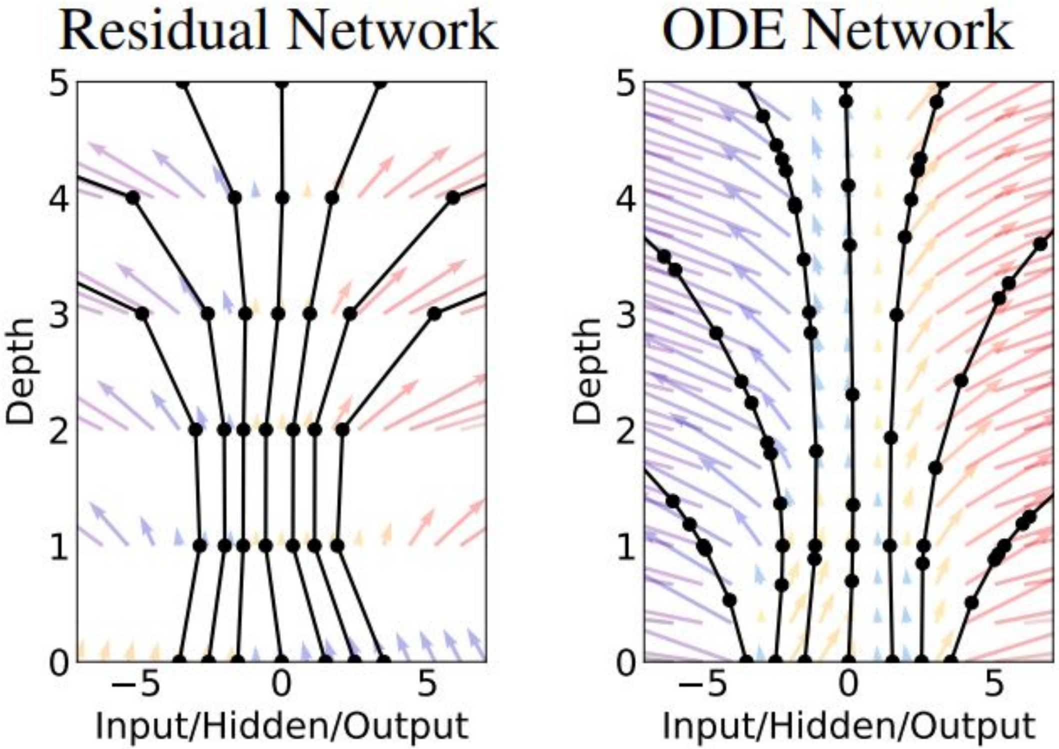
Continuous latent dynamics

If we add **infinite number of layers** and take **infinitely small steps**, we end up with a continuous dynamics on the hidden states:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

Note, that the right side of the equation is a neural network.

Continuous latent dynamics



Source: Chen, Ricky TQ, et al., 2018

Example - MNIST classifier

Let us consider a problem of classifying MNIST digits.

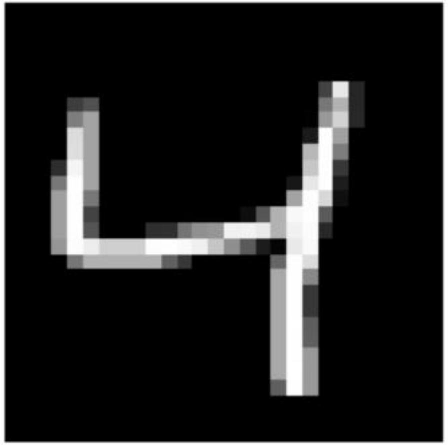
We can define a dynamic function between input (image) and output (class label):

4

Example - MNIST classifier

Let us consider a problem of classifying MNIST digits.

We can define a dynamic function between input (image) and output (class label):



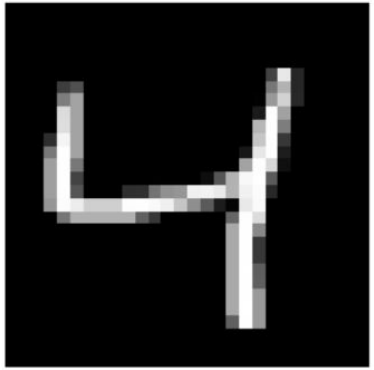
$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

4

$$x(0) = x_0$$

$$x(T) = y$$

Example - MNIST classifier



$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

4

$$x(0) = x_0 \qquad\qquad x(T) = y$$

Once we found the dynamic function, we can classify image using:

$$x(T) = x(0) + \int_0^T f(x(t), t, \theta)dt$$

Dynamic function optimization

We can easily* train our neural net using standard numerical methods (such as RK4).

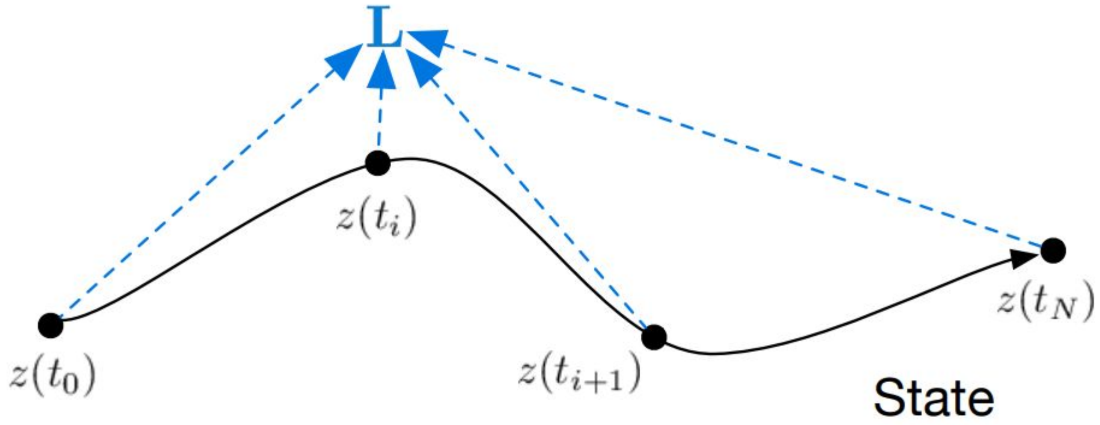
Let us assume we want to optimize some loss function L:

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right)$$

Dynamic function optimization

We can easily* train our neural net using standard numerical methods (such as RK4).

Let us assume we want to optimize some loss function L :



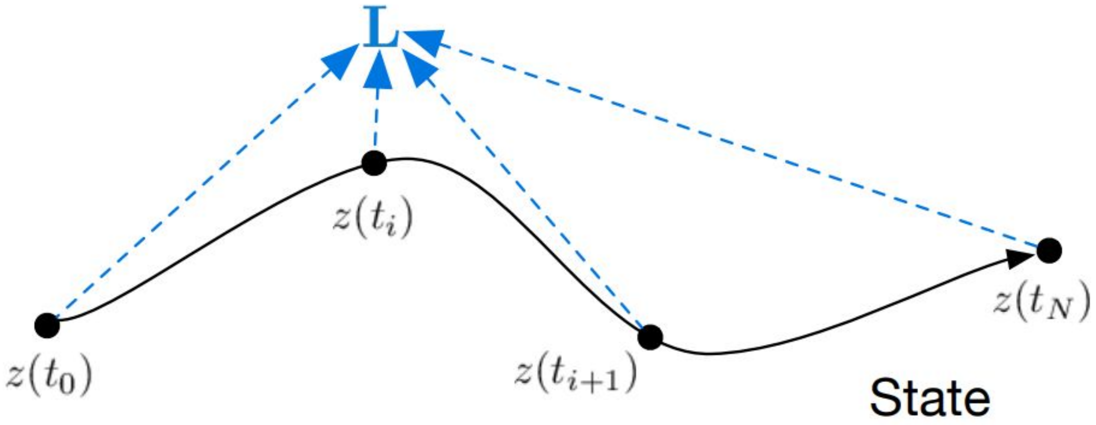
Source: Chen, Ricky TQ, et al., 2018

Dynamic function optimization

We can easily* train our neural net using standard numerical methods (such as RK4).

Let us assume we want to optimize some loss function L :

***We need to remember every step!**



Source: Chen, Ricky TQ, et al., 2018

Adjoint method

To make an optimization step, we need gradients $\frac{dL}{d\theta}$.

Adjoint method

To make an optimization step, we need gradients $\frac{dL}{d\theta}$.

First, let us determine how the loss depends on the hidden state $\mathbf{z}(t)$.

Let us define **adjoint** as:

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$$

Adjoint method

To make an optimization step, we need gradients $\frac{dL}{d\theta}$.

First, let us determine how the loss depends on the hidden state $\mathbf{z}(t)$.
Let us define **adjoint** as:

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$$

It turns out that the adjoint follows its own dynamic function:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

Adjoint method

To make an optimization step, we need gradients $\frac{dL}{d\theta}$.

First, let us determine how the loss depends on the hidden state $\mathbf{z}(t)$.

Let us define **adjoint** as:

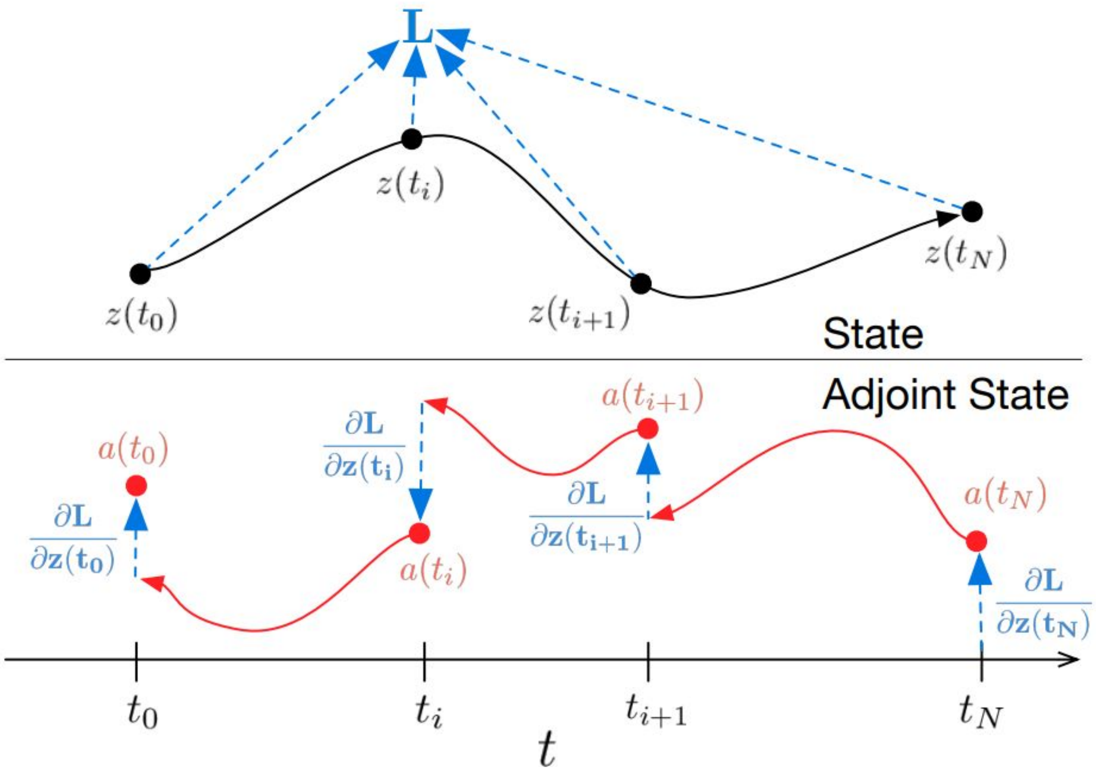
$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$$

It turns out that the adjoint follows its own dynamic function:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

One more pass through an ODE solver!

Adjoint method



Adjoint method

Finally, we can calculate derivatives of the loss function with respect to model's parameters:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Adjoint method

Finally, we can calculate derivatives of the loss function with respect to model's parameters:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Couple of remarks:

- It works with **O(1) memory**!

Adjoint method

Finally, we can calculate derivatives of the loss function with respect to model's parameters:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Couple of remarks:

- It works with **O(1) memory**!
- We can use it only on **imperfect optimization** problems (optimums are bad)!

Adjoint method

Finally, we can calculate derivatives of the loss function with respect to model's parameters:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Couple of remarks:

- It works with **O(1) memory**!
- We can use it only on **imperfect optimization** problems (optimums are bad)!
- For **less complicated models**, we should avoid this method due to the **slower training**!

Adjoint method

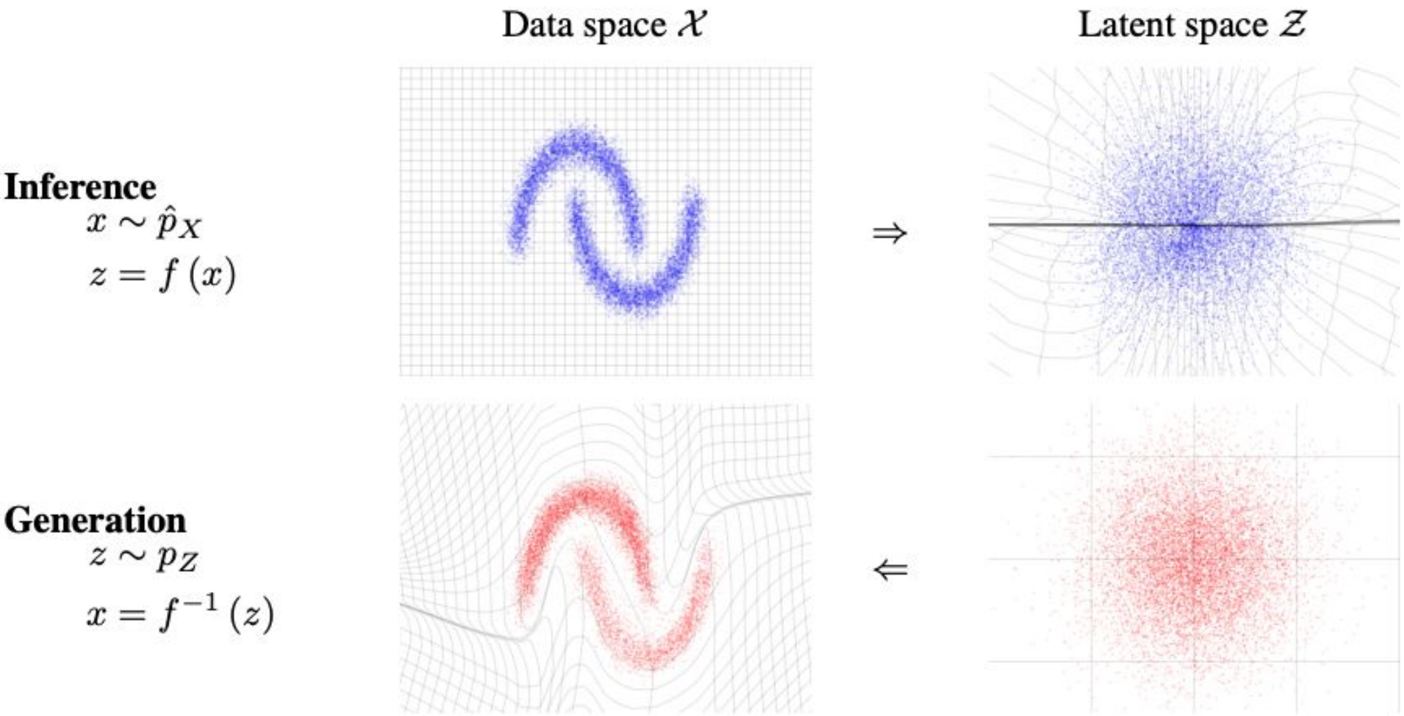
Theory

torchdiffeq

Source: link

Continuous Normalizing Flow (CNF)

Normalizing flows



Source: Dinh, Laurent et al., 2016

Normalizing flows

We can express p_X (likelihood of the data) using **change of variable formula**:

$$p_X(x) = p_Z(f(x))\left|det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|$$

We need f to be a bijection.

Normalizing flows

We can express p_X (likelihood of the data) using **change of variable formula**:

$$p_X(x) = p_Z(f(x)) \left| det\left(\frac{\partial f(x)}{\partial x^T} \right) \right|$$

We need f to be a bijection. But also:

- Easily invertible.
- Simple form of the determinant of the Jacobian.

Real NVP

Let $f = f_n \circ f_{n-1} \circ ... \circ f_1$, where f_i is defined as:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

where (x_1, x_2) is some partition of the input.

Real NVP

Let $f = f_n \circ f_{n-1} \circ \ldots \circ f_1$, where f_i is defined as:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

where (x_1, x_2) is some partition of the input.

- Easily invertible!
- Simple form of the determinant of the Jacobian:

$$\det\left(\frac{\partial f_i(x)}{\partial x^T}\right) = \exp\left(\sum_{j=1}^{d} s(x_1)_j\right)$$

Real NVP

Meh…

Let $f = f_n \circ f_{n-1} \circ \ldots \circ f_1$, where f_i is defined as:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

Meh…

where (x_1, x_2) is some partition of the input.

- Easily invertible!
- Simple form of the determinant of the Jacobian:

$$\det\left(\frac{\partial f_i(x)}{\partial x^T}\right) = \exp\left(\sum_{j=1}^{d} s(x_1)_j\right)$$

Instantaneous change of variables

Now, if we make the transformation continuous in time:

$$\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$$

we are also interested how log-probability changes in time. Instantaneous change of variable theorem shows that:

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

Continuous Normalizing Flow (CNF)

1. **Training** (Data distribution → Prior distribution):

$$\mathbf{z}_0 = \mathbf{z}_T + \int_T^0 f(\mathbf{z}(t), t)dt$$

Continuous Normalizing Flow (CNF)

1. **Training** (Data distribution → Prior distribution):

$$\mathbf{z}_0 = \mathbf{z}_T + \int_T^0 f(\mathbf{z}(t), t)dt$$

2. **Loss function**:

$$\log(p(\mathbf{z}_T)) = \log(p(\mathbf{z}_0)) - \int_0^T \mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}dt\right)$$

Continuous Normalizing Flow (CNF)

1. **Training** (Data distribution → Prior distribution):

$$\mathbf{z}_0 = \mathbf{z}_T + \int_T^0 f(\mathbf{z}(t), t)dt$$

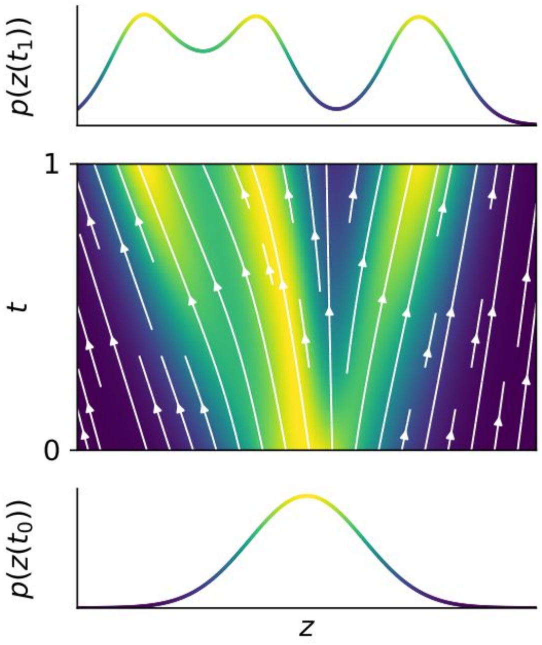
2. **Loss function**:

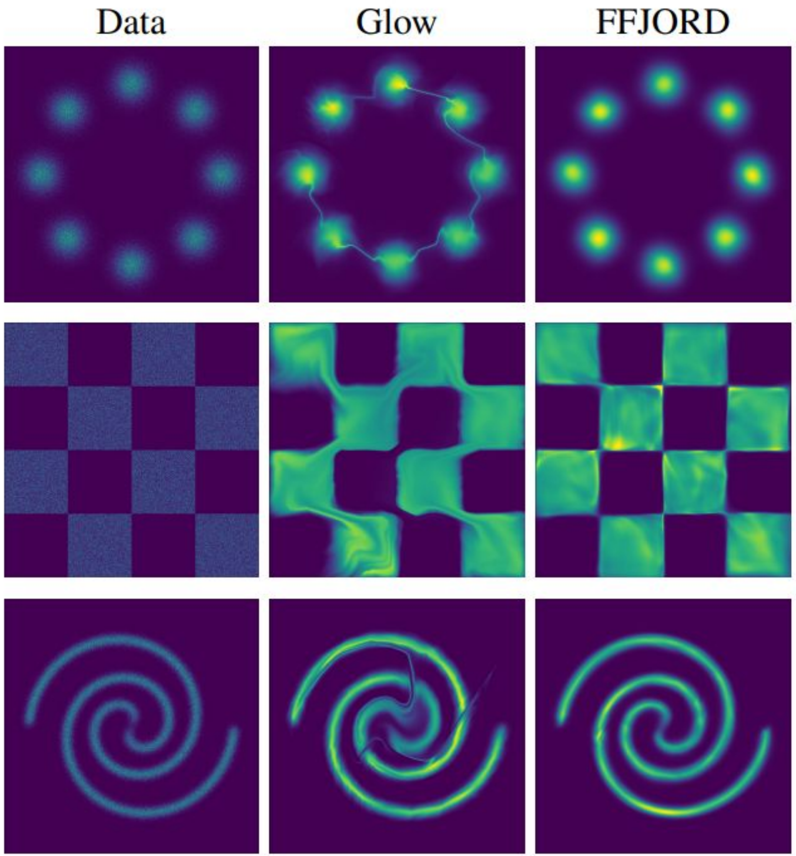
$$\log(p(\mathbf{z}_T)) = \log(p(\mathbf{z}_0)) - \int_0^T \mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}dt\right)$$

3. **Sampling** (Prior distribution → Data distribution):

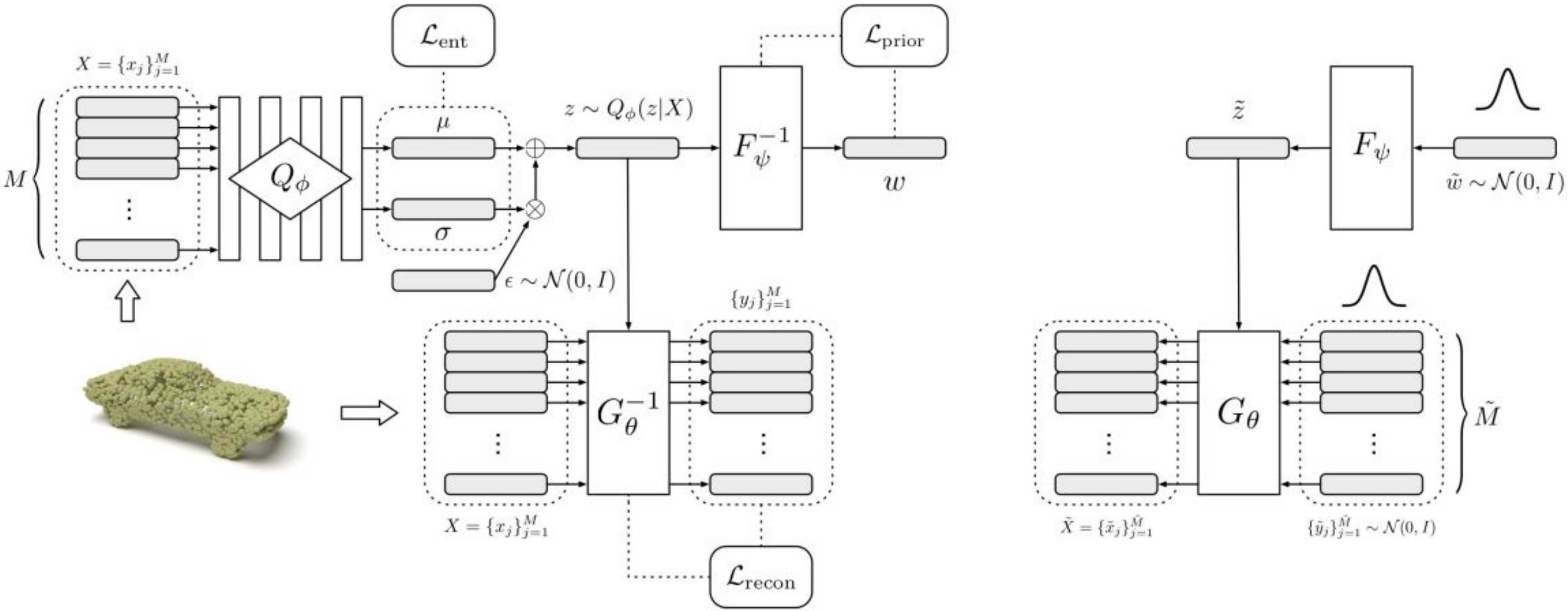
$$\mathbf{z}_T = \mathbf{z}_0 + \int_0^T f(\mathbf{z}(t), t)dt$$

CNF example - FFJORD





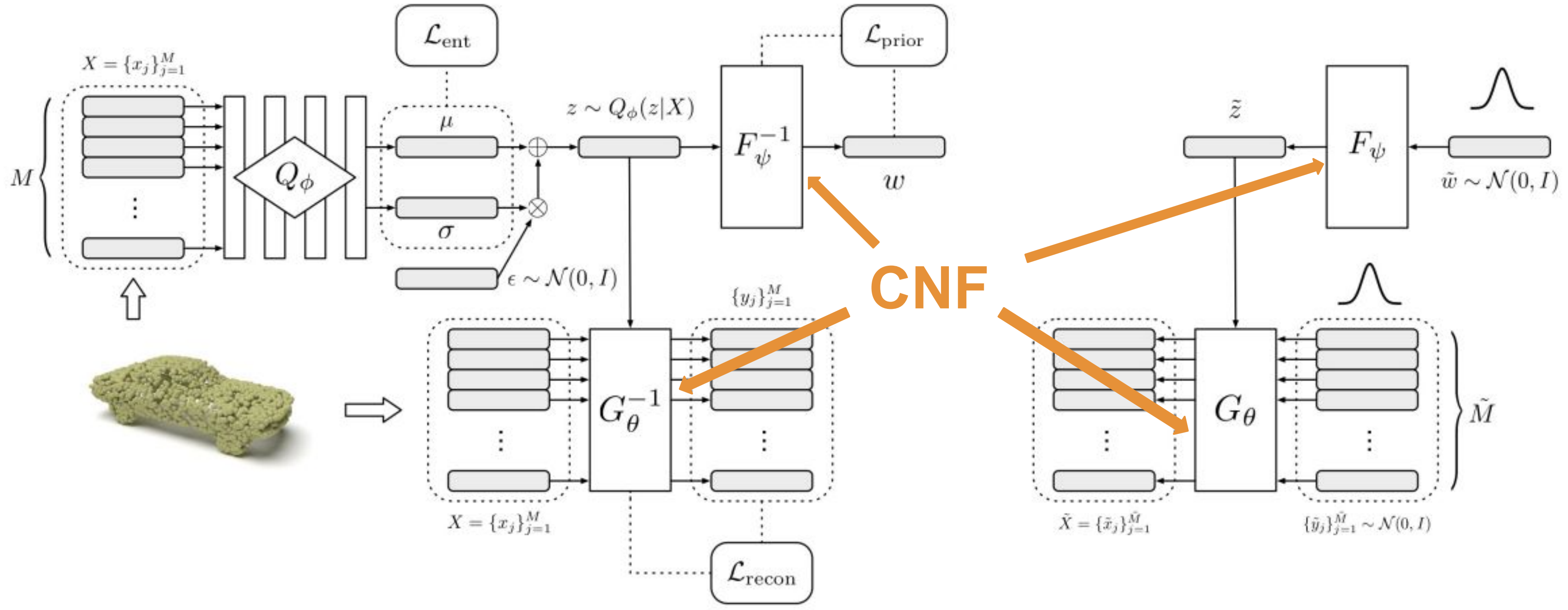
CNF example - PointFlow



(a) Training (Auto-encoding)

(a) Test (Sampling)

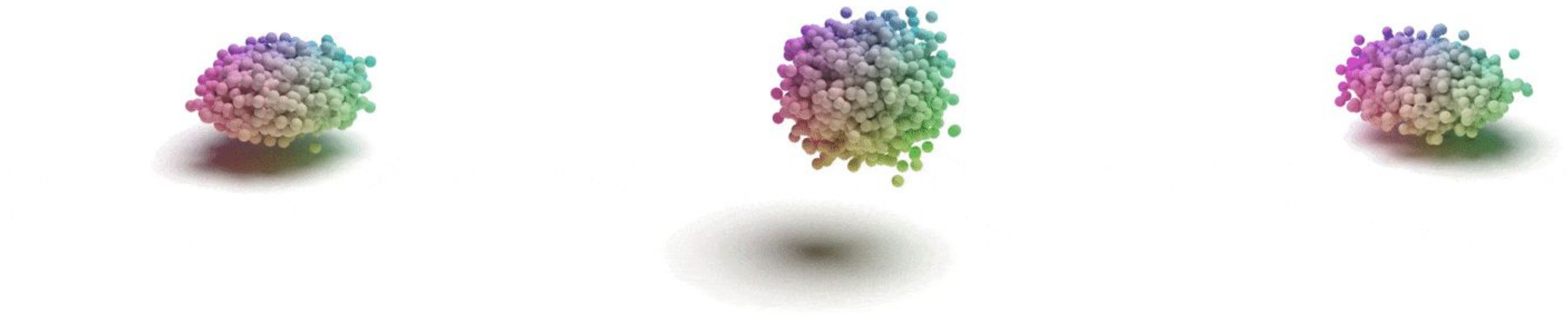
CNF example - PointFlow



(a) Training (Auto-encoding)

(a) Test (Sampling)

CNF example - PointFlow



Thank you!