Neural Ordinary Differential Equations

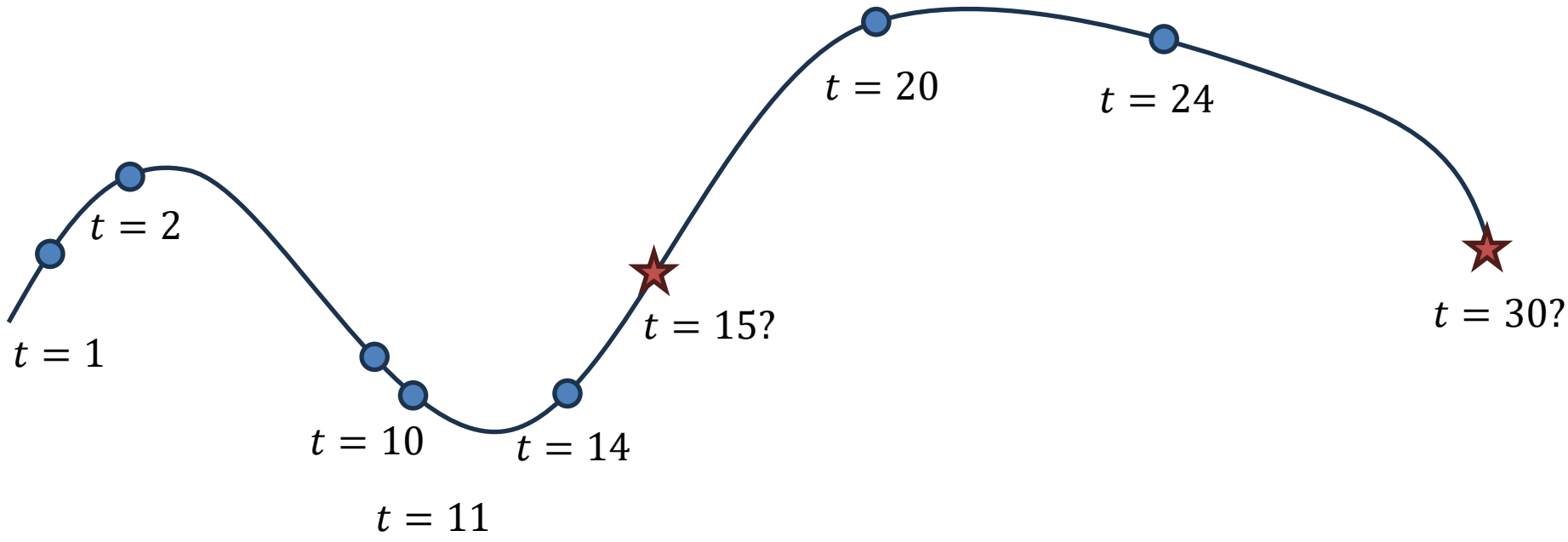
CIDA Lab.

Presentation: Baekryun Seong

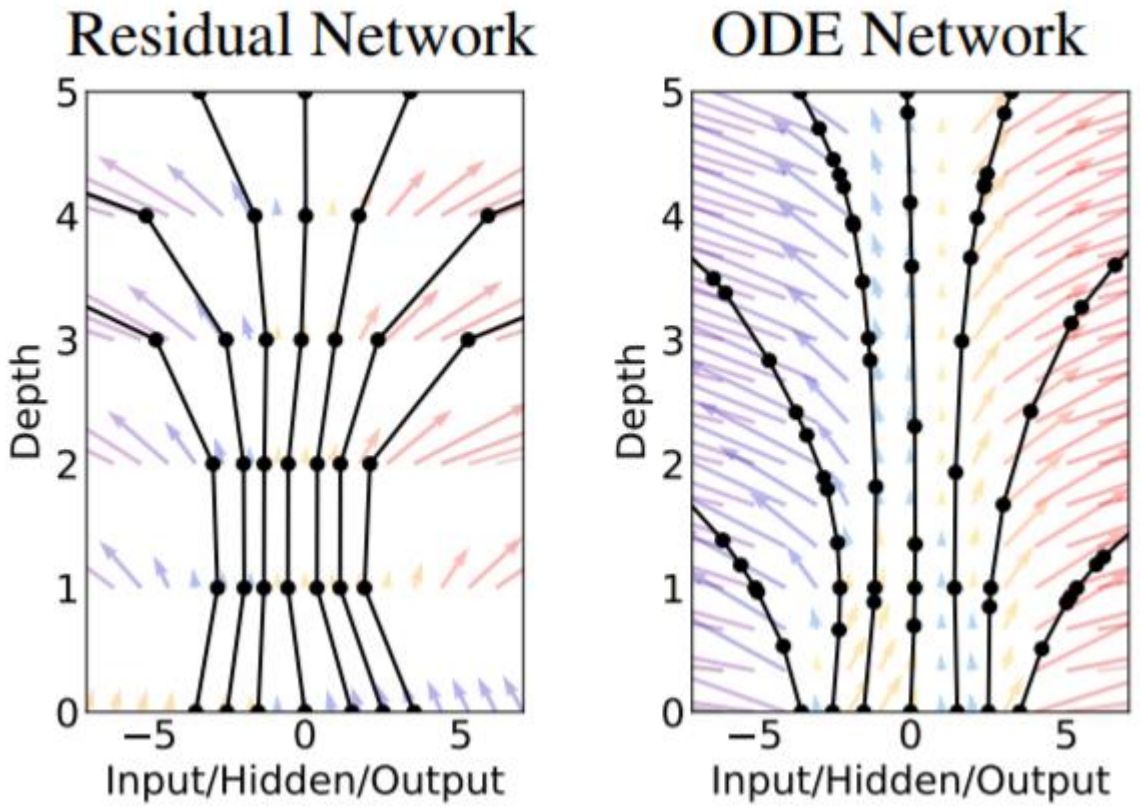
2024.12.27

Table of Contents

Can a recurrent (or residual) neural network predict the future with irregular time points?

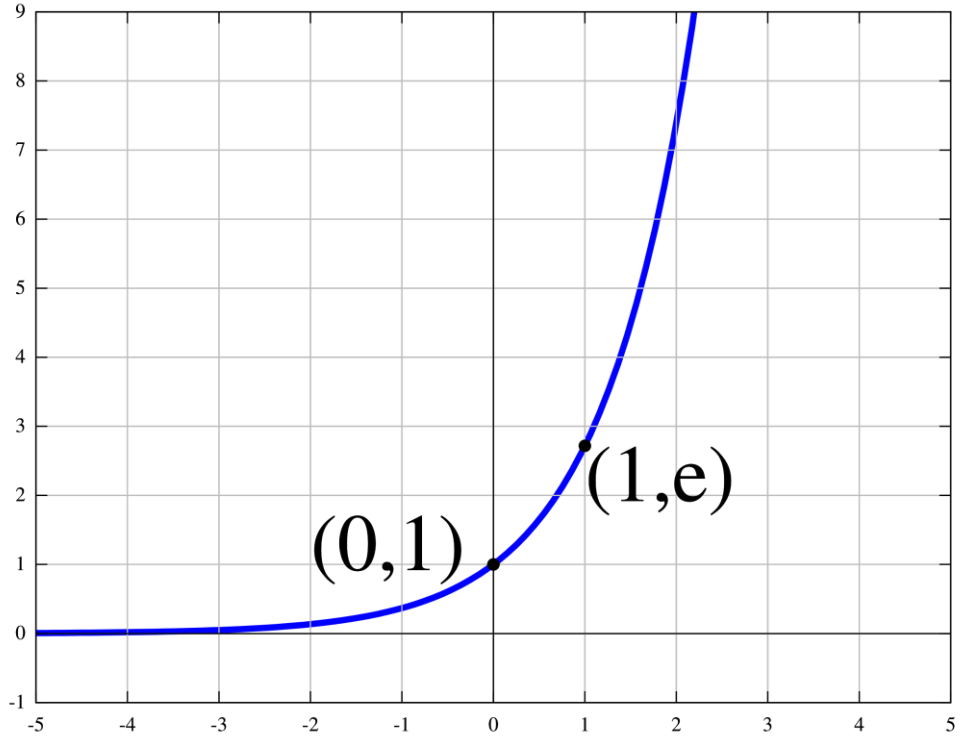


"In classical neural networks, layers are arranged in a sequence indexed by natural numbers. In neural ODEs, however, ***layers form a continuous family indexed by positive real numbers****."*



Neural differential equation - Wikipedia, obtained 2024.12.27.

Preliminary



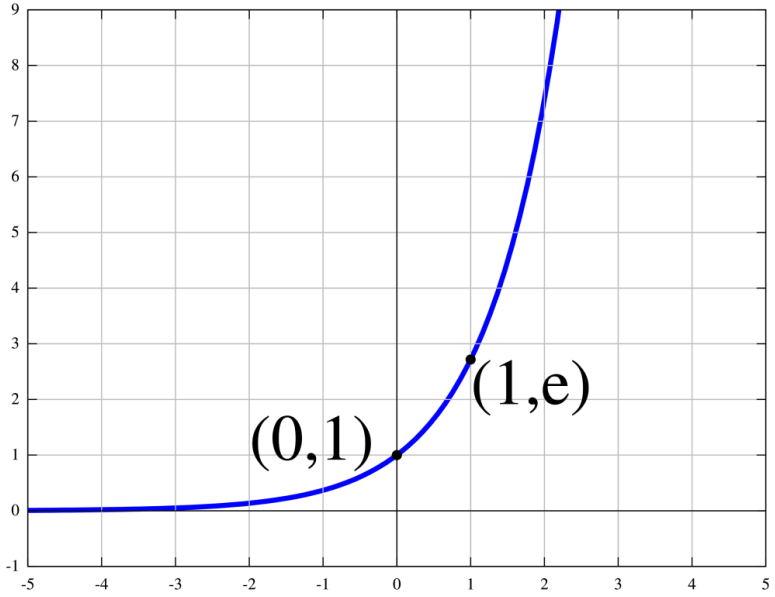
$$\frac{dy}{dx}(x) = y(x)$$

지수 함수 - 위키백과, 우리 모두의 백과사전, obtained 2024.12.27.

Ordinary Differential Equation (ODE) is an equation

that consists of single-valued function and its derivatives.

ODEs have functions as solutions.

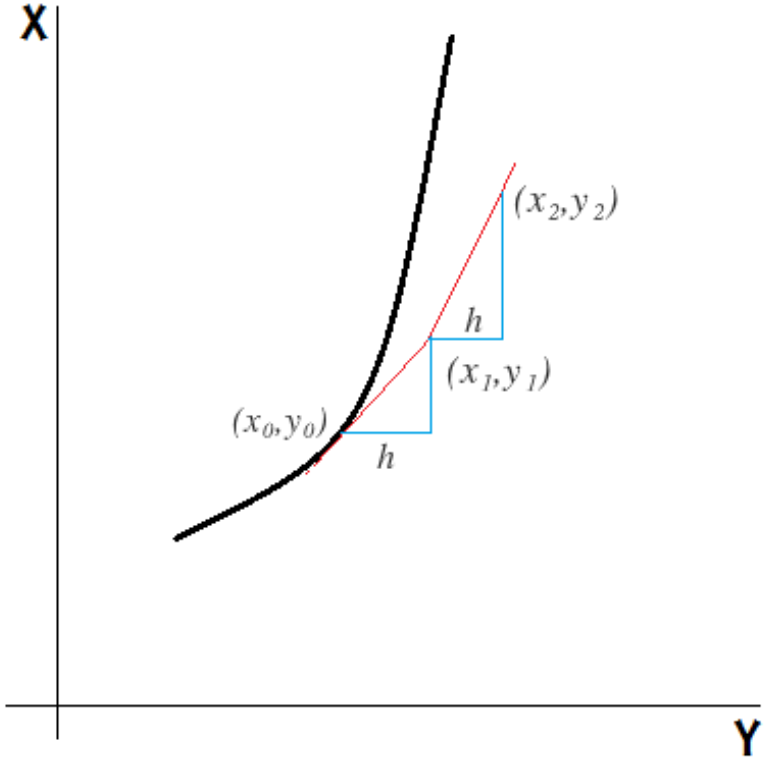


$$y'(x) = \frac{dy}{dx}(x) = y(x)$$

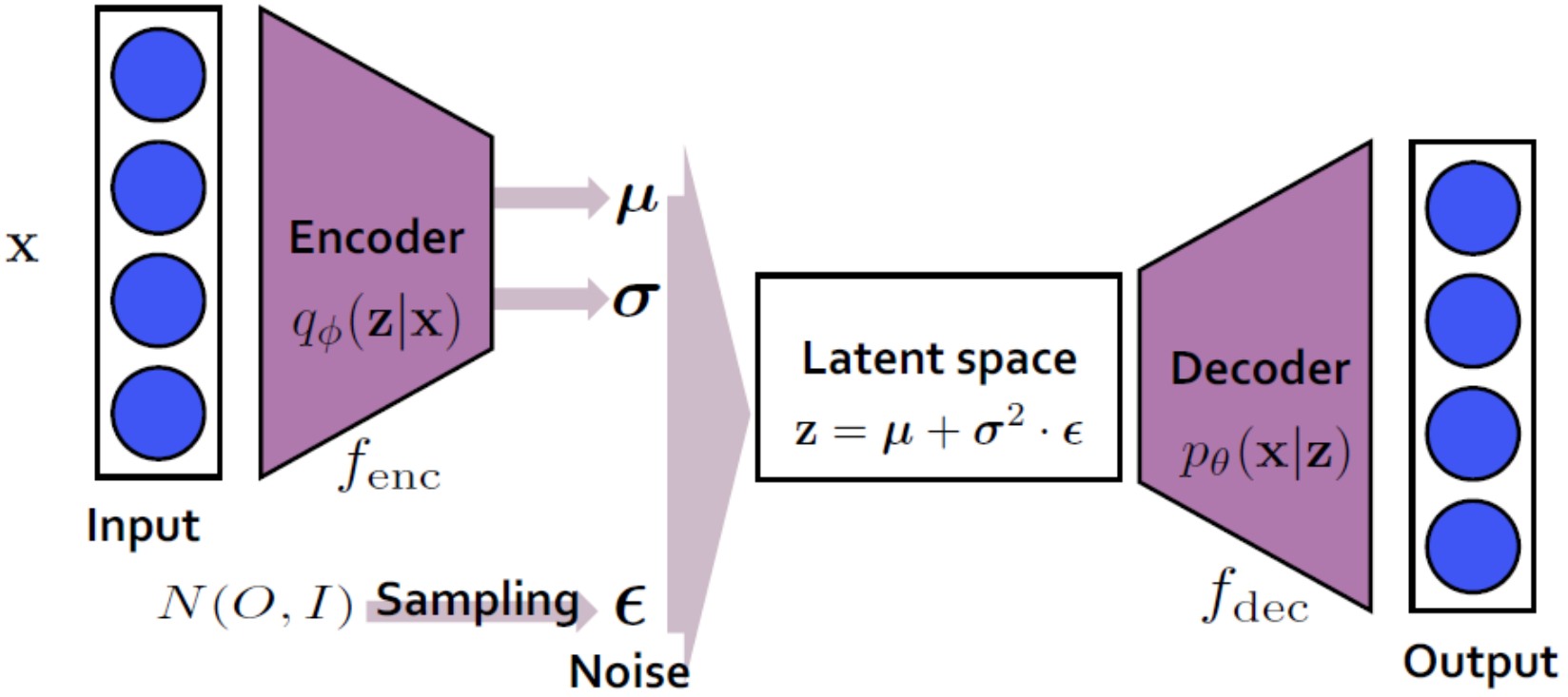
$$y(x) = Ce^{x}$$

In general, we can’t get the exact function that satisfies the ODE.

Instead, we approximately calculate the value with Euler’s method.



Differential Equations - Euler's Method - Small Step Size | Brilliant Math & Science Wiki



Code:

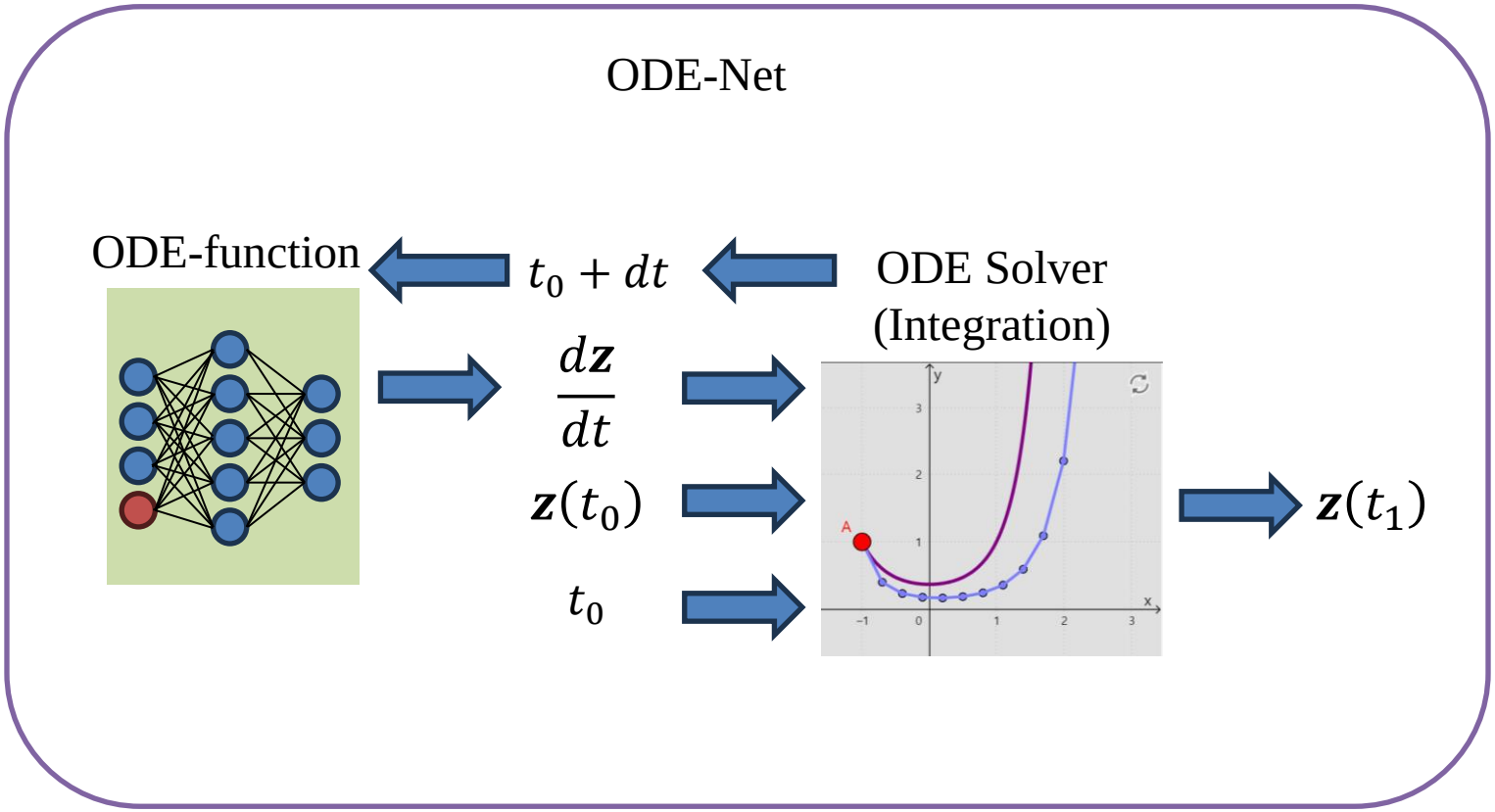
[minimal_vae/vae_mnist.py at master · adler-j/minimal_vae · GitHub]

Replacing Residual Networks with ODEs for Supervised Learning

Table 1: Performance on MNIST. †From LeCun et al. (1998).

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Structure of ODE Network



Replacing Residual Networks with ODEs for Supervised Learning

```python
class ConcatConv2d(nn.Module):

    def __init__(self, dim_in, dim_out, ksize=3, stride=1, padding=0, dilation=1, groups=1, bias=True, transpose=False):
        super(ConcatConv2d, self).__init__()
        module = nn.ConvTranspose2d if transpose else nn.Conv2d
        self._layer = module(
            dim_in + 1, dim_out, kernel_size=ksize, stride=stride, padding=padding, dilation=dilation, groups=groups,
            bias=bias
        )

    def forward(self, t, x):
        tt = torch.ones_like(x[:, :1, :, :]) * t
        ttx = torch.cat([tt, x], 1)
        return self._layer(ttx)


class ODEfunc(nn.Module):

    def __init__(self, dim):
        super(ODEfunc, self).__init__()
        self.norm1 = norm(dim)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm2 = norm(dim)
        self.conv2 = ConcatConv2d(dim, dim, 3, 1, 1)
        self.norm3 = norm(dim)
        self.nfe = 0

    def forward(self, t, x):
        self.nfe += 1
        out = self.norm1(x)
        out = self.relu(out)
        out = self.conv1(t, out)
        out = self.norm2(out)
        out = self.relu(out)
        out = self.conv2(t, out)
        out = self.norm3(out)
        return out
```

torchdiffeq/examples/odenet_mnist.py at master · rtqichen/torchdiffeq, obtained 2024.12.27.

Replacing Residual Networks with ODEs for Supervised Learning

```python
class ODEBlock(nn.Module):

    def __init__(self, odefunc):
        super(ODEBlock, self).__init__()
        self.odefunc = odefunc
        self.integration_time = torch.tensor([0, 1]).float()

    def forward(self, x):
        self.integration_time = self.integration_time.type_as(x)
        out = odeint(self.odefunc, x, self.integration_time, rtol=args.tol, atol=args.tol)
        return out[1]

    @property
    def nfe(self):
        return self.odefunc.nfe

    @nfe.setter
    def nfe(self, value):
        self.odefunc.nfe = value
```

torchdiffeq/examples/odenet_mnist.py at master · rtqichen/torchdiffeq, obtained 2024.12.27.

Generative Latent Function Time-series Model

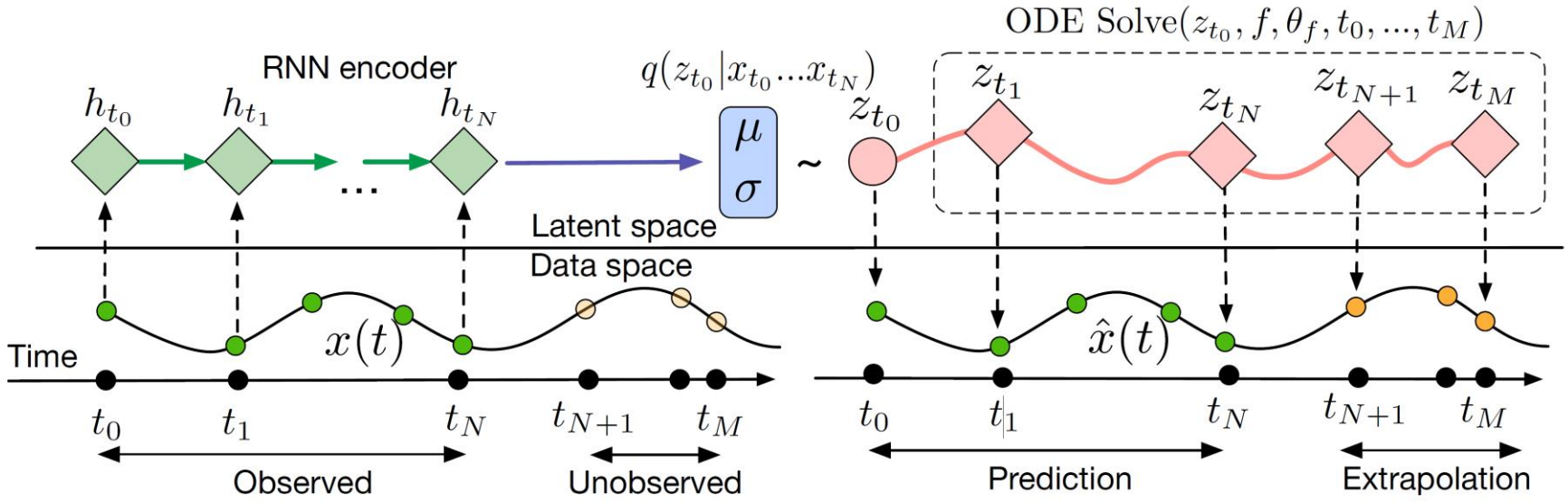


Figure 6: Computation graph of the latent ODE model.

Generative Latent Function Time-series Model with Irregular Time Points

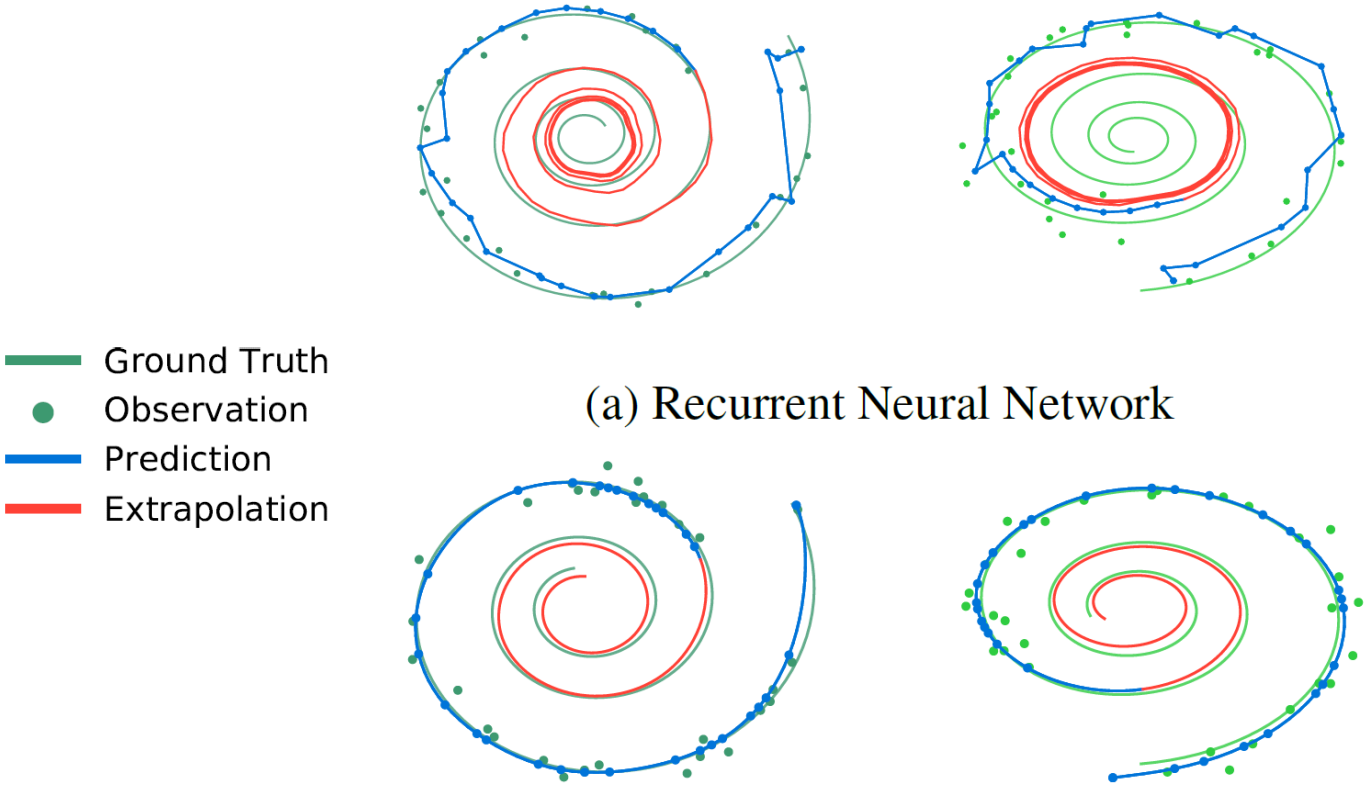
```
try:
    for itr in range(1, args.niters + 1):
        optimizer.zero_grad()
        # backward in time to infer q(z_0)
        h = rec.initHidden().to(device)
        for t in reversed(range(samp_trajs.size(1))):
            obs = samp_trajs[:, t, :]
            out, h = rec.forward(obs, h)
        qz0_mean, qz0_logvar = out[:, :latent_dim], out[:, latent_dim:]
        epsilon = torch.randn(qz0_mean.size()).to(device)
        z0 = epsilon * torch.exp(.5 * qz0_logvar) + qz0_mean

        # forward in time and solve ode for reconstructions
        pred_z = odeint(func, z0, samp_ts).permute(1, 0, 2)
        pred_x = dec(pred_z)

        # compute loss
        noise_std_ = torch.zeros(pred_x.size()).to(device) + noise_std
        noise_logvar = 2. * torch.log(noise_std_).to(device)
        logpx = log_normal_pdf(
            samp_trajs, pred_x, noise_logvar).sum(-1).sum(-1)
        pz0_mean = pz0_logvar = torch.zeros(z0.size()).to(device)
        analytic_kl = normal_kl(qz0_mean, qz0_logvar,
                                pz0_mean, pz0_logvar).sum(-1)
        loss = torch.mean(-logpx + analytic_kl, dim=0)
        loss.backward()
        optimizer.step()
        loss_meter.update(loss.item())
```

torchdiffeq/examples/latent_ode.py at master · rtqichen/torchdiffeq, obtained 2024.12.27.

Generative Latent Function Time-series Model with Irregular Time Points



(a) Recurrent Neural Network

(b) Latent Neural Ordinary Differential Equation

Disccusion

Reconstructing the state trajectory by running the dynamics backwards can introduce extra numerical error if the reconstructed trajectory diverges from the original.

This problem can be solved using checkpointing.