demo_dkf_mv

December 5, 2021

1 Deep Kalman Filter

- https://arxiv.org/pdf/1511.05121.pdf
- https://arxiv.org/pdf/1609.09869.pdf
- DKFDynamicSystem class:
  https://github.com/Google-Health/records-research/blob/master/state-space-model/models.py
- MultivariateNormal:
  https://pytorch.org/docs/stable/distributions.html

[ ]:
```
import pandas as pd
import torch
import torch.nn as nn
from torch.distributions import MultivariateNormal
```

1.1 Model

1.1.1 GatedTransition

Used when computing prior probability

[ ]:
```
class GatedTransition(nn.Module):
    def __init__(self, z_dim, hid_dim):
        super(GatedTransition, self).__init__()

        self.gate = nn.Sequential(nn.Linear(z_dim, hid_dim),
                                  nn.ReLU(),
                                  nn.Linear(hid_dim, z_dim),
                                  nn.Sigmoid())

        self.proposed_mean = nn.Sequential(nn.Linear(z_dim, hid_dim),
                                           nn.ReLU(),
                                           nn.Linear(hid_dim, z_dim))

        self.z_to_mu = nn.Linear(z_dim, z_dim)
```

```
        # modify the default initialization of z_to_mu
        # so that it starts out as the identity function
        self.z_to_mu.weight.data = torch.eye(z_dim)
        self.z_to_mu.bias.data = torch.zeros(z_dim)

        self.z_to_logvar = nn.Linear(z_dim, z_dim)
        self.relu = nn.ReLU()

    def forward(self, z_t_1):
        #
        gate = self.gate(z_t_1)
        proposed_mean = self.proposed_mean(z_t_1)
        mu = (1 - gate) * self.z_to_mu(z_t_1) + gate * proposed_mean
        logvar = self.z_to_logvar(self.relu(proposed_mean))
        # sampling
        eps = torch.randn(z_t_1.size())
        z_t = mu + eps * torch.exp(.5 * logvar)
        return z_t, mu, logvar
```

1.1.2 Combiner

Used when computing posterior probability

[ ]:
```
class Combiner(nn.Module):
    # PostNet
    def __init__(self, z_dim, hid_dim):
        super(Combiner, self).__init__()
        self.z_dim = z_dim
        self.z_to_hidden = nn.Linear(z_dim, hid_dim)
        self.hidden_to_mu = nn.Linear(hid_dim, z_dim)
        self.hidden_to_logvar = nn.Linear(hid_dim, z_dim)
        self.tanh = nn.Tanh()

    def forward(self, z_t_1, h_rnn):
        # combine the rnn hidden state with a transformed version of z_t_1
        h_combined = 0.5 * (self.tanh(self.z_to_hidden(z_t_1)) + h_rnn)
        # use the combined hidden state
        # to compute the mean used to sample z_t
        mu = self.hidden_to_mu(h_combined)
        # use the combined hidden state
        # to compute the scale used to sample z_t
        logvar = self.hidden_to_logvar(h_combined)
        eps = torch.randn(z_t_1.size())
        z_t = mu + eps * torch.exp(.5 * logvar)
        return z_t, mu, logvar
```

1.1.3 Emitter

[ ]:
```
class Emitter(nn.Module):
    def __init__(self, z_dim, hid_dim, input_dim) -> None:
        super().__init__()
        self.input_dim = input_dim
        self.z_to_hidden = nn.Linear(z_dim, hid_dim)
        self.hidden_to_hidden = nn.Linear(hid_dim, hid_dim)
        self.hidden_to_input_mu = nn.Linear(hid_dim, input_dim)
        self.logvar = nn.Parameter(torch.ones(input_dim))
        self.relu = nn.ReLU()

    def forward(self, z_t):
        h1 = self.relu(self.z_to_hidden(z_t))
        h2 = self.relu(self.hidden_to_hidden(h1))
        mu = self.hidden_to_input_mu(h2)
        # return mu  # x_t
        eps = torch.randn(z_t.size(0), self.input_dim)
        x_t = mu + eps * torch.exp(.5 * self.logvar)
        return x_t, mu, self.logvar
```

[ ]:
```
class Encoder(nn.Module):
    def __init__(self) -> None:
        super().__init__()
```

1.1.4 Main class

- encoder

[ ]:
```
class DKF(nn.Module):
    # Structured Inference Networks
    # Current version ignores backward RNN outputs
    def __init__(self, input_dim, z_dim=50, trans_dim=30, emission_dim=30,
                 rnn_dim=100, num_rnn_layers=1) -> None:

        super().__init__()
        self.input_dim = input_dim
        self.z_dim = z_dim
        self.trans_dim = trans_dim
        self.emission_dim = emission_dim
        self.rnn_dim = rnn_dim
        self.num_rnn_layers = num_rnn_layers

        self.trans = GatedTransition(z_dim, trans_dim)
        self.emitter = Emitter(z_dim, emission_dim, input_dim)
```

```
        self.combiner = Combiner(z_dim, rnn_dim)

        self.z_0 = nn.Parameter(torch.zeros(z_dim))
        self.z_q_0 = nn.Parameter(torch.zeros(z_dim))
        self.h_0 = nn.Parameter(torch.zeros(1, 1, rnn_dim))

        # corresponding learning 'l' in the original code
        self.rnn = nn.RNN(input_size=input_dim,
                          hidden_size=rnn_dim,
                          nonlinearity="relu",
                          batch_first=True,
                          bidirectional=False,
                          num_layers=num_rnn_layers)

    def kl_div(self, mu1, logvar1, mu2=None, logvar2=None):

        if mu2 is None:
            mu2 = torch.zeros(1, device=mu1.device)

        if logvar2 is None:
            logvar2 = torch.zeros(1, device=mu1.device)

        return torch.sum(0.5 * (
            logvar2 - logvar1 + (torch.exp(logvar1) + (mu1 - mu2).pow(2))
            / torch.exp(logvar2) - torch.ones(1, device=mu1.device)
        ), 1)

    def infer(self, x):

        batch_size, T_max, x_dim = x.size()
        h_0 = self.h_0.expand(1, batch_size, self.rnn_dim).contiguous()
        rnn_out, h_n = self.rnn(x, h_0)

        z_prev = self.z_q_0.expand(batch_size, self.z_q_0.size(0))
        kl_states = torch.zeros((batch_size, T_max))
        rec_losses = torch.zeros((batch_size, T_max))

        for t in range(T_max):
            # p(z_t|z_{t-1})
            z_prior, z_prior_mu, z_prior_logvar = self.trans(z_prev)
            # q(z_t|z_{t-1},x_{t:T})
            z_t, z_mu, z_logvar = self.combiner(z_prev, rnn_out[:, t])
            # p(x_t|z_t)
            x_t, x_mu, x_logvar = self.emitter(z_t)

            # compute loss
            kl_states[:, t] = self.kl_div(
```

```
                z_mu, z_logvar, z_prior_mu, z_prior_logvar)
            rec_losses[:, t] = nn.MSELoss(reduction='none')(
                x_t.contiguous().view(-1),
                # x_mu.contiguous().view(-1),
                x[:, t].contiguous().view(-1)
            ).view(batch_size, -1).mean(dim=1)

            z_prev = z_t

        return rec_losses.mean(), kl_states.mean()

    def filter(self, x, num_sample=100):

        # Outputs
        x_hat = torch.zeros(x.size())  # predictions
        x_025 = torch.zeros(x.size())
        x_975 = torch.zeros(x.size())

        batch_size, T_max, x_dim = x.size()
        assert batch_size == 1
        z_prev = self.z_0.expand(num_sample, self.z_0.size(0))

        h_0 = self.h_0.expand(1, 1, self.rnn_dim).contiguous()
        rnn_out, _ = self.rnn(x, h_0)
        rnn_out = rnn_out.expand(num_sample,
            rnn_out.size(1), rnn_out.size(2))

        for t in range(T_max):
            # z_t: (num_sample, z_dim)
            z_t, z_mu, z_logvar = self.combiner(z_prev, rnn_out[:, t])
            x_t, x_mu, x_logvar = self.emitter(z_t)
            # x_hat[:, t] = x_mu

            x_covar = torch.diag(torch.sqrt(torch.exp(.5 * x_logvar)))
            x_samples = MultivariateNormal(
                x_mu, covariance_matrix=x_covar).sample()
            # # sampling z_t and computing quantiles
            # x_samples = MultivariateNormal(
            #     loc=x_mu, covariance_matrix=x_covar).sample_n(num_sample)

            x_hat[:, t] = x_samples.mean(0)
            x_025[:, t] = x_samples.quantile(0.025, 0)
            x_975[:, t] = x_samples.quantile(0.975, 0)

            # x_hat[:, t] = x_t.mean(0)
            # x_025[:, t] = x_t.quantile(0.025, 0)
            # x_975[:, t] = x_t.quantile(0.975, 0)
```

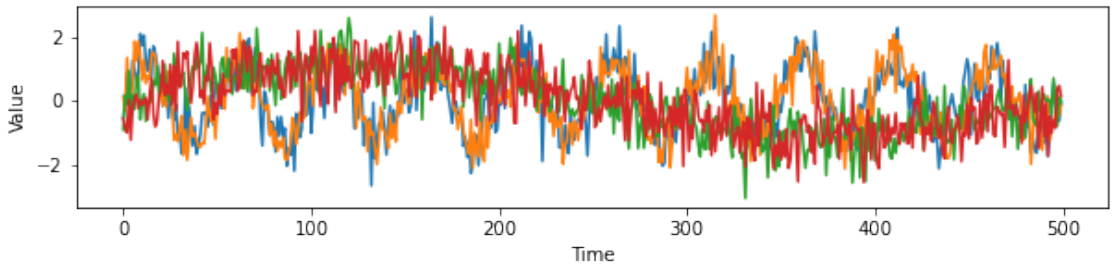
```
            z_prev = z_t
            # z_prev = z_mu

        return x_hat, x_025, x_975

    def predict(self, x, pred_steps=1, num_sample=100):
        """ x should contain the prediction period
        """
        # Outputs
        x_hat = torch.zeros(x.size())  # predictions
        x_025 = torch.zeros(x.size())
        x_975 = torch.zeros(x.size())

        batch_size, T_max, x_dim = x.size()
        assert batch_size == 1
        z_prev = self.z_0.expand(num_sample, self.z_0.size(0))

        h_0 = self.h_0.expand(1, 1, self.rnn_dim).contiguous()
        rnn_out, _ = self.rnn(x[:, :T_max-pred_steps], h_0)
        rnn_out = rnn_out.expand(num_sample,
            rnn_out.size(1), rnn_out.size(2))

        for t in range(T_max - pred_steps):
            # z_t: (num_sample, z_dim)
            z_t, z_mu, z_logvar = self.combiner(z_prev, rnn_out[:, t])
            x_t, x_mu, x_logvar = self.emitter(z_t)

            x_covar = torch.diag(torch.sqrt(torch.exp(.5 * x_logvar)))
            x_samples = MultivariateNormal(
                x_mu, covariance_matrix=x_covar).sample()

            x_hat[:, t] = x_samples.mean(0)
            x_025[:, t] = x_samples.quantile(0.025, 0)
            x_975[:, t] = x_samples.quantile(0.975, 0)

            z_prev = z_mu

        for t in range(T_max - pred_steps, T_max):
            rnn_out, _ = self.rnn(x[:, :t], h_0)
            rnn_out = rnn_out.expand(
                num_sample, rnn_out.size(1), rnn_out.size(2))

            z_t_1, z_mu, z_logvar = self.combiner(z_prev, rnn_out[:, -1])
            z_t, z_mu, z_logvar = self.trans(z_t_1)
            x_t, x_mu, x_logvar = self.emitter(z_t)
```

```
            x_covar = torch.diag(torch.sqrt(torch.exp(.5 * x_logvar)))
            x_samples = MultivariateNormal(
                x_mu, covariance_matrix=x_covar).sample()

            x_hat[:, t] = x_samples.mean(0)
            x_025[:, t] = x_samples.quantile(0.025, 0)
            x_975[:, t] = x_samples.quantile(0.975, 0)

        return x_hat, x_025, x_975

    def train_step(self, x, annealing_factor=0.1):
        self.train()
        # self.rnn.train()
        rec_loss, kl_loss = self.infer(x)
        total_loss = rec_loss + annealing_factor * kl_loss
        self.optimizer.zero_grad()
        total_loss.backward()
        # nn.utils.clip_grad_norm_(self.parameters(), 5.)
        self.optimizer.step()
        return rec_loss.item(), kl_loss.item(), total_loss.item()

    def validation_step(self, x, annealing_factor=0.1):
        self.eval()
        rec_loss, kl_loss = self.infer(x)
        total_loss = rec_loss + annealing_factor * kl_loss
        return rec_loss.item(), kl_loss.item(), total_loss.item()

    def fit(self, x, x_val=None, num_epochs=100, annealing_factor=0.1,
            verbose_step=1, eval_step=1, check_point_path=None,
            patience=20, learning_rate=0.01):

        self.optimizer = torch.optim.Adam(
            self.parameters(), lr=learning_rate)

        losses = []
        kl_losses = []
        rec_losses = []
        val_losses = []
        val_kl_losses = []
        val_rec_losses = []

        for epoch in range(num_epochs):
            try:
                res = self.train_step(x, annealing_factor=annealing_factor)
                losses.append(res[2])
                kl_losses.append(res[1])
                rec_losses.append(res[0])
```

```
            if epoch % verbose_step == verbose_step - 1:
                message =  f'Epoch= {epoch+1}/{num_epochs}, '
                message += f'loss= {res[2]:.3f}, '
                message += f'mse= {res[0]:.3f}, '
                message += f'kld= {res[1]:.3f}'
                print(message)

            if x_val is not None:
                val_res = self.validation_step(x_val, annealing_factor)
                val_losses.append(val_res[2])
                val_kl_losses.append(val_res[1])
                val_rec_losses.append(val_res[0])

            if epoch % eval_step == eval_step - 1 and x_val is not None:
                message = f'\tval_loss= {val_res[2]:.3f}, '
                message += f'val_mse= {val_res[0]:.3f}, '
                message += f'val_kld= {val_res[1]:.3f}'
                print(message)

        except KeyboardInterrupt:
            break

    history = {'loss': losses,
               'kl_loss': kl_losses,
               'rec_loss': rec_losses}

    if x_val is not None:
        history.update({'val_loss': val_losses,
                        'val_kl_loss': val_kl_losses,
                        'val_rec_loss': val_rec_losses})

    return history

def save_model(self, filename):
    """ dkf.pth """
    torch.save(self.to('cpu').state_dict(), filename)

def load_model(self, filename):
    self.load_state_dict(torch.load(filename))

def get_config(self):
    return {
        'input_dim': self.input_dim,
        'z_dim': self.z_dim,
        'trans_dim': self.trans_dim,
        'emission_dim': self.emission_dim,
```

```
            'rnn_dim': self.rnn_dim,
            'num_rnn_layers': self.num_rnn_layers
        }
```

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import scale
# import warnings
# warnings.filterwarnings('ignore')

T = 500  # sequence length
observations = 2*np.sin(np.linspace(0, 20*np.pi, T))
interventions = 2*np.sin(np.linspace(0, 2*np.pi, T))
data = np.vstack([observations, observations*1.2, interventions,␣
 ↪interventions*0.85]).T
data += np.random.randn(*data.shape)
# data[:, 2:] = preprocessing.minmax_scale(data[:, 2:])
data = scale(data)

plt.figure(figsize=(10, 2))
plt.plot(data)
plt.xlabel('Time')
plt.ylabel('Value')
plt.show()
```



[ ]:
```
x = torch.FloatTensor(data).reshape(1, *data.shape)
x_train = torch.FloatTensor(data[:400]).reshape(1, 400, data.shape[1])
x_val   = torch.FloatTensor(data[400:450]).reshape(1, 50, data.shape[1])
```

[ ]:
```
dkf = DKF(input_dim=4, z_dim=20, rnn_dim=20, trans_dim=20, emission_dim=20)
```

[ ]:
```
history = dkf.fit(x_train, x_val, num_epochs=200, annealing_factor=0.1)
```

```
Epoch= 1/200, loss= 4.100, mse= 3.770, kld= 3.291
        val_loss= 4.503, val_mse= 4.264, val_kld= 2.391
Epoch= 2/200, loss= 4.032, mse= 3.781, kld= 2.511
```

```
        val_loss= 4.293, val_mse= 4.108, val_kld= 1.853
Epoch= 3/200, loss= 3.908, mse= 3.705, kld= 2.032
        val_loss= 4.221, val_mse= 4.063, val_kld= 1.582
Epoch= 4/200, loss= 3.581, mse= 3.425, kld= 1.565
        val_loss= 3.934, val_mse= 3.805, val_kld= 1.297
Epoch= 5/200, loss= 3.822, mse= 3.691, kld= 1.311
        val_loss= 3.925, val_mse= 3.809, val_kld= 1.158
Epoch= 6/200, loss= 3.897, mse= 3.788, kld= 1.099
        val_loss= 3.146, val_mse= 3.051, val_kld= 0.949
Epoch= 7/200, loss= 3.544, mse= 3.450, kld= 0.937
        val_loss= 4.292, val_mse= 4.209, val_kld= 0.824
Epoch= 8/200, loss= 3.588, mse= 3.504, kld= 0.837
        val_loss= 3.916, val_mse= 3.839, val_kld= 0.778
Epoch= 9/200, loss= 3.547, mse= 3.474, kld= 0.731
        val_loss= 3.356, val_mse= 3.283, val_kld= 0.733
Epoch= 10/200, loss= 3.432, mse= 3.361, kld= 0.703
        val_loss= 4.087, val_mse= 4.001, val_kld= 0.866
Epoch= 11/200, loss= 3.460, mse= 3.384, kld= 0.756
        val_loss= 3.972, val_mse= 3.871, val_kld= 1.010
Epoch= 12/200, loss= 3.422, mse= 3.333, kld= 0.894
        val_loss= 2.940, val_mse= 2.821, val_kld= 1.192
Epoch= 13/200, loss= 3.517, mse= 3.414, kld= 1.028
        val_loss= 3.573, val_mse= 3.438, val_kld= 1.347
Epoch= 14/200, loss= 3.079, mse= 2.964, kld= 1.149
        val_loss= 3.509, val_mse= 3.394, val_kld= 1.149
Epoch= 15/200, loss= 3.240, mse= 3.146, kld= 0.933
        val_loss= 2.891, val_mse= 2.791, val_kld= 0.998
Epoch= 16/200, loss= 3.288, mse= 3.204, kld= 0.835
        val_loss= 3.034, val_mse= 2.950, val_kld= 0.838
Epoch= 17/200, loss= 2.972, mse= 2.892, kld= 0.799
        val_loss= 3.295, val_mse= 3.220, val_kld= 0.751
Epoch= 18/200, loss= 3.228, mse= 3.152, kld= 0.769
        val_loss= 3.612, val_mse= 3.525, val_kld= 0.875
Epoch= 19/200, loss= 3.120, mse= 3.046, kld= 0.740
        val_loss= 2.865, val_mse= 2.782, val_kld= 0.823
Epoch= 20/200, loss= 2.931, mse= 2.854, kld= 0.776
        val_loss= 3.495, val_mse= 3.414, val_kld= 0.808
Epoch= 21/200, loss= 3.068, mse= 2.995, kld= 0.730
        val_loss= 3.256, val_mse= 3.173, val_kld= 0.826
Epoch= 22/200, loss= 2.884, mse= 2.814, kld= 0.700
        val_loss= 3.054, val_mse= 2.971, val_kld= 0.825
Epoch= 23/200, loss= 2.922, mse= 2.853, kld= 0.692
        val_loss= 3.229, val_mse= 3.140, val_kld= 0.892
Epoch= 24/200, loss= 2.951, mse= 2.872, kld= 0.794
        val_loss= 2.639, val_mse= 2.537, val_kld= 1.024
Epoch= 25/200, loss= 3.019, mse= 2.938, kld= 0.818
        val_loss= 3.689, val_mse= 3.556, val_kld= 1.333
Epoch= 26/200, loss= 2.874, mse= 2.771, kld= 1.036
```

```
        val_loss= 3.162, val_mse= 3.033, val_kld= 1.286
Epoch= 27/200, loss= 2.737, mse= 2.623, kld= 1.137
        val_loss= 2.976, val_mse= 2.855, val_kld= 1.206
Epoch= 28/200, loss= 2.765, mse= 2.648, kld= 1.170
        val_loss= 3.058, val_mse= 2.942, val_kld= 1.157
Epoch= 29/200, loss= 2.655, mse= 2.532, kld= 1.231
        val_loss= 2.708, val_mse= 2.594, val_kld= 1.140
Epoch= 30/200, loss= 2.699, mse= 2.573, kld= 1.254
        val_loss= 2.432, val_mse= 2.306, val_kld= 1.262
Epoch= 31/200, loss= 2.577, mse= 2.452, kld= 1.247
        val_loss= 3.033, val_mse= 2.894, val_kld= 1.396
Epoch= 32/200, loss= 2.586, mse= 2.464, kld= 1.223
        val_loss= 3.093, val_mse= 2.935, val_kld= 1.585
Epoch= 33/200, loss= 2.715, mse= 2.592, kld= 1.231
        val_loss= 2.225, val_mse= 2.096, val_kld= 1.288
Epoch= 34/200, loss= 2.715, mse= 2.601, kld= 1.143
        val_loss= 2.667, val_mse= 2.559, val_kld= 1.085
Epoch= 35/200, loss= 2.608, mse= 2.501, kld= 1.071
        val_loss= 2.343, val_mse= 2.222, val_kld= 1.217
Epoch= 36/200, loss= 2.642, mse= 2.531, kld= 1.107
        val_loss= 2.665, val_mse= 2.557, val_kld= 1.075
Epoch= 37/200, loss= 2.339, mse= 2.227, kld= 1.123
        val_loss= 2.204, val_mse= 2.089, val_kld= 1.153
Epoch= 38/200, loss= 2.341, mse= 2.234, kld= 1.063
        val_loss= 2.448, val_mse= 2.330, val_kld= 1.180
Epoch= 39/200, loss= 2.479, mse= 2.367, kld= 1.125
        val_loss= 2.392, val_mse= 2.258, val_kld= 1.342
Epoch= 40/200, loss= 2.321, mse= 2.195, kld= 1.254
        val_loss= 2.353, val_mse= 2.219, val_kld= 1.345
Epoch= 41/200, loss= 2.305, mse= 2.175, kld= 1.303
        val_loss= 2.337, val_mse= 2.204, val_kld= 1.332
Epoch= 42/200, loss= 2.270, mse= 2.150, kld= 1.194
        val_loss= 2.323, val_mse= 2.198, val_kld= 1.248
Epoch= 43/200, loss= 2.286, mse= 2.160, kld= 1.261
        val_loss= 2.408, val_mse= 2.290, val_kld= 1.179
Epoch= 44/200, loss= 2.293, mse= 2.176, kld= 1.174
        val_loss= 2.105, val_mse= 1.995, val_kld= 1.095
Epoch= 45/200, loss= 2.201, mse= 2.089, kld= 1.124
        val_loss= 2.446, val_mse= 2.345, val_kld= 1.007
Epoch= 46/200, loss= 2.300, mse= 2.186, kld= 1.131
        val_loss= 2.319, val_mse= 2.198, val_kld= 1.210
Epoch= 47/200, loss= 2.147, mse= 2.030, kld= 1.175
        val_loss= 2.578, val_mse= 2.474, val_kld= 1.037
Epoch= 48/200, loss= 2.151, mse= 2.042, kld= 1.086
        val_loss= 2.463, val_mse= 2.347, val_kld= 1.161
Epoch= 49/200, loss= 2.305, mse= 2.194, kld= 1.110
        val_loss= 2.367, val_mse= 2.236, val_kld= 1.308
Epoch= 50/200, loss= 2.132, mse= 2.023, kld= 1.088
```

```
        val_loss= 2.368, val_mse= 2.254, val_kld= 1.143
Epoch= 51/200, loss= 2.076, mse= 1.979, kld= 0.971
        val_loss= 2.124, val_mse= 2.030, val_kld= 0.945
Epoch= 52/200, loss= 2.211, mse= 2.104, kld= 1.065
        val_loss= 2.176, val_mse= 2.055, val_kld= 1.208
Epoch= 53/200, loss= 2.146, mse= 2.040, kld= 1.068
        val_loss= 1.985, val_mse= 1.892, val_kld= 0.928
Epoch= 54/200, loss= 2.084, mse= 1.978, kld= 1.061
        val_loss= 2.344, val_mse= 2.232, val_kld= 1.114
Epoch= 55/200, loss= 2.106, mse= 1.997, kld= 1.085
        val_loss= 2.258, val_mse= 2.154, val_kld= 1.036
Epoch= 56/200, loss= 1.930, mse= 1.825, kld= 1.049
        val_loss= 2.293, val_mse= 2.157, val_kld= 1.363
Epoch= 57/200, loss= 2.082, mse= 1.966, kld= 1.155
        val_loss= 2.054, val_mse= 1.936, val_kld= 1.172
Epoch= 58/200, loss= 1.970, mse= 1.858, kld= 1.122
        val_loss= 2.241, val_mse= 2.128, val_kld= 1.133
Epoch= 59/200, loss= 2.111, mse= 2.003, kld= 1.079
        val_loss= 2.359, val_mse= 2.247, val_kld= 1.123
Epoch= 60/200, loss= 1.958, mse= 1.858, kld= 1.006
        val_loss= 2.343, val_mse= 2.233, val_kld= 1.103
Epoch= 61/200, loss= 2.066, mse= 1.962, kld= 1.039
        val_loss= 2.155, val_mse= 2.051, val_kld= 1.041
Epoch= 62/200, loss= 1.919, mse= 1.812, kld= 1.073
        val_loss= 2.061, val_mse= 1.958, val_kld= 1.036
Epoch= 63/200, loss= 1.965, mse= 1.859, kld= 1.060
        val_loss= 1.822, val_mse= 1.724, val_kld= 0.985
Epoch= 64/200, loss= 2.015, mse= 1.914, kld= 1.018
        val_loss= 1.735, val_mse= 1.634, val_kld= 1.006
Epoch= 65/200, loss= 1.952, mse= 1.854, kld= 0.981
        val_loss= 1.760, val_mse= 1.672, val_kld= 0.873
Epoch= 66/200, loss= 1.834, mse= 1.733, kld= 1.017
        val_loss= 1.679, val_mse= 1.600, val_kld= 0.789
Epoch= 67/200, loss= 1.907, mse= 1.810, kld= 0.968
        val_loss= 1.941, val_mse= 1.850, val_kld= 0.914
Epoch= 68/200, loss= 1.927, mse= 1.819, kld= 1.077
        val_loss= 1.972, val_mse= 1.881, val_kld= 0.904
Epoch= 69/200, loss= 2.006, mse= 1.906, kld= 0.994
        val_loss= 1.989, val_mse= 1.880, val_kld= 1.081
Epoch= 70/200, loss= 1.823, mse= 1.726, kld= 0.971
        val_loss= 1.648, val_mse= 1.564, val_kld= 0.842
Epoch= 71/200, loss= 1.859, mse= 1.755, kld= 1.034
        val_loss= 1.750, val_mse= 1.667, val_kld= 0.828
Epoch= 72/200, loss= 1.816, mse= 1.715, kld= 1.010
        val_loss= 1.831, val_mse= 1.742, val_kld= 0.884
Epoch= 73/200, loss= 1.891, mse= 1.785, kld= 1.067
        val_loss= 1.830, val_mse= 1.744, val_kld= 0.858
Epoch= 74/200, loss= 1.895, mse= 1.793, kld= 1.016
```

```
        val_loss= 1.743, val_mse= 1.664, val_kld= 0.794
Epoch= 75/200, loss= 1.869, mse= 1.768, kld= 1.013
        val_loss= 1.821, val_mse= 1.741, val_kld= 0.800
Epoch= 76/200, loss= 1.868, mse= 1.762, kld= 1.055
        val_loss= 1.954, val_mse= 1.874, val_kld= 0.798
Epoch= 77/200, loss= 1.891, mse= 1.794, kld= 0.972
        val_loss= 2.033, val_mse= 1.959, val_kld= 0.738
Epoch= 78/200, loss= 1.830, mse= 1.731, kld= 0.989
        val_loss= 2.046, val_mse= 1.952, val_kld= 0.943
Epoch= 79/200, loss= 1.788, mse= 1.686, kld= 1.018
        val_loss= 1.708, val_mse= 1.621, val_kld= 0.870
Epoch= 80/200, loss= 1.794, mse= 1.691, kld= 1.031
        val_loss= 2.010, val_mse= 1.913, val_kld= 0.966
Epoch= 81/200, loss= 1.888, mse= 1.784, kld= 1.035
        val_loss= 1.653, val_mse= 1.557, val_kld= 0.965
Epoch= 82/200, loss= 1.798, mse= 1.699, kld= 0.991
        val_loss= 1.490, val_mse= 1.391, val_kld= 0.990
Epoch= 83/200, loss= 1.912, mse= 1.808, kld= 1.039
        val_loss= 1.757, val_mse= 1.677, val_kld= 0.793
Epoch= 84/200, loss= 1.797, mse= 1.697, kld= 0.999
        val_loss= 1.705, val_mse= 1.627, val_kld= 0.775
Epoch= 85/200, loss= 1.771, mse= 1.675, kld= 0.957
        val_loss= 1.530, val_mse= 1.444, val_kld= 0.860
Epoch= 86/200, loss= 1.691, mse= 1.589, kld= 1.014
        val_loss= 1.523, val_mse= 1.429, val_kld= 0.938
Epoch= 87/200, loss= 1.786, mse= 1.694, kld= 0.917
        val_loss= 1.834, val_mse= 1.737, val_kld= 0.971
Epoch= 88/200, loss= 1.695, mse= 1.598, kld= 0.972
        val_loss= 1.717, val_mse= 1.617, val_kld= 0.999
Epoch= 89/200, loss= 1.683, mse= 1.584, kld= 0.996
        val_loss= 1.619, val_mse= 1.524, val_kld= 0.951
Epoch= 90/200, loss= 1.694, mse= 1.596, kld= 0.983
        val_loss= 1.622, val_mse= 1.544, val_kld= 0.776
Epoch= 91/200, loss= 1.686, mse= 1.588, kld= 0.975
        val_loss= 1.566, val_mse= 1.467, val_kld= 0.987
Epoch= 92/200, loss= 1.621, mse= 1.529, kld= 0.919
        val_loss= 1.786, val_mse= 1.712, val_kld= 0.739
Epoch= 93/200, loss= 1.636, mse= 1.537, kld= 0.992
        val_loss= 1.752, val_mse= 1.665, val_kld= 0.861
Epoch= 94/200, loss= 1.655, mse= 1.557, kld= 0.988
        val_loss= 1.669, val_mse= 1.587, val_kld= 0.818
Epoch= 95/200, loss= 1.653, mse= 1.550, kld= 1.029
        val_loss= 1.528, val_mse= 1.445, val_kld= 0.829
Epoch= 96/200, loss= 1.562, mse= 1.465, kld= 0.973
        val_loss= 1.693, val_mse= 1.602, val_kld= 0.907
Epoch= 97/200, loss= 1.697, mse= 1.607, kld= 0.900
        val_loss= 1.844, val_mse= 1.747, val_kld= 0.968
Epoch= 98/200, loss= 1.565, mse= 1.474, kld= 0.910
```

```
        val_loss= 1.402, val_mse= 1.314, val_kld= 0.886
Epoch= 99/200, loss= 1.590, mse= 1.493, kld= 0.968
        val_loss= 1.572, val_mse= 1.474, val_kld= 0.983
Epoch= 100/200, loss= 1.592, mse= 1.495, kld= 0.965
        val_loss= 1.370, val_mse= 1.270, val_kld= 0.997
Epoch= 101/200, loss= 1.531, mse= 1.434, kld= 0.970
        val_loss= 1.593, val_mse= 1.483, val_kld= 1.104
Epoch= 102/200, loss= 1.597, mse= 1.504, kld= 0.930
        val_loss= 1.496, val_mse= 1.416, val_kld= 0.807
Epoch= 103/200, loss= 1.578, mse= 1.487, kld= 0.906
        val_loss= 1.598, val_mse= 1.508, val_kld= 0.907
Epoch= 104/200, loss= 1.550, mse= 1.443, kld= 1.067
        val_loss= 1.599, val_mse= 1.539, val_kld= 0.602
Epoch= 105/200, loss= 1.595, mse= 1.494, kld= 1.012
        val_loss= 1.681, val_mse= 1.600, val_kld= 0.810
Epoch= 106/200, loss= 1.553, mse= 1.460, kld= 0.929
        val_loss= 1.382, val_mse= 1.293, val_kld= 0.888
Epoch= 107/200, loss= 1.407, mse= 1.311, kld= 0.957
        val_loss= 1.476, val_mse= 1.400, val_kld= 0.763
Epoch= 108/200, loss= 1.495, mse= 1.412, kld= 0.837
        val_loss= 1.262, val_mse= 1.192, val_kld= 0.695
Epoch= 109/200, loss= 1.478, mse= 1.387, kld= 0.912
        val_loss= 1.587, val_mse= 1.509, val_kld= 0.788
Epoch= 110/200, loss= 1.518, mse= 1.433, kld= 0.848
        val_loss= 1.328, val_mse= 1.253, val_kld= 0.750
Epoch= 111/200, loss= 1.469, mse= 1.374, kld= 0.946
        val_loss= 1.461, val_mse= 1.386, val_kld= 0.755
Epoch= 112/200, loss= 1.405, mse= 1.315, kld= 0.900
        val_loss= 1.614, val_mse= 1.515, val_kld= 0.993
Epoch= 113/200, loss= 1.445, mse= 1.360, kld= 0.853
        val_loss= 1.582, val_mse= 1.508, val_kld= 0.743
Epoch= 114/200, loss= 1.454, mse= 1.366, kld= 0.876
        val_loss= 1.415, val_mse= 1.350, val_kld= 0.653
Epoch= 115/200, loss= 1.506, mse= 1.416, kld= 0.896
        val_loss= 1.304, val_mse= 1.229, val_kld= 0.748
Epoch= 116/200, loss= 1.405, mse= 1.316, kld= 0.892
        val_loss= 1.309, val_mse= 1.237, val_kld= 0.719
Epoch= 117/200, loss= 1.426, mse= 1.330, kld= 0.954
        val_loss= 1.343, val_mse= 1.260, val_kld= 0.833
Epoch= 118/200, loss= 1.484, mse= 1.398, kld= 0.867
        val_loss= 1.479, val_mse= 1.395, val_kld= 0.843
Epoch= 119/200, loss= 1.462, mse= 1.371, kld= 0.910
        val_loss= 1.549, val_mse= 1.465, val_kld= 0.848
Epoch= 120/200, loss= 1.475, mse= 1.387, kld= 0.889
        val_loss= 1.394, val_mse= 1.306, val_kld= 0.878
Epoch= 121/200, loss= 1.361, mse= 1.277, kld= 0.837
        val_loss= 1.378, val_mse= 1.310, val_kld= 0.679
Epoch= 122/200, loss= 1.354, mse= 1.263, kld= 0.910
```

```
        val_loss= 1.386, val_mse= 1.308, val_kld= 0.788
Epoch= 123/200, loss= 1.520, mse= 1.425, kld= 0.947
        val_loss= 1.489, val_mse= 1.415, val_kld= 0.742
Epoch= 124/200, loss= 1.357, mse= 1.268, kld= 0.890
        val_loss= 1.186, val_mse= 1.098, val_kld= 0.876
Epoch= 125/200, loss= 1.376, mse= 1.288, kld= 0.881
        val_loss= 1.433, val_mse= 1.355, val_kld= 0.782
Epoch= 126/200, loss= 1.391, mse= 1.300, kld= 0.910
        val_loss= 1.374, val_mse= 1.302, val_kld= 0.718
Epoch= 127/200, loss= 1.378, mse= 1.294, kld= 0.845
        val_loss= 1.457, val_mse= 1.387, val_kld= 0.696
Epoch= 128/200, loss= 1.495, mse= 1.404, kld= 0.912
        val_loss= 1.589, val_mse= 1.493, val_kld= 0.958
Epoch= 129/200, loss= 1.376, mse= 1.282, kld= 0.933
        val_loss= 1.389, val_mse= 1.308, val_kld= 0.808
Epoch= 130/200, loss= 1.287, mse= 1.200, kld= 0.868
        val_loss= 1.240, val_mse= 1.168, val_kld= 0.720
Epoch= 131/200, loss= 1.381, mse= 1.293, kld= 0.876
        val_loss= 1.861, val_mse= 1.775, val_kld= 0.864
Epoch= 132/200, loss= 1.384, mse= 1.294, kld= 0.908
        val_loss= 1.404, val_mse= 1.328, val_kld= 0.758
Epoch= 133/200, loss= 1.304, mse= 1.222, kld= 0.822
        val_loss= 1.299, val_mse= 1.228, val_kld= 0.711
Epoch= 134/200, loss= 1.301, mse= 1.213, kld= 0.877
        val_loss= 1.369, val_mse= 1.297, val_kld= 0.725
Epoch= 135/200, loss= 1.318, mse= 1.237, kld= 0.812
        val_loss= 1.325, val_mse= 1.249, val_kld= 0.758
Epoch= 136/200, loss= 1.355, mse= 1.276, kld= 0.784
        val_loss= 1.472, val_mse= 1.388, val_kld= 0.839
Epoch= 137/200, loss= 1.285, mse= 1.206, kld= 0.795
        val_loss= 1.476, val_mse= 1.387, val_kld= 0.885
Epoch= 138/200, loss= 1.370, mse= 1.283, kld= 0.869
        val_loss= 1.288, val_mse= 1.218, val_kld= 0.698
Epoch= 139/200, loss= 1.360, mse= 1.272, kld= 0.884
        val_loss= 1.273, val_mse= 1.194, val_kld= 0.794
Epoch= 140/200, loss= 1.356, mse= 1.266, kld= 0.899
        val_loss= 1.324, val_mse= 1.247, val_kld= 0.774
Epoch= 141/200, loss= 1.272, mse= 1.184, kld= 0.873
        val_loss= 1.256, val_mse= 1.173, val_kld= 0.831
Epoch= 142/200, loss= 1.246, mse= 1.159, kld= 0.866
        val_loss= 1.335, val_mse= 1.260, val_kld= 0.746
Epoch= 143/200, loss= 1.226, mse= 1.140, kld= 0.865
        val_loss= 1.328, val_mse= 1.250, val_kld= 0.782
Epoch= 144/200, loss= 1.307, mse= 1.225, kld= 0.819
        val_loss= 1.382, val_mse= 1.307, val_kld= 0.755
Epoch= 145/200, loss= 1.259, mse= 1.171, kld= 0.874
        val_loss= 1.151, val_mse= 1.071, val_kld= 0.800
Epoch= 146/200, loss= 1.282, mse= 1.197, kld= 0.848
```

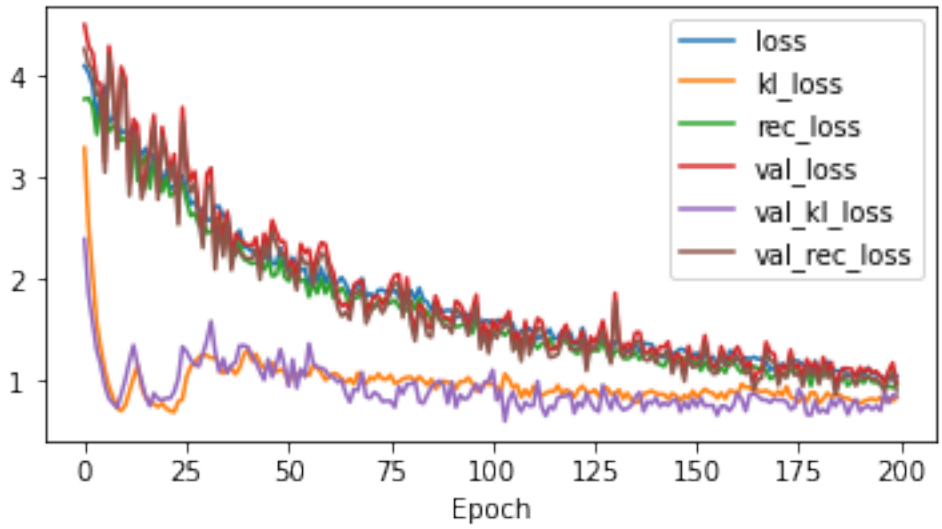
```
        val_loss= 1.270, val_mse= 1.188, val_kld= 0.821
Epoch= 147/200, loss= 1.313, mse= 1.233, kld= 0.799
        val_loss= 1.233, val_mse= 1.163, val_kld= 0.694
Epoch= 148/200, loss= 1.300, mse= 1.215, kld= 0.858
        val_loss= 1.073, val_mse= 0.994, val_kld= 0.792
Epoch= 149/200, loss= 1.258, mse= 1.176, kld= 0.824
        val_loss= 1.340, val_mse= 1.267, val_kld= 0.725
Epoch= 150/200, loss= 1.242, mse= 1.152, kld= 0.899
        val_loss= 1.211, val_mse= 1.129, val_kld= 0.820
Epoch= 151/200, loss= 1.274, mse= 1.190, kld= 0.837
        val_loss= 1.220, val_mse= 1.146, val_kld= 0.744
Epoch= 152/200, loss= 1.186, mse= 1.098, kld= 0.878
        val_loss= 1.242, val_mse= 1.160, val_kld= 0.829
Epoch= 153/200, loss= 1.279, mse= 1.188, kld= 0.916
        val_loss= 1.444, val_mse= 1.369, val_kld= 0.749
Epoch= 154/200, loss= 1.198, mse= 1.109, kld= 0.887
        val_loss= 1.236, val_mse= 1.160, val_kld= 0.755
Epoch= 155/200, loss= 1.236, mse= 1.149, kld= 0.876
        val_loss= 1.142, val_mse= 1.067, val_kld= 0.756
Epoch= 156/200, loss= 1.169, mse= 1.087, kld= 0.827
        val_loss= 1.139, val_mse= 1.073, val_kld= 0.661
Epoch= 157/200, loss= 1.215, mse= 1.127, kld= 0.880
        val_loss= 1.336, val_mse= 1.261, val_kld= 0.750
Epoch= 158/200, loss= 1.158, mse= 1.076, kld= 0.822
        val_loss= 1.228, val_mse= 1.153, val_kld= 0.750
Epoch= 159/200, loss= 1.291, mse= 1.201, kld= 0.900
        val_loss= 1.215, val_mse= 1.145, val_kld= 0.703
Epoch= 160/200, loss= 1.284, mse= 1.195, kld= 0.896
        val_loss= 1.258, val_mse= 1.182, val_kld= 0.756
Epoch= 161/200, loss= 1.208, mse= 1.122, kld= 0.861
        val_loss= 1.333, val_mse= 1.246, val_kld= 0.876
Epoch= 162/200, loss= 1.173, mse= 1.077, kld= 0.967
        val_loss= 1.247, val_mse= 1.165, val_kld= 0.819
Epoch= 163/200, loss= 1.228, mse= 1.134, kld= 0.934
        val_loss= 1.136, val_mse= 1.066, val_kld= 0.701
Epoch= 164/200, loss= 1.211, mse= 1.117, kld= 0.934
        val_loss= 1.185, val_mse= 1.099, val_kld= 0.863
Epoch= 165/200, loss= 1.233, mse= 1.143, kld= 0.898
        val_loss= 0.995, val_mse= 0.907, val_kld= 0.871
Epoch= 166/200, loss= 1.211, mse= 1.120, kld= 0.904
        val_loss= 1.090, val_mse= 1.005, val_kld= 0.847
Epoch= 167/200, loss= 1.215, mse= 1.126, kld= 0.890
        val_loss= 1.109, val_mse= 1.028, val_kld= 0.809
Epoch= 168/200, loss= 1.189, mse= 1.099, kld= 0.904
        val_loss= 1.379, val_mse= 1.298, val_kld= 0.810
Epoch= 169/200, loss= 1.104, mse= 1.016, kld= 0.881
        val_loss= 1.279, val_mse= 1.200, val_kld= 0.785
Epoch= 170/200, loss= 1.131, mse= 1.046, kld= 0.853
```

```
        val_loss= 1.264, val_mse= 1.182, val_kld= 0.821
Epoch= 171/200, loss= 1.142, mse= 1.053, kld= 0.897
        val_loss= 1.036, val_mse= 0.955, val_kld= 0.806
Epoch= 172/200, loss= 1.167, mse= 1.081, kld= 0.856
        val_loss= 1.154, val_mse= 1.073, val_kld= 0.809
Epoch= 173/200, loss= 1.136, mse= 1.042, kld= 0.942
        val_loss= 1.108, val_mse= 1.028, val_kld= 0.792
Epoch= 174/200, loss= 1.182, mse= 1.098, kld= 0.838
        val_loss= 1.302, val_mse= 1.230, val_kld= 0.718
Epoch= 175/200, loss= 1.197, mse= 1.115, kld= 0.826
        val_loss= 1.124, val_mse= 1.050, val_kld= 0.737
Epoch= 176/200, loss= 1.190, mse= 1.111, kld= 0.790
        val_loss= 1.165, val_mse= 1.073, val_kld= 0.917
Epoch= 177/200, loss= 1.159, mse= 1.074, kld= 0.847
        val_loss= 1.027, val_mse= 0.943, val_kld= 0.841
Epoch= 178/200, loss= 1.134, mse= 1.054, kld= 0.794
        val_loss= 1.210, val_mse= 1.144, val_kld= 0.667
Epoch= 179/200, loss= 1.036, mse= 0.948, kld= 0.884
        val_loss= 1.122, val_mse= 1.056, val_kld= 0.661
Epoch= 180/200, loss= 1.193, mse= 1.105, kld= 0.880
        val_loss= 1.182, val_mse= 1.102, val_kld= 0.800
Epoch= 181/200, loss= 1.093, mse= 1.015, kld= 0.777
        val_loss= 1.033, val_mse= 0.964, val_kld= 0.686
Epoch= 182/200, loss= 1.133, mse= 1.051, kld= 0.819
        val_loss= 1.230, val_mse= 1.158, val_kld= 0.716
Epoch= 183/200, loss= 1.188, mse= 1.100, kld= 0.879
        val_loss= 1.095, val_mse= 1.027, val_kld= 0.683
Epoch= 184/200, loss= 1.125, mse= 1.041, kld= 0.840
        val_loss= 1.181, val_mse= 1.100, val_kld= 0.811
Epoch= 185/200, loss= 1.099, mse= 1.013, kld= 0.861
        val_loss= 0.931, val_mse= 0.857, val_kld= 0.747
Epoch= 186/200, loss= 1.088, mse= 1.003, kld= 0.858
        val_loss= 1.073, val_mse= 0.991, val_kld= 0.824
Epoch= 187/200, loss= 1.120, mse= 1.037, kld= 0.829
        val_loss= 1.122, val_mse= 1.044, val_kld= 0.781
Epoch= 188/200, loss= 1.031, mse= 0.949, kld= 0.826
        val_loss= 1.120, val_mse= 1.053, val_kld= 0.668
Epoch= 189/200, loss= 1.077, mse= 0.997, kld= 0.804
        val_loss= 1.114, val_mse= 1.037, val_kld= 0.761
Epoch= 190/200, loss= 1.053, mse= 0.973, kld= 0.798
        val_loss= 1.031, val_mse= 0.949, val_kld= 0.815
Epoch= 191/200, loss= 1.078, mse= 1.000, kld= 0.779
        val_loss= 1.132, val_mse= 1.063, val_kld= 0.694
Epoch= 192/200, loss= 1.057, mse= 0.978, kld= 0.797
        val_loss= 1.088, val_mse= 1.015, val_kld= 0.734
Epoch= 193/200, loss= 1.130, mse= 1.049, kld= 0.813
        val_loss= 1.121, val_mse= 1.046, val_kld= 0.753
Epoch= 194/200, loss= 1.067, mse= 0.985, kld= 0.823
```

```
        val_loss= 1.080, val_mse= 1.008, val_kld= 0.716
Epoch= 195/200, loss= 1.064, mse= 0.984, kld= 0.804
        val_loss= 1.092, val_mse= 1.018, val_kld= 0.749
Epoch= 196/200, loss= 1.046, mse= 0.963, kld= 0.837
        val_loss= 1.040, val_mse= 0.970, val_kld= 0.704
Epoch= 197/200, loss= 1.019, mse= 0.940, kld= 0.796
        val_loss= 0.980, val_mse= 0.892, val_kld= 0.887
Epoch= 198/200, loss= 1.027, mse= 0.944, kld= 0.831
        val_loss= 1.094, val_mse= 1.017, val_kld= 0.779
Epoch= 199/200, loss= 1.010, mse= 0.930, kld= 0.799
        val_loss= 1.174, val_mse= 1.089, val_kld= 0.856
Epoch= 200/200, loss= 1.043, mse= 0.960, kld= 0.832
        val_loss= 0.986, val_mse= 0.901, val_kld= 0.849
```

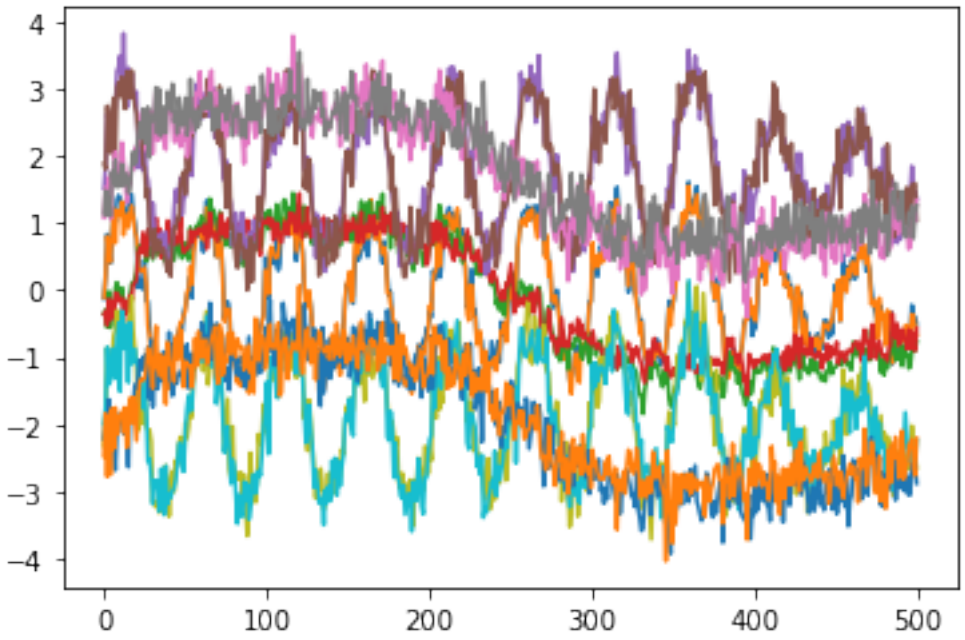
[ ]:
```
pd.DataFrame(history).plot(figsize=(6, 3), xlabel='Epoch')
```

[ ]:
```
<AxesSubplot:xlabel='Epoch'>
```

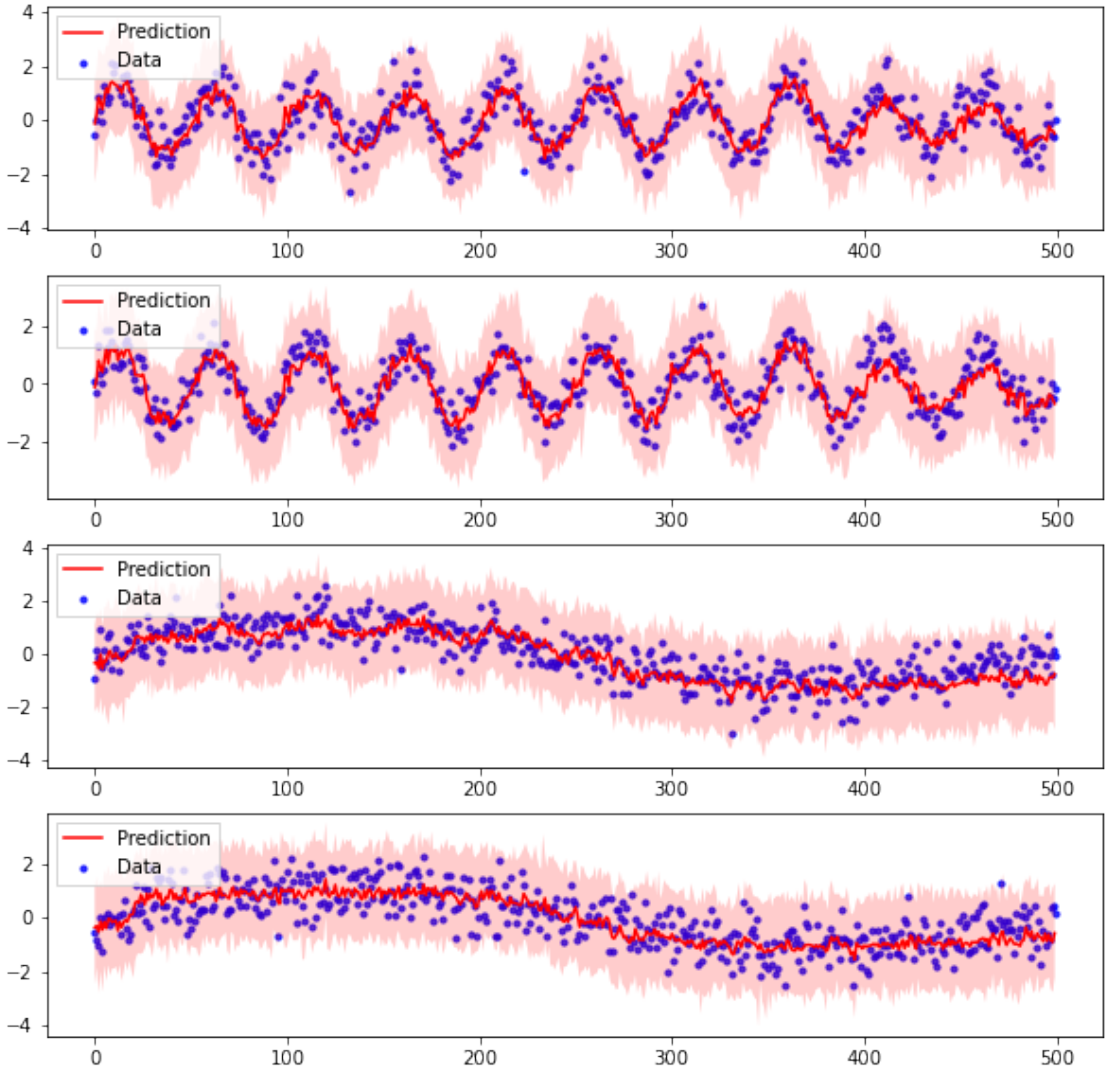


[ ]:
```
# x_hat = dkf.generate(x_train)
# x_hat, x_025, x_975 = dkf.filter(x_train)
x_hat, x_025, x_975 = dkf.predict(x, 100)
x_hat = x_hat.detach().numpy()[0]
x_025 = x_025.detach().numpy()[0]
x_975 = x_975.detach().numpy()[0]
plt.plot(x_hat)
plt.plot(x_975)
plt.plot(x_025)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f8a8198d130>,
 <matplotlib.lines.Line2D at 0x7f8a8198d0d0>,
 <matplotlib.lines.Line2D at 0x7f8a81e70790>,
 <matplotlib.lines.Line2D at 0x7f8a8180baf0>]
```



[ ]:
```
fig, ax = plt.subplots(4, figsize=(10, 10))

for i, axi in enumerate(ax):
    axi.scatter(
        np.arange(data.shape[0]),
        data[:, i], s=10, alpha=0.8, label='Data', c='b')
    axi.plot(x_hat[:, i], label='Prediction', c='r')
    axi.fill_between(np.arange(x_hat.shape[0]), x_025[:, i], x_975[:, i],
                     facecolor='r', alpha=0.2)
    axi.legend(loc='upper left', fancybox=False)

plt.show()
```


Prediction
Data
4
2
0
−2
−4
0
100
200
300
400
500

[ ]: