Bayesian Neural Networks for Time Series Forecasting

Billy Hernawan

STAT 618 Final Project

*Chandra & He. Bayesian neural networks for stock price forecasting before and during COVID-19 pandemic. *PLoS One* **2021**
Chandra et al. Langevin-gradient parallel tempering for Bayesian neural learning. *Neurocomputing* **2019**
Jospin et al. Hands-on Bayesian neural networks-a tutorial for deep learning users. *IEEE Comp. Intell. Mag.* **2022**
***: Primary paper**

State-space reconstruction of time series

Objective is a multi-step prediction $y \in \mathbb{R}^m$ based on previous input of $x \in \mathbb{R}^m$

Given

N: length of time series

m: embedding dimension

T: time lag

For the very first instance

$$\bar{\boldsymbol{x}}_1 = [x_1, x_2, x_3, \dots, x_m]$$

$$\boldsymbol{y_1} = [x_{m+1}, x_{m+2}, x_{m+3}, \dots, x_{m+n}]$$

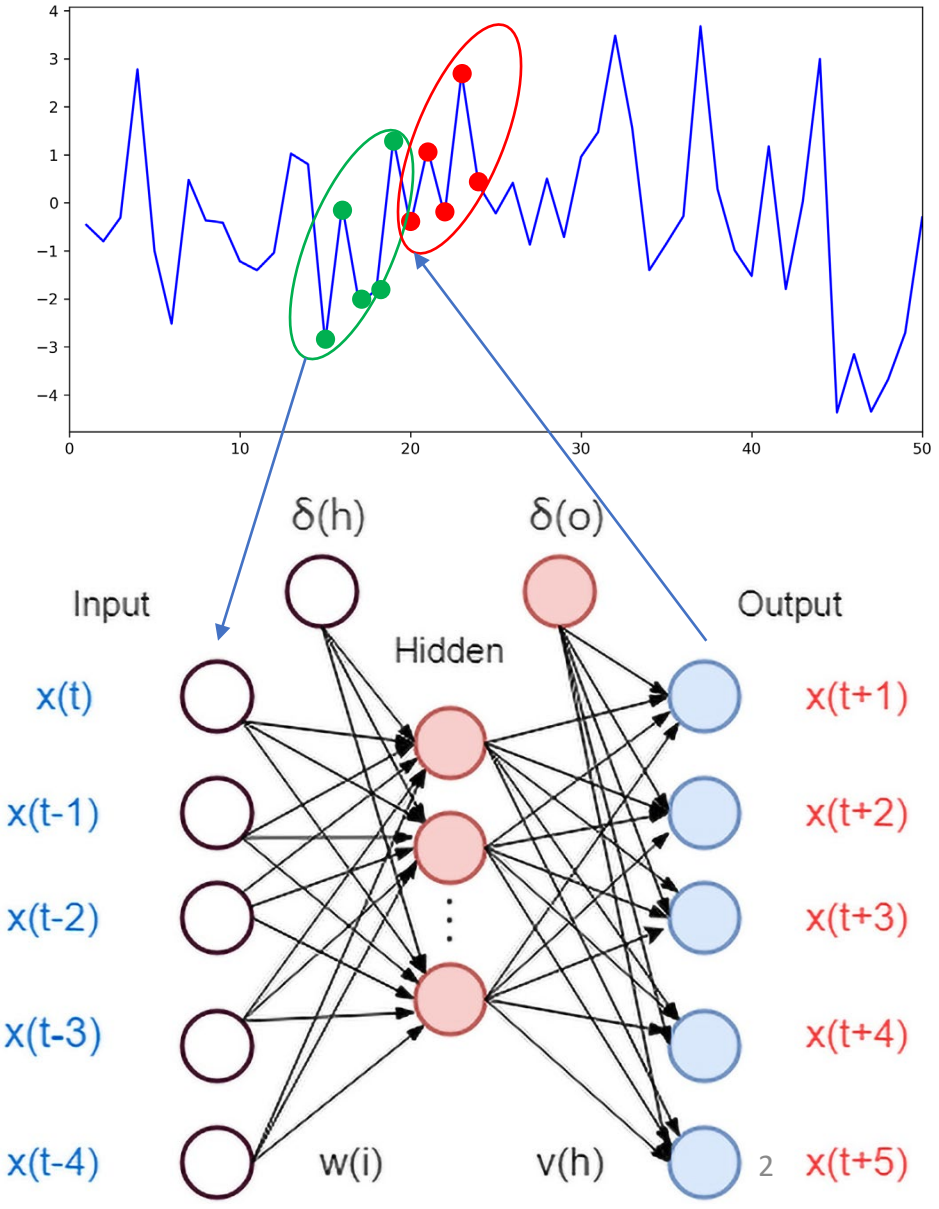
For all of the instance

$$\bar{\boldsymbol{x}}_t = [x_{1+(t-1)T}, x_{2+(t-1)T}, x_{3+(t-1)T}, \dots, x_{m+(t-1)T}]$$

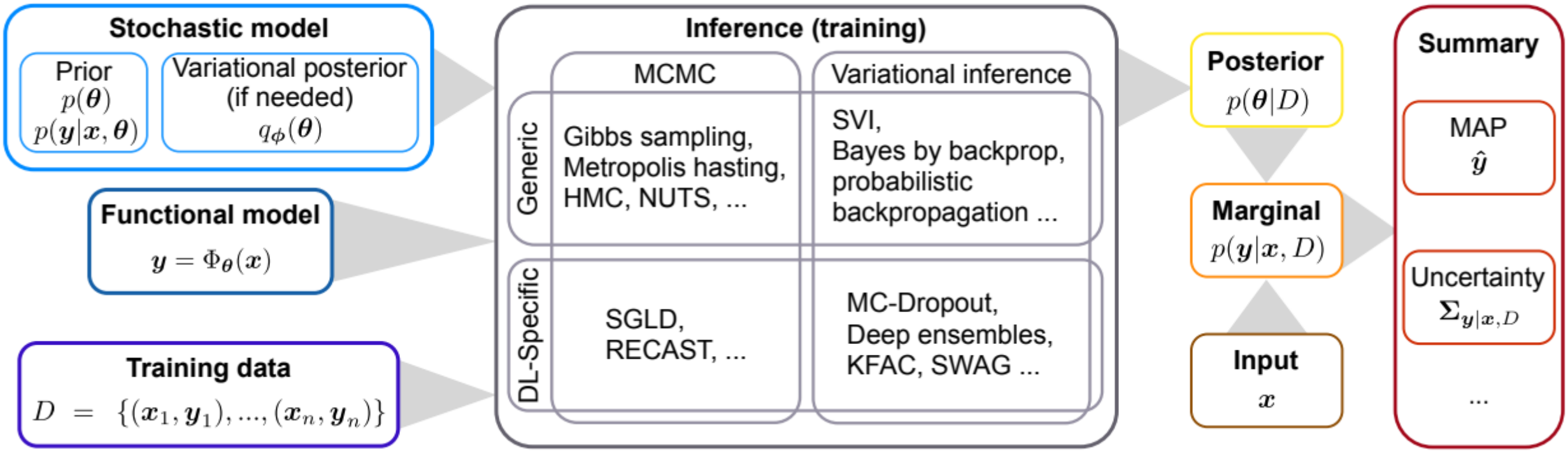
$$\boldsymbol{y_t} = [x_{m+1+(t-1)T}, x_{m+2+(t-1)T}, x_{m+3+(t-1)T}, \dots, x_{m+n+(t-1)T}]$$

Embedding dimension controls the input layer size

Output layer size depends on the number of desired prediction head of time



Bayesian Neural Network-1 (Overview)



Bayesian Neural Network-2 (Framework)

$$f(\bar{\boldsymbol{x}}_t) = g\left(\delta_o + \sum_{h=1}^{H} v_h * g\left(\delta_h + \sum_{h=1}^{H} w_{ih}\,\bar{x}_{t,i}\right)\right)$$

Adaptive Moment Estimation (Adam):

a_{k-1}: learning rate
β_1: first moment estimates
β_2: second moment estimates

$$w_k = w_{k-1} - a_{k-1}\frac{\sqrt{(1-\beta_2^k)}}{\sqrt{(1-\beta_1^k)}}\frac{\beta_1 u_{k-2} + (1-\beta_1)\nabla f(w_{k-1})}{\sqrt{\beta_2 n_{k-2} + (1-\beta_2)\nabla f(w_{k-1})^2} + \epsilon}$$

$$\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\delta})$$

Likelihood function

$$p(\boldsymbol{y_s}|\boldsymbol{\theta}) = -\frac{1}{(2\pi\tau^2)^{\frac{S}{2}}}\exp\left(-\frac{1}{2\tau^2}\sum_{t\in S}\left(\boldsymbol{y_t} - f(\bar{\boldsymbol{x}}_t)\right)^2\right)$$

τ: noise as parameter

S: training instances

Prior function

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{-\frac{1}{2\sigma^2}\left(\sum_{i=1}^{M}\theta\right)\right\} \times \tau^{2(1+v_1)}\exp\left(\frac{-v_2}{\tau^2}\right)$$

M: # network parameter
v_1: User-defined
v_2: User-defined

Bayesian Neural Network-3 (Parallel Tempering)

Parallel tempering MCMC: Parallel replica samplers to sample multi-modal posterior distributions

Each replica is assigned a temperature values where higher temperature have higher probability to accept weaker proposals

$$\boldsymbol{\Theta} = (\theta^1, \dots, \theta^R)$$

$$T \in [0,1]$$

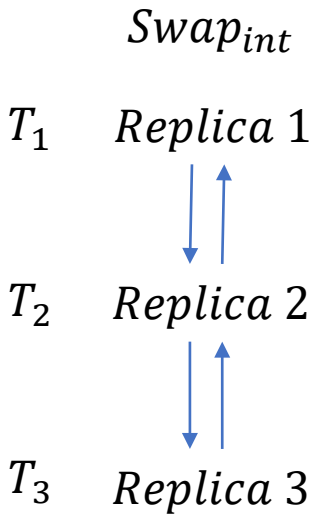
$$p(\theta|\boldsymbol{D})^T$$

$T = 0$: Uniform stationary dist.
$T = 1$: Posterior

Smaller temperature level-> global exploration
Larger temperature level-> local exploration

Replica implementation has been dropped due to time constraint

Parallel execution



Replica swap determined by Metropolis-Hasting acceptance criterion

Replica posterior distributions are combined after convergence

Bayesian Neural Network-4 (Langevin-Gradient MC)

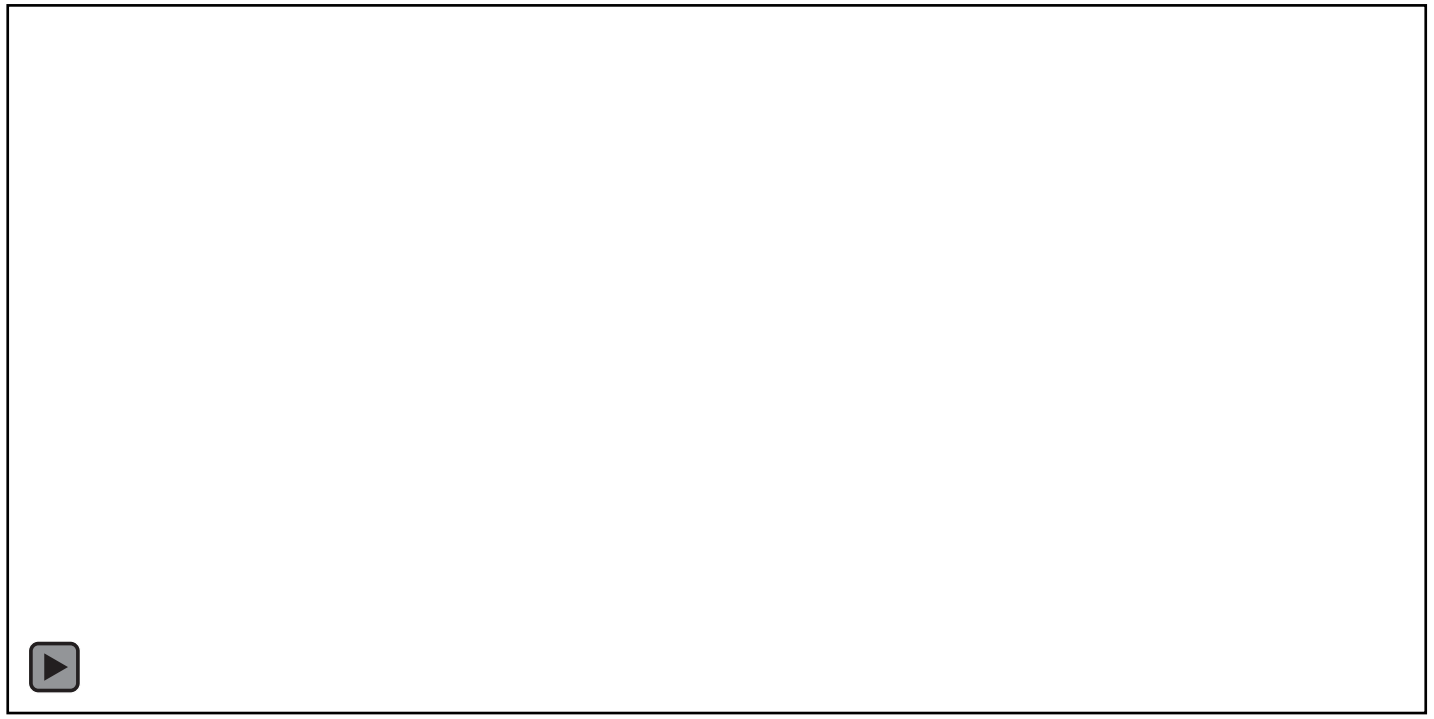
Differentiable Posterior

$$\boldsymbol{dX_t} = \nabla \log \pi(x)\, dt + \sqrt{2} dB_t$$

Drift Term

Standard Brownian Motion

SGD w/ noise



Implementation

$$\theta_r^p \sim q(\theta|\theta_r^k)$$

where

$$\boldsymbol{\theta}^p \sim \mathcal{N}(\bar{\boldsymbol{\theta}}^k, \Sigma_\theta)$$

$$\bar{\boldsymbol{\theta}}^k = \boldsymbol{\theta}^k + r\nabla E(\boldsymbol{\theta}^k)$$

$$E = \sum_{i \in S} \left(y_i - f(\bar{x}_i)\right)^2$$

$$\nabla E\left(\boldsymbol{\theta}^k\right) = \left(\frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_L}\right)$$

One-step gradient over Gaussian noise

Data Exploration-1

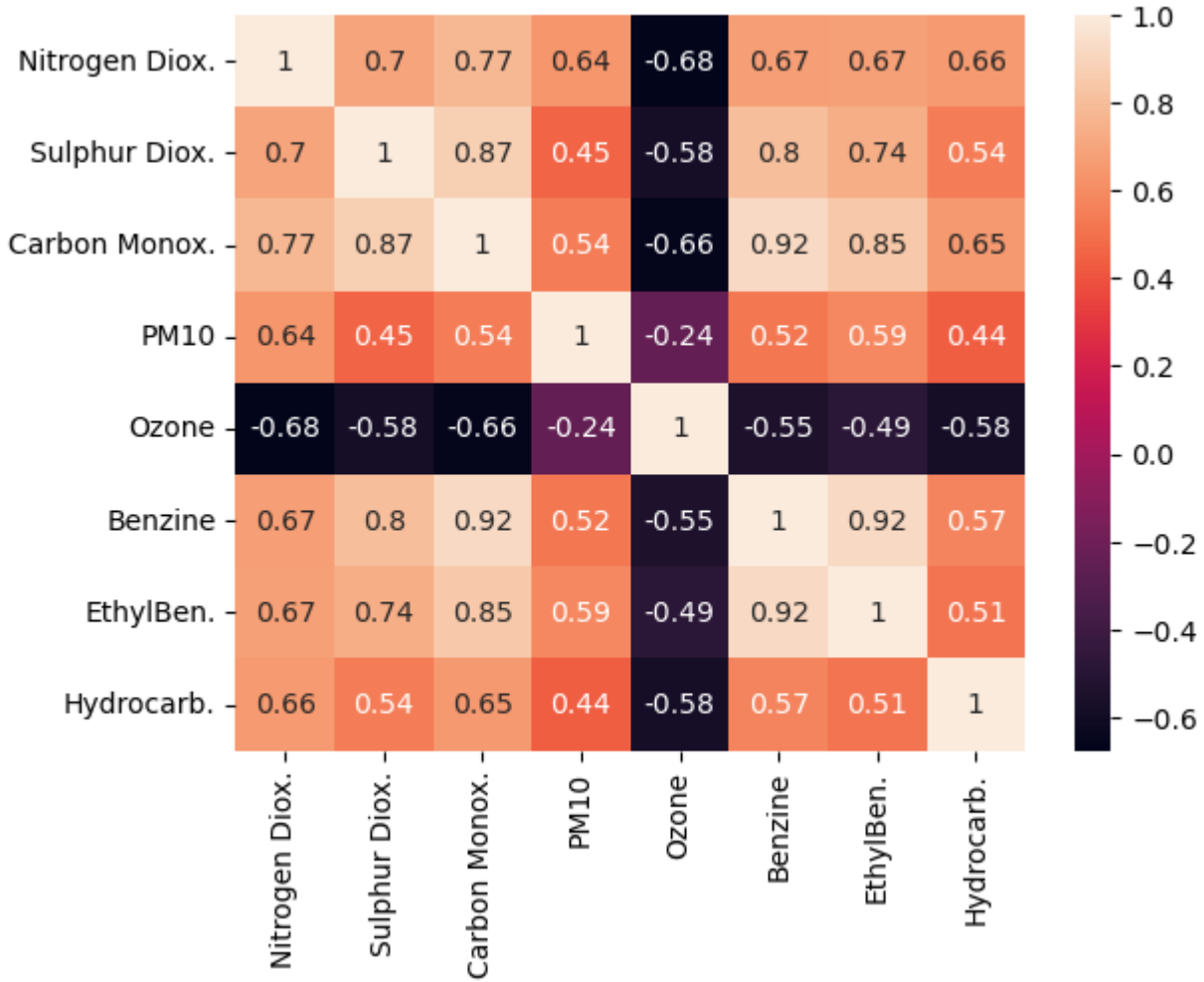
Air Quality in Madrid (2001-2018) Dataset from Kaggle

- 217871 rows
- $SO_2, CO, NO, NO_2, PM25, PM10, NO_x, O_3$, and etc
- Readings given per station at different time points (minutes)
- Data is aggregated to yield daily reading of all of the station.

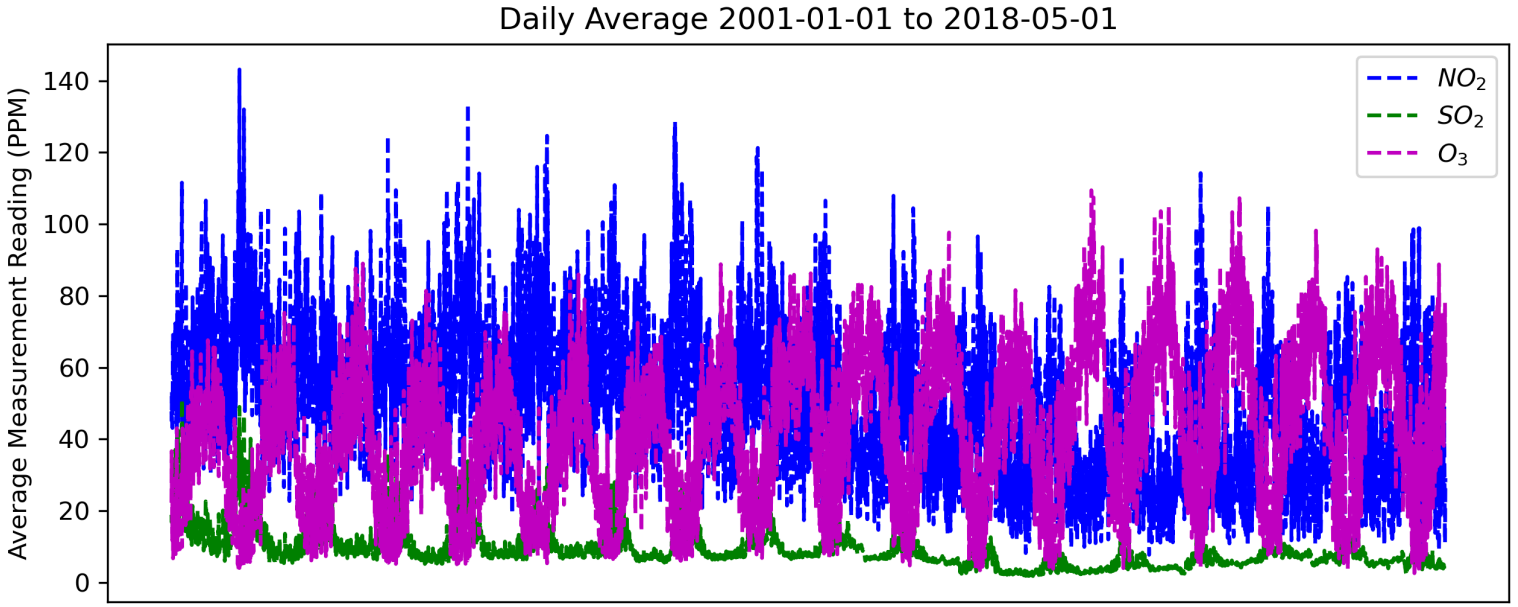
- Training dataset: 2001-01-01 to 2014-08-27

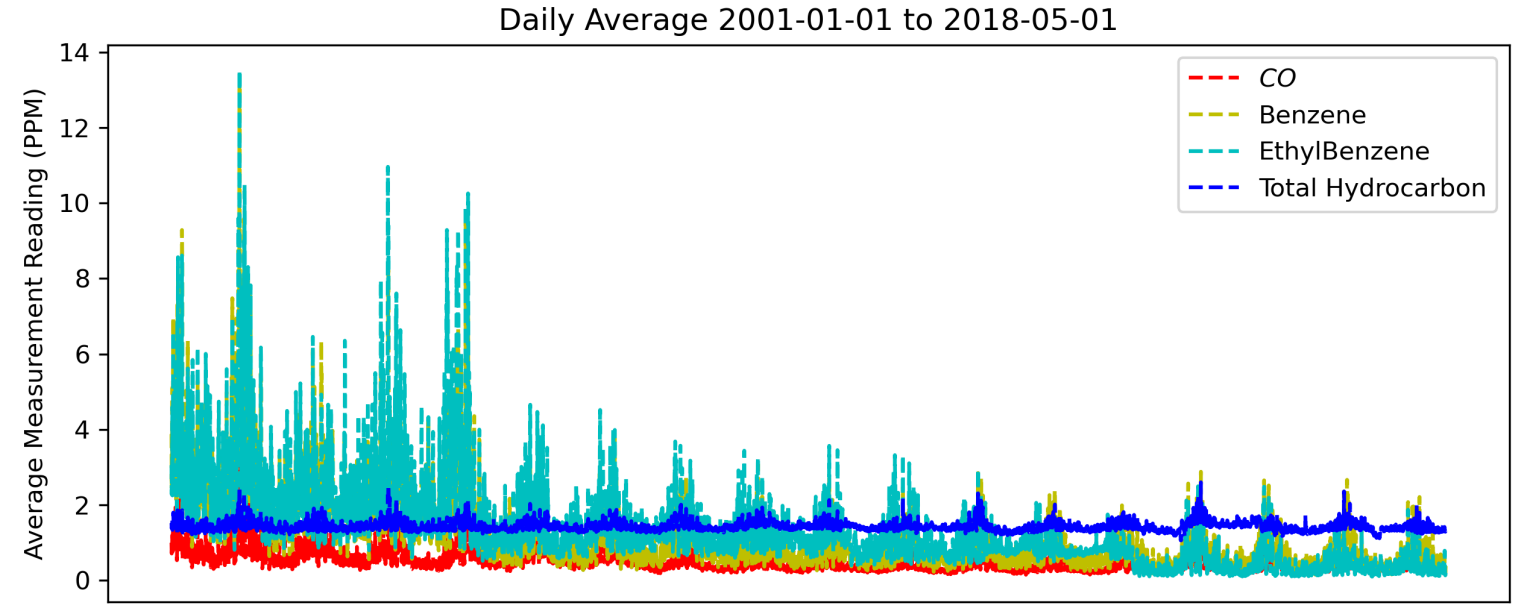
- Test dataset: 2014-08-28 to 2017-05-20

- Validation dataset: 2017-05-21 to 2018-05-01



Data Exploration-2





Objective & Implementation-1

Objective:

1. Investigate if a single sampler can still capture multi-modal distribution of the model parameters
2. Investigate if a single sampler affects prediction quality adversely

Implementation

- Authors original implementation is in Python with parallel computing environment. In this study, I implemented a stripped-down version of their algorithm in Julia (removing the replica sampler)
- Train wall clock time: 4.5 hours
- Samples in simulation: 300,000

Network topology:
Input layer: 5
Hidden layer: 10
Output layer: 5

Data:
Train: $x \in \mathbb{R}^{4991x10}$
Test: $y \in \mathbb{R}^{991x10}$

Sigmoid activation function:

$$z(x) = \frac{1}{1 + e^{-x}}$$

Total: 115 parameters

$w_1 \in \mathbb{R}^{5x10}$
$b_1 \in \mathbb{R}^{10}$
$w_2 \in \mathbb{R}^{10x5}$
$b_2 \in \mathbb{R}^{5}$

Objective & Implementation-2

Helper function (neural net+Langevin grad)

```
using Distributions, LinearAlgebra, DelimitedFiles
#Bayesian Neural Network

function network(topo, train, test, learn_rate)
    #initialize all fixed variable
    return topo, train, test, learn_rate
end

function weight_init(topo)
    weight_layer_1=rand(Normal(0, 1), (topo[1],topo[2]))./sqrt(topo[1]);
    #bias first layer
    bias_1=rand(Normal(0, 1), (1,topo[2]))./sqrt(topo[2]);
    #weight second layer
    weight_layer_2=rand(Normal(0, 1), (topo[2],topo[3]))./sqrt(topo[2]);
    #bias second layer
    bias_2=rand(Normal(0, 1), (1,topo[3]))./sqrt(topo[2]);
    #concatenate weights and biases
    weight_output=[weight_layer_1, bias_1, weight_layer_2, bias_2];
    return weight_output
end
# layer=weight_output(topo);

function sigmoid(x)
    return 1 ./ (1 .+ exp.(-x))
end

function compute_output(topo)
    #compute hidden and last layer output
    hidden_output=zeros(1,topo[2]);
    last_output=zeros(1,topo[3]);
    return last_output
end

function sampleEr(topo, actualout)
    error=compute_output(topo) .- actualout;
    sqerror=sum(error.^2)/topo[3];
    return sqerror
end

function forward_pass(X, topo, weight_vec)
    z_1=(reshape(X,(1,topo[1])))*weight_vec[1] .+ weight_vec[2];
    hid_out=sigmoid(z_1);
    z_2=hid_out*weight_vec[3] .+ weight_vec[4];
    last_out=sigmoid(z_2);
    return hid_out,last_out
end


function backward_pass(inp, desired, topo, weight_vec, learn_rate)
    hid_out, last_out=forward_pass(inp, topo, weight_vec);
    out_delta=((reshape(desired,(1,topo[3]))) .- last_out) .* (last_out .* (1 .- last_out));
    hid_delta=out_delta*transpose(weight_vec[3]) .* (hid_out .* (1 .- hid_out));
    h_o_layer=2 # hidden to output layer
    for i in 1:topo[h_o_layer]
        for j in 1:topo[h_o_layer+1]
            #update weight layer 2
            weight_vec[3][i,j] += learn_rate * out_delta[j] * hid_out[i]
        end
    end
    for k in 1:topo[h_o_layer+1]
        weight_vec[4][k] += -1 * learn_rate * out_delta[k]
    end
```

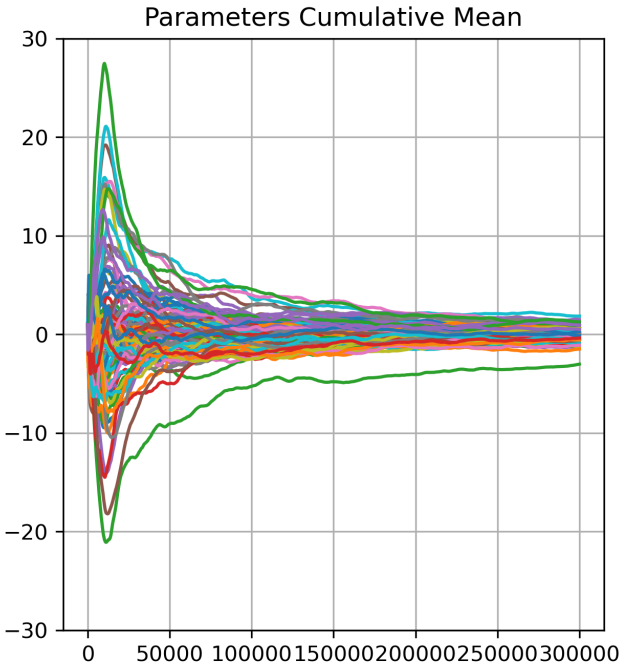
Initialization

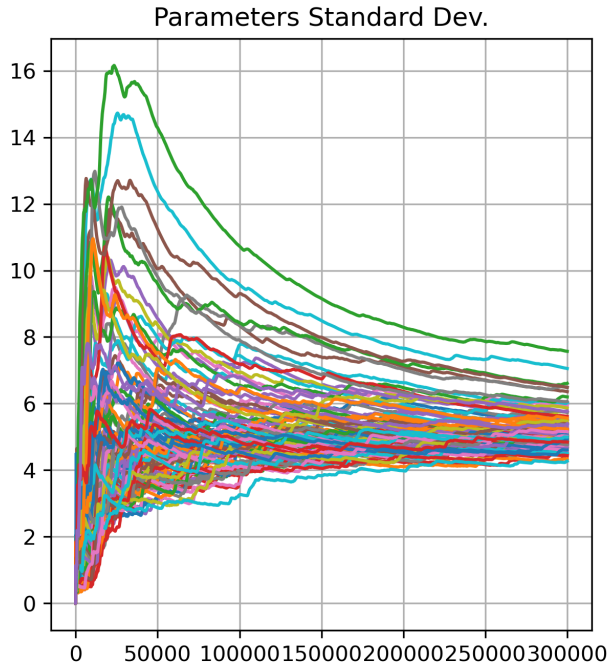
```
#initialization
## def run
# Load training and test data
train=readdlm("train_data_mat.txt");
test=readdlm("test_data_mat.txt");
trainsize=size(train)[1];
testsize=size(test)[1];
topo, train, test, learn_rate=network([5,10,5], train, test, 0.05);
depth=1;
w_size=topo[1]*topo[2]+topo[2]+topo[2]*topo[3]+topo[3];
w=rand(Normal(0, 1), (1,w_size));
w_proposal=rand(Normal(0, 1), (1,w_size));
weight_vec=weight_init(topo);
step_w = 0.25;
step_eta=0.2;
langevin_count=0;
pred_train=evaluate_proposal(train, w, topo, weight_vec);
pred_test=evaluate_proposal(test, w, topo, weight_vec);
y_train = train[:, topo[1]+1: topo[1] + last(topo)];
y_test = test[:, topo[1]+1: topo[1] + last(topo)];
eta=log(var(pred_train-y_train));
tau_pro=exp(eta);
sigma_squared=25;
nu_1=nu_2=0;
sigma_diagmat=zeros((w_size,w_size));
sigma_diagmat[diagind(sigma_diagmat)].=step_w;
delta_likelihood=0.5;
prior_current=prior_likelihood(sigma_squared, nu_1, nu_2, w, tau_pro, topo);
adapt_temp=2;
samples=150000;
likelihood,pred_train,rmsetrain,indi_rmsetrain,maetrain,mapetrain=likelihood_func(train, w, tau_pro, topo, adapt_
_,pred_test,rmsetest,indi_rmsetest,maetest,mapetest=likelihood_func(test, w, tau_pro, topo, adapt_temp);
trainacc=0;
testacc=0;
prop_list=zeros((samples,length(w)));
likeh_list=zeros((samples,2));
likeh_list[1,1:2]=[-100,-100];
surg_likeh_list=zeros((samples,2));
accept_list=zeros(samples);
pos_w=ones((samples,w_size));
lhood_list=zeros((samples,1));
surrogate_list=zeros((samples,1));
fxtrain_samples=ones((samples, trainsize, topo[3]));
fxtest_samples=ones((samples,testsize, topo[3]));
rmse_train=zeros(samples);
indi_rmse_train=zeros((samples,topo[3]));
mae_train=zeros(samples);
mape_train=zeros(samples);
rmse_test=zeros(samples);
indi_rmse_test=zeros((samples,topo[3]));
mae_test=zeros(samples);
mape_test=zeros(samples);
acc_train=zeros(samples);
acc_test=zeros(samples);
num_accepted=0;
langevin_count=0;
l_prob=0.5;
```

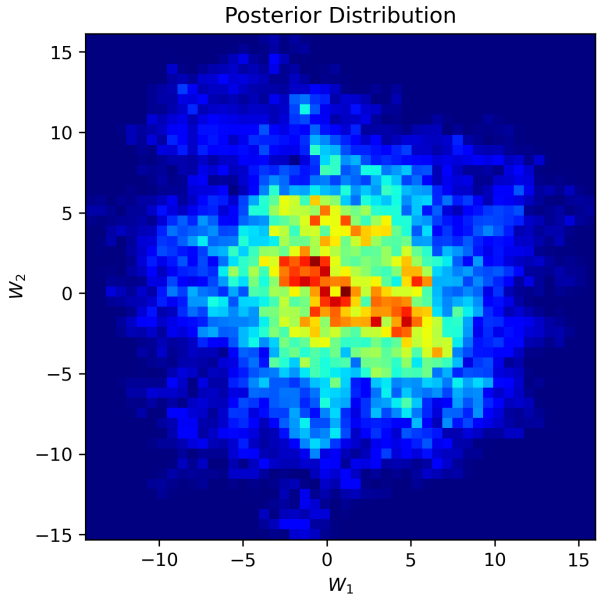
Main loop

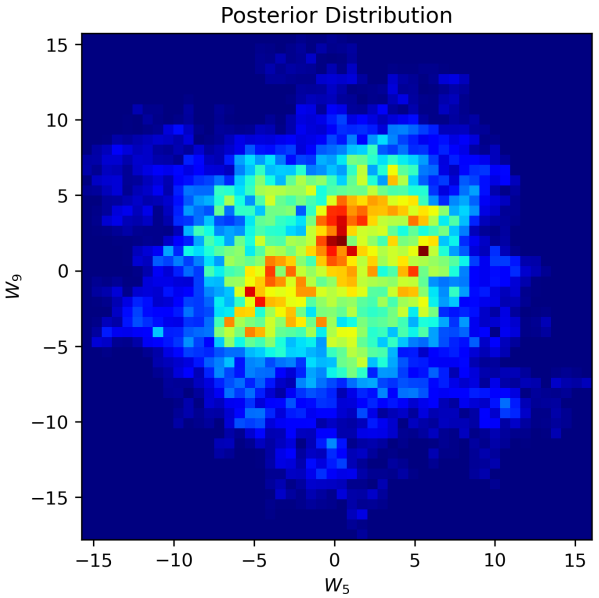
```
for i in 2:samples
    lx=rand(Uniform(0, 1));
    #using langevin gradient
    if lx<l_prob
        w_gd=langevin_gradient(train, w, depth, topo, weight_vec, learn_rate);
        w_proposal=rand.(Normal.(w_gd, step_w));
        w_prop_gd=langevin_gradient(train, w_proposal, depth, topo, weight_vec, learn_rate);
        wc_delta=(w-w_prop_gd);
        wp_delta=(w_proposal-w_gd);
        #compute variance
        sigma_sq=step_w^2;

        first=-0.5 * sum(wc_delta.^2)/sigma_sq;
        second= -0.5 * sum(wp_delta.^2)/sigma_sq;

        scaling_factor=0.25;

        diff_prop=first-second;
        diff_prop=diff_prop*scaling_factor/adapt_temp;
        langevin_count = langevin_count + 1
    else
        diff_prop=0;
        w_proposal=rand.(Normal.(w, step_w));
    end
    eta_pro=eta + rand(Normal(0, step_eta), 1)[1];
    tau_pro=exp(eta_pro);
    likelihood_proposal,pred_train,rmsetrain,indi_rmsetrain,maetrain,mapetrain=likelihood_func(train, w_proposal, tau_pro, topo, adapt_temp);
    _,pred_test,rmsetest,indi_rmsetest,maetest,mapetest=likelihood_func(test, w_proposal, tau_pro, topo, adapt_temp);
    prior_prop=prior_likelihood(sigma_squared, nu_1, nu_2, w_proposal, tau_pro, topo);
    diff_prior=prior_prop-prior_current;
    diff_likelihood=likelihood_proposal-likelihood;
    surg_likeh_list[i,1]=likelihood_proposal*adapt_temp;
    mh_prob=min(1, exp(diff_likelihood+diff_prior+diff_prop));
    accept_list[i]=num_accepted;
    u=rand(Uniform(0, 1));
    prop_list[i,1:w_size]=w_proposal;
    likeh_list[i,1]=likelihood_proposal;
    if u<mh_prob
        num_accepted=num_accepted+1;
        likelihood=likelihood_proposal;
        prior_current=prior_prop;
        w=w_proposal;
        eta=eta_pro;
        acc_train[i]=0;
        acc_test[i]=0;

        pos_w[i,1:size(pos_w)[2]]=w_proposal;
        fxtrain_samples[i,:,:]=pred_train;
        fxtest_samples[i,:,:]=pred_test;
        rmse_train[i]=rmsetrain;
        rmse_test[i]=rmsetest;
        indi_rmse_train[i,1:size(indi_rmse_train)[2]]=indi_rmsetrain;
        indi_rmse_test[i,1:size(indi_rmse_test)[2]]=indi_rmsetest;
        mae_train[i]=maetrain;
        mae_test[i]=maetest;
        mape_train[i]=mapetrain;
        mape_test[i]=mapetest;
    else
        pos_w[i,1:size(pos_w)[2]]=pos_w[i-1,1:size(pos_w)[2]];
        fxtrain_samples[i,:,:]=fxtrain_samples[i-1,:,:];
        fxtest_samples[i,:,:]=fxtest_samples[i-1,:,:];
        rmse_train[i]=rmse_train[i-1];
```

Result-1 (Convergence & Posterior)

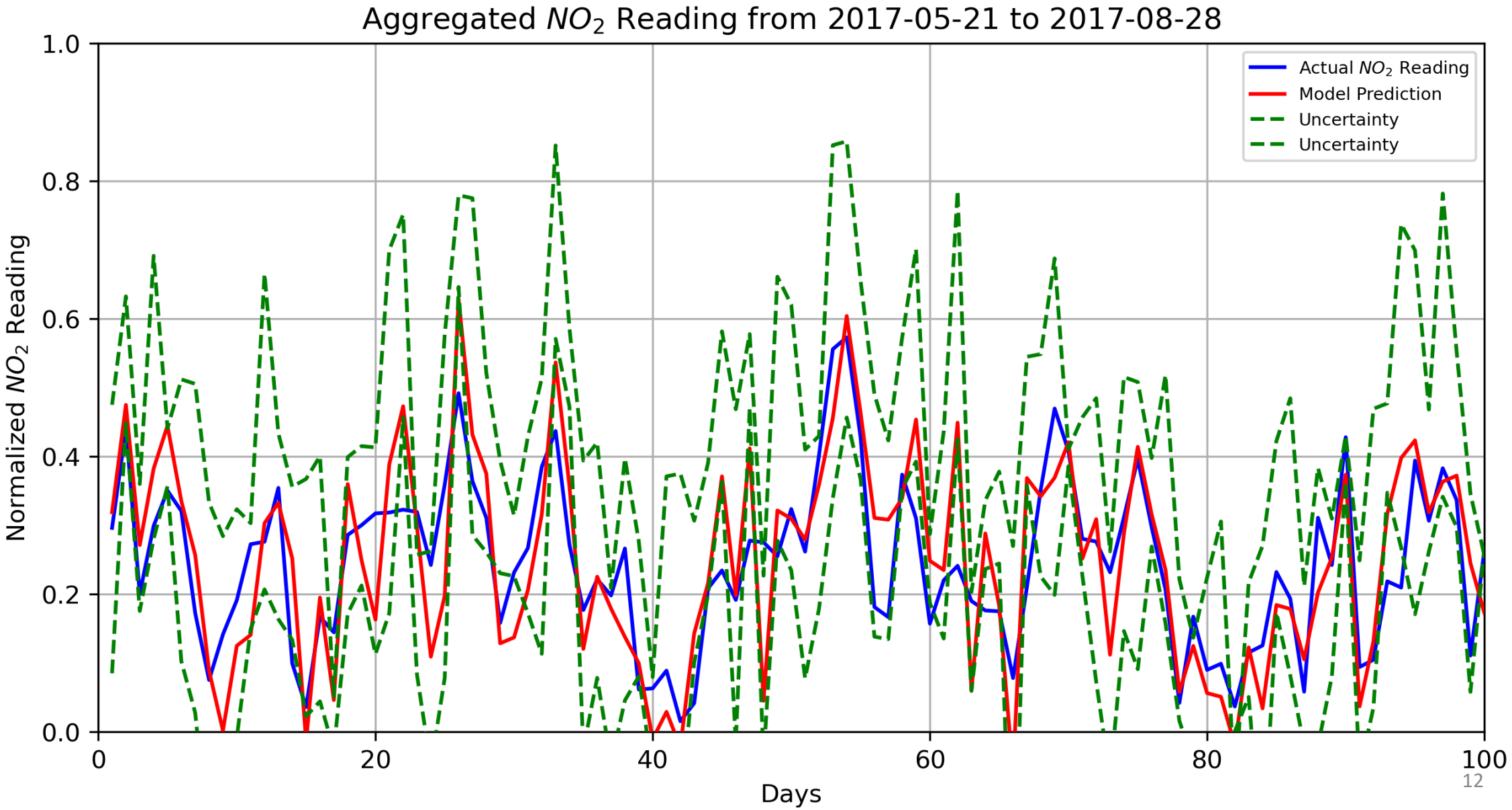




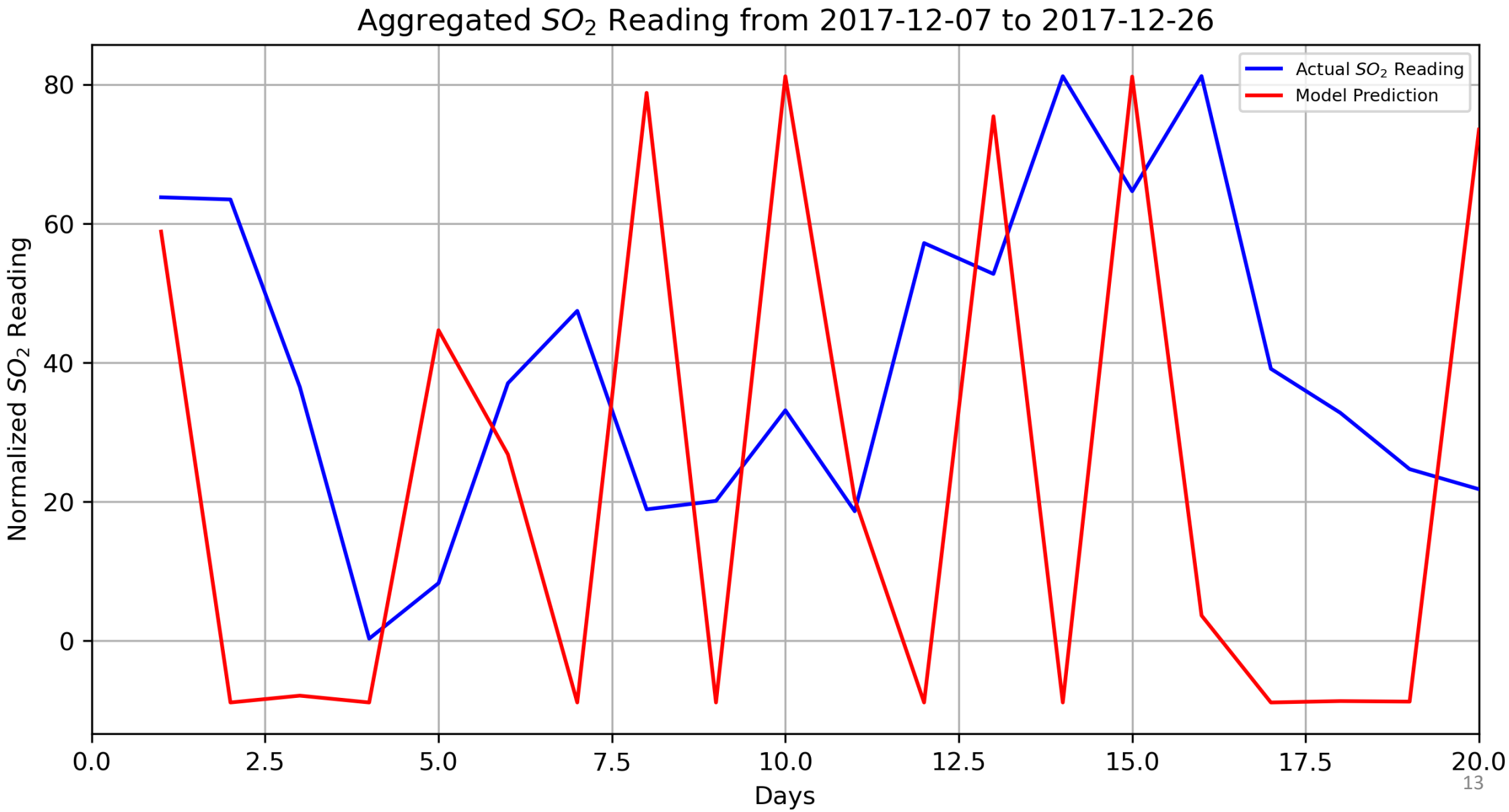




Result-2 (NO_2 Prediction)



Result-3 (SO_2 Prediction)



Conclusion

- Need further refinement of hyperparameters
- Could use a more suitable RNN with LSTM and GRU to get better prediction
- Can be benchmark against purely random walk MCMC (i.e., Multiple-Try Metropolis)
- Changing the number of multi-step prediction ahead of time