# PyTorch Based Implementation of Semi-Model-Based Reinforcement Learning with Uncertainty

Simon Lund, Sophia Sigethy, Georg Staber, and Malte Wilhelm

28.09.2021

Ludwig Maximilian University of Munich

# Table of Contents

# Introduction

# Motivation

Real life data is often noisy, scarce and unreliable

# Motivation

Real life data is often noisy, scarce and unreliable

Prediction quality of neural networks is highly dependent on quality of training data

# Motivation

Real life data is often noisy, scarce and unreliable

Prediction quality of neural networks is highly dependent on quality of training data

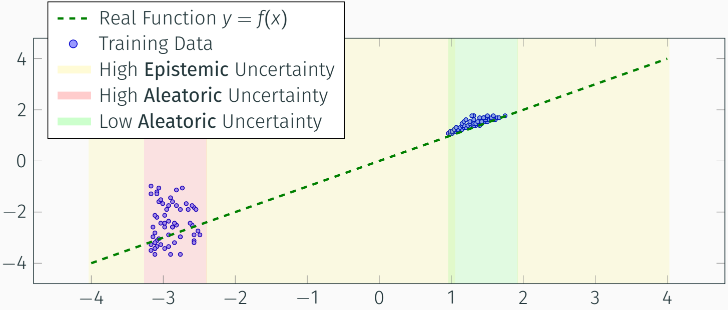
Model becomes uncertain regarding $y = f(x)$

# Motivation

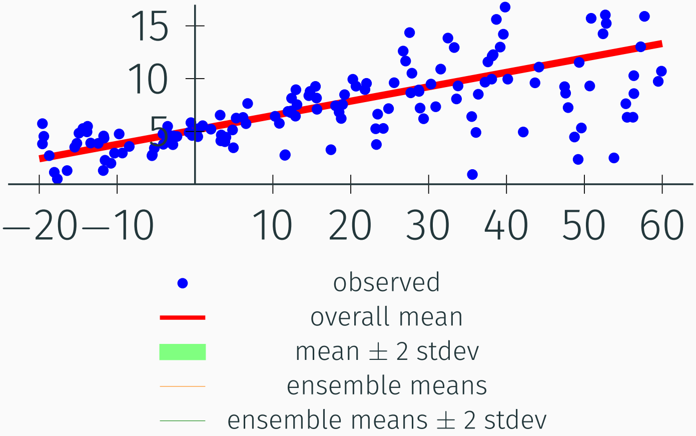
Real life data is often noisy, scarce and unreliable

Prediction quality of neural networks is highly dependent on quality of training data

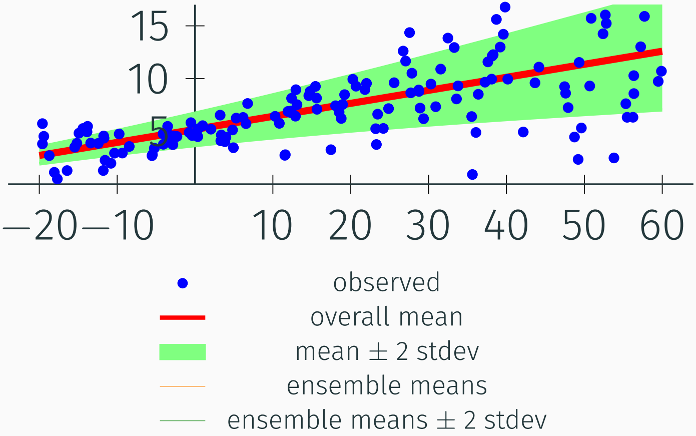
Model becomes uncertain regarding $y = f(x)$



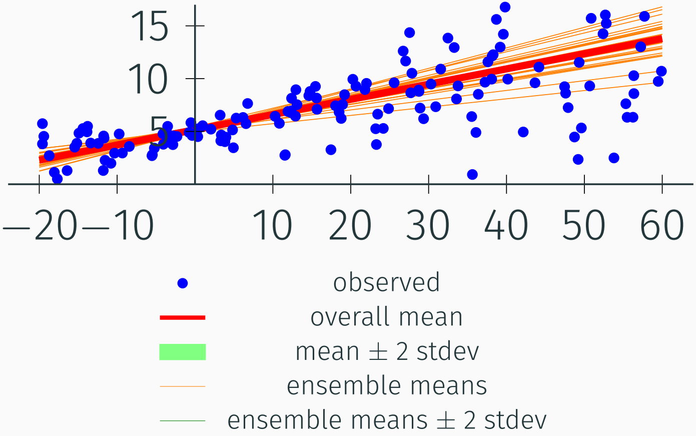
# No Uncertainty (linear regression)



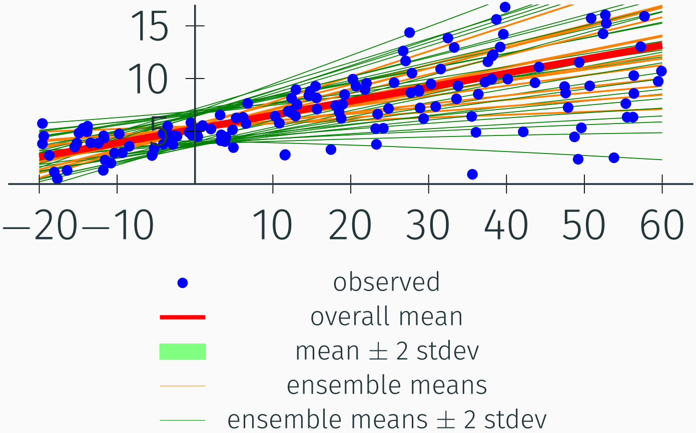
# Aleatoric Uncertainty



# Epistemic Uncertainty



# Aleatoric and Epistemic Uncertainty



# Concept

# Workflow

1. Sample data from noisy Cartpole Environment

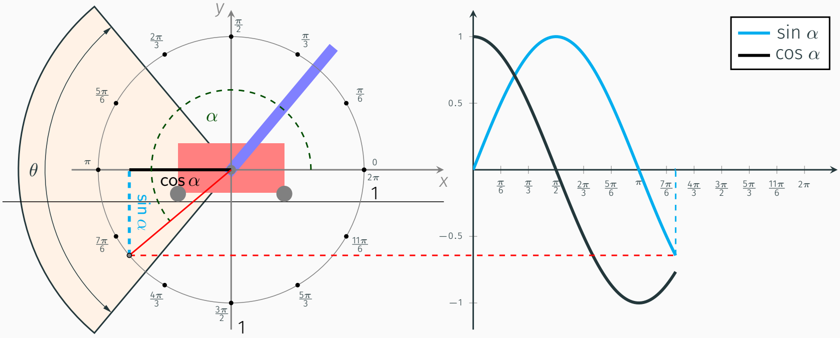
# Workflow

1. Sample data from noisy Cartpole Environment
2. Learn neural network for one-step dynamics

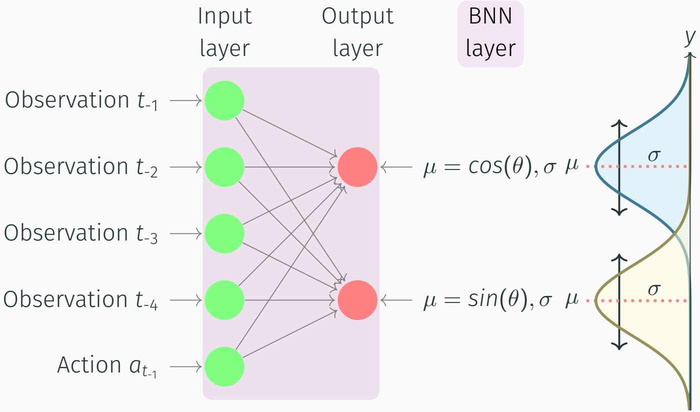
# Workflow

1. Sample data from noisy Cartpole Environment
2. Learn neural network for one-step dynamics
3. Train RL policy against dynamics model

# Data Sampling



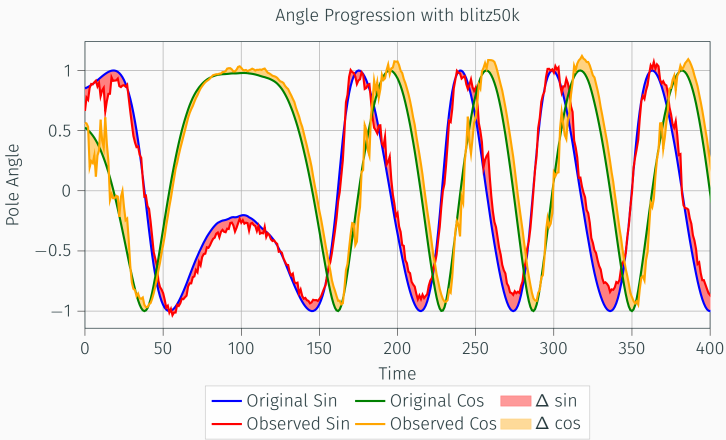
# Neural Network Architecture



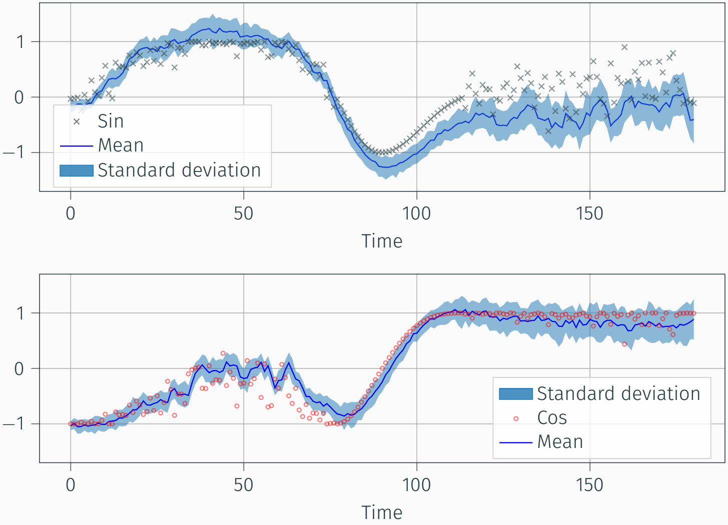
# Experiments & Results

# BNN predictions: 1k time series

# BNN predictions: 50k time series



# BNN Uncertainty: 75k time series



DEMO TIME

# Conclusion

# Take Aways

1. RL agent detects noisy sector and avoids it

# Take Aways

1. RL agent detects noisy sector and avoids it
2. RL agent has trouble solving both environments

# Take Aways

1. RL agent detects noisy sector and avoids it
2. RL agent has trouble solving both environments
3. Use PyTorch ;-)

Questions?

# References

P. Sountsov, C. Suter, J. Burnim and J. V. Dillon. *Regression with Probabilistic Layers in TensorFlow Probability — The TensorFlow Blog*. 12th Mar. 2019. URL: *https://blog.tensorflow.org/2019/03/regression-with-probabilistic-layers-in.html* (visited on 2nd Sept. 2021).

A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto and N. Dormann. *Stable Baselines3*. *https://github.com/DLR-RM/stable-baselines3*. 2019.

# References ii


Â. Lovatto. *angelolovatto/gym-cartpole-swingup: A simple, continuous-control environment for OpenAI Gym*. 7th July 2019. URL: `https://github.com/angelolovatto/gym-cartpole-swingup` (visited on 2nd Sept. 2021).

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba. “OpenAI Gym”. In: (2016). eprint: `arXiv:1606.01540`. URL: `https://gym.openai.com/`.

C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra. *Weight Uncertainty in Neural Networks*. 2015. arXiv: `1505.05424 [stat.ML]`.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: *http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf*.

# Backup Slides

# Creating a variational regressor class

```python
@variational_estimator
class BayesianRegressor(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.blinear = BayesianLinear(input_dim, output_dim)

    def forward(self, x):
        return self.blinear(x)
```

# Defining a confidence interval evaluating function

```python
def evaluate_regression(regressor, x, y, samples=25, std_multiplier=2):
    preds = [regressor(x) for i in range(samples)]
    preds = torch.stack(preds)
    means = preds.mean(axis=0)
    stds = preds.std(axis=0)
    ci_upper = means + (std_multiplier * stds)
    ci_lower = means - (std_multiplier * stds)
    ic_acc = (ci_lower <= y) * (ci_upper >= y)
    ic_acc = ic_acc.float().mean()
    return ic_acc, (ci_upper >= y).float().mean(), (ci_lower <= y).float().mean()
```

# Creating our regressor and loading data

```
optimizer = optim.Adam(regressor.parameters(), lr=0.01)
criterion = torch.nn.MSELoss()
complexity_cost_weight = 1. / x_train.shape[0]
```

# Main training and evaluating loop

```python
losses = []
for epoch in tqdm(range(100)):
    new_epoch = True
    for i, (datapoints, labels) in enumerate(dataloader_train):
        optimizer.zero_grad()

        loss = regressor.sample_elbo(
            inputs=datapoints,
            labels=labels,
            criterion=criterion,
            sample_nbr=3,
            complexity_cost_weight=complexity_cost_weight
        )

        loss.backward()
        optimizer.step()
```