

UNCERT

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

1. Introduction
2. Concept
3. Experiments & Results
4. Implementation
5. Conclusion

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**

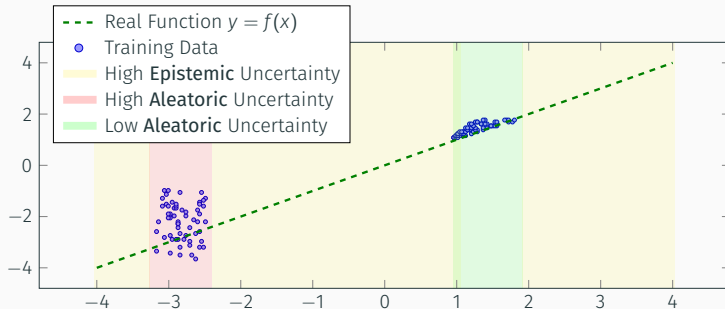
Model becomes **uncertain** regarding $y = f(x)$

Motivation

Real life data is often **noisy, scarce and unreliable**

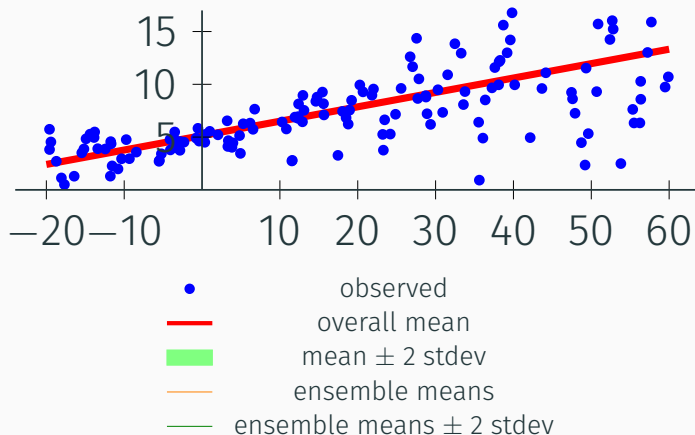
Prediction quality of neural networks is highly dependent on **quality of training data**

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



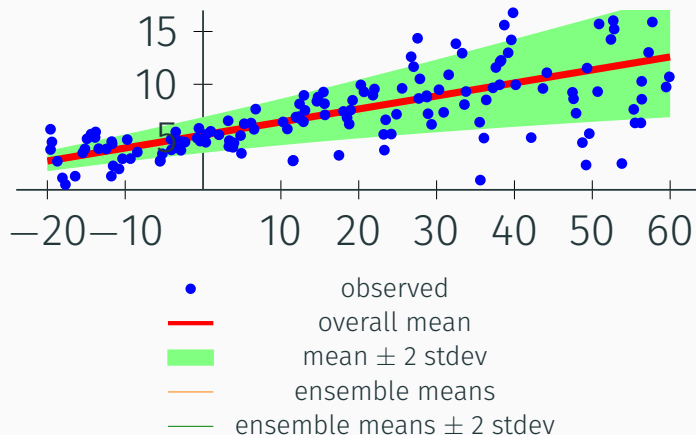
No Uncertainty (linear regression)

Mean captures the overall trend, but misses that y becomes more variable as x becomes larger.



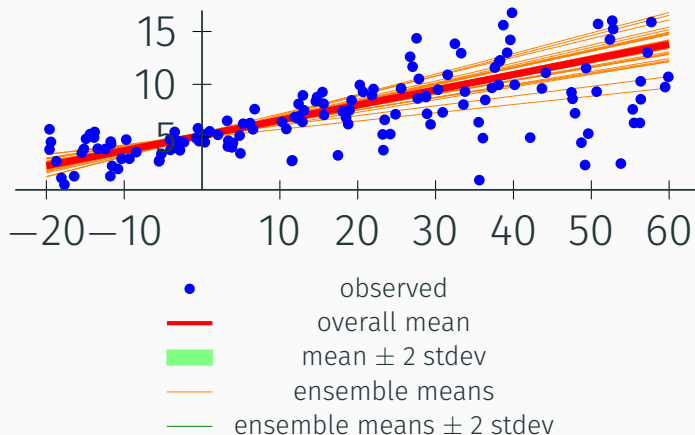
Aleatoric Uncertainty

Standard deviation represents variation inherent to the underlying process. This uncertainty can not be reduced.



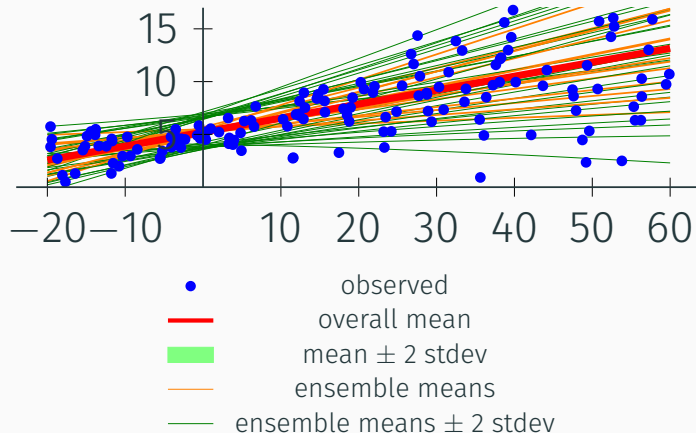
Epistemic Uncertainty

Each line represents a **different random draw** of the model parameters from the posterior distribution. Model is uncertain about the linear relationship.



Aleatoric and Epistemic Uncertainty

This model correctly predicts more variability as towards the extremes of x .



Convolutional Neural Network (CNN) vs. Bayesian Neural Network (BNN)

BNN has weights and biases that are probability distributions instead of single fixed values. These are updated during every output calculation.

Convolutional Neural Network (CNN) vs. Bayesian Neural Network (BNN)

BNN has weights and biases that are probability distributions instead of single fixed values. These are updated during every output calculation.

1. Built-in variability makes them **resistant to model overfitting** (when presented with previously unseen data).

Convolutional Neural Network (CNN) vs. Bayesian Neural Network (BNN)

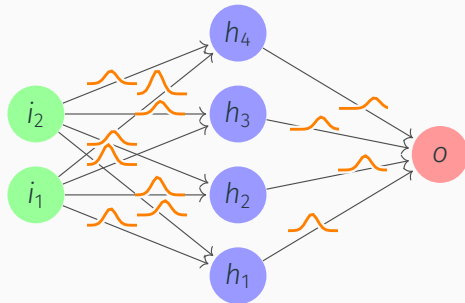
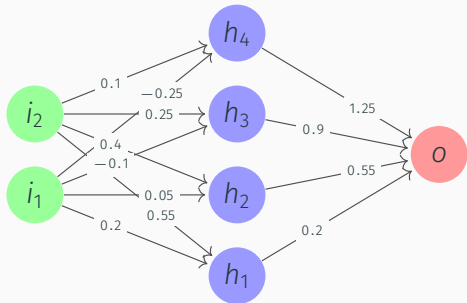
BNN has weights and biases that are probability distributions instead of single fixed values. These are updated during every output calculation.

1. Built-in variability makes them **resistant to model overfitting** (when presented with previously unseen data).
2. You can identify the **uncertainty** of predictions.

Convolutional Neural Network (CNN) vs. Bayesian Neural Network (BNN)

BNN has weights and biases that are probability distributions instead of single fixed values. These are updated during every output calculation.

1. Built-in variability makes them **resistant to model overfitting** (when presented with previously unseen data).
2. You can identify the **uncertainty** of predictions.



Concept

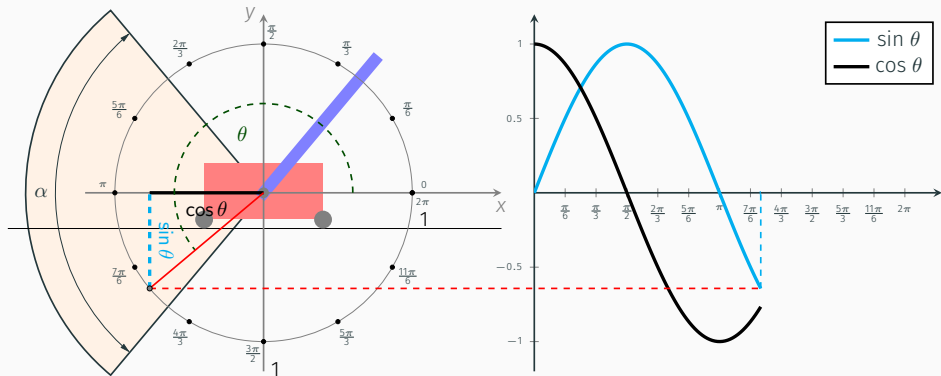
1. Sample data from noisy Cartpole Environment

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

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

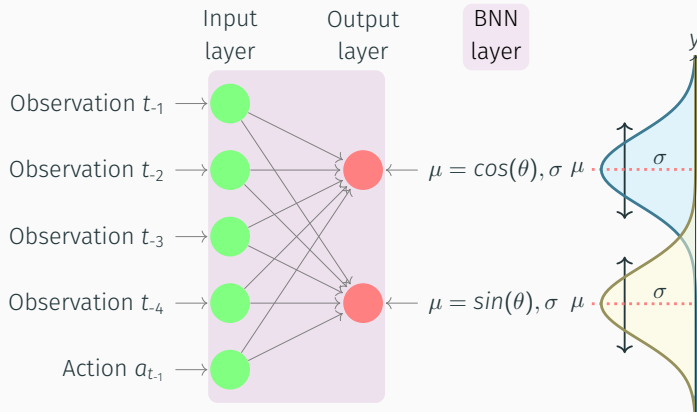
Data Sampling

CartPole superimposed with the unit circle.¹ α is the **noisy** section. The angle of the pole θ can be decomposed into **sine and cosine**. Action space is quantized.



¹Actual environment has the circle turned 90° anti-clockwise.

Bayesian Neural Network Architecture



Observation includes x position of the cart, its velocity, the angle of the pendulum θ and its angular velocity. 21 input nodes (5 observation parameters * 4 time steps + 1 action = 21) and two outputs. Action $a_{t_{-1}}$ will take the agent from observation t_{-1} to observation t_0 .

Experiments & Results

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
-----------	-------

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
<code>noisy sector</code>	0 - π (left half of unit circle)

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
<code>noisy sector</code>	0 - π (left half of unit circle)
<code>noise offset</code>	0.3

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
<code>noisy sector</code>	0 - π (left half of unit circle)
<code>noise offset</code>	0.3
<code>observation space</code>	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
<code>noisy sector</code>	0 - π (left half of unit circle)
<code>noise offset</code>	0.3
<code>observation space</code>	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)
<code>action space</code>	discrete, 10 actions

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
noisy sector	0 - π (left half of unit circle)
noise offset	0.3
observation space	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)
action space	discrete, 10 actions
NN epochs	100

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
noisy sector	0 - π (left half of unit circle)
noise offset	0.3
observation space	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)
action space	discrete, 10 actions
NN epochs	100
time series length	4

Experiment Configuration

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
noisy sector	0 - π (left half of unit circle)
noise offset	0.3
observation space	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)
action space	discrete, 10 actions
NN epochs	100
time series length	4
reward function	[centered, right, boundaries, best, cos, xpos_theta_uncert]

Table 1: Hyperparameter used to run the experiments.

Parameter	Value
noisy sector	0 - π (left half of unit circle)
noise offset	0.3
observation space	continuous, 5 dimensional (xpos, xdot, theta dot, theta sin, theta xos)
action space	discrete, 10 actions
NN epochs	100
time series length	4
reward function	[centered, right, boundaries, best, cos, xpos_theta_uncert]
RL algorithms	PPO

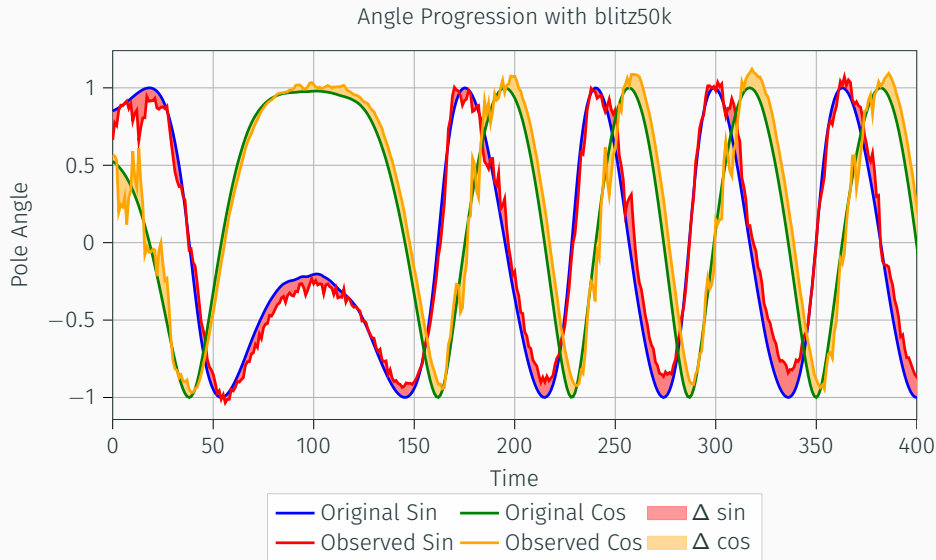
Experiments & Results

Bayesian Neural Network.
Random actions.

BNN predictions: 1k time series, random actions



BNN predictions: 50k time series, random actions



Experiments & Results

Well-trained PPO agent.

50k time series training data.

75k time steps learning.

Without Uncertainty

<https://youtu.be/Xuhp5B6EwWY>

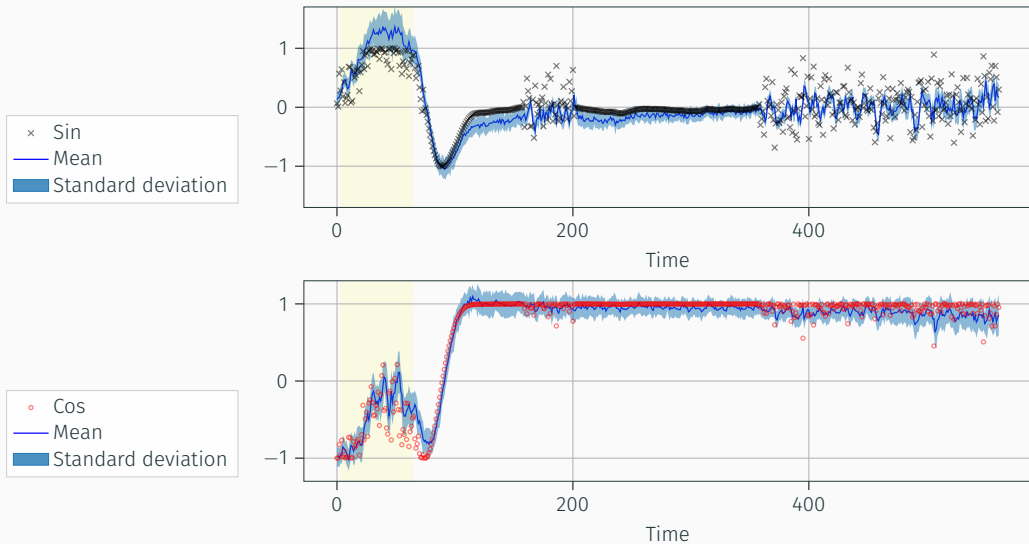
With Uncertainty

<https://youtu.be/vAlEon3I5lw>

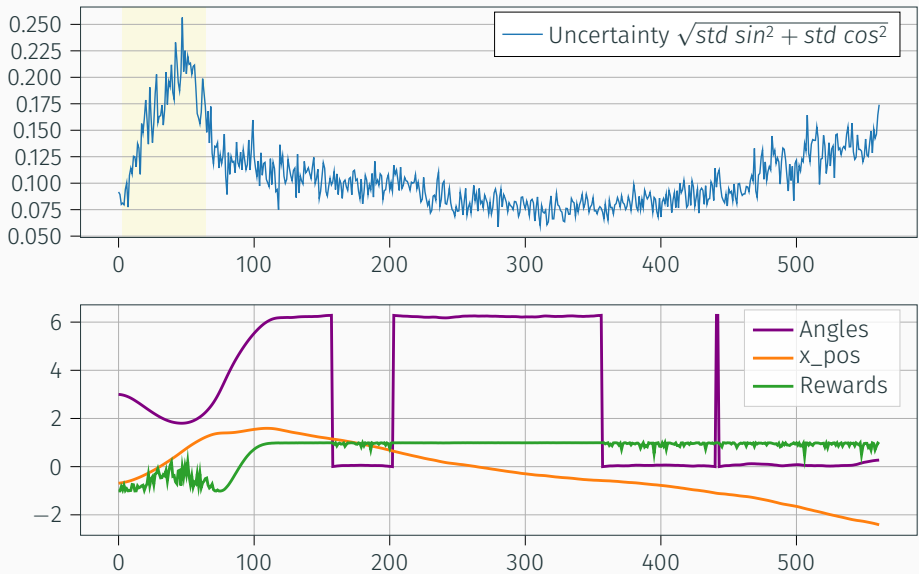
$$reward = \cos(\alpha) - \left(\sqrt{\sigma_{\sin}^2 + \sigma_{\cos}^2} * \epsilon \right)$$

```
1 reward = cosine - (math.sqrt(std_sin ** 2 + std_cos ** 2)*factor)
```

BNN Uncertainty: PPO 75k training steps



BNN Uncertainty: PPO 75k training steps



Implementation

“Talk is cheap. Show me the code.”

Linus Torvalds

Conclusion

1. RL agent detects noisy sector and avoids it

1. RL agent detects noisy sector and avoids it
2. RL agent is able to "solve" the environment even with uncertainty




1. RL agent detects noisy sector and avoids it
2. RL agent is able to "solve" the environment even with uncertainty
3. Use PyTorch ;-)


Questions?²

²Source code, presentation and an exhaustive report are at github.com/github-throwaway/ARL-Model-RL-Unsicherheit

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.

-  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](https://arxiv.org/abs/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

Backup Slides

BLiTZ Implementation

Creating a variational regressor class

```
1 @variational_estimator
2 class BayesianRegressor(nn.Module):
3     def __init__(self, input_dim, output_dim):
4         super().__init__()
5         self.blinear = BayesianLinear(input_dim, output_dim)
6
7     def forward(self, x):
8         return self.blinear(x)
```

Defining a confidence interval evaluating function

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

Creating our regressor and loading data

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

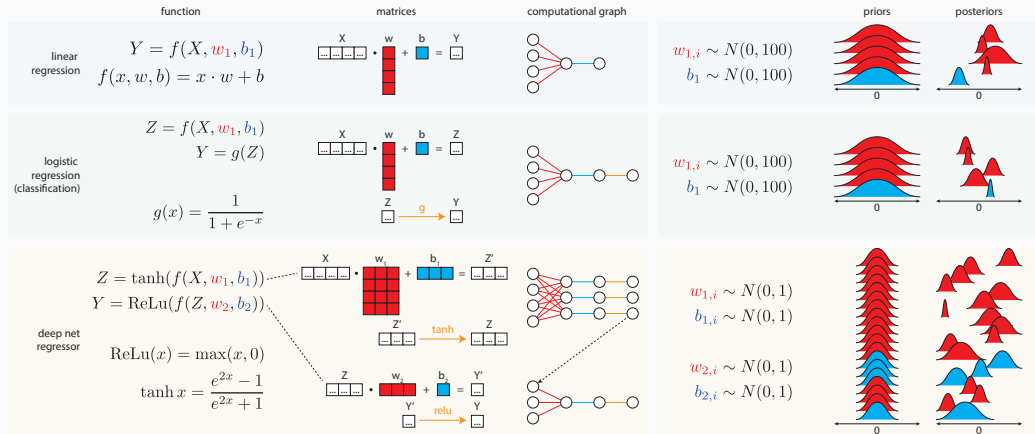
Main training and evaluating loop

```
1 losses = []
2 for epoch in tqdm(range(100)):
3     new_epoch = True
4     for i, (datapoints, labels) in enumerate(dataloader_train):
5         optimizer.zero_grad()
6
7         loss = regressor.sample_elbo(
8             inputs=datapoints,
9             labels=labels,
10            criterion=criterion,
11            sample_nbr=3,
12            complexity_cost_weight=complexity_cost_weight
13        )
14
15        loss.backward()
16        optimizer.step()
```

Backup Slides

Cheatsheet

Cheatsheet³



³<https://github.com/ericmjl/bayesian-deep-learning-demystified>

Backup Slides

Distributions

Probability distributions entangled by Bayes' rule

$$P(\mathbf{w} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathbf{w})P(\mathbf{w})}{P(\mathcal{D})} \quad \text{where } \mathcal{D} = (x_i, y_i)$$

