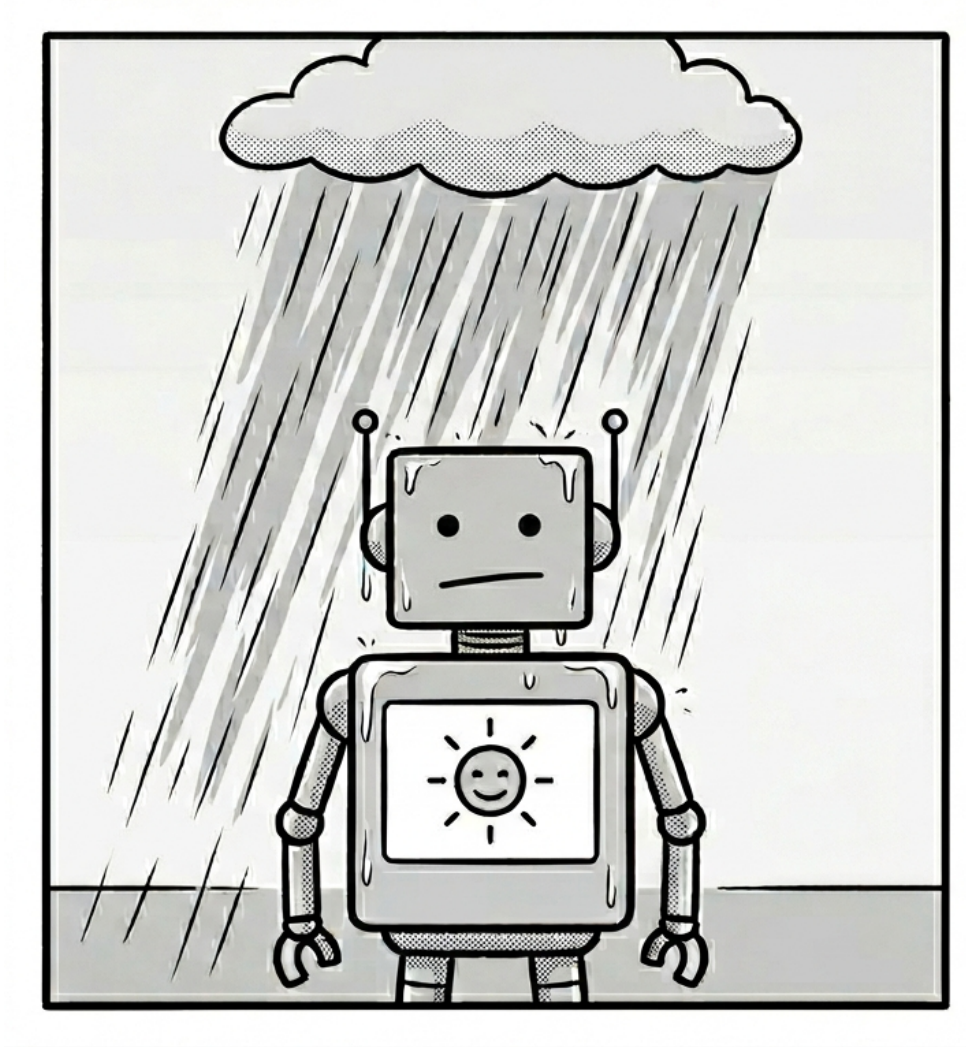
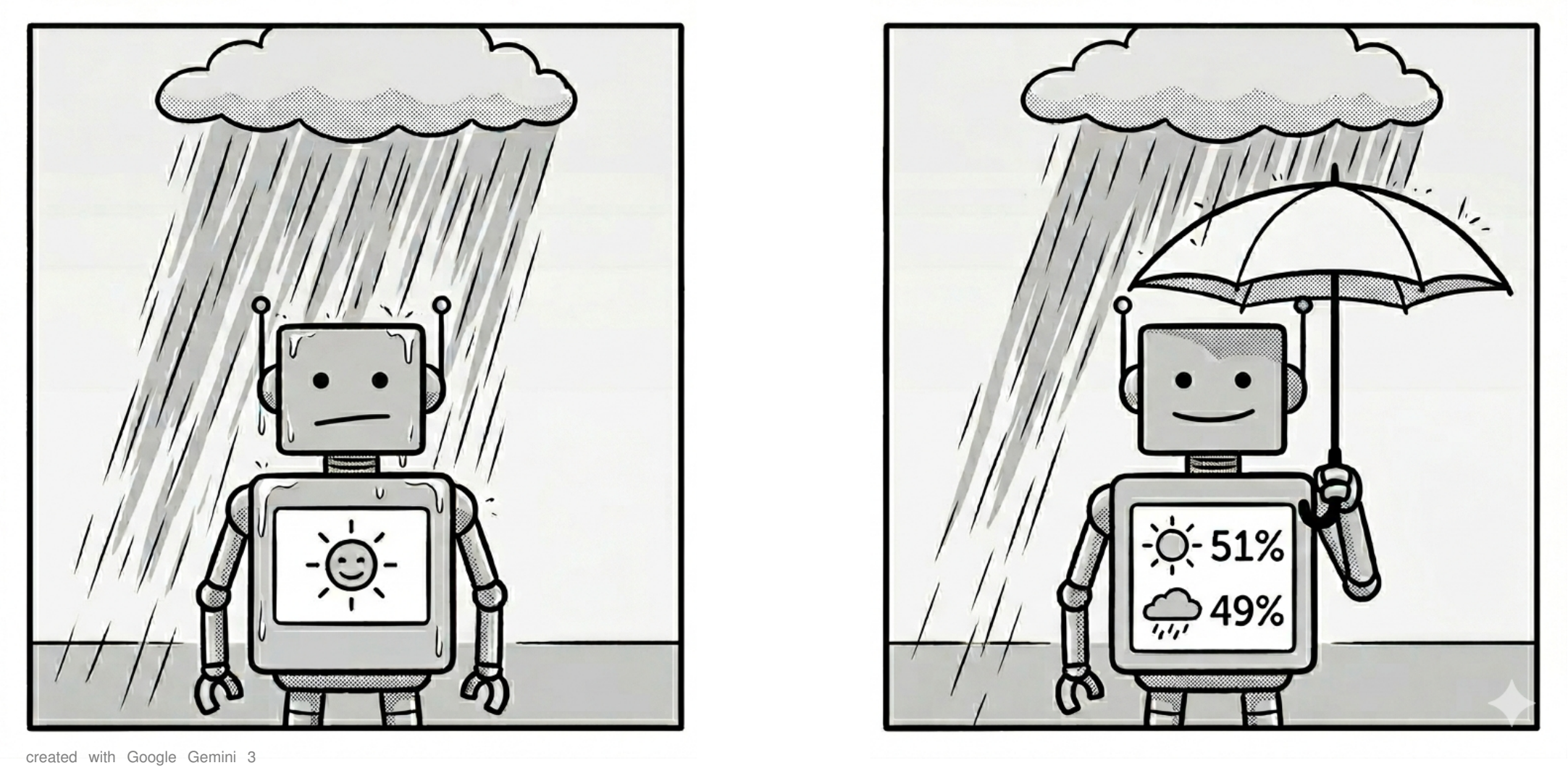
An Introduction to Variational Inference in PyTorch

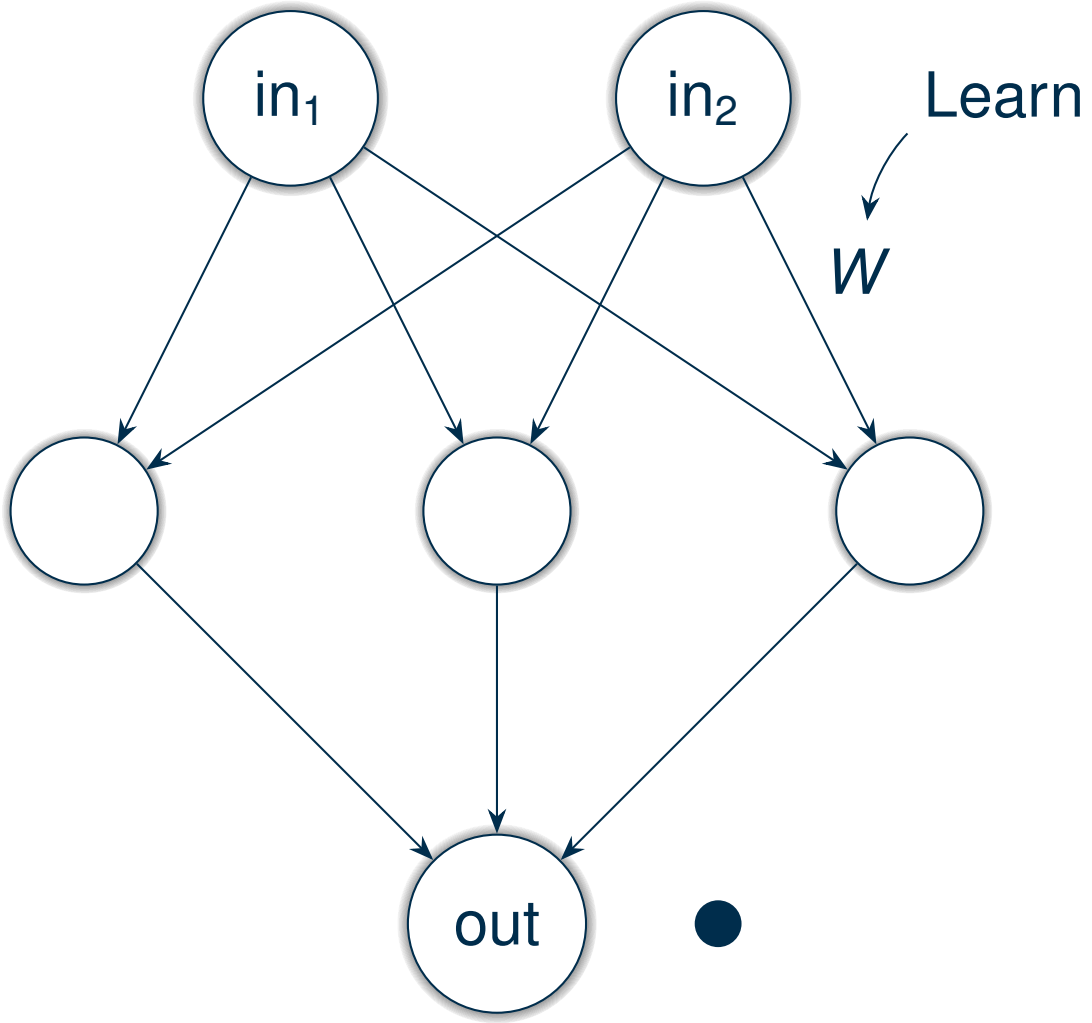
Lars Heyen, Robust and Efficient AI, SCC | 04/07/2026



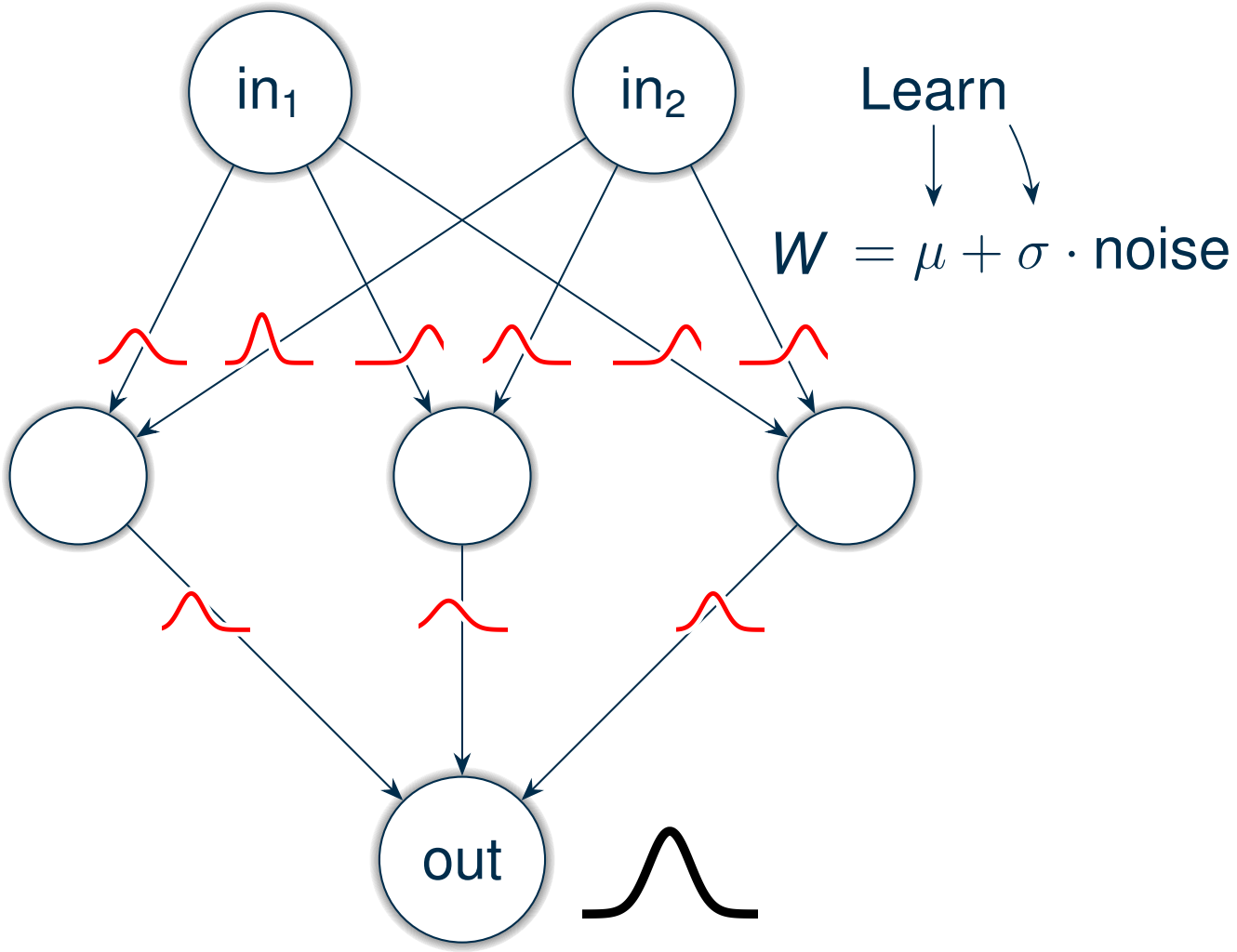


created with Google Gemini 3

Neural Networks



Neural Networks with Variational Inference



The Math

Model output:

$$g(y \mid x_i) = \int \mathrm{d}W \; \underbrace{p(y \mid x_i, W)}_{\substack{\text{predictive distribution} \\ \text{(given weights)}}} \cdot \underbrace{p(W \mid X, Y)}_{\substack{\text{weight posterior} \\ \text{(given dataset)}}}$$

Variational approximation:

$$W \sim q(W \mid \lambda) = \mathcal{N}(W \mid \mu, \Sigma) \approx p(W \mid X, Y)$$

Loss ansatz:

$$\mathcal{L} = \mathrm{KL}(q(\cdot \mid \lambda) \mid\mid p(\cdot \mid X, Y)) = \int \mathrm{d}W \, q(W \mid \lambda) \log\left(\frac{q(W \mid \lambda)}{p(W \mid X, Y)}\right)$$

Bayes' Theorem:

$$p(W \mid X, Y) = \frac{\overbrace{p(Y \mid X, W)}^{\text{likelihood}} \overbrace{p(W)}^{\text{prior}}}{\underbrace{p(Y \mid X)}_{\text{evidence}}}$$

Evidence Lower Bound (ELBO):

$$\mathrm{ELBO} = \mathbb{E}_{W \sim q}[\log p(Y \mid X, W)] - \mathrm{KL}(q(\cdot \mid \lambda) \mid\mid p(\cdot))$$

The Math

Model output:

$$g(y \mid x_i) = \int \mathrm{d}W \underbrace{p(y \mid x_i, W)}_{\substack{\text{predictive distribution} \\ \text{(given weights)}}} \cdot \underbrace{p(W \mid X, Y)}_{\substack{\text{weight posterior} \\ \text{(given dataset)}}}$$

Variational approximation:

$$W \sim q(W \mid \lambda) = \mathcal{N}(W \mid \mu, \Sigma) \approx p(W \mid X, Y)$$

Loss ansatz:

$$\mathcal{L} = \mathrm{KL}(q(\cdot \mid \lambda) \mid\mid p(\cdot \mid X, Y)) = \int \mathrm{d}W \, q(W \mid \lambda) \log\left(\frac{q(W \mid \lambda)}{p(W \mid X, Y)}\right)$$

Bayes' Theorem:

$$p(W \mid X, Y) = \frac{\overbrace{p(Y \mid X, W)}^{\text{likelihood}} \overbrace{p(W)}^{\text{prior}}}{\underbrace{p(Y \mid X)}_{\text{evidence}}}$$

Evidence Lower Bound (ELBO):

$$\mathrm{ELBO} = \mathbb{E}_{W \sim q}[\log p(Y \mid X, W)] - \mathrm{KL}(q(\cdot \mid \lambda) \mid\mid p(\cdot))$$

Easy to get wrong

The Math

Model output:

$$g(y \mid x_i) = \int \mathrm{d}W \underbrace{p(y \mid x_i, W)}_{\substack{\text{predictive distribution} \\ \text{(given weights)}}} \cdot \underbrace{p(W \mid X, Y)}_{\substack{\text{weight posterior} \\ \text{(given dataset)}}}$$

Variational approximation:

$$W \sim q(W \mid \lambda) = \mathcal{N}(W \mid \mu, \Sigma) \approx p(W \mid X, Y)$$

Loss ansatz:

$$\mathcal{L} = \mathrm{KL}(q(\cdot \mid \lambda) \,||\, p(\cdot \mid X, Y)) = \int \mathrm{d}W \, q(W \mid \lambda) \log\left(\frac{q(W \mid \lambda)}{p(W \mid X, Y)}\right)$$

Bayes' Theorem:

$$p(W \mid X, Y) = \frac{\overbrace{p(Y \mid X, W)}^{\text{likelihood}} \overbrace{p(W)}^{\text{prior}}}{\underbrace{p(Y \mid X)}_{\text{evidence}}}$$

Evidence Lower Bound (ELBO):

$$\mathrm{ELBO} = \mathbb{E}_{W \sim q}[\log p(Y \mid X, W)] - \mathrm{KL}(q(\cdot \mid \lambda) \,||\, p(\cdot))$$

Easy to get wrong $\rightarrow$ **torch-blue**

In Practice

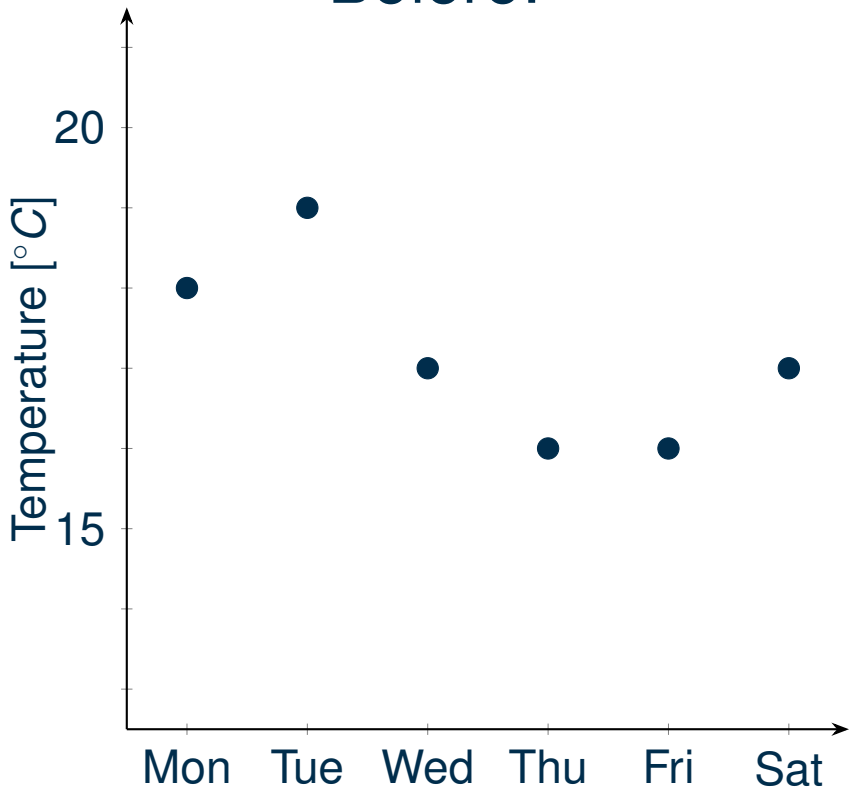
```
from torch import nn


# Define model
model = nn.Linear(in_dim, out_dim).to(device)


# Set up regression task loss

loss_fn = nn.MSELoss()

# Training loop
train_model(model, loss_fn, training_data)
```

In Practice: VI with torch-blue

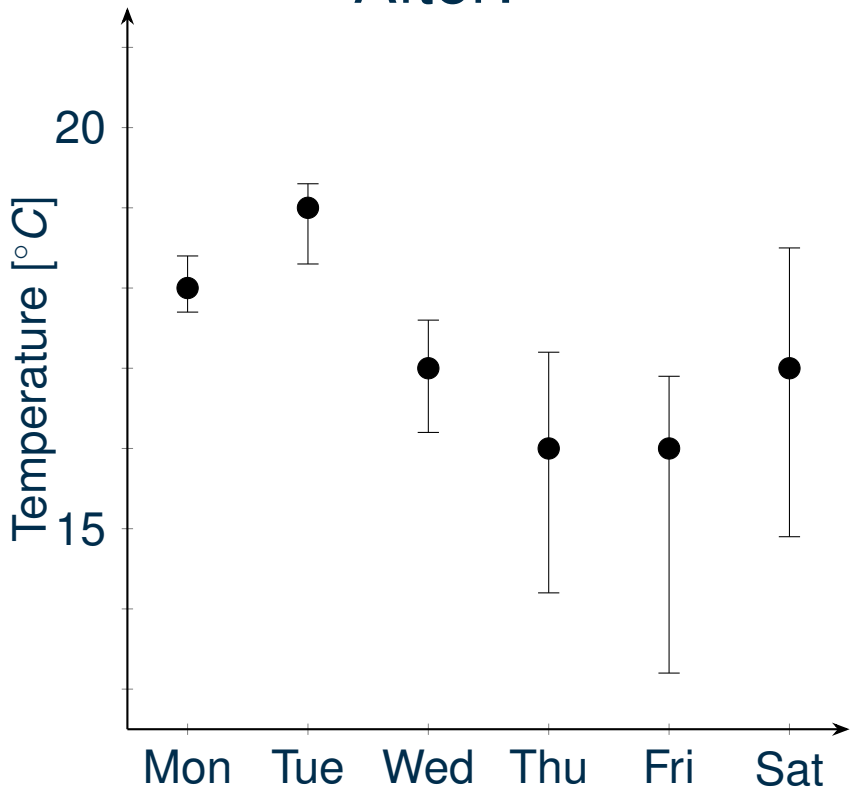
```python
from torch import nn
import torch_blue.vi as vi

# Define model
model = nn.Linear(in_dim, out_dim).to(device)
vi.convert_to_vimodule(model)

# Set up regression task loss
predictive_distribution = vi.distributions.Normal()
loss_fn = vi.KullbackLeiblerLoss(predictive_distribution, len(training_data))

# Training loop
train_model(model, loss_fn, training_data)
```

Temperature Forecasts – Now with Error Bars

Before:

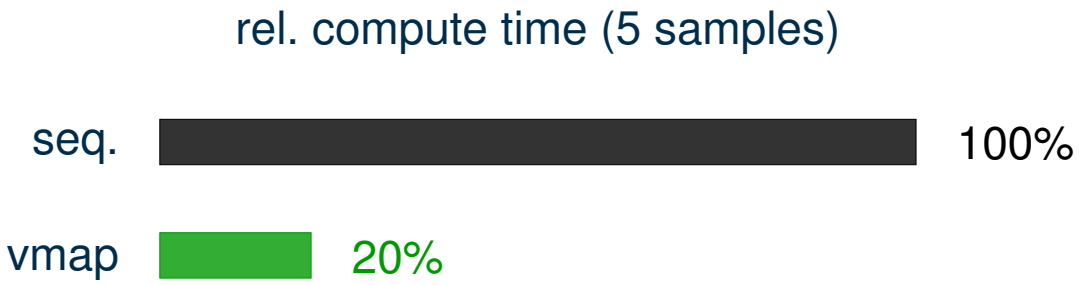


After:



Vectorized Random Sampling

```python
def sampled_forward(self, x, num_samples):
    extended_x = x.expand(num_samples, *x.shape)
    return torch.vmap(self.forward, randomness="different")(extended_x)
```



Further Advantages of torch-blue

- ☑ PyTorch-like interface for manual model building
- ☑ Autoconversion functionality for PyTorch models
- ☑ Compatible with PyTorch’s distributed computing functionality
- ☑ Vectorized random sampling for more efficient GPU utilization
- ☑ Easy implementation of custom distributions

If you are interested in Variational Inference now

torch-blue

```
pip install torch-blue
```

PyPI: `torch-blue`
ReadTheDocs: `torch-blue.readthedocs.io`

`github.com/RAI-SCC/torch_blue`
`torch_blue` on github