Introduction to Pyro

By Giulia Caglia of Objective Platform

Goals for Today

1. Understanding the basic building blocks of Pyro
2. Building a simple Linear Regression Model in Pyro
3. Understanding the difference between an Autoencoder and Variational Autoencoder
4. Building a Variational Autoencoder in Pyro

-> Keep the theory high-level -> Focus on practical coding exercises

Part 0: Setting up the Workspace

Setting up the Workspace

Option 1: Local Jupyter Lab

1. Git clone the repo to your local machine
2. [Optional] create a new virtual environment with your favourite environment management tool
3. Install python requirements with `pip3 install -r requirements.txt` or `poetry install` if you use poetry
4. [Optional] Install graphviz, e.g. with `sudo apt install graphviz` on Ubuntu
5. Start a local Jupyter server and get ready to hack!

Option 2: Colab Environment

1. Go to Google Colab
2. Click on File -> Open Notebook -> GitHub
3. Enter github URL: https://github.com/pyladiesams/pyro-may2023
4. Find the part I and II notebooks and copy this code into the first cell of each of them.
5. Execute the first two cells.

Part I: Linear Regression in Pyro

Introduction: What is Pyro?

- Framework for probabilistic programing in Python
- Thin wrapper around PyTorch -> Comparable usage and syntax
- Flexible, lightweight framework
- Allows us to reason about uncertainty
    - Accepts prior assumptions (i.e. prior distributions) across parameters
    - Facilitates regularization of parameters
    - Can perform well in very high dimensional but sparse environments
    - Estimates posterior distributions of parameters and predictions

Building Blocks of Pyro

- PyroModule → Subclass or register torch modules as Pyro Modules (similar to torch.nn.Module)

- PyroSample → Specify distribution to draw samples from

- pyro.plate → Sample copies of a random variable

- SVI → Main inference algorithm of Pyro

PyroModule

- Used to subclass torch modules as Pyro modules
- Parameters of PyroModules are registered with Pyro's parameter store and can be regularized with PyroSample (see next slide)
- Initialized Pytorch modules can also be registered with pyro with pyro.module(name: str, object: nn.Module)

```python
from torch import nn
from pyro.nn import PyroModule

assert issubclass(PyroModule[nn.Linear], nn.Linear)
assert issubclass(PyroModule[nn.Linear], PyroModule)
```

```
In [25]: pyro.get_param_store().get_all_param_names()
Out[25]: dict_keys([])

In [26]: _ = PyroModule[nn.Linear](3, 1)(X)

In [27]: pyro.get_param_store().get_all_param_names()
Out[27]: dict_keys(['weight', 'bias'])
```

PyroSample

- Initializes a tensor, that will draw a sample from a specified distribution whenever called!
- Regularized through a Pyro distribution
- Use expand and to_event to shape the output
- Use PyroParam for PyroModules, or pyro.sample for other variables

```
from pyro.nn import PyroSample


class BayesianRegression(PyroModule):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = PyroModule[nn.Linear](in_features, out_features)
        self.linear.weight = PyroSample(
            dist.Normal(0., 1.).expand([out_features, in_features]).to_event(2)
            )
        self.linear.bias = PyroSample(
            dist.Normal(0., 10.).expand([out_features]).to_event(1)
            )

    def forward(self, x, y=None):
        sigma = pyro.sample("sigma", dist.Uniform(0., 10.))
        mean = self.linear(x).squeeze(-1)
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Normal(mean, sigma), obs=y)
        return mean
```

PyroSample

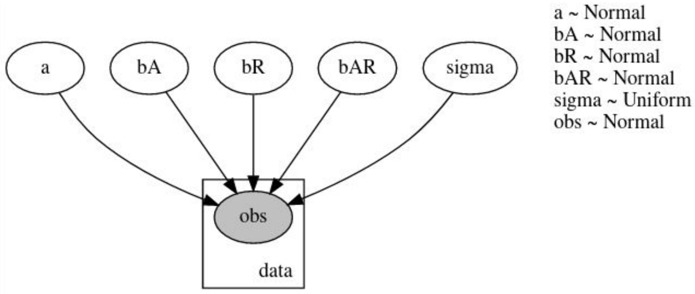
- Initializes a tensor, that will draw a sample from a specified distribution whenever called!
- Regularized through a Pyro distribution
- Use `expand` and `to_event` to shape the output
- Use PyroParam for `PyroModules`, or `pyro.sample` for other variables

```
from pyro.nn import PyroSample


class BayesianRegression(PyroModule):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = PyroModule[nn.Linear](in_features, out_features)
        self.linear.weight = PyroSample(
            dist.Normal(0., 1.).expand([out_features, in_features]).to_event(2)
            )
        self.linear.bias = PyroSample(
            dist.Normal(0., 10.).expand([out_features]).to_event(1)
            )

    def forward(self, x, y=None):
        sigma = pyro.sample("sigma", dist.Uniform(0., 10.))
        mean = self.linear(x).squeeze(-1)
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Normal(mean, sigma), obs=y)
        return mean
```

pyro.plate



- Technical implementation of plate notation in graphical models
- Context manager used to indicate repeated, independent sampling
- Usage: with pyro.plate(name: str, n_samples: int)

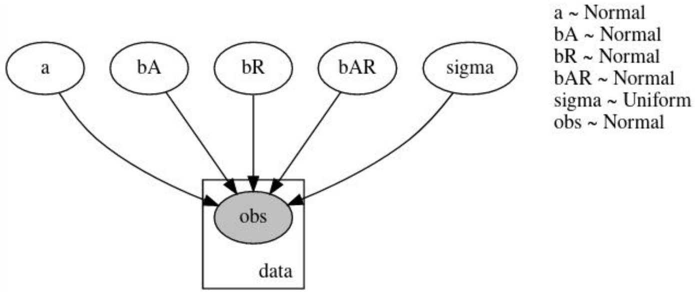
```
from pyro.nn import PyroSample


class BayesianRegression(PyroModule):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = PyroModule[nn.Linear](in_features, out_features)
        self.linear.weight = PyroSample(
            dist.Normal(0., 1.).expand([out_features, in_features]).to_event(2)
            )
        self.linear.bias = PyroSample(
            dist.Normal(0., 10.).expand([out_features]).to_event(1)
            )

    def forward(self, x, y=None):
        sigma = pyro.sample("sigma", dist.Uniform(0., 10.))
        mean = self.linear(x).squeeze(-1)
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Normal(mean, sigma), obs=y)
        return mean
```

pyro.plate

- Technical implementation of plate notation in graphical models
- Context manager used to indicate repeated, independent sampling
- Usage: with pyro.plate(name: str, n_samples: int)




```python
from pyro.nn import PyroSample


class BayesianRegression(PyroModule):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = PyroModule[nn.Linear](in_features, out_features)
        self.linear.weight = PyroSample(
            dist.Normal(0., 1.).expand([out_features, in_features]).to_event(2)
            )
        self.linear.bias = PyroSample(
            dist.Normal(0., 10.).expand([out_features]).to_event(1)
            )

    def forward(self, x, y=None):
        sigma = pyro.sample("sigma", dist.Uniform(0., 10.))
        mean = self.linear(x).squeeze(-1)
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Normal(mean, sigma), obs=y)
        return mean
```

SVI

- Stochastic Variational Inference as main inference algorithm of Pyro
- Finds approximation of intractable posterior by maximizing ELBO
- Facilitates flexible inference through guides
- Basic use cases make use of Pyro’s Autoguides

```python
model = MyModel()
svi = pyro.infer.SVI(
    model=model,
    guide=pyro.infer.autoguide.AutoDiagonalNormal(model),
    optim=pyro.optim.Adam(optim_args={"lr": 1e-3}),
    loss=pyro.infer.Trace_ELBO()
)

total_loss = 0
for epoch in range(EPOCHS):
    total_loss += svi.step(X, y)
```

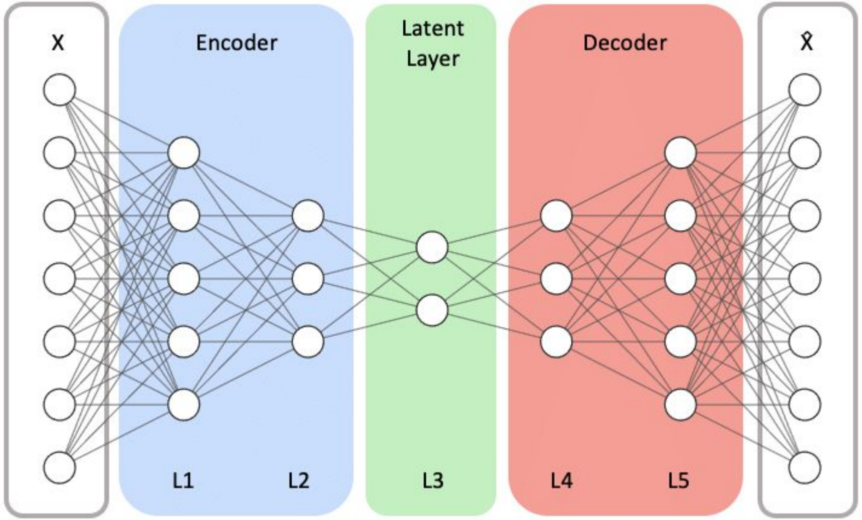
Time to get coding!

→ Find the first tasks in “workshop/Part 1 - Linear Modelling with Pyro - Empty Template.ipynb”

Let’s take a break of ~15 minutes to ask questions, network, have a snack, …

Part II: Variational Autoencoders in Pyro

Autoencoders



- Learns efficient data representations
- Compresses input into a lower-dimensional space
- Minimizes reconstruction error to learn meaningful features

→ E.g. used for dimensionality reduction

→ No regularization will lead to non-continuous latent space

Song, Youngrok & Hyun, Sangwon & Cheong, Yun-Gyung. (2021). Analysis of Autoencoders for Network Intrusion Detection. Sensors. 21. 4294. 10.3390/s21134294.

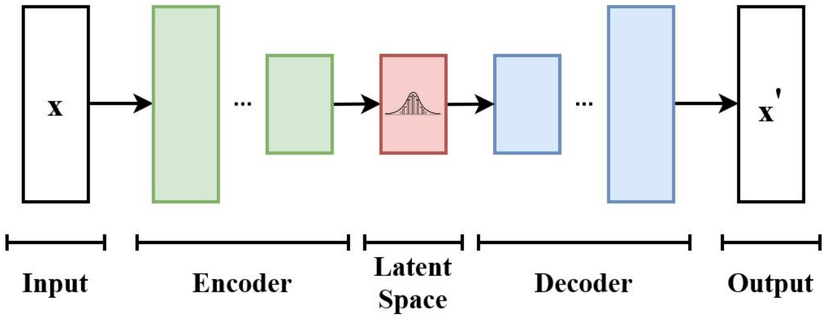
Original Digit

Reconstructed Digit

Latent Space samples

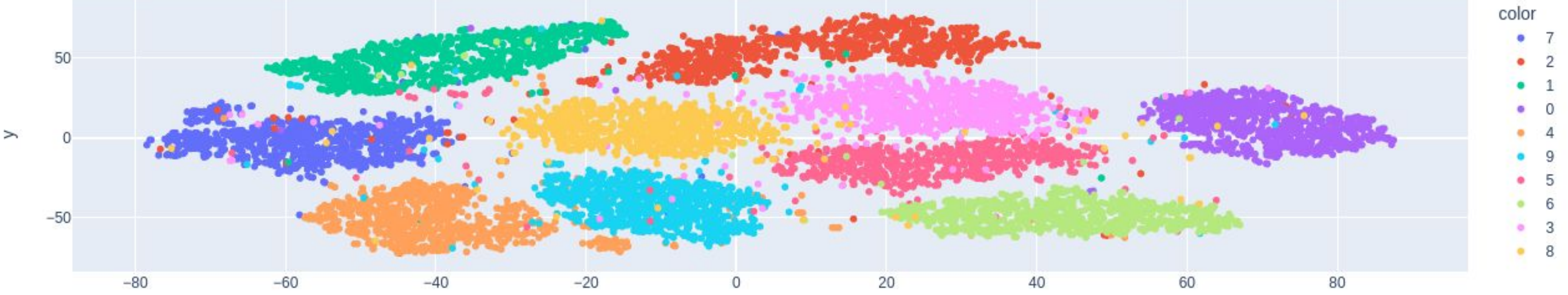
Examples of Autoencoder Outputs

Variational Autoencoders



- Variational Autoencoders (VAEs) are architecturally similar to regular Autoencoders
- VAEs model a latent space as probability distribution
- Probability distribution serves as regularization of latent space

→ Facilitates finite, complete and continuous latent space for interpolation and sampling



T-SNE Embedding of Latent Space of Variational Autoencoder

Original Digit

Reconstructed Digits

Latent Space samples

Examples of VAE Outputs

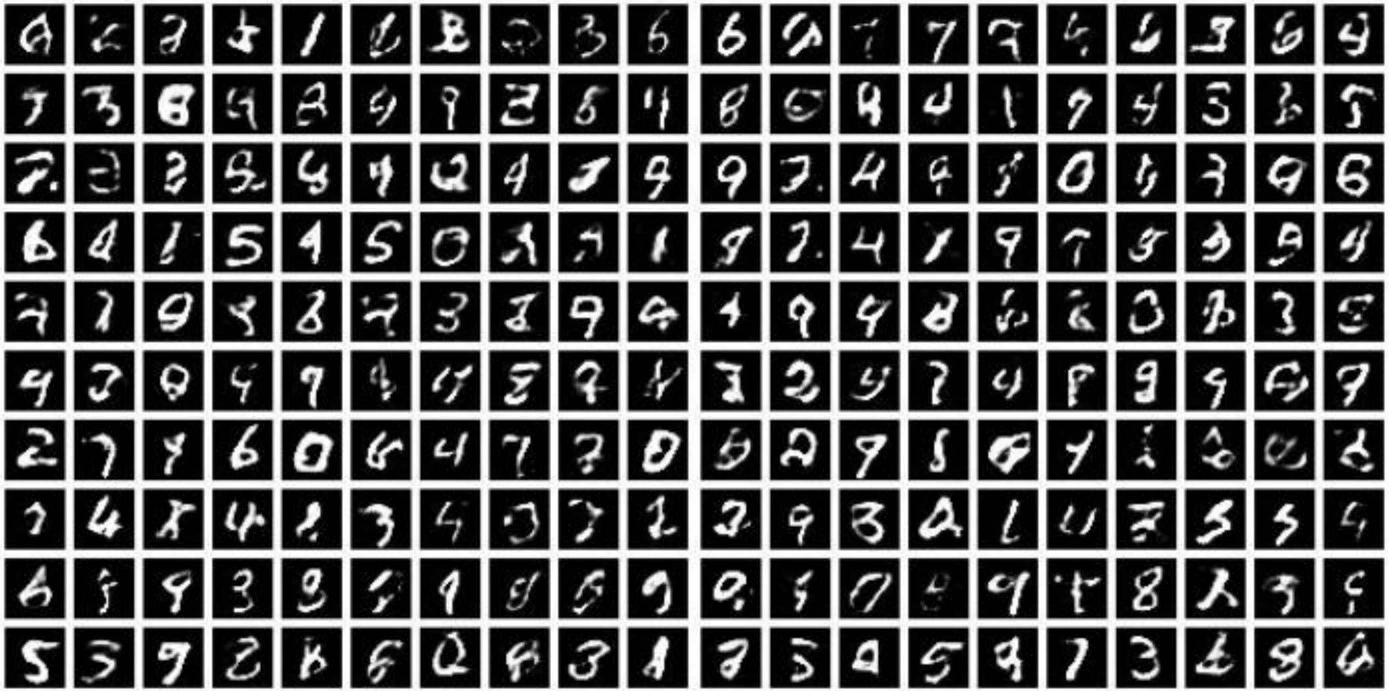


Figure 4: Samples from generative model.

Samples from [VAE in Pyro Docs]

Time to get coding!

→ Find the second tasks in “workshop/Part 2 - Variational Autoencoder - Empty Template.ipynb”

Thank you for dropping by!