Probabilistic programming II
Inference with Pyro

Thomas Hamelryck
January 2023

Overview

- Pyro
  - Deep probabilistic programming
- Stochastic Variational inference (SVI)
- Hamiltonian Monte Carlo / NUTS
- [Stein variational inference]
- Example
  - Coin flip model
  - p(heads)
- Exercise: Bayesian neural network
  - Fisher’s iris data set
  - Classification problem

Pyro PPL

Why Pyro?

- pyro.ai
- Universal probabilistic programming language (PPL)
  - Built upon PyTorch
  - Python based
  - Adds Bayesian inference to deep learning
  - Uber labs/Broad institute (MIT, Harvard)
- Freely available, easy installation
  - `pip install pyro-pp`
  - Google colab, Anaconda

Pyro: Universal deep PPL

- **Universal**
  - Can represent any computable probability distribution.
- **Scalable**
  - Scales to large data sets with little overhead.
- **Minimal**
  - Implemented with a small core of powerful, composable abstractions.
- **Flexible**
  - Aims for automation when you want it, control when you need it.

Specifying a probabilistic model in Pyro

Iris data set

- **Fisher's Iris data set** is a multivariate data set introduced by the British statistician Ronald Fisher in 1936 (linear discriminant analysis, LDA).

Iris Versicolor **Iris Setosa** **Iris Virginica**

Iris data set

- Based on (150) measurements of petal and sepal width and length (4-vector), **classify** the flower as Versicolor, Setosa or Virginica (3 classes)

Iris Versicolor Iris Setosa Iris Virginica

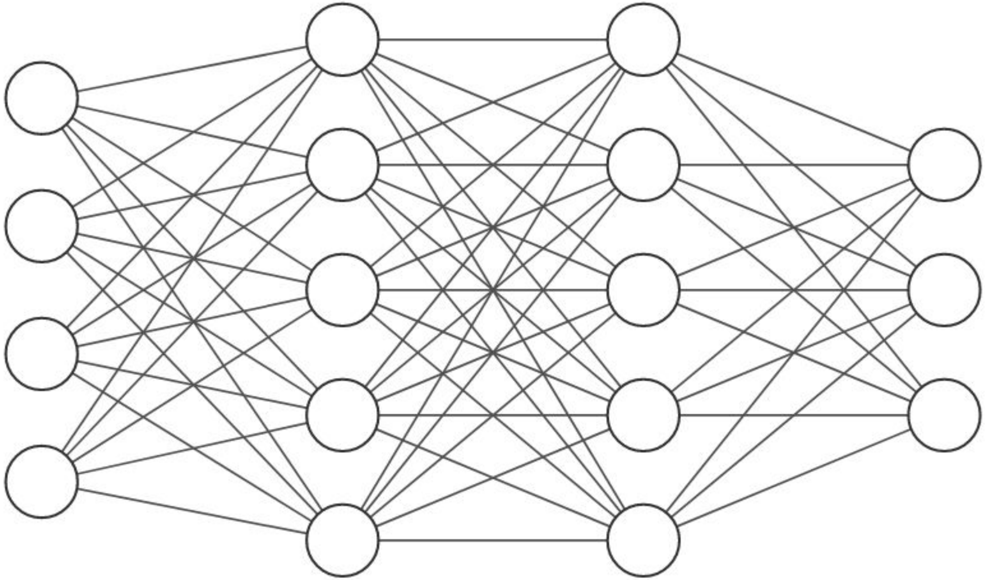
Iris classification problem

- x
  - Independent variable / input
  - 4 measurements of width and length of petals and sepals
  - Vector of 4 floats
- y
  - Dependent variable / output
  - What we want to predict
  - 3 iris classes: Versicolor, Setosa or Virginica
  - Categories: 0, 1, 2

Bayesian neural network for Iris

- 2 hidden layers, 5 neurons, parameters $\boldsymbol{\theta}$

Petals & sepals

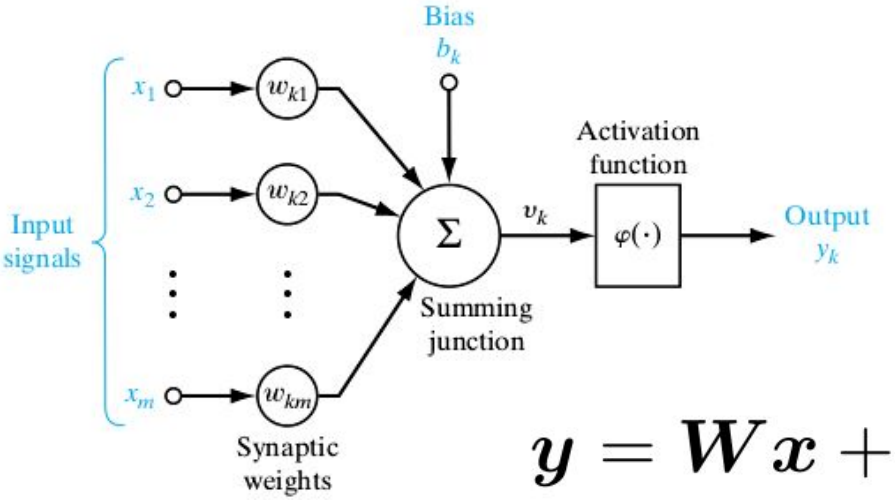


Probabilities of
3 iris classes
+
Uncertainty over all parameters

Input Layer $\in \mathbb{R}^4$ Hidden Layer $\in \mathbb{R}^5$ Hidden Layer $\in \mathbb{R}^5$ Output Layer $\in \mathbb{R}^3$

The humble digital neuron…

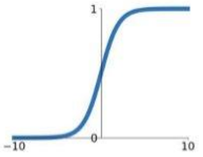
- Calculates the weighted sum of the inputs
- Applies a nonlinear function to the sum



$$y = Wx + b$$

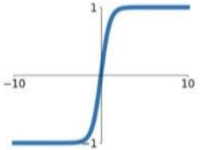
Sigmoid

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



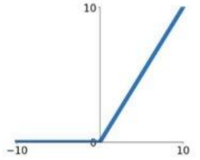
tanh

$\tanh(x)$



ReLU

$\max(0, x)$



Picture: The Men Who Stare at Codes/Shruti Jadon

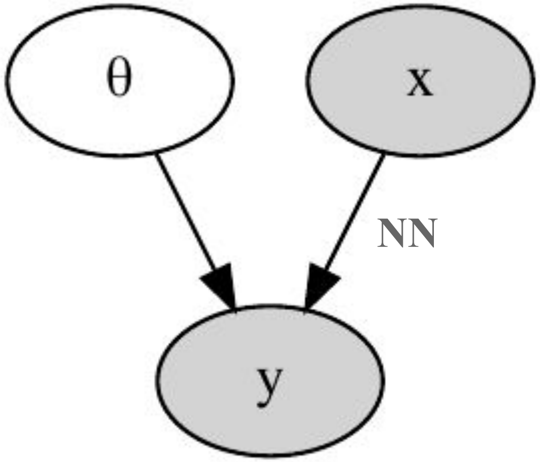
Bayesian inference

- We use a NN for classification in the Iris exercise
- Ideally, we want to be Bayesian about its parameters
    - We want a posterior distribution over the network's parameters
- **Posterior ∝ Likelihood × Prior**

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{x}) \pi(\boldsymbol{\theta})$$

MAP, variational Bayes, Bayes

- A NN maps **x** to y
  - Shaded=observed
- θ=parameters of the NN
- **ML or MAP estimate**
  - Maximum a posteriori
  - Single, most likely value
- **Variational Bayes estimate**
  - Diagonal Gaussian, custom guide
- **Full Bayesian estimate**
  - Samples from the posterior



NN model in Pyro

```
def model(x, y=None):
        ...
        # Priors (layer 1)
        w1=pyro.sample("w1", pdist.Normal(0, 1).expand([x_dim, h_dim]).to_event(2))
        b1=pyro.sample("b1", pdist.Normal(0, 1).expand([h_dim]).to_event(1))
        ...
        # NN
        h1=torch.tanh((x @ w1) + b1)
        h2=torch.tanh((h1 @ w2) + b2)
        logits=(h2 @ w3 + b3)
        # Categorical likelihood
        with pyro.plate("labels", n):
            obs=pyro.sample("obs", pdist.Categorical(logits=logits), obs=y)
```

Prediction

- **Posterior predictive distribution:** predict **y**' from a new input **x**'

$$p(\boldsymbol{y}' \mid \boldsymbol{x}', \boldsymbol{x}, \boldsymbol{y}) \propto \int p(\boldsymbol{y}' \mid \boldsymbol{\theta}, \boldsymbol{x}')p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{x})\mathrm{d}\theta$$

  - MAP: single value $\boldsymbol{\theta}_{\mathrm{MAP}}$
  - Variational Bayes: $\boldsymbol{\theta}$ is Gaussian distributed (or custom guide)
  - Full Bayes: set of samples for $\boldsymbol{\theta}$
    - Integral becomes a sum over samples
- Pyro provides a **single interface** for all cases
  - `pyro.infer.Predictive`

NN model in Pyro

```python
def model(x, y=None):
        ...
        # Layer 1
        w1=pyro.sample("w1", pdist.Normal(0, 1).expand([x_dim, h_dim]).to_event(2))
        b1=pyro.sample("b1", pdist.Normal(0, 1).expand([h_dim]).to_event(1))
        ...
        # NN
        h1=torch.tanh((x @ w1) + b1)
        h2=torch.tanh((h1 @ w2) + b2)
        logits=(h2 @ w3 + b3)
        # Categorical likelihood
        with pyro.plate("labels", n):
            obs=pyro.sample( "obs", pdist.Categorical(logits=logits),  obs=y)
```

The pyro.sample primitive

- Primitive **stochastic function**
    - **Random variable**
- Returns a sample from the specified distribution
    - This is a **named sample**
- The behavior of this stochastic function can be changed at runtime depending on how it is being used

```
w=pyro.sample("w", pyro.distributions.Normal(0, 1))
```

expand

- Suppose you want to sample a tensor with shape (3,4) with elements sampled from the same normal distribution.
  - This can be done using `expand`
  - All the random variables will be conditionally independent
  - **`batch_shape==(3,4), event_shape==()`**

```
d=Normal(0,1).expand((3,4))
x=d.sample()
assert x.shape==(3,4)
assert d.log_prob(x).shape==(3,4)
```

to_event

- `to_event` allows creating dependent random variables
  - The `log_prob` method will only produce a single number for each event
  - Works on the left-most dimension
  - **`batch_shape==(3), event_shape==(4)`**

```
d=Normal(0,1).expand((3,4)).to_event(1)
x=d.sample()
assert x.shape==(3,4)
assert d.log_prob(x).shape==(3,)
```

pyro.plate

- `pyro.plate` declares **conditional independence**
  - This is important to make automated inference efficient
- In our case, all observations y_i are IID given $\mathbf{x}_i$ and $\boldsymbol{\theta}$
  - IID=independent and identically distributed

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$$

pyro.plate

- `pyro.plate` declares **conditional independence**

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$$

- **Likelihood** for IID data uses `pyro.plate`:

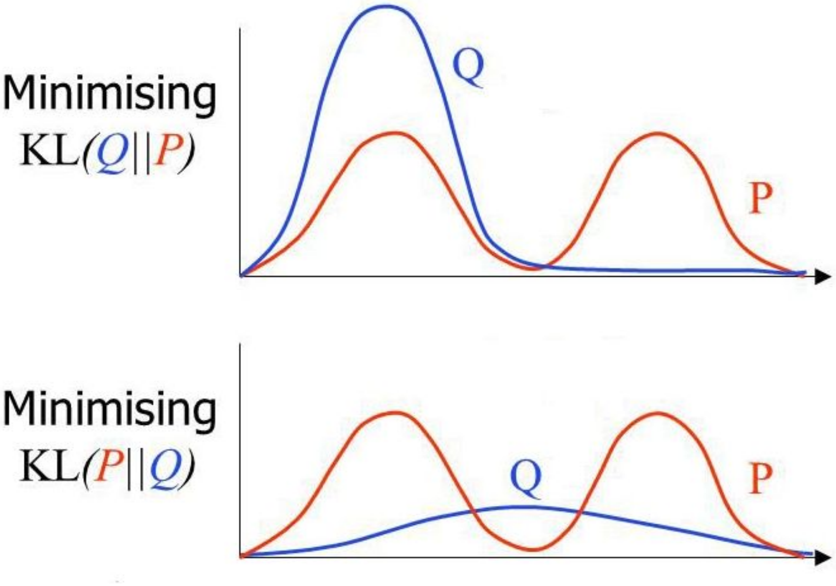
Number of observations

```
with pyro.plate("labels", n):
    obs=pyro.sample("obs",
        pdist.Categorical(logits=theta), obs=y)
```

Variational Inference

Automatic inference

$$\frac{\partial \theta}{\partial t} = \frac{\partial E_{kin}}{\partial p} = \frac{p}{m}$$
$$\frac{\partial p}{\partial t} = -\frac{\partial E_{pot}}{\partial \theta}$$



Sampling
Hamiltonian Monte Carlo / NUTS (2011)

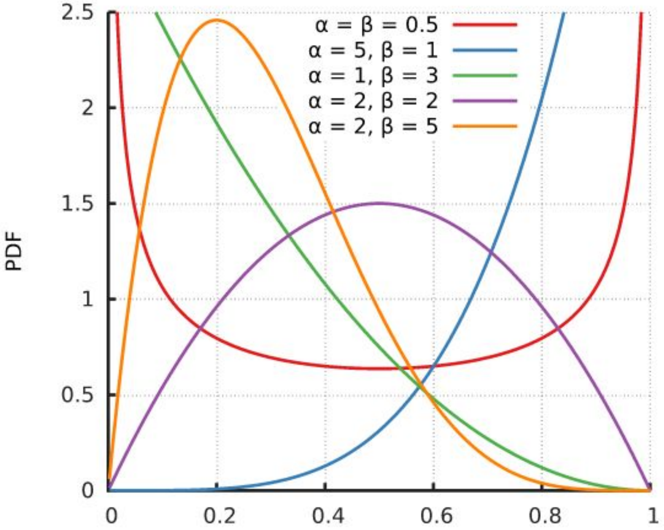
Optimisation
Stochastic Variational Inference (SVI)

Pictures: Mathieu Lê / John Winn, Microsoft Research Cambridge (adapted)

Coin flip problem

- Data: N coin flips, count vector **c**
    - 0=Heads (total count H), 1=Tails (T)
- What is the probability *h* of throwing heads?
    - We want a Bayesian estimate
    - Posterior distribution over *h*

$$p(h \mid \boldsymbol{c}) \propto p(\boldsymbol{c} \mid h)\pi(h)$$
$$= \prod_{i=1}^{n} \mathrm{Ber}(c_i \mid h)\pi(h)$$
$$= h^H (1-h)^T \mathrm{Beta}(h)$$



Evidence lower bound (ELBO)

- Maximizing ELBO=maximizing marginal likelihood p(**x**)
- **x**=data, **z**=latent variable, ϕ=guide parameters, θ=model parameters

Probabilistic model:

$$p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$$

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\
&= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})}\frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
&= \underbrace{\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})}\right]\right]}_{\substack{=\mathcal{L}_{\boldsymbol{\theta},\phi}(\mathbf{x}) \\ \text{(ELBO)}}} + \underbrace{\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))}
\end{aligned}$$

Picture: [Kingma & Welling, 2019]

Model / guide

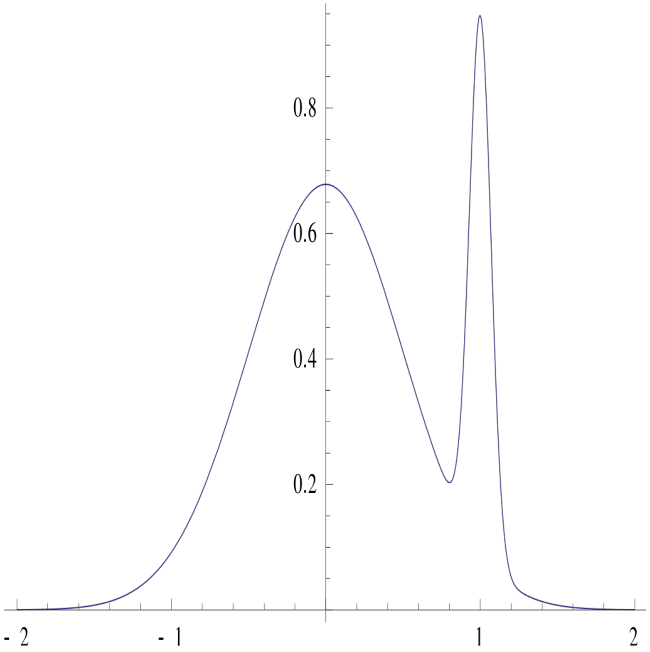
- The **model p(y,z)** is our probabilistic model
  - y=data, z=latent variable
- The **guide q(z)** is a simplified model that approximates the posterior
- One can use predefined **AutoGuides**
  - AutoDelta
  - AutoDiagonalNormal
  - AutoGuideList
- ...or your own guide
  - **pyro.param**

```
def model():
    ...
    pyro.sample("z", ...)

def guide():
    ...
    q=pyro.param("q", ...)
    pyro.sample("z", ...)
```

AutoDelta

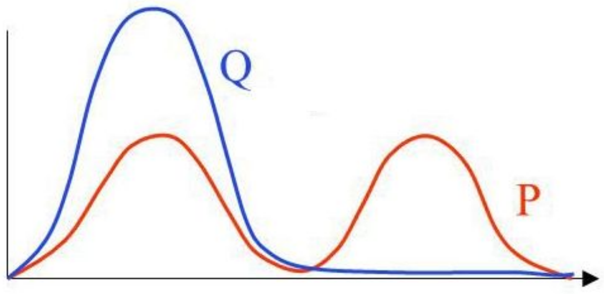
- AutoDelta provides a point estimate
  - MAP estimate
  - The guide is a **delta function**
- Depending on the shape of the posterior:
  - A good approximation
  - A disastrous simplification




```
# MAP estimate of model parameter
guide=pyro.infer.autoguide.AutoDelta(model)
```

AutoDiagonalNormal

- The posterior is approximated by independent Gaussian distributions
  - One for each parameter




```
# Variational estimate using diagonal normal
guide=pyro.infer.autoguide.AutoDiagonalNormal(model)
```

Inference

- The parameters of the guide (`pyro.param`) are iteratively optimized using stochastic variational inference (SVI)
  - **ELBO**: evidence lower bound
  - Lower bound to marginalized likelihood p(**x**)

```
adam=pyro.optim.Adam({"lr": 0.01})
svi=pyro.infer.SVI(model, guide, adam, loss=pyro.infer.Trace_ELBO())
...
for j in range(0, MAXIT):
    loss=svi.step(x, y)
```

pyro.infer.Predictive

- Once the guide's parameters (`pyro.param`) are optimized, we can use the guide for prediction.
    - Uses the approximate posterior predictive distribution

```
posterior_predictive=pyro.infer.Predictive(
    model,
    guide=guide,
    num_samples=S,
    return_sites=["obs"])(
x_test, None)
```

Passed to model

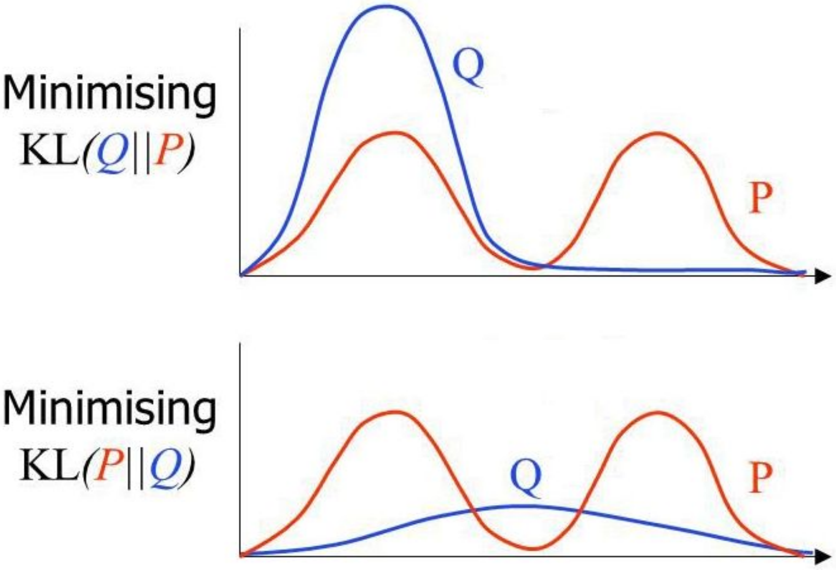
Coin flip SVI example

Colab coin flip SVI code

Full Bayesian inference with NUTS

Automatic inference

$$\frac{\partial\theta}{\partial t} = \frac{\partial E_{kin}}{\partial p} = \frac{p}{m}$$
$$\frac{\partial p}{\partial t} = -\frac{\partial E_{pot}}{\partial\theta}$$



Sampling
Hamiltonian Monte Carlo / NUTS (2011)

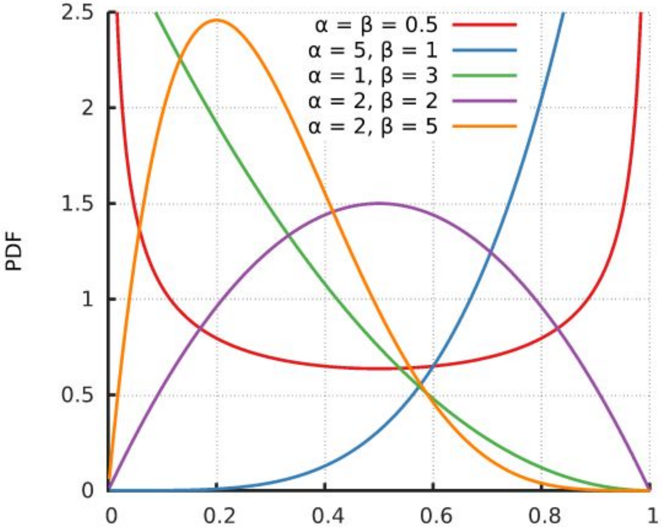
Optimisation
Stochastic Variational Inference (SVI)

Pictures: Mathieu Lê / John Winn, Microsoft Research Cambridge (adapted)

Coin flip problem

- Data: N coin flips, count vector **c**
    - 0=Heads (total count H), 1=Tails (T)
- What is the probability *h* of throwing heads?
    - We want a Bayesian estimate
    - Posterior distribution over *h*

$$
\begin{aligned}
p(h \mid \boldsymbol{c}) &\propto p(\boldsymbol{c} \mid h)\pi(h) \\
&= \prod_{i=1}^{n} \mathrm{Ber}(c_i \mid h)\pi(h) \\
&= h^H (1-h)^T \mathrm{Beta}(h)
\end{aligned}
$$



Monte Carlo & Bayes

- The Bayesian posterior is often unavailable as a closed-form expression.
- Monte Carlo methods approximate the posterior using samples.
  - Fast computers made this approach mainstream.
- The core idea is simple: approximate an expectation using **samples**.

$$\mathbb{E}\left[f(x)\right] = \int f(x)p(x)\mathrm{d}x \approx \frac{1}{S}\sum_{s=1}^{S} f(x_s)$$

Hamiltonian Monte Carlo

- The parameters of the model are interpreted as the **position of a particle** in a force field with added **momentum** and simulated using Hamilton's equations.

$$\frac{\mathrm{d}q}{\mathrm{d}t} = +\frac{\partial H}{\partial p} = \frac{\partial K}{\partial p}$$

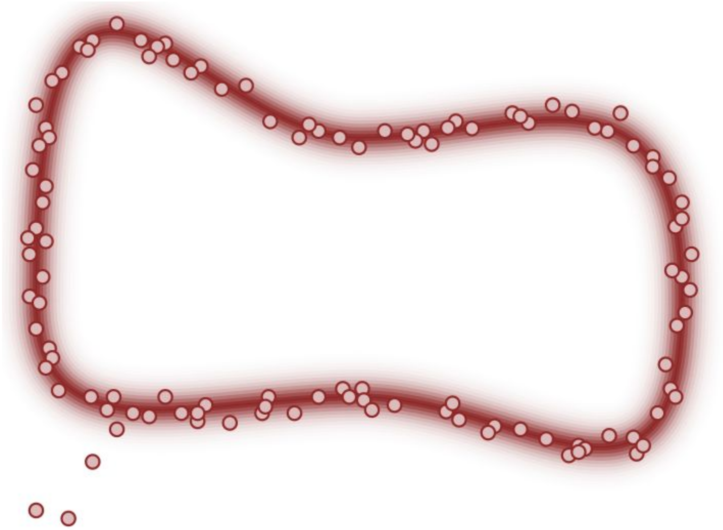
$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial H}{\partial q} = -\frac{\partial K}{\partial q} - \frac{\partial V}{\partial q}$$

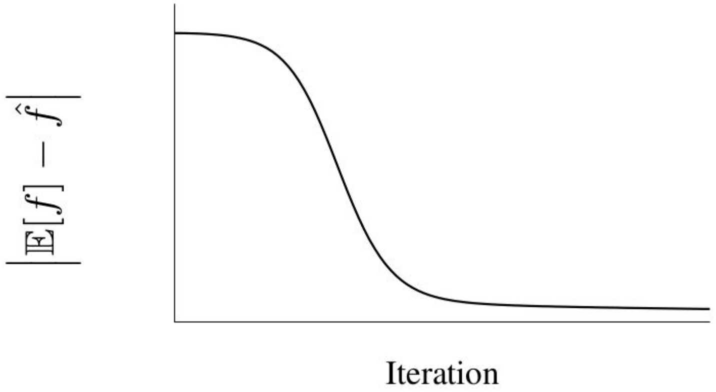
Sir William Rowan Hamilton
(1805 - 1865)

Pictures: M. Betancourt & Wikipedia

Markov chain Monte Carlo

- Samples approximate the posterior
  - **Typical set**





Picture: M. Betancourt

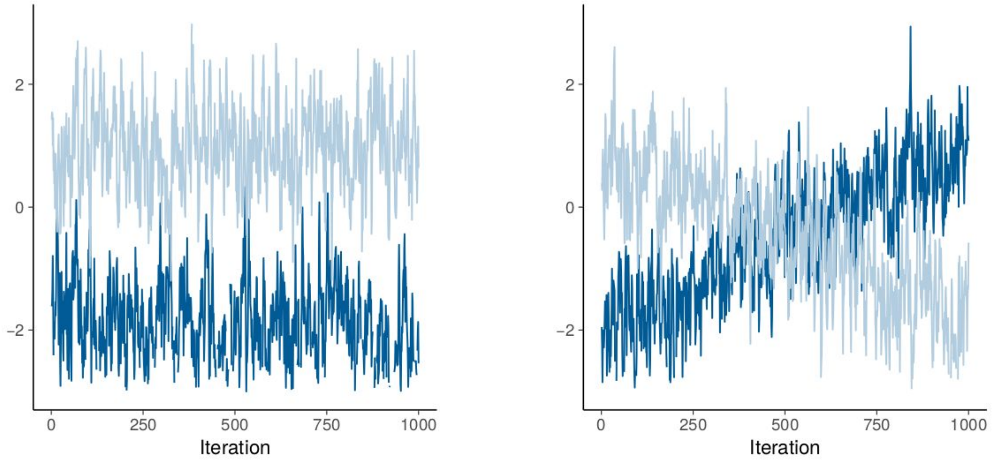
Pyro: MCMC with NUTS

- NUTS: No U-turn sampling
  - An automated version of Hamiltonian Monte Carlo

```
nuts_kernel=pyro.infer.NUTS(model, jit_compile=False)
mcmc=pyro.infer.MCMC(nuts_kernel, num_samples=S,
    num_chains=C, warmup_steps=W)
# Run on training set
mcmc.run(x, y)
```

Convergence diagnostics

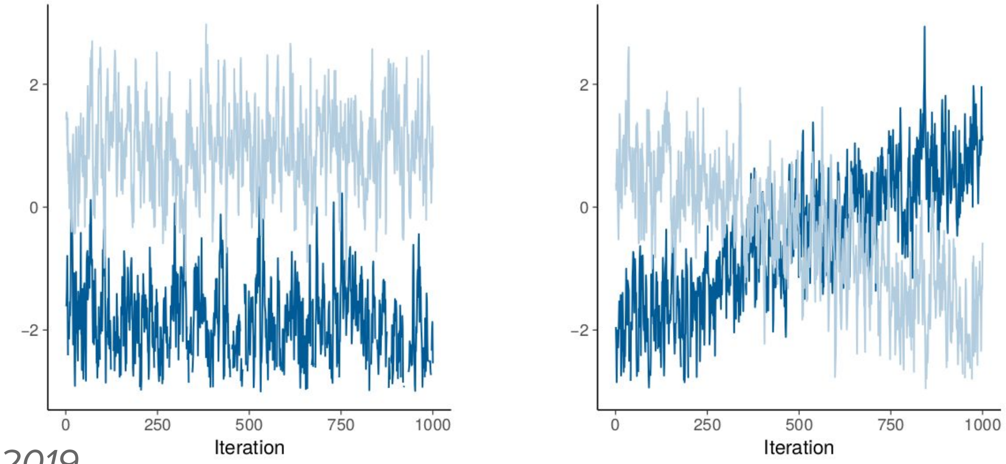
- MCMC is guaranteed to converge to the posterior for infinite samples.
  - But there are rarely any strong guarantees for finite samples.
- Diagnostics are needed, for example from running multiple chains.
  - **Trace plots**



Picture: Vehtari et al., 2019

Trace plots

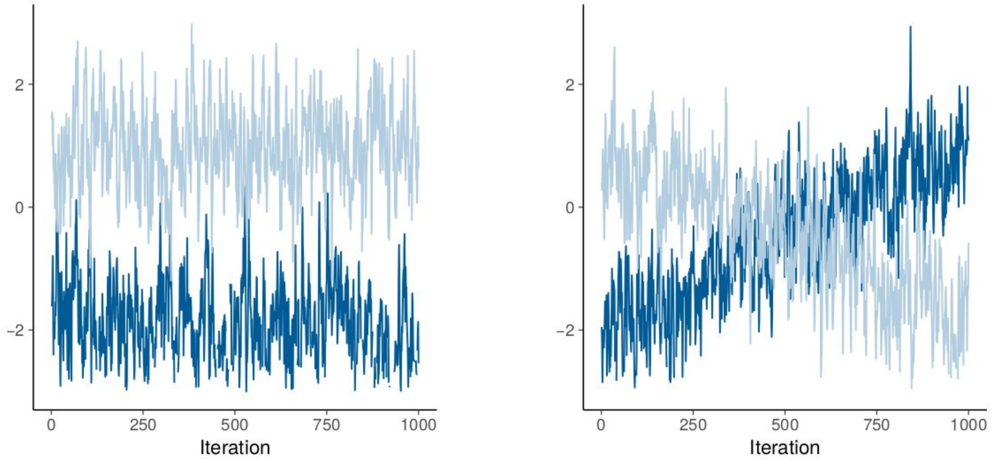
- **Left:** trace plots look stable, but did not converge to the same distribution.
- **Right:** trace plots are not stationary, though they seem to cover similar distributions.



Picture: Vehtari et al., 2019

Trace plots

- We need both **between-sequence** and **within-sequence** diagnostics.
- We can’t visually inspects trace plots of 1000s of variables.
  - We need **numerical summaries**.
  - Most common: R-hat (≅1), Effective sample size (ESS-bulk>400)



Picture: Vehtari et al., 2019

Diagnostics with Arviz

- https://arviz-devs.github.io/arviz/
- ArviZ is a Python package for **exploratory analysis of Bayesian models**.
- Includes functions for **posterior analysis**, data storage, **sample diagnostics**, **model checking**, and comparison.
- The goal is to provide **backend-agnostic tools** for diagnostics and visualizations of Bayesian inference in Python, by first converting inference data into xarray objects.

pyro.infer.Predictive

- We make predictions from the posterior samples
  - Again, using `pyro.infer.Predictive` like for SVI

```
posterior_samples=mcmc.get_samples()

posterior_predictive=pyro.infer.Predictive(
    model, posterior_samples)(
 x_test, None)

y_pred=posterior_predictive['obs']
```

Coin flip NUTS example

Colab coin flip NUTS code

Iris NN exercise

Iris data set

- **Fisher's Iris data set** is a multivariate data set introduced by the British statistician Ronald Fisher in 1936.

Iris Versicolor Iris Setosa Iris Virginica

Iris data set

- Based on (150) measurements of petal and sepal width and length (4-vector), **classify** the flower as Versicolor, Setosa or Virginica (3 classes)

Iris Versicolor　　Iris Setosa　　Iris Virginica

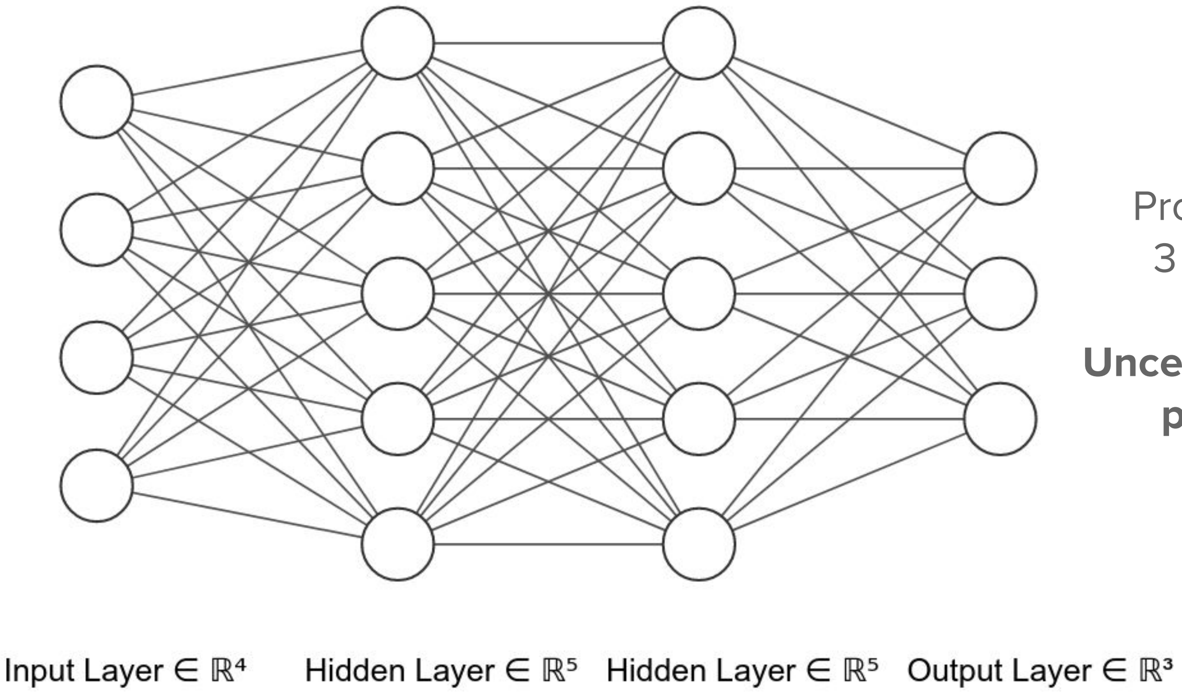
Iris classification problem

- x
  - Independent variable / input
  - 4 measurements of width and length of petals and sepals
  - Vector of 4 floats
- y
  - Dependent variable / output
  - What we want to predict
  - 3 iris classes: Versicolor, Setosa or Virginica
  - Categories: 0, 1, 2

Bayesian neural network for Iris

- 2 hidden layers, 5 neurons

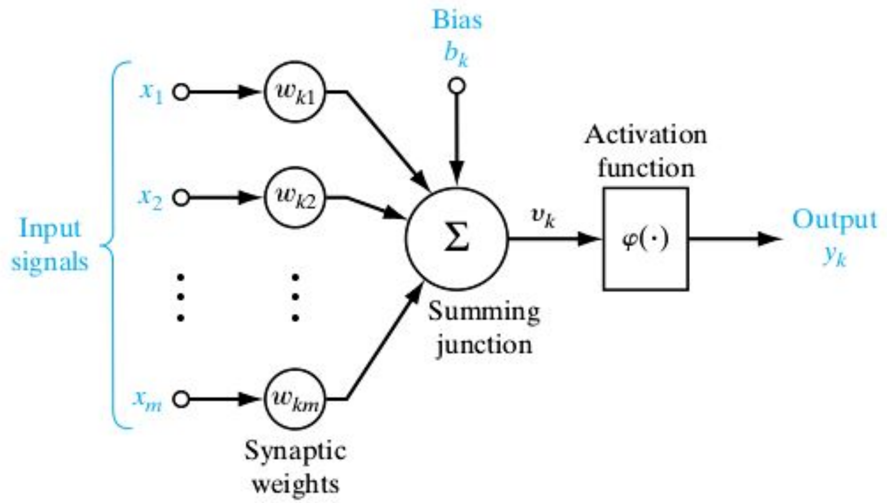
Petals & sepals

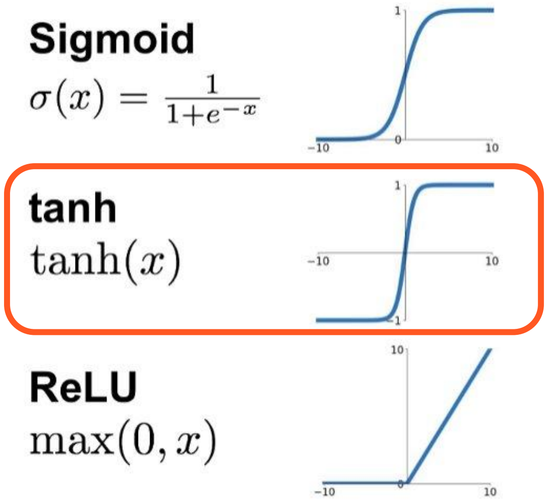


Probabilities of
3 iris classes
+
Uncertainty over all parameters

The humble digital neuron...

- Calculates the weighted sum of the inputs
- Applies a nonlinear function to the sum





Picture: The Men Who Stare at Codes/Shruti Jadon

Data handling

- We get the data from **sklearn**
- Map data to **torch.tensor**

```
iris=sklearn.datasets.load_iris()
x_all=torch.tensor(iris.data, dtype=torch.float)
y_all=torch.tensor(iris.target, dtype=torch.int)
```

Test set & training set

- We need to test our model on data not used in inference
  - Overfitting

```
# Make training and test set
x, x_test, y, y_test =
    sklearn.model_selection.train_test_split(
    x_all, y_all, test_size=0.33, random_state=42)
```

Evaluation

- For each input x_{TEST} in the test set, we sample a set of predictions y_{PRED}
- We use the logits for prediction
    - We need to register the deterministic variable

```
# NN
 h1=torch.tanh((x @ w1) + b1)
 h2=torch.tanh((h1 @ w2) + b2)
 logits=(h2 @ w3 + b3)
 # Save deterministc variable (logits) in trace
 pyro.deterministic("logits", logits)
 # Categorical likelihood
 with pyro.plate("labels", n):
     obs=pyro.sample("obs", pdist.Categorical(logits=logits), obs=y)
```

Iris SVI exercise

Colab iris SVI code

Iris SVI exercise I

- Identify the likelihood and the priors in the model.
- What is characteristic for the likelihood in the model code?
- Are the distributions of priors and likelihoods appropriate?
  - Note: consider the type of the distributions, not their parameters
- What does `pyro.plate` accomplish?
- Do we get a point estimate or a Bayesian posterior?
- Explain the shape of the posterior predictive tensor.
  - See output at the end.

Iris SVI exercise I: solution

- Identify the likelihood and the priors in the model.
  - **Categorical, Normal**
- What is characteristic for the likelihood in the model code?
  - **obs=y**
- Are the distributions of priors and likelihoods appropriate?
  - Note: consider the type of the distributions, not their parameters
  - **yes**
- What does `pyro.plate` accomplish?
  - **It specifies conditional independence**

Iris SVI exercise I: solution

- Do we get a point estimate or a Bayesian posterior?
  - **A point estimate (AutoDelta)**
- Explain the shape of the posterior predictive tensor.
  - See output at the end.
  - **torch.Size([500, 1, 50, 3])**
  - **500 = number of samples**
  - **1 = number of chains**
  - **50 = test set size**
  - **3 = logits**

Iris SVI exercise II

- Let’s try to improve the results.
  - The network has only two layers. Add a middle layer (layer 2).
- Are the parameters of the prior distributions appropriate?
- Evaluate the final prediction results.

Iris SVI exercise II: solution

- Let’s try to improve the results.
  - The network has only two layers. Add a middle layer (layer 2).
  - Solution code
- Are the parameters of the prior distributions appropriate?
  - **No, the standard deviation for the Normal priors on the weights was too high (100). Change it to 1.**
- Evaluate the final prediction results.
  - **From 0.7 to 0.9 accuracy**

Iris NUTS exercise (afternoon)

- Take the Iris-SVI model, make a copy, and modify it so we use NUTS instead of SVI for inference.
  - Use the coin flip NUTS implementation as an example to guide you.
  - Check the quality of the sampling with arviz.
- How is the posterior represented in SVI and NUTS?
- What are the advantages and disadvantages of SVI and NUTS?
- For the Iris problem, which method makes most sense and why?
  - Compare with the coin flip problem.

Conclusions

Pyro & probabilistic programming

- Formulate model
  - `pyro.sample`, `pyro.plate`,....
- Automatic inference
  - Stochastic variational inference (SVI)
  - Hamiltonian Monte Carlo (NUTS)
- From classic Bayesian to deep generative models
  - Variational autoencoders
  - Deep Gaussian processes
  - Deep Markov models
  - ...
- https://pyro.ai/examples/