Other Retrieval Methods

Simon Pfreundschuh

December 19, 2017

Overview

Background

- We still want to solve the retrieval problem:

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) \tag{1}$$

- Rodgers tells use how to do that if
  - $p(\mathbf{x}), p(\mathbf{y}|\mathbf{x})$ is Gaussian.
  - We have a(n at most *moderately non-linear*) foward model F
- But practically F should also
  - provide Jacobians,
  - not be too computationally complex.

Background

The Solutions

- Markov Chain Monte Carlo:
  - If you have a forward model and need the *full posterior distribution*.
- Database Retrievals:
  - (Bayesian) Monte Carlo Integration (BMCI)
  - Machine Learning
  - If performance is critical or if you don't have a forward model.

Markov Chain Monte Carlo

Advantages

- Direct sampling of the a posteriori distribution $p(\mathbf{x}|\mathbf{y})$
- No Jacobians required
- Not based on any assumptions (except a priori, of course)

Disadvantages

- Very Slow

Markov Chain Monte Carlo

- General method to sample from arbitrary distributions
- Sequential sampling: Next sample depends only on current state
  - *Markov chain property*
- Samples converge to target distribution

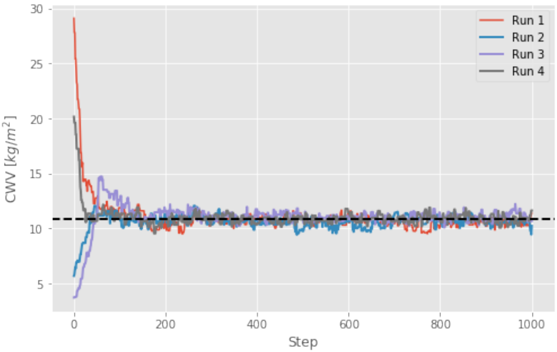
Basic Idea

- Sample *proposal state* from a proposal distribution (often a random walk).
- Accept or discard proposal depending on change in likelihood
- Repeat until convergence

Markov Chain Monte Carlo

A Retrieval Example

- Retrieval of integrated column water vapor from passive microwave observations
- 5 Channels: 23.8 GHz, 88.2 GHz, 165.5 GHz, 2×183.3 GHz
- Retrieve temperature and water vapor profiles



Markov Chain Monte Carlo

Let $p(\mathbf{x}|\mathbf{y})$ be the posterior we want to sample from and $J(\mathbf{x}_j|\mathbf{x}_i)$ a *symmetric proposal distribution*, i.e. satisfying $J_t(\mathbf{x}_j|\mathbf{x}_i) = J(\mathbf{x}_i|\mathbf{x}_j)$.

Metropolis Algorithm

1. Draw a starting point $\mathbf{x}_0$ with $p(\mathbf{x}_0|\mathbf{y}) > 0$
2. Iterate for $t = 1, \dots, n$:
   2.1 Sample a proposal $\mathbf{x}^*$ from the proposal distribution $J_t(\mathbf{x}|\mathbf{x}_{t-1})$.
   2.2 Calculate

$$r = \frac{p(\mathbf{x}^*|\mathbf{y})}{p(\mathbf{x}_{i-1}|\mathbf{y})} \tag{2}$$

   2.3 Set

$$\mathbf{x}_t = \begin{cases} \mathbf{x}^* & \text{with probability } \min(r, 1) \\ \mathbf{x}_{t-1} & \text{otherwise.} \end{cases} \tag{3}$$

Markov Chain Monte Carlo

Why does it work?

A Markov chain is guaranteed to have a *unique stationary distribution* if

- it is *aperiodic*,
- not *transient*,
  - i.e. there is no state that is not recurrent,
- *irreducible*,
  - i.e. there is no state for which there is a non-reachable state.

Thus only need to show that the stationary distribution is the posterior $p(\mathbf{x}|\mathbf{y})$.

Markov Chain Monte Carlo

Why does it work?

- Assume $p(\mathbf{x}_t|\mathbf{y}) = p(\mathbf{x}|\mathbf{y})$, i.e. the true posterior
- Consider the probability for a transition from $\mathbf{x}_{t+1}$ to $\mathbf{x}_t$:

$$p(\mathbf{x}_{t+1}, \mathbf{x}_t|\mathbf{y}) = p(\mathbf{x}_t|\mathbf{y}) J(\mathbf{x}_{t+1}|\mathbf{x}_t) \min\left(\frac{p(\mathbf{x}_{t+1}|\mathbf{y})}{p(\mathbf{x}_t|\mathbf{y})}, 1\right) \tag{4}$$

$$= \operatorname*{argmax}_{\mathbf{x}_t, \mathbf{x}_{t+1}} \{p(\mathbf{x}|\mathbf{y})\} J(\mathbf{x}_{t+1}|\mathbf{x}_t) \tag{5}$$

  - This is symmetric as well: $p(\mathbf{a}, \mathbf{b}) = p(\mathbf{b}, \mathbf{a})$
- Symmetry of the joint distribution implies equality of the marginal distributions:

$$p(\mathbf{x}_{t+1}|\mathbf{y}) = p(\mathbf{x}_t|\mathbf{y}) = p(\mathbf{x}|\mathbf{y}) \tag{6}$$

Markov Chain Monte Carlo

Things to Consider

- If the posterior probability of a proposed state is higher than that of the current state, the proposal is always accepted.
  - The state will move towards high posterior densities.
- Algorithm needs time to reach stationary distribution.
  - Samples from *warm up* phase must be discarded.
- Consecutive samples are not independent.
  - Keep only every nth sample

Bayesian Monte Carlo Integration (BMCI)

- MCMC is (conceptually) nice, but also inherently slow.
- BMCI uses a database of *precomputed simulations* or *observations*.

General Idea

- Use a database of pairs $(\mathbf{y}, \mathbf{x})$ of observations and known $\mathbf{x}$
- Use importance sampling to transform samples in database to samples of the posterior

Bayesian Monte Carlo Integration (BMCI)

Consider the expected value $\mathcal{E}_{\mathbf{x}|\mathbf{y}}\{f(\mathbf{x})\}$ of a function f computed with respect to the a posteriori distribution $p(\mathbf{x}|\mathbf{y})$:

$$\int f(\mathbf{x}')p(\mathbf{x}'|\mathbf{y})\,d\mathbf{x}' \tag{7}$$

Using Bayes theorem, the integral can be computed as

$$\begin{aligned}
\int f(\mathbf{x}')p(\mathbf{x}'|\mathbf{y})\,d\mathbf{x}' &= \int f(\mathbf{x}')\frac{p(\mathbf{y}|\mathbf{x}')p(\mathbf{x}')}{\int p(\mathbf{y}|\mathbf{x}'')\,d\mathbf{x}''}\,d\mathbf{x}' &(8)\\
&= \int f(\mathbf{x}')w(\mathbf{y},\mathbf{x})p(\mathbf{x}')\,d\mathbf{x}' &(9)\\
&= \mathcal{E}_{\mathbf{x}}\{f(\mathbf{x})w(\mathbf{y},\mathbf{x})\} &(10)
\end{aligned}$$

Bayesian Monte Carlo Integration (BMCI)

If the database is distributed according to our a priori assumtions, we can thus approximate any integral over the posterior distribution by:

$$\int f(\mathbf{x}')p(\mathbf{x}'|\mathbf{y})\, d\mathbf{x}' \approx \sum_{i=1}^{n} f(\mathbf{x}_i)w(\mathbf{y}, \mathbf{x}_i) \tag{11}$$

Bayesian Monte Carlo Integration (BMCI)

The Weighting Function

- Assuming the database is exact up to a zero-mean, Gaussian error with covariance matrix $\mathbf{S}_e$ the weighting function $w(\mathbf{y}, \mathbf{x})$ is given by:

$$w(\mathbf{y}, \mathbf{x}_i) = \frac{1}{C} \cdot \exp\left\{ -\frac{(\mathbf{y} - \mathbf{y}_i)^T \mathbf{S}_e^{-1} (\mathbf{y} - \mathbf{y}_i)}{2} \right\} \qquad (12)$$

with normalization factor C

$$C = \int w(\mathbf{y}, \mathbf{x}) d\mathbf{x} \approx \sum_{i=1}^{n} w(\mathbf{y}, \mathbf{x}_i) \qquad (13)$$

Bayesian Monte Carlo Integration (BMCI)

The Retrieval

- This can be used to retrieve the mean and variance of the posterior distribution:

$$\bar{x} = \mathcal{E}_{x|\mathbf{y}}\{x\} \approx \sum_{i=1}^{n} w(\mathbf{y}, x_i)x_i \tag{14}$$

$$\text{var}(x) = \mathcal{E}_{x|\mathbf{y}}\{(x - \bar{x})^2\} \approx \sum_{i=1}^{n} w(\mathbf{y}, x_i)(x_i - \mathcal{E}_{x|\mathbf{y}}\{x\})^2 \tag{15}$$

- Or even the CDF of the posterior:

$$F_{\mathbf{x}|\mathbf{y}}(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} p(\mathbf{x}')d\mathbf{x}' \tag{16}$$

$$\approx \sum_{\mathbf{x}_i < \mathbf{x}} w(\mathbf{y}, \mathbf{x}_i) \tag{17}$$

Bayesian Monte Carlo Integration (BMCI)

Things to Consider

- A very large database may be required to truthfully represent the a priori and provide sufficient a posteriori statistics.
  - Solution: Weighting/clustering of database samples
- Traversing the database can take quite some time.
  - Solution: Sorting the database in a smart way

Bayesian Monte Carlo Integration (BMCI)

Example: Global Precipitation Measurement (GMP) Retrieval

International satellite mission to provide next-generation observations of rain and snow worldwide every three hours.

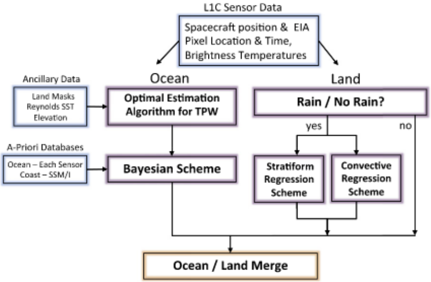


Figure: GPROF 2010 Retrieval Algorithm Flow (Kummerow et al. 2015).

Bayesian Monte Carlo Integration (BMCI)

Example: Global Precipitation Measurement (GMP) Retrieval

- Over Ocean:
  - Uses simulated database generated from profiles observed by the TRMM precipitation radar
  - Input: TBs, total precipitable water (TPR) from OEM, sea surface temperature (SST) from NWP
  - database with 65×10^6 entries stratified into SST/TPR bins of width 1 K / 1 mm.
  - Clustering algorithm used on bins to improve retrieval speed

Bayesian Monte Carlo Integration (BMCI)

Example: Global Precipitation Measurement (GMP) Retrieval

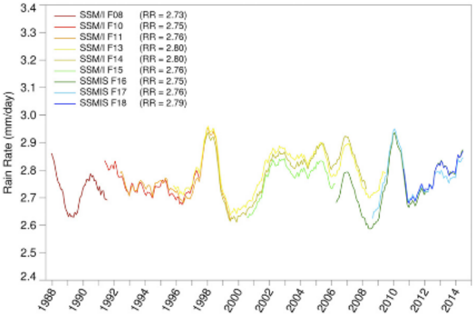


Figure: Trends in oceanic precipitation (Kummerow et al. 2015).

Bayesian Monte Carlo Integration (BMCI)

Example: Global Precipitation Measurement (GMP) Retrieval

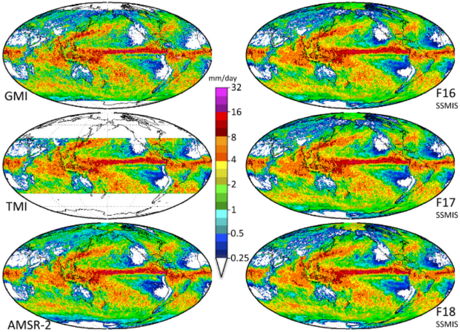


FIG. 10. April–June 2014 average mean surface precipitation of the GPM constellation conical sensors.

Figure: Precip from GPROF (Kummerow et al. 2015).

Machine Learning

Idea

- Try to learn the inverse method $\mathbf{x} = R(\mathbf{y})$ directly from the data.
- Regression is an old problem: Plenty of methods to choose from
  - Traditional regression analysis
  - Machine Learning

Neural Networks

- Universal estimators that compute a vector of output activations $\mathbf{y} = F_{NN}(\mathbf{x}_i, \mathbf{W}_i, \boldsymbol{\theta}_i)$ from a vector of input activations $\mathbf{x}$:

$$\mathbf{x}_0 = \mathbf{x} \tag{18}$$

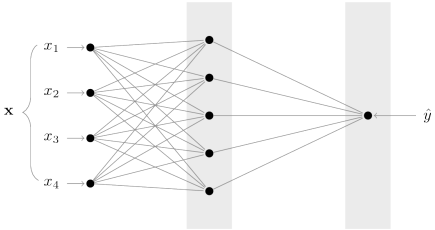
$$\mathbf{x}_i = f_i\left(\mathbf{W}_i \mathbf{x}_{i-1} + \boldsymbol{\theta}_i\right) \tag{19}$$

$$\mathbf{y} = \mathbf{x}_n \tag{20}$$

- Weight matrices $\mathbf{W}_i$ and bias vectors $\boldsymbol{\theta}_i$ are *learnable parameters* of the network.

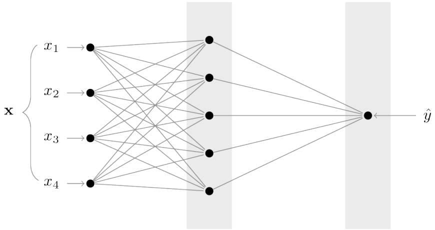
Machine Learning

Neural Networks



Machine Learning

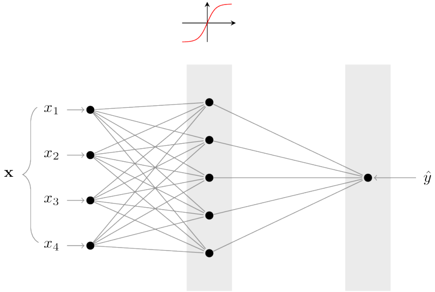
Neural Networks



$$\mathbf{u}_1 = \quad \mathbf{W}_1\mathbf{x} + \boldsymbol{\theta}_1$$

Machine Learning

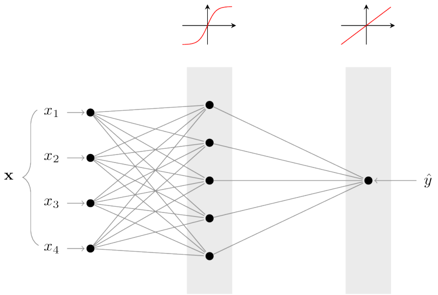
Neural Networks



$$\mathbf{u}_1 = f\left(\mathbf{W}_1 \mathbf{x} + \boldsymbol{\theta}_1\right)$$

Machine Learning

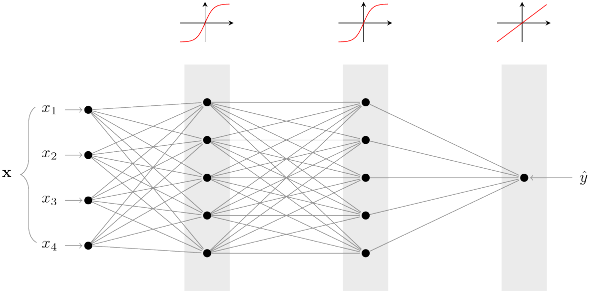
Neural Networks



$\mathbf{u}_1 = f\left(\mathbf{W}_1\mathbf{x} + \boldsymbol{\theta}_1\right)$ $\quad$ $\mathbf{u}_2 = f\left(\mathbf{W}_2\mathbf{u}_1 + \boldsymbol{\theta}_2\right)$

Machine Learning

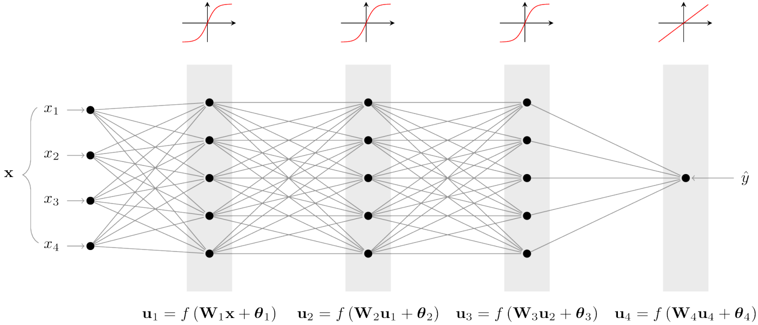
Neural Networks



$\mathbf{u}_1 = f\left(\mathbf{W}_1\mathbf{x} + \boldsymbol{\theta}_1\right)$ $\mathbf{u}_2 = f\left(\mathbf{W}_2\mathbf{u}_1 + \boldsymbol{\theta}_2\right)$ $\mathbf{u}_3 = f\left(\mathbf{W}_3\mathbf{u}_3 + \boldsymbol{\theta}_3\right)$

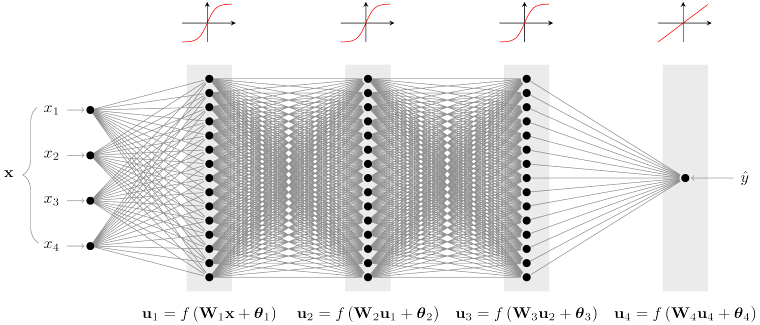
Machine Learning

Neural Networks



Machine Learning

Neural Networks



Machine Learning

Neural Networks

- (Not so) Recent Trends
  - Deep networks
  - End-to-end learning
- Deep Learning
  - Complex models, large amounts of data
  - Enabled through minibatch leaning (independence of dataset size) and fast (parallel) CPUs (GPUs)

Machine Learning

Neural Networks Training

- Supervised learning: Minimize mean of loss function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ over training set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$.

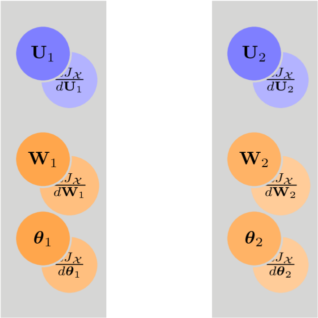
$$\underset{\mathbf{W}_i, \boldsymbol{\theta}_i}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{N} \mathcal{L}(F_{NN}(\mathbf{x}_i, \mathbf{W}_i, \boldsymbol{\theta}_i), \mathbf{y}_i) \tag{21}$$

- Use gradient information for efficient training
- Perform training on randomized minibatches (subsets of the training set)

Machine Learning

Neural Networks

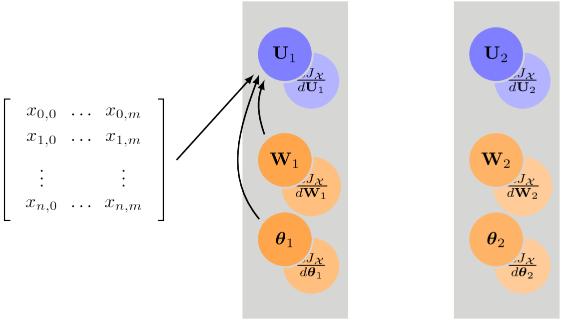
$$\begin{bmatrix} x_{0,0} & \cdots & x_{0,m} \\ x_{1,0} & \cdots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \cdots & x_{n,m} \end{bmatrix}$$



Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \theta_1\right)$$

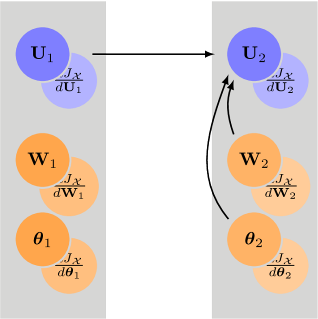


Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \theta_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$

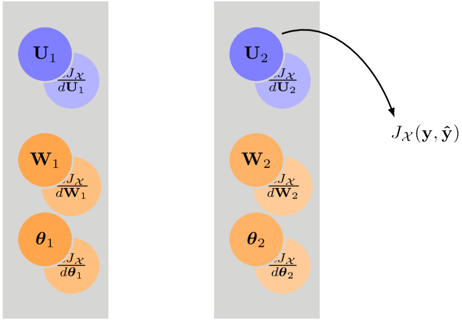


Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \cdots & x_{0,m} \\ x_{1,0} & \cdots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \cdots & x_{n,m} \end{bmatrix}$$

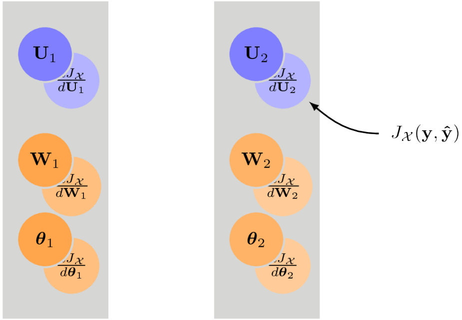


Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$

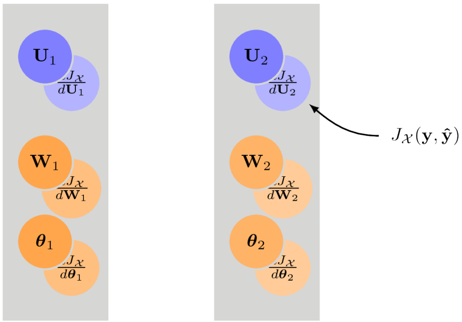


Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$

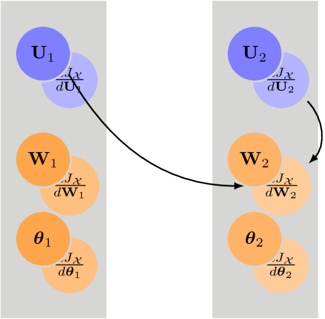


Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$



$$J_{\mathcal{X}}(\mathbf{y}, \hat{\mathbf{y}})$$

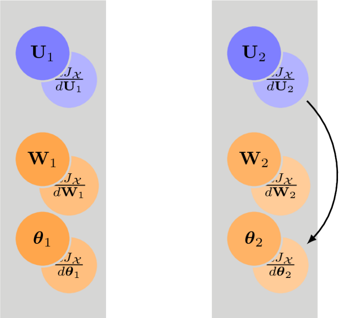
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_2} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)^T \mathbf{U}_1$$

Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$



$J_{\mathcal{X}}(\mathbf{y}, \hat{\mathbf{y}})$

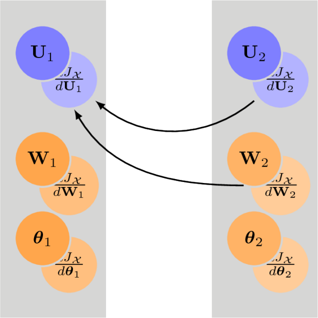
$$\frac{dJ_{\mathcal{X}}}{d\boldsymbol{\theta}_2} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)^T \mathbf{1}$$

Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$



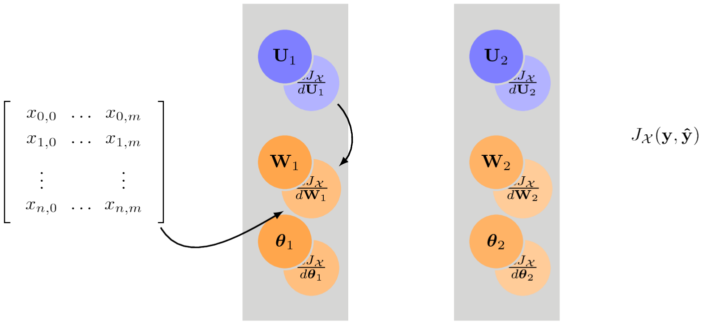
$$J_{\mathcal{X}}(\mathbf{y}, \hat{\mathbf{y}})$$

$$\frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)\mathbf{W}_2$$

Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

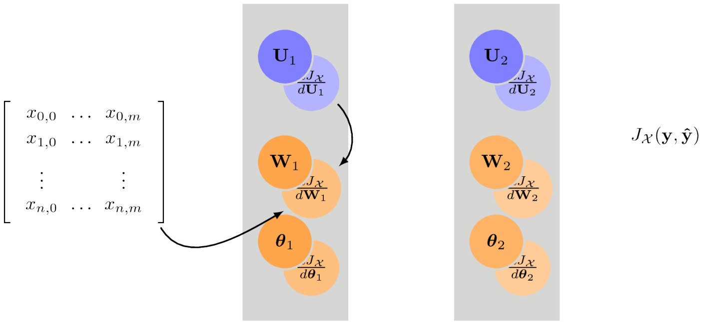


$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_1} = \left(\mathbf{f}_1' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1}\right)^T \mathbf{X}$$

Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$



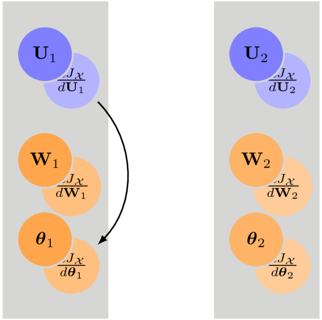
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_1} = \left(\mathbf{f}_1' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1}\right)^T \mathbf{X}$$

Machine Learning

Neural Networks

$$\mathbf{U}_1 = f_1\left(\mathbf{X}\mathbf{W}_1^T + \boldsymbol{\theta}_1\right) \qquad \mathbf{U}_2 = f_2\left(\mathbf{U_1}\mathbf{W}_2^T + \boldsymbol{\theta}_2\right)$$

$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$



$$J_{\mathcal{X}}(\mathbf{y}, \hat{\mathbf{y}})$$

$$\frac{dJ_{\mathcal{X}}}{d\boldsymbol{\theta}_1} = \left(\mathbf{f}_1' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1}\right)^T \mathbf{1}$$

Machine Learning

Neural Networks

Advantages

- Computational performance
  - Optimized CPU/GPU codes readily available
- Flexibility

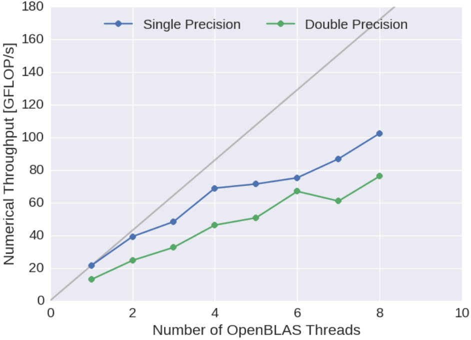
Disadvantages

- Need hyperparameter tuning for optimal performance
- More-or-less black box models

Machine Learning

Neural Networks Performance

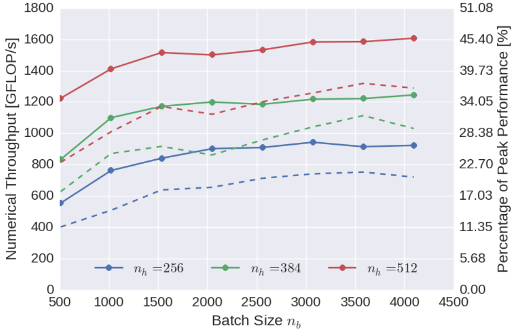
- Intel Xeon Processor E5-1680 v4



Machine Learning

Neural Networks Performance

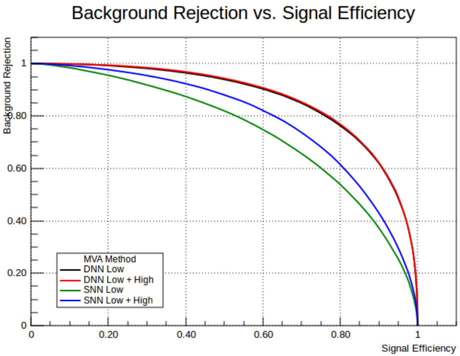
- NVIDIA Tesla K20 (Single Precision)



Machine Learning

Neural Network Example (Particle Physics)

- Neural network trained on simulated detector signals (momenta of decay products)
- Shallow (SNN) and deep (DNN) neural networks
- Trained with and without hand crafted high-level features (Low, High)



Machine Learning

Neural Networks

- Advantages:
  - Simple
  - Fast
    - Packages providing optimized code readily available
  - Flexible
- Disadvantages:
  - Need hyperparameter tuning for optimal performance
  - Black box model

My Current Research

Motivation

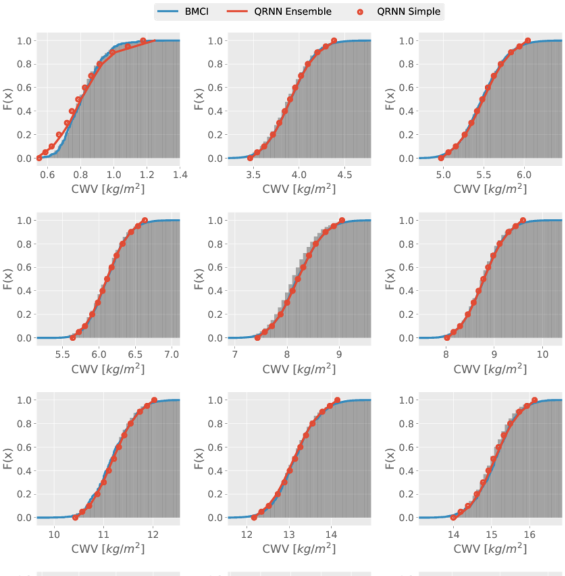
- Neural networks are nice but they usually only yield a single value $\mathbf{x}$ for the retrieval
- Is it possible to instead retrieve the (approximate) posterior $p(\mathbf{x}|\mathbf{y})$ using a neural network approach?
- Relevant for the retrieval of (frozen) hydrometeors from passive microwave observations

Approach

- Learn qunatiles of the posterior distribution (quantile regression)

My Current Research

Preliminary Results



References

Christian D. Kummerow et al. "The Evolution of the Goddard Profiling Algorithm to a Fully Parametric Scheme". In: *Journal of Atmospheric and Oceanic Technology* 32.12 (2015), pp. 2265–2280. DOI: 10.1175/JTECH-D-15-0039.1. eprint: https://doi.org/10.1175/JTECH-D-15-0039.1. URL: https://doi.org/10.1175/JTECH-D-15-0039.1.