Other Retrieval Methods

Simon Pfreundschuh

December 18, 2017

Overview

1. Background
2. Markov Chain Monte Carlo
3. Bayesian Monte Carlo Integration
4. Machine Learning

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
  - $p(\mathbf{x})$ 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

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*
- Sample converge to target distribution

Basic Idea

- Sample *propsal 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

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

Metropolis Algorithm

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

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

   2.3 Set

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

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 $p(\mathbf{x}_i|\mathbf{y}) = p(\mathbf{x}|\mathbf{y})$, i.e. the true posterior
- Then:

$$p(\mathbf{x}_i, \mathbf{x}_{i+1}|\mathbf{y}) = p(\mathbf{x}_i|\mathbf{y})J(\mathbf{x}_{i+1}, \mathbf{x}_i)\min(\frac{p(\mathbf{x}_{i+1}|\mathbf{y})}{p(\mathbf{x}_i|\mathbf{y})}, 1) \tag{4}$$

$$= \underset{\mathbf{x}_i, \mathbf{x}_{i+1}}{\operatorname{argmax}}\{p(\mathbf{x}|\mathbf{y})\}J(\mathbf{x}_{i+1}, \mathbf{x}_i) \tag{5}$$

  - 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:

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

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: *warm-up* phase
- Consecutive samples are not independent.

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:

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

with normalization factor C

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

Bayesian Monte Carlo Integration (BMCI)

The Retrieval

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

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

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

- 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.

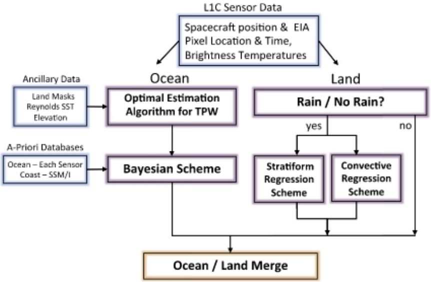


Figure: GPROF 2010 Retrieval Algorithm Flow (**gprof**).

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

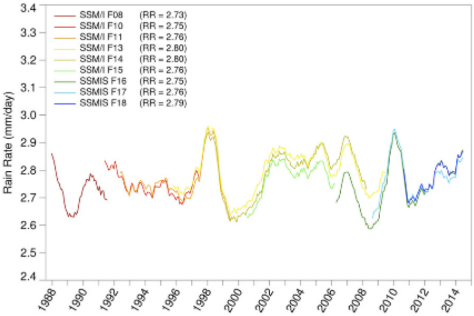


Figure: Trends in oceanic precipitation (**gprof**).

Bayesian Monte Carlo Integration (BMCI)

Example: Global Precipitation Measurement (GMP) Retrieval

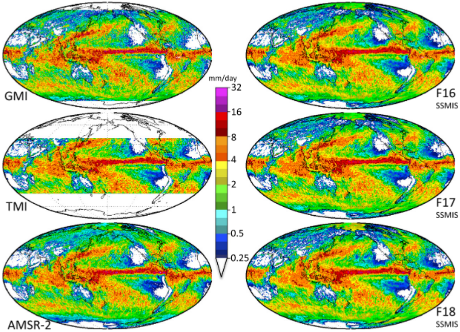


FIG. 10. April–June 2014 average mean surface precipitation of the GPM constellation conical sensors.

Figure: Precip from GPROF (**gprof**).

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 $\mathbf{y} = f_{NN}(\mathbf{x})$ from a vector of input activations $\mathbf{x}$:

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

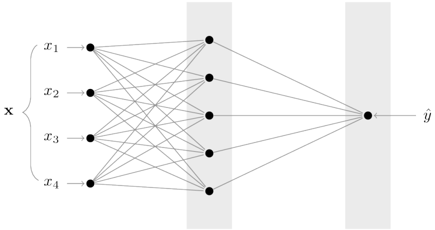
$$\mathbf{x}_i = f_i\left(\mathbf{W}_i\mathbf{x}_{i-1} + \theta_i\right) \tag{19}$$

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

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

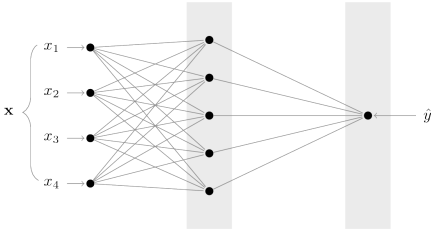
Machine Learning

Neural Networks



Machine Learning

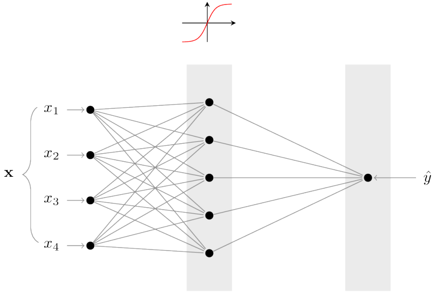
Neural Networks



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

Machine Learning

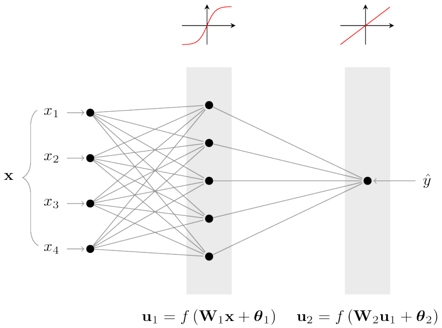
Neural Networks



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

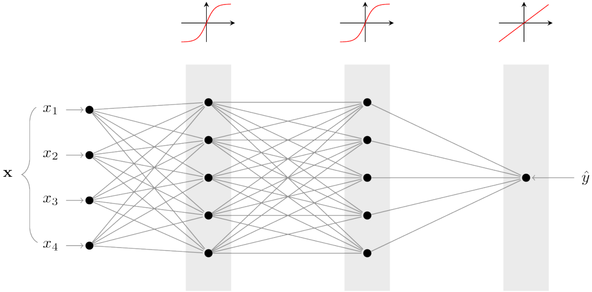
Machine Learning

Neural Networks



Machine Learning

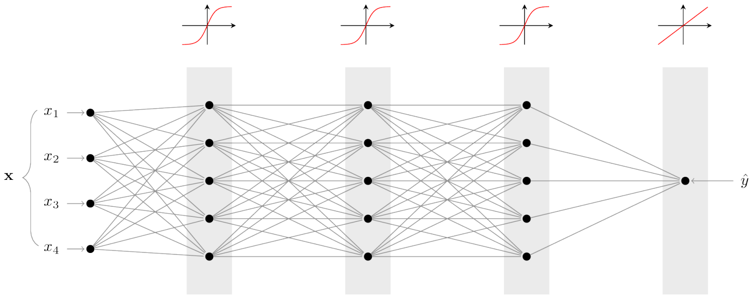
Neural Networks



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

Machine Learning

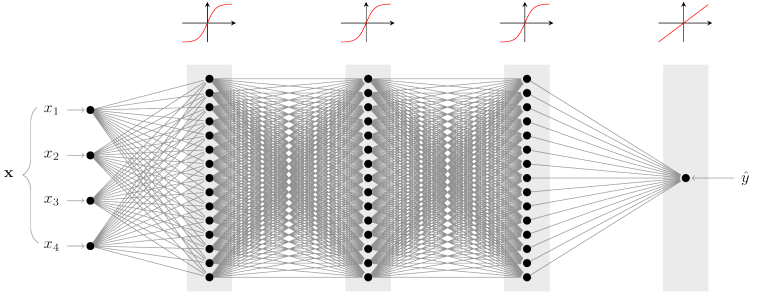
Neural Networks



$\mathbf{u}_1 = f\left(\mathbf{W}_1\mathbf{x} + \boldsymbol{\theta}_1\right)$ $\mathbf{u}_2 = f\left(\mathbf{W}_2\mathbf{u}_1 + \boldsymbol{\theta}_2\right)$ $\mathbf{u}_3 = f\left(\mathbf{W}_3\mathbf{u}_2 + \boldsymbol{\theta}_3\right)$ $\mathbf{u}_4 = f\left(\mathbf{W}_4\mathbf{u}_4 + \boldsymbol{\theta}_4\right)$

Machine Learning

Neural Networks



$$\mathbf{u}_1 = f\left(\mathbf{W}_1\mathbf{x} + \boldsymbol{\theta}_1\right) \quad \mathbf{u}_2 = f\left(\mathbf{W}_2\mathbf{u}_1 + \boldsymbol{\theta}_2\right) \quad \mathbf{u}_3 = f\left(\mathbf{W}_3\mathbf{u}_2 + \boldsymbol{\theta}_3\right) \quad \mathbf{u}_4 = f\left(\mathbf{W}_4\mathbf{u}_4 + \boldsymbol{\theta}_4\right)$$

Machine Learning

Neural Networks

(Not so) Recent Trends

- Deep networks
- End-to-end learning
- Very hot topic currently

Deep Learning

- Complex models, large amounts of data
- Enabled through batch leaning (independence of dataset size) and fast (parallel) CPUs (GPUs)

Machine Learning

Neural Networks

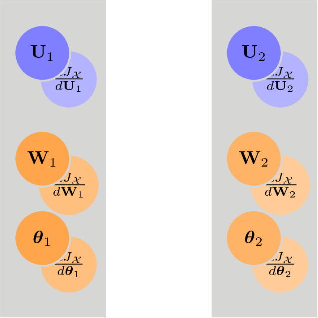
Training

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

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

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

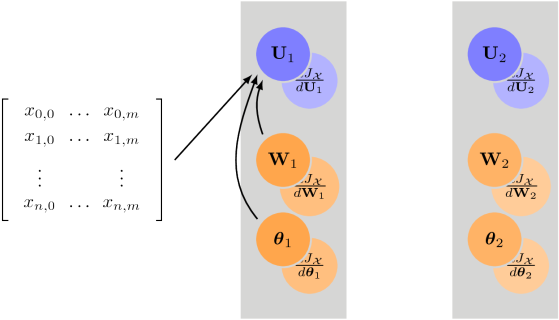
$$\begin{bmatrix} x_{0,0} & \dots & x_{0,m} \\ x_{1,0} & \dots & x_{1,m} \\ \vdots & & \vdots \\ x_{n,0} & \dots & x_{n,m} \end{bmatrix}$$



Machine Learning

Neural Networks

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

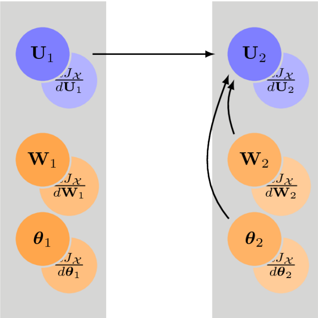


Machine Learning

Neural Networks

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

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

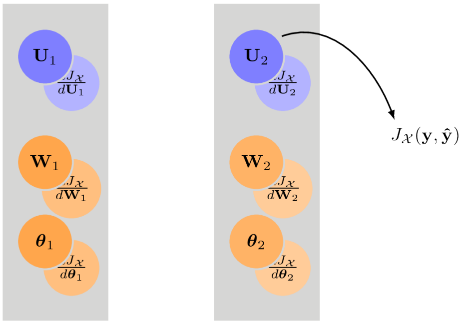


Machine Learning

Neural Networks

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

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

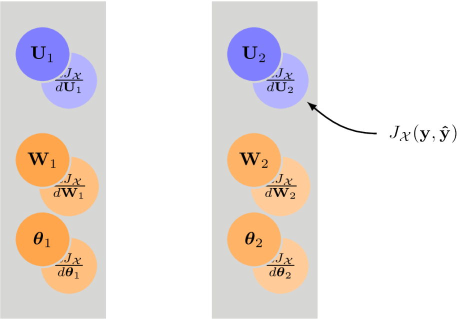


Machine Learning

Neural Networks

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

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

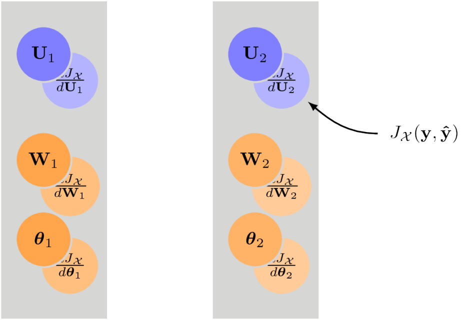


Machine Learning

Neural Networks

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

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

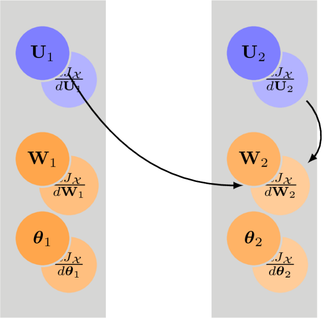


Machine Learning

Neural Networks

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

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



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

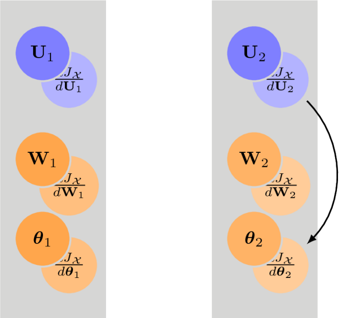
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_2} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)^T \mathbf{U}_1$$

Machine Learning

Neural Networks

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

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



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

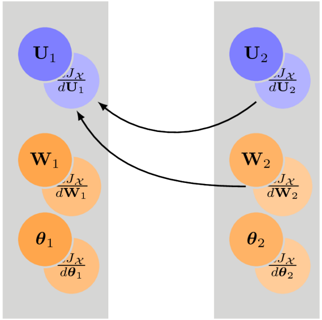
$$\frac{dJ_{\mathcal{X}}}{d\boldsymbol{\theta}_2} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)^T \mathbf{1}$$

Machine Learning

Neural Networks

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

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



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

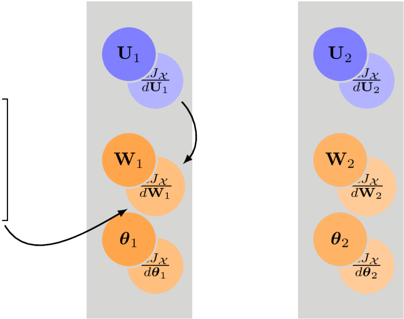
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1} = \left(\mathbf{f}_2' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_2}\right)\mathbf{W}_2$$

Machine Learning

Neural Networks

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

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



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

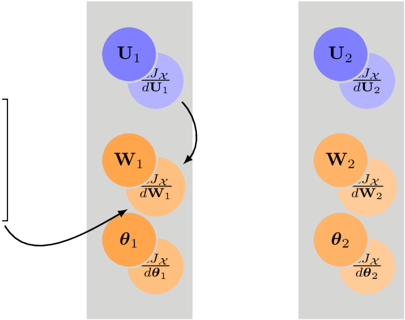
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_1} = \left(\mathbf{f}_1' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1}\right)^T \mathbf{X}$$

Machine Learning

Neural Networks

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

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



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

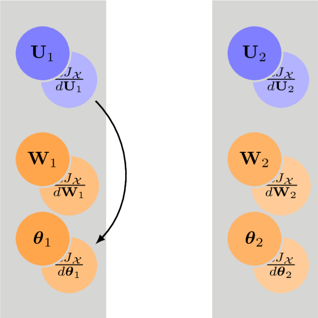
$$\frac{dJ_{\mathcal{X}}}{d\mathbf{W}_1} = \left(\mathbf{f}_1' \odot \frac{dJ_{\mathcal{X}}}{d\mathbf{U}_1}\right)^T \mathbf{X}$$

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

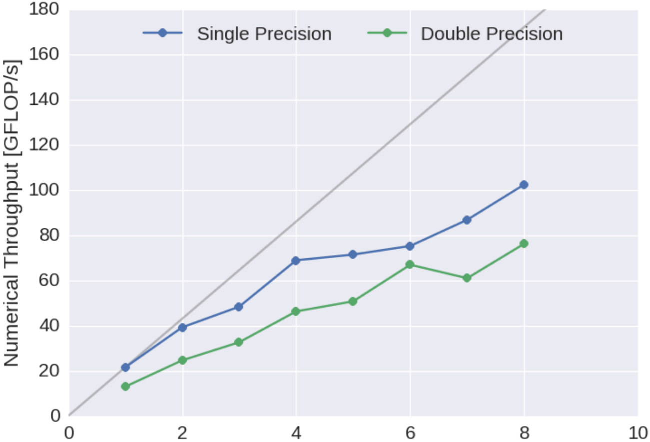
- Need hyperparameter tuning for optimal performance
- More-or-less black box models

Machine Learning

Neural Networks

NN Performance on Intel Xeon Processor E5-1680 v4

Example

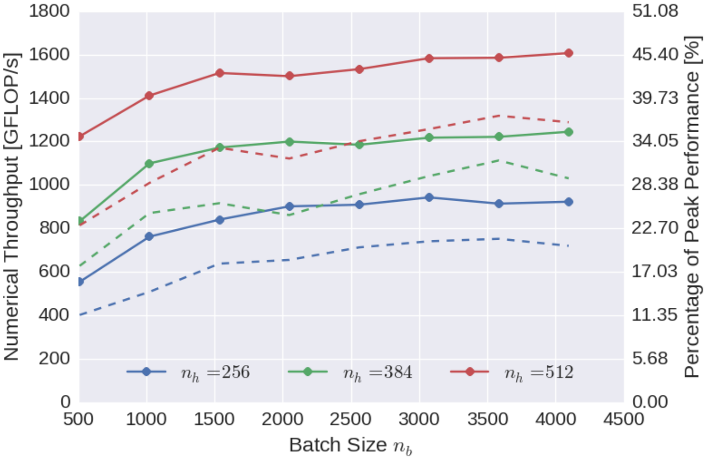


Machine Learning

Neural Networks

Example

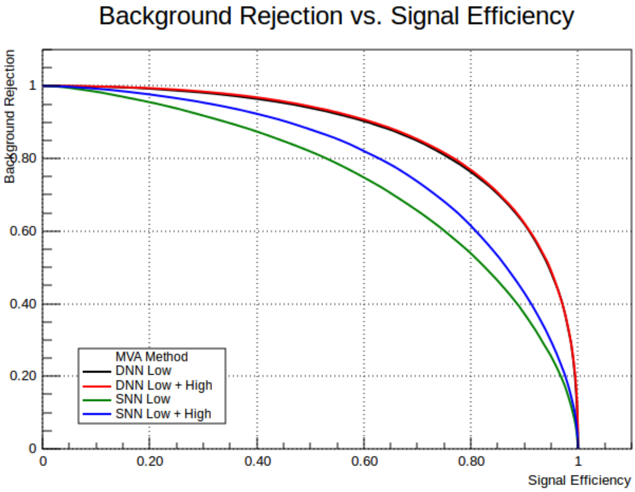
NN Performance on NVIDIA Tesla K20



Machine Learning

Neural Networks

Example



References