Fitting IceCube Ice Model Parameters with Gradient Descent

Alexander Harnisch
Astroparticle School 2018
October 6, 2018
Introduction

- IceCube detects neutrinos by measuring the Cherenkov light emitted by secondary particles.
- For that we need precise simulations of photon propagation within the ice.
- For modeling the ice we divide it into layers in $z$ direction.
- There are global ice model parameters and parameters for each layer, mainly scattering and absorption coefficients.
- There is a flasher board on each DOM which we use as light sources for calibration.
Motivation

- Fitting ice parameters by performing iterative grid searches is extremely complex and time-consuming.
- We want to compute gradients of the likelihood, to be able to perform gradient descent instead of grid searches.
- This might reduce the cost and improve scalability.
- For now we focused on bulk ice absorption coefficients.
- We should at least be able to verify the current best fit using this independent method.
General Idea

Data → Loss → Simulation → Fitted Parameters

Gradient → Optimizer

Challenge: Making the red part differentiable.
Photon Propagation Simulation - Before

1. Initialize photons
2. While not hit or absorbed
   2.1 Propagate photon
      - Hit? Stop propagation at DOM contact point
      - Absorbed? Stop propagation at point of absorption
      - Otherwise propagate to next scattering point
   2.2 Scatter photon, if not hit or absorbed
3. Output: DOM responses

See pseudo code on backup-slide 23
Photon Propagation Simulation - Modified to Allow Differentiation

1. Initialize photons
2. While not hit or cut-off reached
   2.1 Propagate photon
      - Hit? Stop propagation at DOM contact point
      - Cut-off reached? Stop propagation
      - Otherwise propagate to next scattering point
      - Log traveled distance in each layer
   2.2 Scatter photon, if not hit or cut-off
3. Intermediate output for each hit: The DOM that was hit and the traveled distance in each layer

Cut-off based on total travel distance and ignore all photons, that were cut-off.
Calculate DOM Response from Intermediate Result

**Input:** Photon travel distances $D_j$ for each layer $j$ and which DOM was hit for each Photon, that hit a DOM.

Calculate the probability for each photon to reach the DOM, after propagation:

$$p_{\text{Hit}} = 1 - p_{\text{Absorbed}} = \exp \left( - \int_0^D a(x) \, dx \right)$$

(1)

where $a_j$ is the absorption coefficient of the $j$-th layer.

**Output:** Summed hit probabilities of all photons for each DOM

$$s_i = \sum_{j=1}^{N_{\text{Hits},i}} p_{\text{Hit},ij}$$

(2)
Getting the Gradient

The loss \( F \) we are trying to minimize is the negative logarithm of the likelihood ratio

\[
-F = \log \mathcal{L}_{\text{Ratio}} .
\]

(3)

We now have a direct chain between a suitable loss function \( F \) and absorption coefficients without any control statements, which we can use to compute the gradient

\[
\nabla_{\vec{a}} F = \cdots
\]

(4)

by using TENSORFLOW’s automatic differentiation (backpropagation). We could also define the gradient by hand, which would be simple but tedious.
Fitting Algorithm

- "True" $a_i$
- PPC
- "Data" Hits
- Choose Flashers
- Modified PPC
- Expected Hits
- Loss
- Fitted $a_i$
- Gradient
- Optimizer

Later: Data

Node
- Forward data flow
- Backward propagation
- Differentiation Node

October 6, 2018
Videos of the Learning Progress

Hyperlinks to videos:

- Initial coefficients all set to $0.008 \text{ m}^{-1}$
  - 10 times faster
- Initial coefficients sampled uniformly between $0.005 \text{ m}^{-1}$ and $0.03 \text{ m}^{-1}$
  - 10 times faster
Fit Stability Test

- Idea: Fit multiple times with random initial values to check for stability
- 8 runs with initial coefficients sampled uniformly between $0.005 \, m^{-1}$ and $0.03 \, m^{-1}$
- The following plots show the mean and standard deviation of the mean of those 8 fit results
Fit Stability Test

- True
- Fitted

Absorption Coefficient / m$^{-1}$

Detector z / m
Fit Stability Test

Absorption Coefficient / m

Detector z / m

1900 1950 2000 2050 2100 2150 2200

October 6, 2018
Conclusion and Outlook

Seems to be working! We should at least be able to verify the current fit with this independent approach.

Problems:

- Uncertainty on LED light output for real data
- Performance: Copying stuff around GPU and CPU memory. Need for general optimization

Ideas:

- Scattering: Many possible approaches
  - Assuming a strong correlation between absorption and scattering
  - Hybrid between grid search and gradient descent approach
  - Estimating the gradient for scattering coefficients, e.g. by resampling the arrival time for each photon
- Anisotropy: Tessellated sphere idea proposed by Martin Rongen
- Possibly do hardware simulation
- Use timing information

Next? Moving on towards using real data and incorporate scattering.
Test Setup

- Two separate PPC executables, one without absorption and the other one unmodified to generate fake data to fit to
- Using the current 3.2 best fit values to generate the fake data
- The scattering coefficients are fixed to the “True” values for the fit
- Anisotropy is disabled
- DOM-Oversizing of 15, should be fine since we don’t use arrival time information yet
- All photons have the same wavelength of 400 nm (PPC WFLA=400)
- Flashing all DOMs on string 36 (inside deep core) to generate batches
- Emitting $1.5 \times 10^7$ photons for each batch
- All following fits were done on the same fake dataset which consists of 347 batches
Fit Stability Test

Detector z / m

October 6, 2018
The Problem with Differentiating the Propagation Loop

- Automatic differentiation works great for arbitrarily complex programs and can deal with control statements.
- There is one caveat: Those control statements must not depend on the target parameters of the differentiation.
  - In that case the structure of the program changes depending on the target parameter, information about other branches of the program is not included in a mathematical derivative of the one branch that was executed.
  - It can still work in some cases (e.g. our first results) but generally there is no guarantee for the gradient to point in the right direction.
The Problem - Super Simple Example

Consider the following program:

```python
def y(x):
    if x == 2:
        return 4
    else:
        return 2*x
```

- In case of $x \neq 2$ automatic differentiation will provide the correct derivative of $\frac{\partial y}{\partial x} = 2$. In case of $x = 2$ the derivative is 0, but we need it to be 2 in that case as well.
- Automatic differentiation simply evaluates the chain rule along the executed branch of the program. This is only fine as long as the branching does not depend on target parameters.
- In this case the problem could be easily resolved by defining the gradient manually or rewriting the program.
The Problem - Our Case

Our case looks like this (simplified pseudo code):

```python
def simulate(photon, l_abs, l_scat):
    d_abs = sample_absorption(l_abs)
    while d_abs > 0 and not hit:
        d_scat = sample_scattering(l_scat)
        hit = check_for_hits(photon, d_scat)
        if hit:
            propagate_to_hit(photon)
        else:
            if d_abs - d_scat > 0:
                propagate(photon, d_scat)
                scatter(photon)
                d_abs -= d_scat
            else:
                propagate(photon, d_abs)
                d_abs = 0
```

- During forward propagation TensorFlow counts the number of loop iterations. When backpropagating the number of loop iterations is therefore a constant.
- This means the gradient “does not know” that changing a scattering or absorption length a little bit would lead to one more/less scattering process, even though the derivative of a single scattering process is correct.
- More on this is explained in the reference given on slide 24 and in a demonstration we wrote.
Tensorflow While Loop - Problem

Problem: $N$ is a constant during backprop. $P$ must not depend on target variables, but it does in our case.

Automatic Differentiation - Introduction

- Set of techniques to evaluate derivatives of functions given by computer programs
- Every computer program can be broken down to a number of basic mathematical operations
- If the derivative of those operations is known, we can use the chain rule to get the derivative of the entire program, which is the basic idea of AD
- It is not symbolic nor numerical differentiation (often confused)
- Can be applied to arbitrarily complex functions, like simulations
Automatic Differentiation - Forward and Reverse Mode

- Two possible approaches: Forward and reverse mode
- Essentially the direction in which we apply the chain rule
- Reverse mode is divided into two phases: forward pass and backward pass
- Backward pass is often called backpropagation in machine learning
- We use TENSORFLOW, which uses reverse mode AD

\[
\frac{\partial f_n}{\partial x} = \frac{\partial f_0}{\partial x} \frac{\partial f_1}{\partial f_0} \cdots \frac{\partial f_n}{\partial f_{n-1}}
\]
Automatic Differentiation Example - Computational Graph

Let’s look at a simple example

\[ y = f(x_1, x_2) = 3x_1x_2 + \ln^2(x_2) \]  

The *computational graph* looks like this
Automatic Differentiation Example - Node Derivatives

We know the derivatives of every node:

\[ y = c + d \]

\[ \frac{\partial y}{\partial c} = 1 \quad \frac{\partial y}{\partial d} = 1 \]

\[ c = 3a \]

\[ \frac{\partial c}{\partial a} = 3 \]

\[ d = b^2 \]

\[ \frac{\partial d}{\partial b} = 2b \]

\[ a = x_1 \cdot x_2 \]

\[ \frac{\partial a}{\partial x_1} = x_2 \quad \frac{\partial a}{\partial x_2} = x_1 \]

\[ b = \ln(x_2) \]

\[ \frac{\partial b}{\partial x_2} = x_2^{-1} \]

\[ x_1 \quad x_2 \]
Automatic Differentiation Example - Forward Pass

Let’s evaluate the gradient for $x_1 = 2$ and $x_2 = e^2$ with reverse mode AD. First we perform the **forward pass** and save all the intermediate results:

\[
\begin{align*}
    y &= c + d \\
    c &= 3a \\
    a &= x_1 \cdot x_2 \\
    x_1 &= 2 \\
    x_2 &= e^2 \\
    d &= b^2 \\
    b &= \ln(x_2) \\
    c &= 6e^2 \\
    c &= 6e^2 + 4 \\
    d &= 4 \\
    a &= 2e^2 \\
    b &= 2 \\
\end{align*}
\]
Automatic Differentiation Example - Backward Pass

To obtain the gradient \((\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2})^T\) we apply the chain rule by traversing the graph in reverse order:

\[
\begin{align*}
y &= 6e^2 + 4 \\
c &= 6e^2 \\
a &= 2e^2 \\
b &= 2 \\
d &= 4
\end{align*}
\]

As expected by doing this we evaluate chain rule terms like \(\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial c} \frac{\partial c}{\partial a} \frac{\partial a}{\partial x_1}\) from outside to inside.