Inverse Volume Rendering

Absorption + Emission, Automatic Differentiation

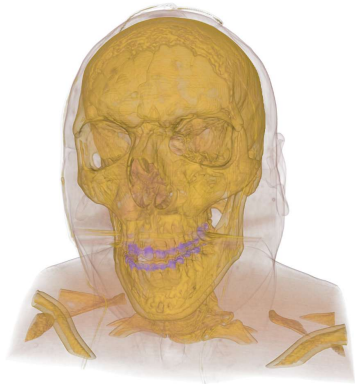
Sebastian Weiß

Lehrstuhl für Computergrafik
Technische Universität München

November 2020

Overview

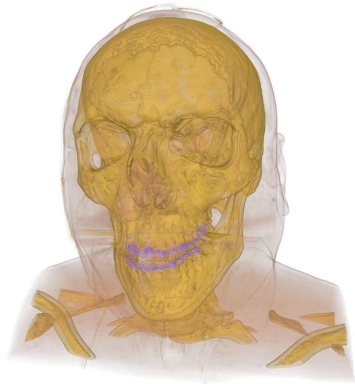
1. Models and Assumptions
   - 1.1. Absorption
   - 1.2. Emission
   - 1.3. Transfer Function



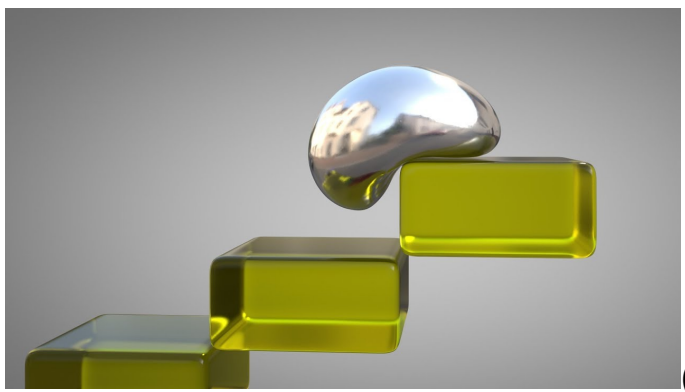
Overview

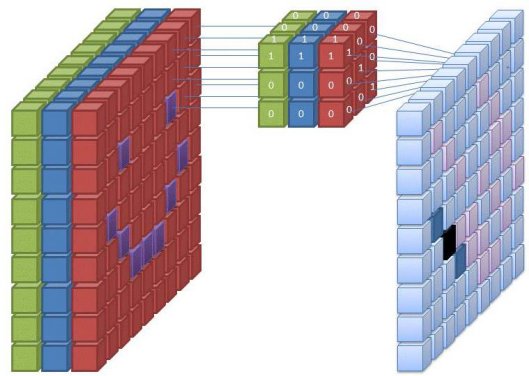
1. Models and Assumptions
   - 1.1. Absorption
   - 1.2. Emission
   - 1.3. Transfer Function





2. Automatic Differentiation
   - 2.1. Forward Mode
   - 2.2. Backward Mode

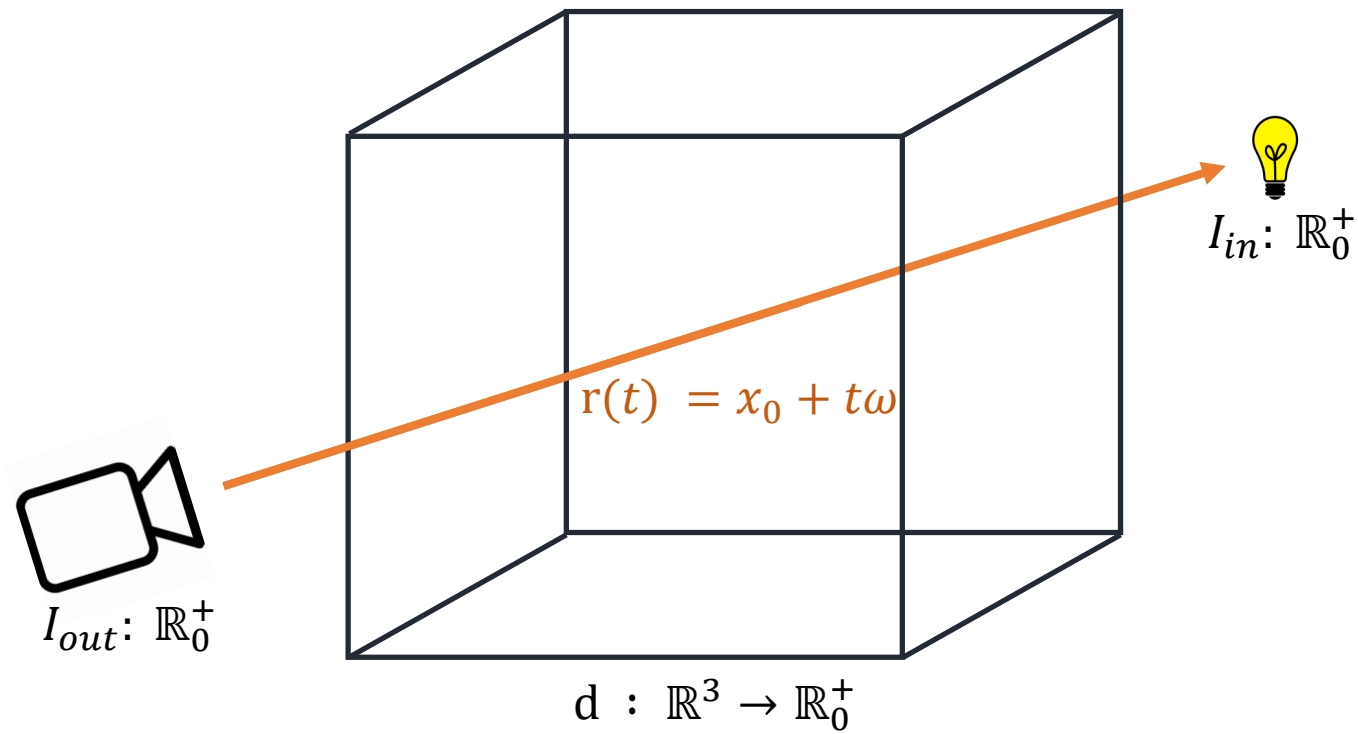

(1)


(2)

(1) https://www.youtube.com/watch?v=HwJyg5o3HGg
(2) https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network_with_Color_Image_Filter.gif

1. Models and Assumptions

1. Scalar Field $d : \mathbb{R}^3 \to \mathbb{R}_0^+$ of densities
2. Ray $r(t) = \boldsymbol{x}_0 + t\boldsymbol{\omega}$, $\boldsymbol{x}_0 \in \mathbb{R}^3, \boldsymbol{\omega} \in S^2$
3. Received Illumination $I_{\text{out}}(\boldsymbol{x}_0, \boldsymbol{\omega}) \in \mathbb{R}_0^+$



1.1 Only Absorption

Assume a medium that only absorbs incoming light, no emission or scattering.

$$I_{\text{out}}(\boldsymbol{x}_0, \boldsymbol{\omega}) = \int_0^{\infty} g(t) \exp\left(-\int_0^t \tau(v(r(t')))dt'\right) dt$$

$$\text{with } g(s) = c(s)\tau(s) + \int_{\Omega} p(\boldsymbol{\omega}, \boldsymbol{\omega}') I_{\text{out}}(r(t), \boldsymbol{\omega}')d\boldsymbol{\omega}'$$

Nelson Max. Optical Models for Direct Volume Rendering. *TVCG* 1995
Kajiya, von Herzen. Ray Tracing Volume Densities. *Computer Graphics* 1984

1.1 Only Absorption

Assume a medium that only absorbs incoming light, no emission or scattering.

$$I_{\text{out}}(\boldsymbol{x}_0, \boldsymbol{\omega}) = \int_0^{\infty} g(t) \exp\left(-\int_0^{t} \tau(v(r(t')))dt'\right) dt$$

$$\text{with } g(s) = c(s)\tau(s) + \int_{\Omega} p(\boldsymbol{\omega}, \boldsymbol{\omega}') I_{\text{out}}(r(t), \boldsymbol{\omega}')d\boldsymbol{\omega}'$$

Set $g(\infty) = I_{\text{in}}$ and zero everywhere else (only absorption).
Set $\tau(d) = d$

Nelson Max. Optical Models for Direct Volume Rendering. *TVCG* 1995
Kajiya, von Herzen. Ray Tracing Volume Densities. *Computer Graphics* 1984

1.1 Only Absorption

Assume a medium that only absorbs incoming light, no emission or scattering.

$$I_{\text{out}}(\boldsymbol{x}_0, \boldsymbol{\omega}) = \int_0^{\infty} g(t) \exp\left(-\int_0^{t} \tau(v(r(t')))dt'\right) dt$$

$$\text{with } g(s) = c(s)\tau(s) + \int_{\Omega} p(\boldsymbol{\omega}, \boldsymbol{\omega}') I_{\text{out}}(r(t), \boldsymbol{\omega}')d\boldsymbol{\omega}'$$

Set $g(\infty) = I_{\text{in}}$ and zero everywhere else (only absorption).
Set $\tau(d) = d$

$$\Rightarrow \text{Transparency } T(s) = \exp\left(-\int_0^{s} d(r(t))dt\right)$$

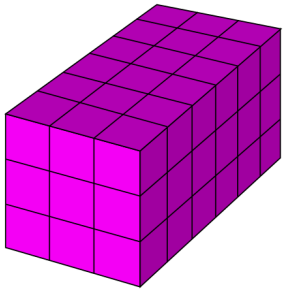
$$\Rightarrow \text{Received Illumination } I_{\text{out}} = T(t_{\text{max}})I_{\text{in}}$$

Nelson Max. Optical Models for Direct Volume Rendering. *TVCG* 1995
Kajiya, von Herzen. Ray Tracing Volume Densities. *Computer Graphics* 1984

1.1 Only Absorption

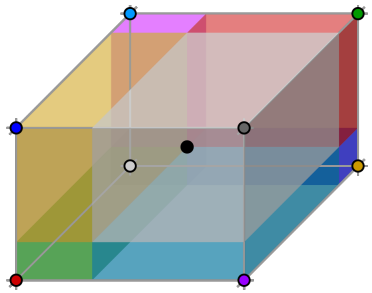
1.1 Only Absorption: Solutions

Densities on a grid v_i



(1)

Trilinear Interpolation $N_i(x)$



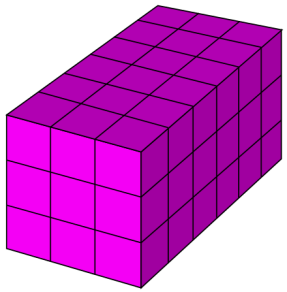
(2)

(1) https://en.wikipedia.org/wiki/Regular_grid

(2) https://en.wikipedia.org/wiki/Trilinear_interpolation

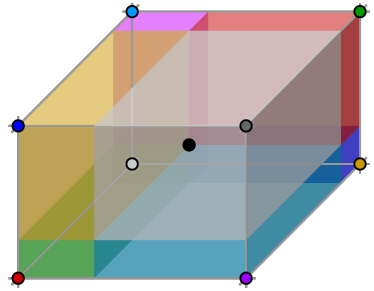
1.1 Only Absorption: Solutions

Densities on a grid v_i



(1)

Trilinear Interpolation $N_i(\mathrm{x})$



(2)

$$\Rightarrow \log I_{\text{out}} = \sum_{i \in \text{voxels}} \sum_{j=1}^{8} v_{ij} \int_{t_0^{(i)}}^{t_1^{(i)}} N_j(r(t))dt$$

Assemble into matrices for all rays:

$$\Rightarrow \log \boldsymbol{I}_{\text{out}} = \boldsymbol{N}\boldsymbol{v}$$

(1) https://en.wikipedia.org/wiki/Regular_grid

(2) https://en.wikipedia.org/wiki/Trilinear_interpolation

1.1 Only Absorption: Differentiation

$$\log \boldsymbol{I}_{\text{out}} = \boldsymbol{N}\ \boldsymbol{v}$$

output images — matrix of voxel-pixel contributions — voxel densities

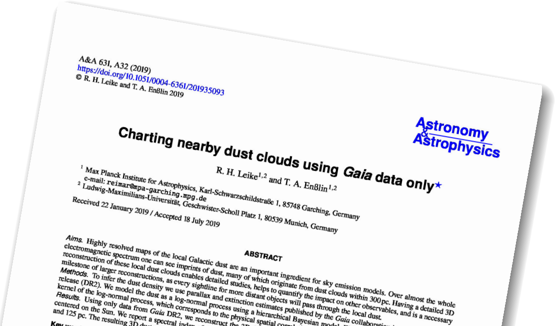
$\Rightarrow$ Linear in v, simply solve it

1.1 Only Absorption: Differentiation

$$\log \boldsymbol{I}_{\text{out}} = \boldsymbol{N} \; \boldsymbol{v}$$

output images — matrix of voxel-pixel contributions — voxel densities

$\Rightarrow$ Linear in v, simply solve it


A&A 631, A32 (2019)
https://doi.org/10.1051/0004-6361/201935093
© R. H. Leike and T. A. Enßlin 2019

Astronomy & Astrophysics

Charting nearby dust clouds using *Gaia* data only★

R. H. Leike[1,2] and T. A. Enßlin[1,2]

[1] Max Planck Institute for Astrophysics, Karl-Schwarzschildstraße 1, 85748 Garching, Germany
e-mail: reimar@mpa-garching.mpg.de
[2] Ludwig-Maximilians-Universität, Geschwister-Scholl Platz 1, 80539 Munich, Germany

Received 22 January 2019 / Accepted 18 July 2019

ABSTRACT

Aims. Highly resolved maps of the local Galactic dust are an important ingredient for sky emission models. Over almost the whole electromagnetic spectrum one can see imprints of dust, many of which originate from dust clouds within 300 pc. Having a detailed 3D reconstruction of these local dust clouds enables detailed studies, helps to quantify the impact on other observables, and is a necessary milestone of larger reconstructions, as every sightline for more distant objects will pass through the local dust.
Methods. To infer the dust density we use parallax and extinction estimates published by the *Gaia* collaboration in the second data release (DR2). We model the dust as a log-normal process using a hierarchical Bayesian model. ... kernel of the log-normal process, which corresponds to the physical spatial correlation ...
Results. Using only data from *Gaia* DR2, we reconstruct the 3D ... centered on the Sun. We report a spectral index ... and 125 pc. The resulting 3D dust ...

1.2 With Emission

Now every point in space also emits light $g(d)$
Usually dependent on $\tau(d)$: $g(d) := c(d)\tau(d)$ for some color $c(d)$.

1.2 With Emission

Now every point in space also emits light $g(d)$
Usually dependent on $\tau(d)$: $g(d) := c(d)\tau(d)$ for some color $c(d)$.

$$I_{\text{out}} = T(t_{\text{max}})I_{\text{in}} + \int_0^{t_{\text{max}}} g(t)T(t)dt$$

$$\text{with } T(s) = \exp\left(-\int_0^s \tau(v(r(t)))dt\right)$$

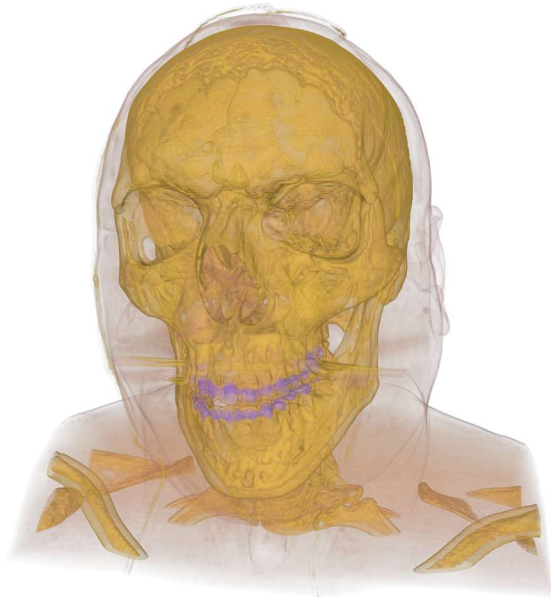
This is a nested integral and cannot be solved analytically.

1.2 With Emission

$$\tau(d) = d, c(d) = d$$

1.2 With Emission

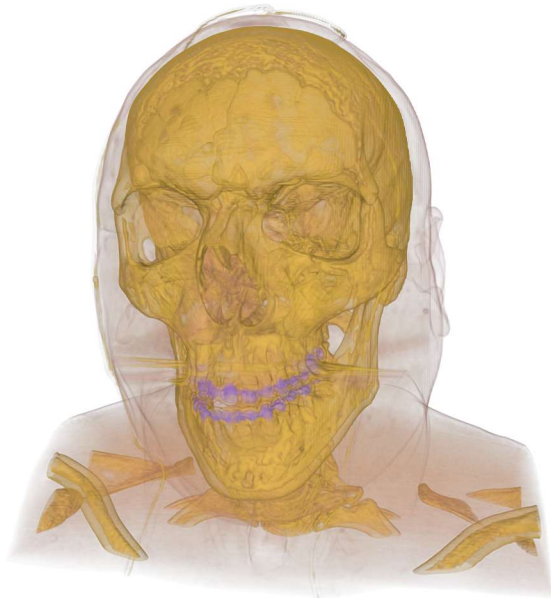
$\tau(d) = d, c(d) = d$



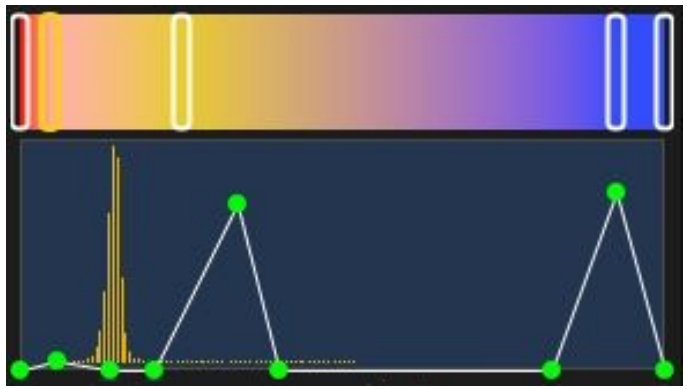
$\tau(d), c(d)$ arbitrary

1.2 With Emission

$\tau(d) = d, c(d) = d$



$\tau(d), c(d)$ arbitrary



2. Automatic Differentiation

$$\boldsymbol{x}_0 = \text{const}$$
$$\boldsymbol{x}_1 = f_1(\boldsymbol{x}_0, \boldsymbol{w}_1)$$
$$\boldsymbol{x}_2 = f_2(\boldsymbol{x}_1, \boldsymbol{w}_2)$$
$$\dots$$
$$\boldsymbol{x}_k = f_k(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)$$

with

$k \in \mathbb{N}$	number of operations
$\boldsymbol{x}_i \in \mathbb{R}^n$	state variables
$\boldsymbol{x}_k \in \mathbb{R}^m$	size of the output / last state
$\boldsymbol{w}_i \in \mathbb{R}^p$	parameters

2. Automatic Differentiation

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_k(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
\qquad \Rightarrow \frac{\mathrm{d}\boldsymbol{x}_k}{\mathrm{d}\boldsymbol{w}_1}
$$

with

$k \in \mathbb{N}$	number of operations
$\boldsymbol{x}_i \in \mathbb{R}^n$	state variables
$\boldsymbol{x}_k \in \mathbb{R}^m$	size of the output / last state
$\boldsymbol{w}_i \in \mathbb{R}^p$	parameters

2.1 AutoDiff: Forward Mode

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

2.1 AutoDiff: Forward Mode

$$\begin{aligned} \boldsymbol{x}_0 &= \text{const} \\ \boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\ \boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\ &\dots \\ \boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k) \end{aligned}$$

Replace each $\boldsymbol{x}_i, \boldsymbol{w}_i$ by

$$\tilde{\boldsymbol{x}}_i = \left\langle \boldsymbol{x}_i, \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle, \quad \tilde{\boldsymbol{w}}_i = \left\langle \boldsymbol{w}_i, \frac{\mathrm{d}\boldsymbol{w}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle$$

Note that $\tilde{\boldsymbol{x}}_0 = \langle \boldsymbol{x}_0, 0 \rangle, \tilde{\boldsymbol{w}}_1 = \langle \boldsymbol{w}_1, 1 \rangle$

2.1 AutoDiff: Forward Mode

$$\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\cdots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}$$

Replace each $\boldsymbol{x}_i, \boldsymbol{w}_i$ by

$$\tilde{\boldsymbol{x}}_i = \left\langle \boldsymbol{x}_i, \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle, \quad \tilde{\boldsymbol{w}}_i = \left\langle \boldsymbol{w}_i, \frac{\mathrm{d}\boldsymbol{w}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle$$

Note that $\tilde{\boldsymbol{x}}_0 = \langle \boldsymbol{x}_0, 0 \rangle$, $\tilde{\boldsymbol{w}}_1 = \langle \boldsymbol{w}_1, 1 \rangle$

Replace a function $\boldsymbol{c} = f(\boldsymbol{a}, \boldsymbol{b})$ by $\tilde{\boldsymbol{c}} = \tilde{f}(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}})$ with

$$\left\langle \boldsymbol{c}, \frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{c}_1} \right\rangle = \left\langle f(a, b), \right.$$
$$\left. J_{f|a}(a, b) \frac{\mathrm{d}\boldsymbol{a}_i}{\mathrm{d}\boldsymbol{w}_1} + J_{f|b}(a, b) \frac{\mathrm{d}\boldsymbol{b}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle$$

2.1 AutoDiff: Forward Mode

$$\boldsymbol{x}_0 = \text{const}$$
$$\boldsymbol{x}_1 = f_1(\boldsymbol{x}_0, \boldsymbol{w}_1)$$
$$\boldsymbol{x}_2 = f_2(\boldsymbol{x}_1, \boldsymbol{w}_2)$$
$$\cdots$$
$$\boldsymbol{x}_k = f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)$$

Replace each $\boldsymbol{x}_i, \boldsymbol{w}_i$ by

$$\tilde{\boldsymbol{x}}_i = \left\langle \boldsymbol{x}_i, \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle, \quad \tilde{\boldsymbol{w}}_i = \left\langle \boldsymbol{w}_i, \frac{\mathrm{d}\boldsymbol{w}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle$$

Note that $\tilde{\boldsymbol{x}}_0 = \langle \boldsymbol{x}_0, 0 \rangle$, $\tilde{\boldsymbol{w}}_1 = \langle \boldsymbol{w}_1, 1 \rangle$

Replace a function $\boldsymbol{c} = f(\boldsymbol{a}, \boldsymbol{b})$ by $\tilde{\boldsymbol{c}} = \tilde{f}(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}})$ with

$$\left\langle \boldsymbol{c}, \frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{c}_1} \right\rangle = \left\langle f(a, b), J_{f|a}(a, b)\frac{\mathrm{d}\boldsymbol{a}_i}{\mathrm{d}\boldsymbol{w}_1} + J_{f|b}(a, b)\frac{\mathrm{d}\boldsymbol{b}_i}{\mathrm{d}\boldsymbol{w}_1} \right\rangle$$

$\Rightarrow O(np)$ memory
$\Rightarrow O(kn^2 + knp)$ computations

2.2 AutoDiff: Backward Mode / Adjoint Method

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

2.2 AutoDiff: Backward Mode / Adjoint Method

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

Assume $x_k \in \mathbb{R}$ is a scalar ($m = 1$), define

$$
\hat{\boldsymbol{x}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{x}_i} \in \mathbb{R}^n, \quad \hat{\boldsymbol{w}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{w}_i} \in \mathbb{R}^p,
$$

Note: $\hat{x}_k = 1$.

2.2 AutoDiff: Backward Mode / Adjoint Method

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

Assume $x_k \in \mathbb{R}$ is a scalar ($m = 1$), define

$$\hat{\boldsymbol{x}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{x}_i} \in \mathbb{R}^n, \quad \hat{\boldsymbol{w}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{w}_i} \in \mathbb{R}^p,$$

Note: $\hat{x}_k = 1$.

1. Forward Pass, compute x_k normally, but save intermediate results

2.2 AutoDiff: Backward Mode / Adjoint Method

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

Assume $x_k \in \mathbb{R}$ is a scalar ($m = 1$), define

$$\hat{\boldsymbol{x}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{x}_i} \in \mathbb{R}^n, \quad \hat{\boldsymbol{w}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{w}_i} \in \mathbb{R}^p,$$

Note: $\hat{x}_k = 1$.

1. Forward Pass, compute x_k normally, but save intermediate results
2. Backward Pass: For $i = k, ..., 1$

$$
\begin{aligned}
\hat{\boldsymbol{x}}_{i-1} &= J_{f_i|\boldsymbol{x}_{i-1}}(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)^T \hat{\boldsymbol{x}}_i \\
\hat{\boldsymbol{w}}_i &= J_{f_i|\boldsymbol{w}_i}(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)^T \hat{\boldsymbol{x}}_i
\end{aligned}
$$

2.2 AutoDiff: Backward Mode / Adjoint Method

$$
\begin{aligned}
\boldsymbol{x}_0 &= \text{const} \\
\boldsymbol{x}_1 &= f_1(\boldsymbol{x}_0, \boldsymbol{w}_1) \\
\boldsymbol{x}_2 &= f_2(\boldsymbol{x}_1, \boldsymbol{w}_2) \\
&\dots \\
\boldsymbol{x}_k &= f_n(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k)
\end{aligned}
$$

Assume $x_k \in \mathbb{R}$ is a scalar ($m = 1$), define

$$\hat{\boldsymbol{x}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{x}_i} \in \mathbb{R}^n, \quad \hat{\boldsymbol{w}}_i = \frac{\mathrm{d}x_k}{\mathrm{d}\boldsymbol{w}_i} \in \mathbb{R}^p,$$

Note: $\hat{x}_k = 1$.

1. Forward Pass, compute x_k normally, but save intermediate results
2. Backward Pass: For $i = k, ..., 1$

$$
\begin{aligned}
\hat{\boldsymbol{x}}_{i-1} &= J_{f_i|\boldsymbol{x}_{i-1}}(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)^T \hat{\boldsymbol{x}}_i \\
\hat{\boldsymbol{w}}_i &= J_{f_i|\boldsymbol{w}_i}(\boldsymbol{x}_{i-1}, \boldsymbol{w}_i)^T \hat{\boldsymbol{x}}_i
\end{aligned}
$$

$\Rightarrow O(kn)$ memory
$\Rightarrow O(kn^2 + np)$ computations

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs
- Compile-time: implement with operator overloading

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass
- Dynamic computation graph Dynamic memory

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs
- Compile-time: implement with operator overloading
- Expensive for large p

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass
- Dynamic computation graph Dynamic memory
- Lots of memory

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs
- Compile-time: implement with operator overloading
- Expensive for large p

$\Rightarrow$ use if k large, p small

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass
- Dynamic computation graph
  Dynamic memory
- Lots of memory

$\Rightarrow$ use if k small, p large

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs
- Compile-time: implement with operator overloading
- Expensive for large p

$\Rightarrow$ use if k large, p small

$\Rightarrow$ Sensitivity Analysis in simulations

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass
- Dynamic computation graph Dynamic memory
- Lots of memory

$\Rightarrow$ use if k small, p large

$\Rightarrow$ Neural networks

2.3 AutoDiff: Comparison

Forward Mode

- $\Rightarrow O(np)$ memory
  $\Rightarrow O(kn^2 + knp)$ ops
- Arbitrary number of outputs
- Compile-time: implement with operator overloading
- Expensive for large p

$\Rightarrow$ use if k large, p small

$\Rightarrow$ Sensitivity Analysis in simulations

Backward Mode

- $\Rightarrow O(kn)$ memory
  $\Rightarrow O(kn^2 + np)$ ops
- Only one output! Or repeat backward pass
- Dynamic computation graph
  Dynamic memory
- Lots of memory

$\Rightarrow$ use if k small, p large

$\Rightarrow$ Neural networks

For DVR we need some hybrid method...