Optimal smoothing for pathwise adjoints

Jonathan Hüser, Shih-Te Yang and Uwe Naumann

Informatik 12: Software and Tools for Computational Engineering
RWTH Aachen University

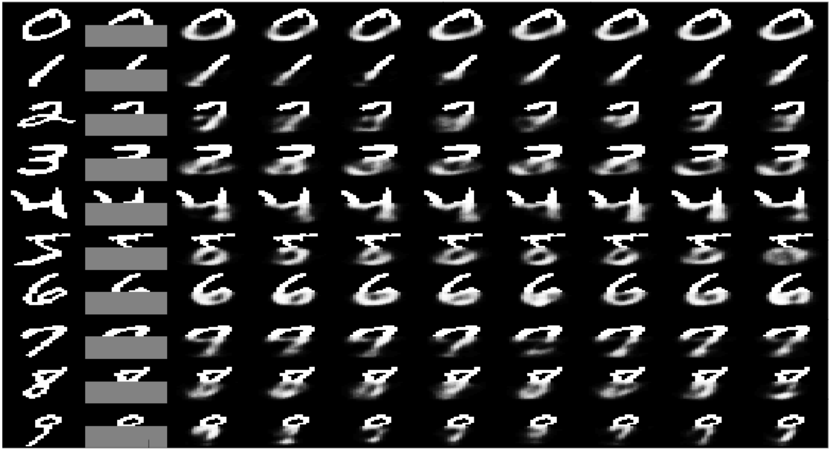
December 8, 2017

Binary Stochastic Neural Network[1]

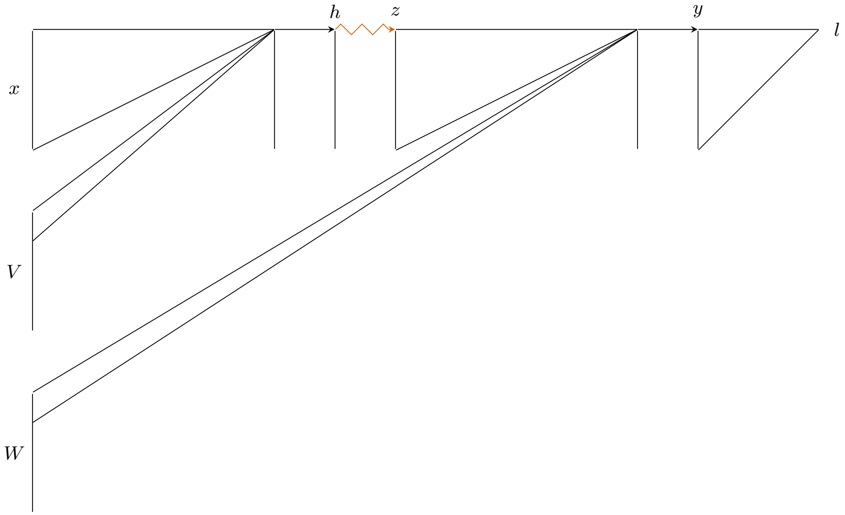
[1] Techniques for Learning Binary Stochastic Feedforward Neural Networks, Raiko et al., 2015

Binary Stochastic Neural Network: Goal

- Generative model for binarized MNIST



Binary Stochastic Neural Network: DAG



$$h = \sigma\left(V \begin{pmatrix} 1 \\ x \end{pmatrix}\right), \quad z_i \sim \text{Bernoulli}(h_i), \quad y = \sigma\left(W \begin{pmatrix} 1 \\ z \end{pmatrix}\right)$$

Binary Stochastic Neural Network: Loss

- Minimize negative loglikelihood of lower pixels

$$\min_{V,W} \mathbb{E}_{x,y} - \log \sum_i P(y \mid W, z^i) P(z^i \mid V, x)$$

Binary Stochastic Neural Network: Loss

- Minimize negative loglikelihood of lower pixels

$$\min_{V,W} \mathbb{E}_{x,y} - \log \sum_i P(y \mid W, z^i) P(z^i \mid V, x)$$

- MC estimator of loss

$$- \log \sum_i P(y \mid W, z^i) P(z^i \mid V, x) = - \log \mathbb{E}_z \; P(y \mid W, z)$$

$$\approx - \log \frac{1}{N} \sum_{j=1}^{N} P(y \mid W, z^j) = L(V, W) = l$$

Binary Stochastic Neural Network: Loss

- Minimize negative loglikelihood of lower pixels

$$\min_{V,W} \mathbb{E}_{x,y} - \log \sum_i P(y \mid W, z^i) P(z^i \mid V, x)$$

- MC estimator of loss

$$- \log \sum_i P(y \mid W, z^i) P(z^i \mid V, x) = - \log \mathbb{E}_z \, P(y \mid W, z)$$

$$\approx - \log \frac{1}{N} \sum_{j=1}^{N} P(y \mid W, z^j) = L(V, W) = l$$

- **Pathwise adjoint** does not give sensitivity

$$V_{(1)} \mathrel{+}= \nabla_V L(V, W)^T \, l_{(1)} = 0$$

Score Function Trick

(Likelihood Ratio Method / REINFORCE[2])

[2] Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning, Williams, 1992

Score Function Trick: Idea

- Bridge sampling gap: Use derivatives of the probability

$$
\begin{aligned}
\nabla_V \mathbb{E}_{z\sim P(g(V))}\, f(z) &= \sum_i f(z^i)\ \nabla_V P(z^i \mid g(V)) \\
&= \sum_i f(z^i)\ \nabla_V \log P(z^i \mid g(V))\ P(z^i \mid g(V)) \\
&\approx \frac{1}{N}\sum_{j=1}^{N} f(z^j)\ \nabla_V \log P(z^j \mid g(V))
\end{aligned}
$$

Score Function Trick: Idea

- Bridge sampling gap: Use derivatives of the probability

$$\begin{aligned}
\nabla_V \mathbb{E}_{z\sim P(g(V))}\, f(z) &= \sum_i f(z^i)\ \nabla_V P(z^i \mid g(V)) \\
&= \sum_i f(z^i)\ \nabla_V \log P(z^i \mid g(V))\ P(z^i \mid g(V)) \\
&\approx \frac{1}{N}\sum_{j=1}^{N} f(z^j)\ \nabla_V \log P(z^j \mid g(V))
\end{aligned}$$

- Seed adjoint $g_{(1)}$ to plug into AD computation

$$\begin{aligned}
V_{(1)} &\mathrel{+}= \nabla_V g(V)^T\ \frac{\partial}{\partial g} \log P(z^j \mid g)\ f(z^j) \\
&= f(z^j)\ \nabla_V \log P(z^j \mid g(V))
\end{aligned}$$

Score Function Trick: Variance Problem

- Score function trick results in a high variance gradient estimator

$$\min_{\theta} \mathbb{E}_z \, (z - 0.49)^2 + c, \quad z \sim \text{Bernoulli}(\theta)$$

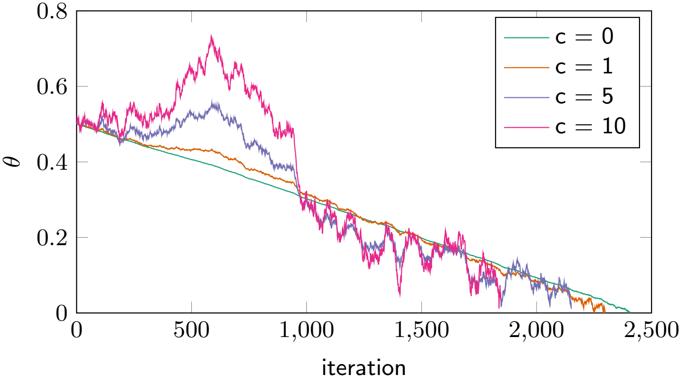
$$\approx \frac{1}{1000} \sum_{j=1}^{1000} (z^j - 0.49)^2 + c$$

- Score function trick results in a high variance gradient estimator

$$\min_{\theta} \mathbb{E}_z \, (z - 0.49)^2 + c, \quad z \sim \text{Bernoulli}(\theta)$$

$$\approx \frac{1}{1000} \sum_{j=1}^{1000} (z^j - 0.49)^2 + c$$

- SGD using score function adjoint



Reparametrization Trick

Reparametrization Trick: Idea

- Bridge sampling gap: Make random variable deterministic function of parameter and nonparametric noise

$$
\begin{aligned}
& z \sim \text{Bernoulli}(g(\theta)) \\
\rightarrow \quad & u \sim \text{Uniform}(0,1) \\
& z = H(g(\theta) - u)
\end{aligned}
$$

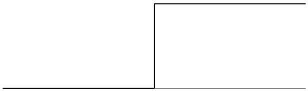
Reparametrization Trick: Idea

- Bridge sampling gap: Make random variable deterministic function of parameter and nonparametric noise

$$\begin{aligned} & z \sim \text{Bernoulli}(g(\theta)) \\ \rightarrow \quad & u \sim \text{Uniform}(0,1) \\ & z = H(g(\theta) - u) \end{aligned}$$

- H is the Heaviside step function

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

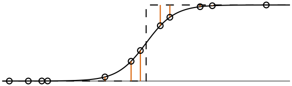


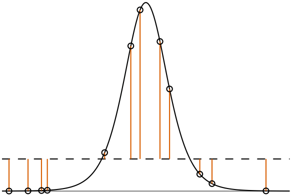
Reparametrization Trick: Smoothing

- Approximate $H(x) \approx \sigma(x/p)$ to obtain a differentiable surrogate model

[3] The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables, Maddison et al., 2017

- Approximate $H(x) \approx \sigma(x/p)$ to obtain a differentiable surrogate model
- Choice of p introduces bias variance tradeoff
  $\rightarrow$ **optimal smoothing**

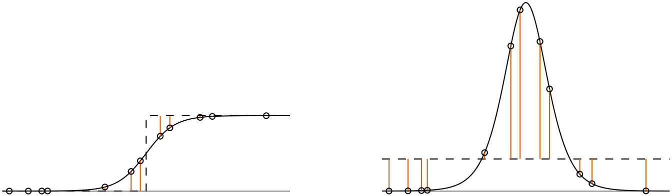




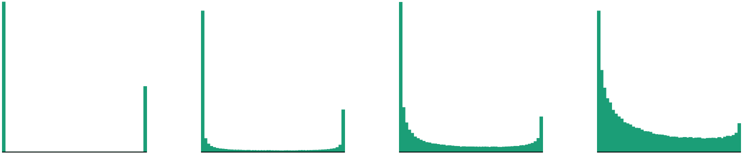
[3] The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables, Maddison et al., 2017

Reparametrization Trick: Smoothing

- Approximate $H(x) \approx \sigma(x/p)$ to obtain a differentiable surrogate model
- Choice of p introduces bias variance tradeoff
  $\rightarrow$ **optimal smoothing**

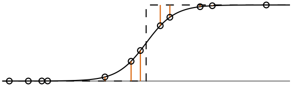


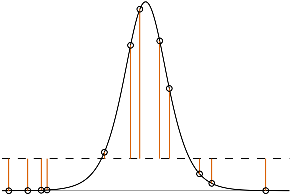
- Resulting z is a continuous relaxation



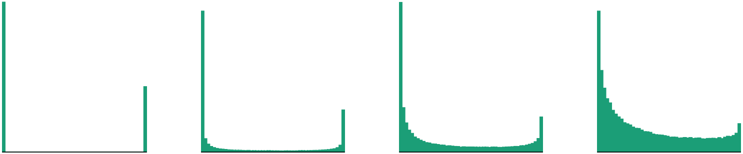
[3] The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables, Maddison et al., 2017

- Approximate $H(x) \approx \sigma(x/p)$ to obtain a differentiable surrogate model
- Choice of p introduces bias variance tradeoff
  $\rightarrow$ **optimal smoothing**





- Resulting z is a continuous relaxation



- CONCRETE[3] relaxation $z = \sigma((\log(\theta/(1-\theta)) + \log(1-u) - \log(u))/p)$

[3] The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables, Maddison et al., 2017

- Smoothing reparametrization trick results in biased gradient estimator

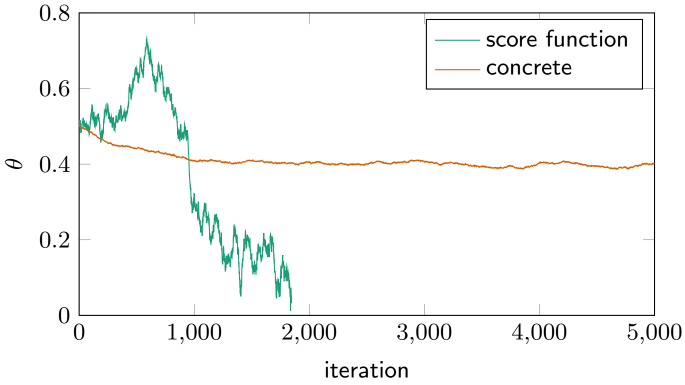
$$\min_{\theta} \ \mathbb{E}_z \ (z - 0.49)^2 + 10, \quad z \sim \mathsf{Bernoulli}(\theta)$$

Reparametrization Trick: Bias Problem

- Smoothing reparametrization trick results in biased gradient estimator

$$\min_\theta \ \mathbb{E}_z \left(z - 0.49\right)^2 + 10, \quad z \sim \text{Bernoulli}(\theta)$$

- SGD using CONCRETE relaxation adjoint with $p = 0.1$



Control Variates

Control Variates: Idea

- Reduce variance using correlated function with "known" mean

$$\mathbb{E}_z f(z) \approx \frac{1}{N}\sum_{j=1}^{N}(f(z^j) - \tilde{f}(z^j)) + \mathbb{E}_z \tilde{f}(z)$$

[4] REBAR: Low-Variance, Unbiased Gradient Estimates for Discrete Latent Variable Models, Tucker et al., 2017

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

- Reduce variance using correlated function with "known" mean

$$\mathbb{E}_z f(z) \approx \frac{1}{N} \sum_{j=1}^{N} (f(z^j) - \tilde{f}(z^j)) + \mathbb{E}_z \tilde{f}(z)$$

- REBAR[4]: Use smoothing as control variate for score function
  $\rightarrow$ low variance unbiased estimator

[4] REBAR: Low-Variance, Unbiased Gradient Estimates for Discrete Latent Variable Models, Tucker et al., 2017

Control Variates: Idea

- Reduce variance using correlated function with "known" mean

$$\mathbb{E}_z f(z) \approx \frac{1}{N}\sum_{j=1}^{N}(f(z^j) - \tilde{f}(z^j)) + \mathbb{E}_z \tilde{f}(z)$$

- REBAR[4]: Use smoothing as control variate for score function
  $\rightarrow$ low variance unbiased estimator
- Roughly as follows (details more involved due to marginalization)

$$\nabla_\theta \mathbb{E}_z L(z) \approx \underbrace{\frac{1}{N}\sum_{j=1}^{N}(L(z^j) - \tilde{L}(z^j))\ \nabla_\theta \log P(z^j \mid \theta)}_{\text{score function adjoint}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \tilde{L}(z^j)}_{\text{smoothed pathwise adjoint}}$$

[4] REBAR: Low-Variance, Unbiased Gradient Estimates for Discrete Latent Variable Models, Tucker et al., 2017

Control Variates: No Problem

- REBAR control variate results in low variance unbiased gradient estimator

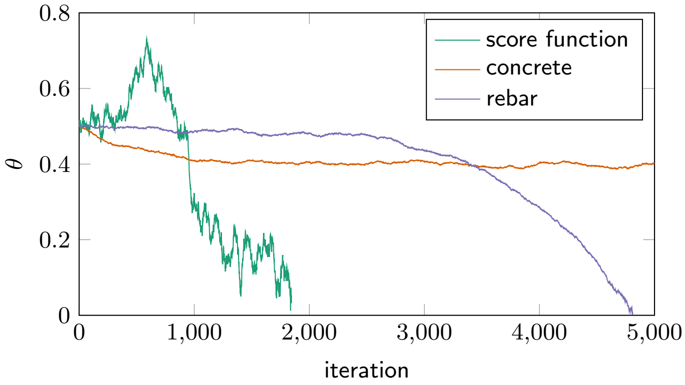
$$\min_{\theta} \; \mathbb{E}_z \; (z - 0.49)^2 + 10, \quad z \sim \text{Bernoulli}(\theta)$$

Control Variates: No Problem

- REBAR control variate results in low variance unbiased gradient estimator

$$\min_{\theta} \ \mathbb{E}_z \ (z - 0.49)^2 + 10, \quad z \sim \text{Bernoulli}(\theta)$$

- SGD using REBAR with $p = 0.1$

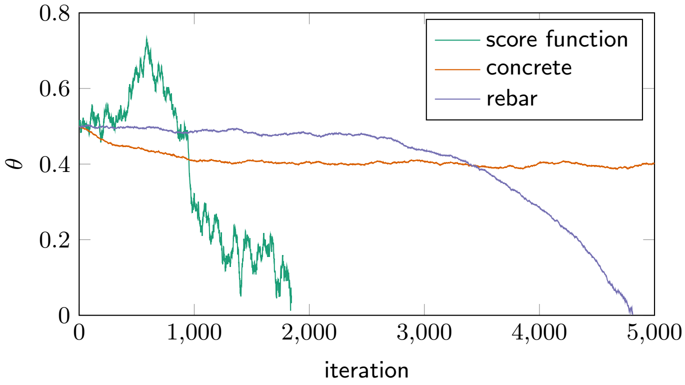


Control Variates: No Problem

- REBAR control variate results in low variance unbiased gradient estimator

$$\min_{\theta} \ \mathbb{E}_z \ (z - 0.49)^2 + 10, \quad z \sim \text{Bernoulli}(\theta)$$

- SGD using REBAR with $p = 0.1$



- **Optimal smoothing:** Parameter p still "unkown"

Control Variates: Adaptive Optimization

- Let $\nabla_\theta L(\theta, p)$ be the REBAR estimator, minimize variance

$$\min_p \; \|\mathbb{E}\left(\nabla_\theta L(\theta,p)^2\right) - \left(\mathbb{E}\,\nabla_\theta L(\theta,p)\right)^2\|_2^2 = G(p) = g$$

Control Variates: Adaptive Optimization

- Let $\nabla_\theta L(\theta, p)$ be the REBAR estimator, minimize variance

$$\min_p \|\mathbb{E}\left(\nabla_\theta L(\theta,p)^2\right) - \left(\mathbb{E}\,\nabla_\theta L(\theta,p)\right)^2\|_2^2 = G(p) = g$$

- To optimize via SGD we can use a one sample gradient estimator

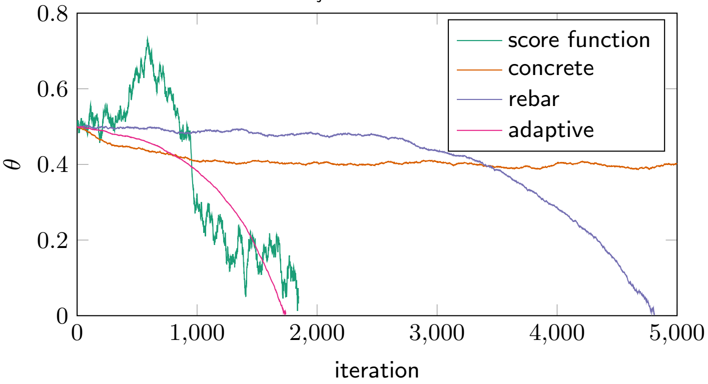
$$\mathbb{E}\left(2\nabla_\theta L(\theta,p)\underbrace{\frac{\partial}{\partial p}\nabla_\theta L(\theta,p)}_{\text{second-order adjoint}}\right) - 2(\mathbb{E}\,\nabla_\theta L(\theta,p))\underbrace{\frac{\partial}{\partial p}\mathbb{E}\,\nabla_\theta L(\theta,p)}_{\text{zero}}$$

- Let $\nabla_\theta L(\theta, p)$ be the REBAR estimator, minimize variance

$$\min_p \|\mathbb{E}\left(\nabla_\theta L(\theta,p)^2\right) - \left(\mathbb{E}\,\nabla_\theta L(\theta,p)\right)^2\|_2^2 = G(p) = g$$

- To optimize via SGD we can use a one sample gradient estimator

$$\mathbb{E}\left(2\nabla_\theta L(\theta,p)\underbrace{\frac{\partial}{\partial p}\nabla_\theta L(\theta,p)}_{\text{second-order adjoint}}\right) - 2(\mathbb{E}\,\nabla_\theta L(\theta,p))\underbrace{\frac{\partial}{\partial p}\mathbb{E}\,\nabla_\theta L(\theta,p)}_{\text{zero}}$$



Second-order Adjoints[5][6]

[5] Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Griewank and Walther, 2008

[6] The Art of Differentiating Computer Programs, Naumann, 2011

- Pathwise adjoint of $l = L(\theta, p)$ is

$$\begin{bmatrix} \theta_{(1)} \\ p_{(1)} \end{bmatrix} \mathrel{+}= \begin{bmatrix} \nabla_\theta L(\theta, p)^T \\ \nabla_p L(\theta, p)^T \end{bmatrix} \; l_{(1)}$$

[7] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

Second-order Adjoints: Reverse-over-reverse 1

- Pathwise adjoint of $l = L(\theta, p)$ is

$$\begin{bmatrix} \theta_{(1)} \\ p_{(1)} \end{bmatrix} += \begin{bmatrix} \nabla_\theta L(\theta,p)^T \\ \nabla_p L(\theta,p)^T \end{bmatrix} \; l_{(1)}$$

- Adjoint of hyper parameter objective $g = G(\theta, p)$ is

$$p_{(2)} += (\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T) \; g_{(2)}$$

[7] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

- Pathwise adjoint of $l = L(\theta, p)$ is

$$\begin{bmatrix} \theta_{(1)} \\ p_{(1)} \end{bmatrix} \mathrel{+}= \begin{bmatrix} \nabla_\theta L(\theta,p)^T \\ \nabla_p L(\theta,p)^T \end{bmatrix} \; l_{(1)}$$

- Adjoint of hyper parameter objective $g = G(\theta, p)$ is

$$p_{(2)} \mathrel{+}= (\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T) \; g_{(2)}$$

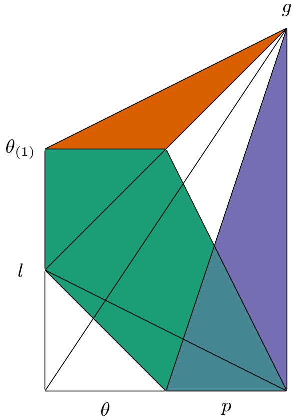
- Implementation via type generic overloading AD (dco/c++[7])

```
dco::ga1s< float >::type x_a1s;
dco::ga1s< dco::ga1s< float >::type >::type x_a1s_a2s;
```

[7] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

- Adjoint of hyper parameter objective $g = G(\theta, p, \theta_{(1)}(\theta, p))$ is

$$p_{(2)} \mathrel{+}= \left(\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T\right) g_{(2)}$$



Second-order Adjoints: Forward-over-reverse 1

- Save memory for second-order adjoints by local replacement with tangent

Second-order Adjoints: Forward-over-reverse 1

- Save memory for second-order adjoints by local replacement with tangent
- Pathwise adjoint of $l = L(\theta, p)$ is

$$\begin{bmatrix} \theta_{(1)} \\ p_{(1)} \end{bmatrix} \mathrel{+}= \begin{bmatrix} \nabla_\theta L(\theta, p)^T \\ \nabla_p L(\theta, p)^T \end{bmatrix} \, l_{(1)} = \nabla L(\theta, p)^T \, l_{(1)}$$

- Save memory for second-order adjoints by local replacement with tangent
- Pathwise adjoint of $l = L(\theta, p)$ is

$$\begin{bmatrix} \theta_{(1)} \\ p_{(1)} \end{bmatrix} += \begin{bmatrix} \nabla_\theta L(\theta,p)^T \\ \nabla_p L(\theta,p)^T \end{bmatrix} \; l_{(1)} = \nabla L(\theta,p)^T \; l_{(1)}$$

- Tangent of pathwise adjoint is

$$\begin{bmatrix} \theta_{(1)}^{(2)} \\ p_{(1)}^{(2)} \end{bmatrix} += l_{(1)}^T \nabla^2 L(\theta,p) \begin{bmatrix} \theta^{(2)} \\ p^{(2)} \end{bmatrix}$$

$$\nabla^2 L(\theta,p) = \begin{bmatrix} \frac{\partial^2 L}{\partial\theta_1 \partial\theta_1} & \cdots & \frac{\partial^2 L}{\partial\theta_1 \partial\theta_{n_\theta}} & \frac{\partial^2 L}{\partial\theta_1 \partial p_1} & \cdots & \frac{\partial^2 L}{\partial\theta_1 \partial p_{n_p}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial^2 L}{\partial\theta_{n_\theta} \partial\theta_1} & \cdots & \frac{\partial^2 L}{\partial\theta_{n_\theta} \partial\theta_{n_\theta}} & \frac{\partial^2 L}{\partial\theta_{n_\theta} \partial p_1} & \cdots & \frac{\partial^2 L}{\partial\theta_{n_\theta} \partial p_{n_p}} \\ \frac{\partial^2 L}{\partial p_1 \partial\theta_1} & \cdots & \frac{\partial^2 L}{\partial p_1 \partial\theta_{n_\theta}} & \frac{\partial^2 L}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 L}{\partial p_1 \partial p_{n_p}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial^2 L}{\partial p_{n_p} \partial\theta_1} & \cdots & \frac{\partial^2 L}{\partial p_{n_p} \partial\theta_{n_\theta}} & \frac{\partial^2 L}{\partial p_{n_p} \partial p_1} & \cdots & \frac{\partial^2 L}{\partial p_{n_p} \partial p_{n_p}} \end{bmatrix}$$

- Adjoint of hyper parameter objective $g = G(\theta, p, \theta_{(1)}(\theta, p))$ is

$$p_{(2)} \mathrel{+}= \left(\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T\right) g_{(2)}$$

- Tangent of pathwise adjoint is

$$\begin{bmatrix} \theta_{(1)}^{(2)} \\ p_{(1)}^{(2)} \end{bmatrix} \mathrel{+}= l_{(1)}^T \nabla^2 L(\theta, p) \begin{bmatrix} \theta^{(2)} \\ p^{(2)} \end{bmatrix}$$

[8] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

- Adjoint of hyper parameter objective $g = G(\theta, p, \theta_{(1)}(\theta, p))$ is

$$p_{(2)} \mathrel{+}= \left(\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T\right) g_{(2)}$$

- Tangent of pathwise adjoint is

$$\begin{bmatrix} \theta_{(1)}^{(2)} \\ p_{(1)}^{(2)} \end{bmatrix} \mathrel{+}= l_{(1)}^T \nabla^2 L(\theta, p) \begin{bmatrix} \theta^{(2)} \\ p^{(2)} \end{bmatrix}$$

- We seed $\theta^{(2)} = \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T$ to get $p_{(1)}^{(2)} = \partial_p \theta_{(1)}(\theta, p)^T \; \theta^{(2)}$

[8] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

- Adjoint of hyper parameter objective $g = G(\theta, p, \theta_{(1)}(\theta, p))$ is

$$p_{(2)} \mathrel{+}= \left(\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T\right) g_{(2)}$$

- Tangent of pathwise adjoint is

$$\begin{bmatrix} \theta_{(1)}^{(2)} \\ p_{(1)}^{(2)} \end{bmatrix} \mathrel{+}= l_{(1)}^T \nabla^2 L(\theta, p) \begin{bmatrix} \theta^{(2)} \\ p^{(2)} \end{bmatrix}$$

- We seed $\theta^{(2)} = \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T$ to get $p_{(1)}^{(2)} = \partial_p \theta_{(1)}(\theta, p)^T \; \theta^{(2)}$
- Local replacement of reverse-over-reverse by forward-over-reverse is an **Adjoint Code Design Pattern (see Poster)**

[8] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

- Adjoint of hyper parameter objective $g = G(\theta, p, \theta_{(1)}(\theta, p))$ is

$$p_{(2)} \mathrel{+}= \left(\partial_p G(\theta, p, \theta_{(1)})^T + \partial_p \theta_{(1)}(\theta, p)^T \; \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T\right) g_{(2)}$$

- Tangent of pathwise adjoint is

$$\begin{bmatrix} \theta_{(1)}^{(2)} \\ p_{(1)}^{(2)} \end{bmatrix} \mathrel{+}= l_{(1)}^T \nabla^2 L(\theta, p) \begin{bmatrix} \theta^{(2)} \\ p^{(2)} \end{bmatrix}$$

- We seed $\theta^{(2)} = \partial_{\theta_{(1)}} G(\theta, p, \theta_{(1)})^T$ to get $p_{(1)}^{(2)} = \partial_p \theta_{(1)}(\theta, p)^T \; \theta^{(2)}$
- Local replacement of reverse-over-reverse by forward-over-reverse is an **Adjoint Code Design Pattern (see Poster)**
- Implementation via type generic overloading AD (dco/c++[8])

```
dco::ga1s< float >::type x_a1s;
dco::ga1s< dco::gt1s< float >::type >::type x_a1s_t2s;
```

[8] dco/c++: Derivative Code by Overloading in C++, Leppkes et al., under review

Thank you