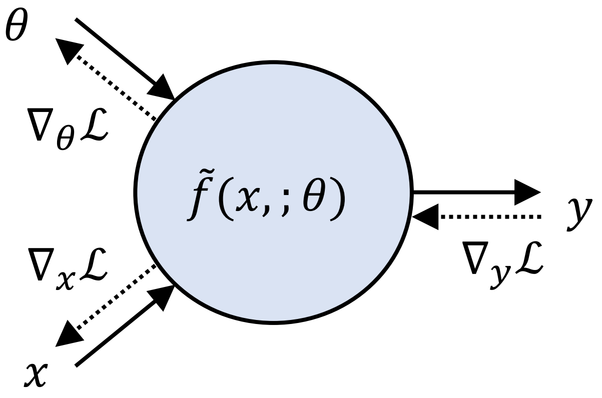
Exploiting Problem Structure in Deep Declarative Networks: Two Case Studies

Stephen Gould

Australian National University

The 1st International Workshop on Optimal Transport and Structured Data Modeling, AAAI, February 2022

Deep declarative networks (DDNs)



In an **imperative node** the input-output relationship is explicitly defined as

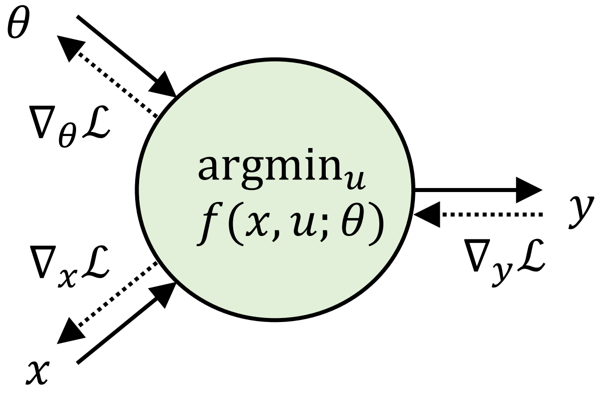
$$y = \tilde{f}(x; \theta)$$

where x is the input and θ are the parameters of the node.

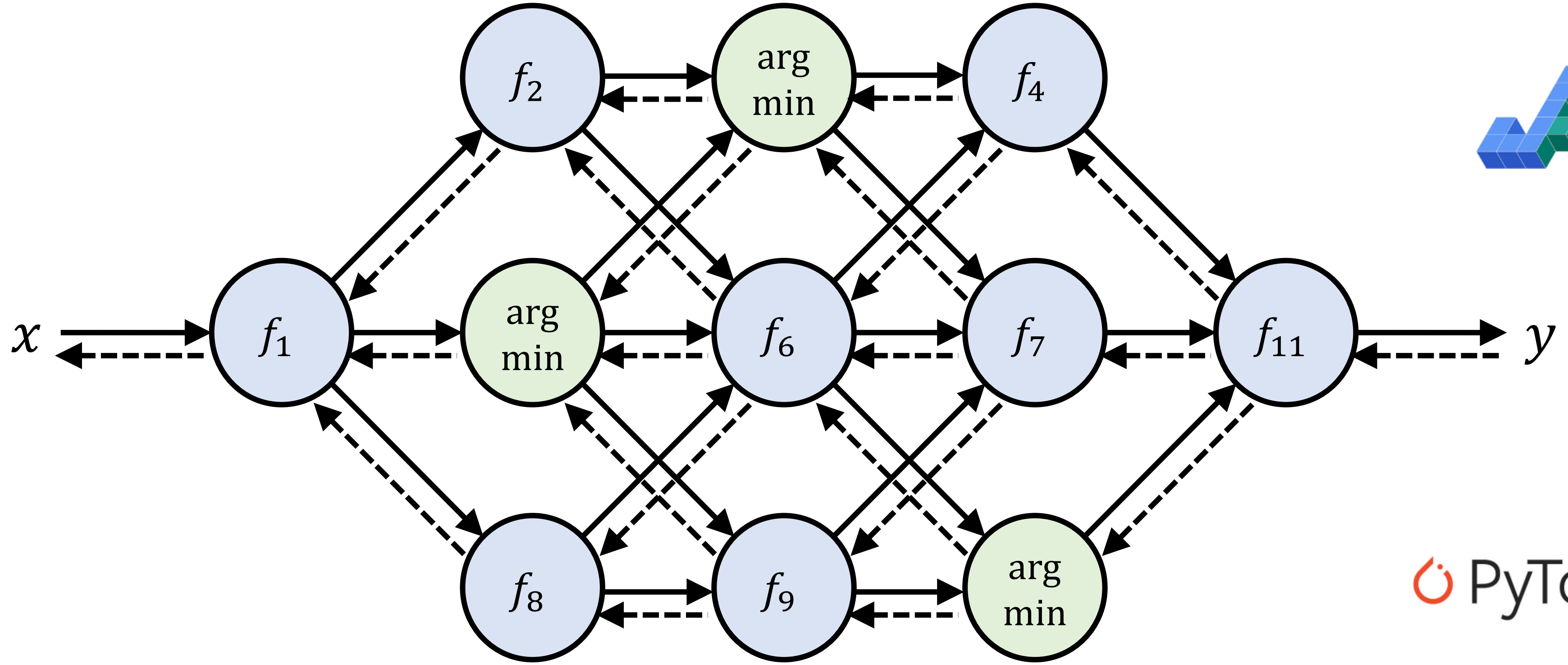
In a **declarative node** the input-output relationship is specified as the solution to an optimization problem

$$y \in \text{argmin}_{u \in C} \; f(x, u; \theta)$$

where f is the objective and C are the constraints.



Imperative and declarative nodes can co-exist



Main technical question for DDNs

How do we compute $\frac{d}{dx}\text{argmin}_{u \in C(x)} f(x, u)$?

Two answers

- Imperative approach: unroll the optimization procedure
  - **Advantages:** very simple, makes use of automatic differentiation so no additional coding is required
  - **Disadvantages:** need to store intermediate calculations in the forward pass, numerical issues when propagating through many iterations, potentially slow, may not be possible if non-differentiable steps are used in the forward pass

- Declarative approach: differentiate the optimality conditions to obtain closed-form expression for the gradient
  - Advantages/disadvantages to be continued…

Main result for (smooth) DDNs

Given optimization problem parametrized by $x \in \mathbb{R}^n$, we define the solution set $Y(x): \mathbb{R}^{\mathrm{n}} \rightrightarrows \mathbb{R}^{\mathrm{m}}$ as

smooth constraints

$$Y(x) = \text{argmin}_u \{\, f(x, u) : h(x, u) = 0_p, g(x, u) \leq 0_q \,\}$$

smooth objective

Then for any regular $y \in Y(x)$,

$$\frac{dy}{dx} = H^{-1}A^T(AH^{-1}A^T)^{-1}(AH^{-1}B - C) - H^{-1}B$$

second partial derivatives of f, h and g

Simplified result

Let $f: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a twice differentiable function and let

$$y(x) \in \text{argmin}_u f(x, u)$$

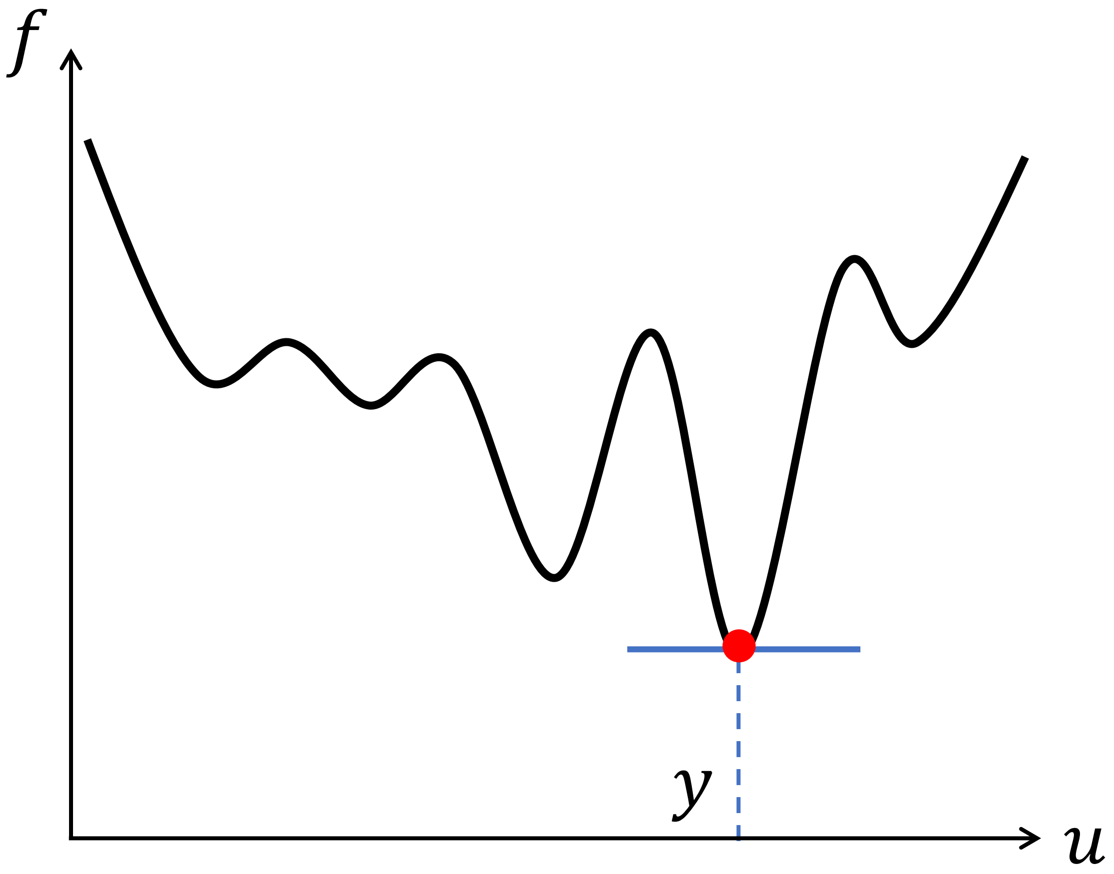
smooth objective

The derivative of f vanishes at (x, y) so by Dini's implicit function theorem (1878)

$$\frac{dy(x)}{dx} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}$$

second partial derivatives of f

Proof sketch



$$y \in \operatorname{argmin}_u f(x,u) \Rightarrow \frac{\partial f(x,y)}{\partial y} = 0$$

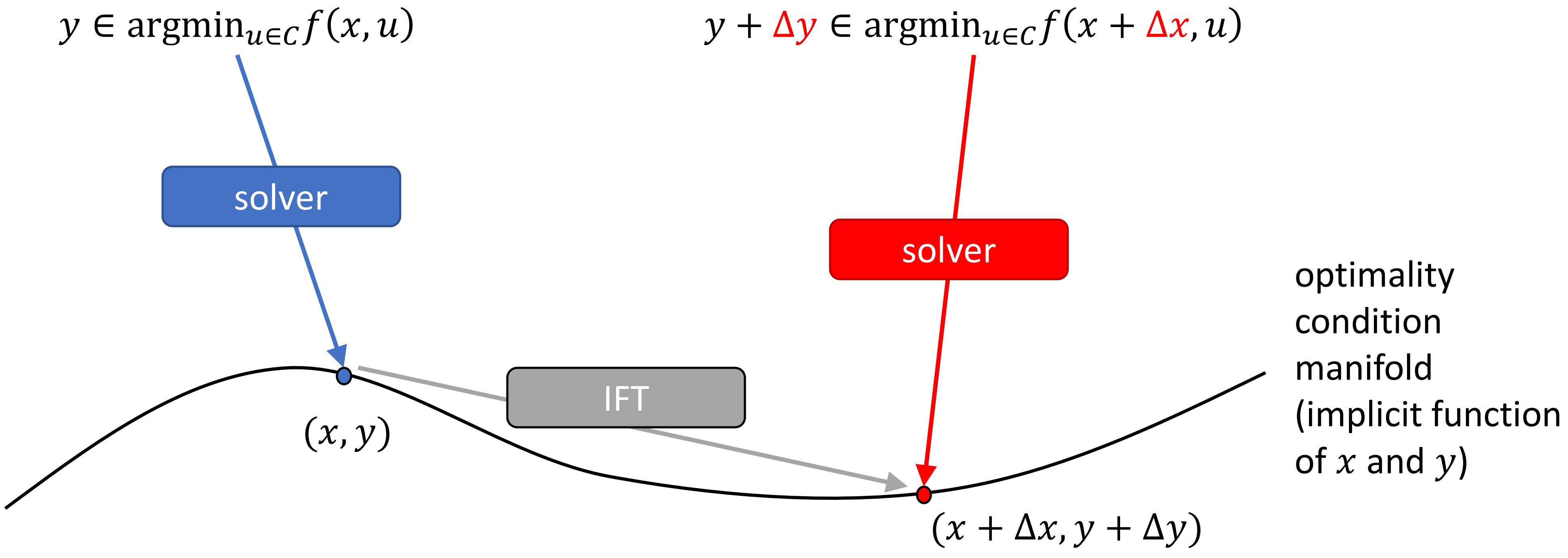
LHS: $\frac{d}{dx}\frac{\partial f(x,y)}{\partial y} = \frac{\partial^2 f(x,y)}{\partial x \partial y} + \frac{\partial^2 f(x,y)}{\partial y^2}\frac{dy}{dx}$

RHS: $\frac{d}{dx} 0 = 0$

Rearranging gives $\frac{dy}{dx} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}$.

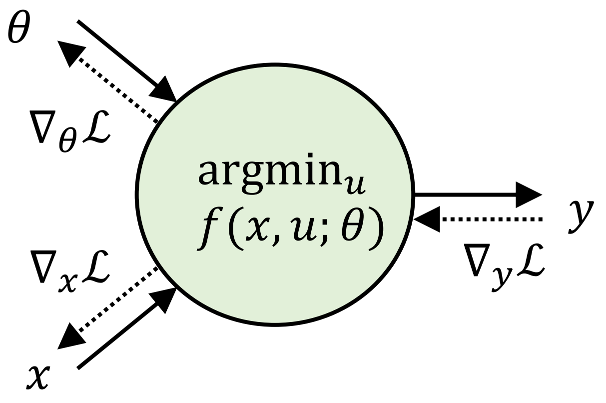
Differentiable Optimization Concept

How does y change as x changes?

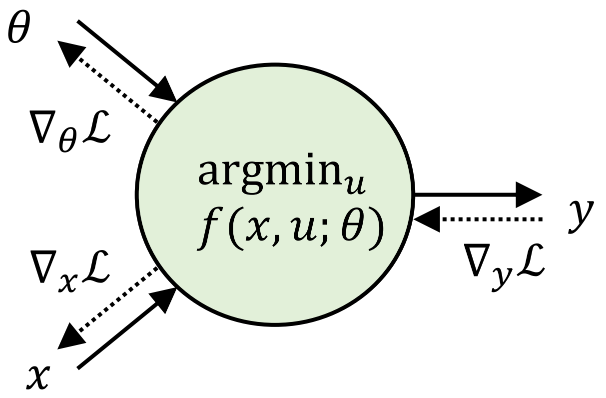


Backward pass summary

$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dy}\frac{dy}{dx}$$



Backward pass summary

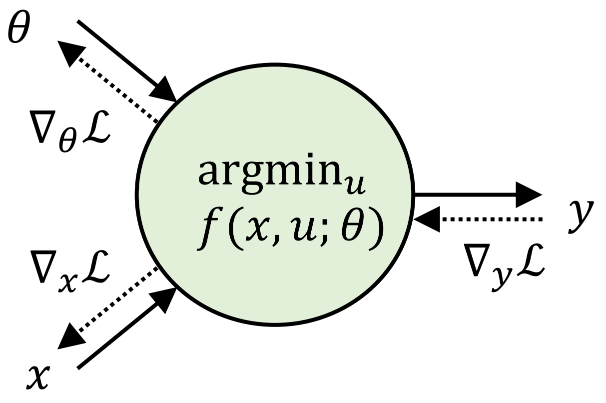


$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dy}\frac{dy}{dx}$$

$$v^T\left(H^{-1}A^T\left(AH^{-1}A^T\right)^{-1}\left(AH^{-1}B - C\right) - H^{-1}B\right)$$

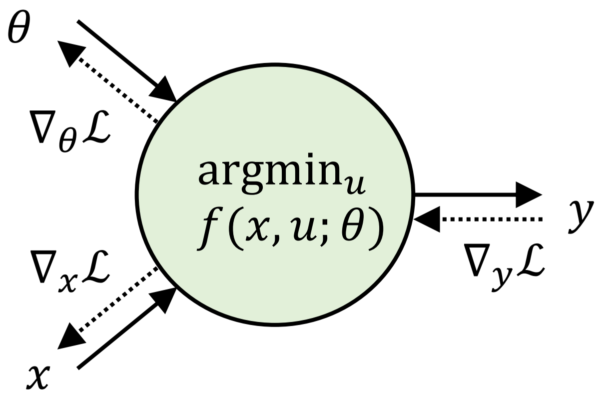
Backward pass summary

$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dy}\frac{dy}{dx}$$



$$v^T \left(H^{-1}A^T(AH^{-1}A^T)^{-1}(AH^{-1}B - C) - H^{-1}B\right)$$

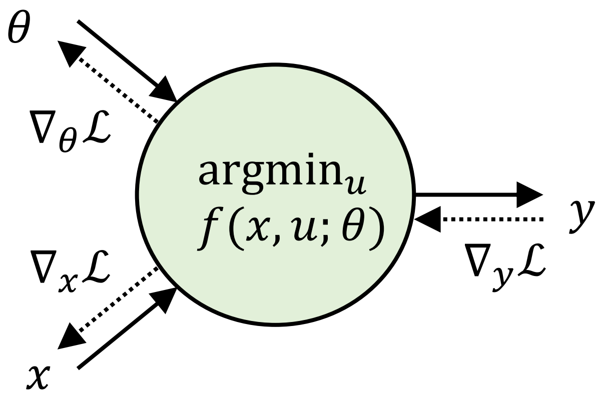
Backward pass summary



$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dy}\frac{dy}{dx}$$

$$v^T\left(H^{-1}A^T(AH^{-1}A^T)^{-1}(AH^{-1}B - C) - H^{-1}B\right)$$

Backward pass summary



$$\frac{d\mathcal{L}}{dx} = \frac{d\mathcal{L}}{dy}\frac{dy}{dx}$$

$$v^T\left(H^{-1}A^T(AH^{-1}A^T)^{-1}(AH^{-1}B - C) - H^{-1}B\right)$$

H: $\frac{d^2 f}{dy^2} - \lambda \frac{d^2 h}{dy^2}$

A: $\frac{dh}{dy}$

C: $\frac{dh}{dx}$

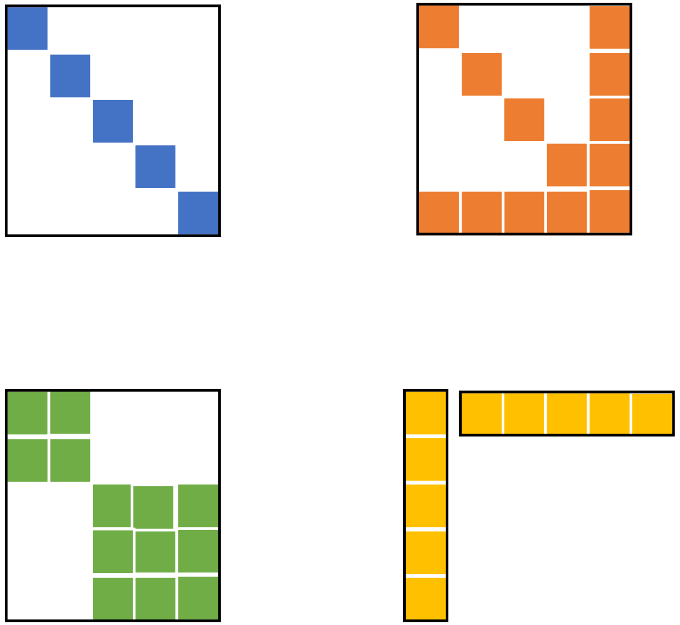
B: $\frac{d^2 f}{dydx} - \lambda \frac{d^2 h}{dydx}$

$$H^{-1}$$

Exploiting structure

- Many problems exhibit structure that can be exploited for H and other terms

- Our paper presents two case studies:
  - Robust vector pooling (omitted in this talk)
  - Optimal transport



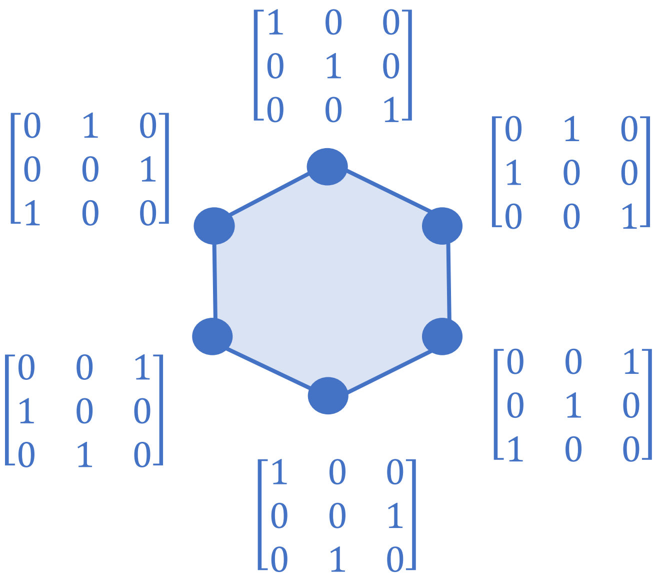
Entropic optimal transport

inner product — regularization

$$\text{minimize} \quad \langle P, M \rangle + \frac{1}{\gamma} \mathrm{KL}(P \| rc^T)$$

$$\text{subject to} \quad P1 = r, \quad P^T 1 = c$$

row sum and column sum constraints



Sinkhorn algorithm

initialize P as $P_{ij} \leftarrow e^{-\gamma M_{ij}}$

repeat until convergence

  for i in $1, \ldots, n$ **do**

    set α_i to the sum of the i-th row of P

    scale the i-th row of P by $\frac{1}{\alpha_i}$

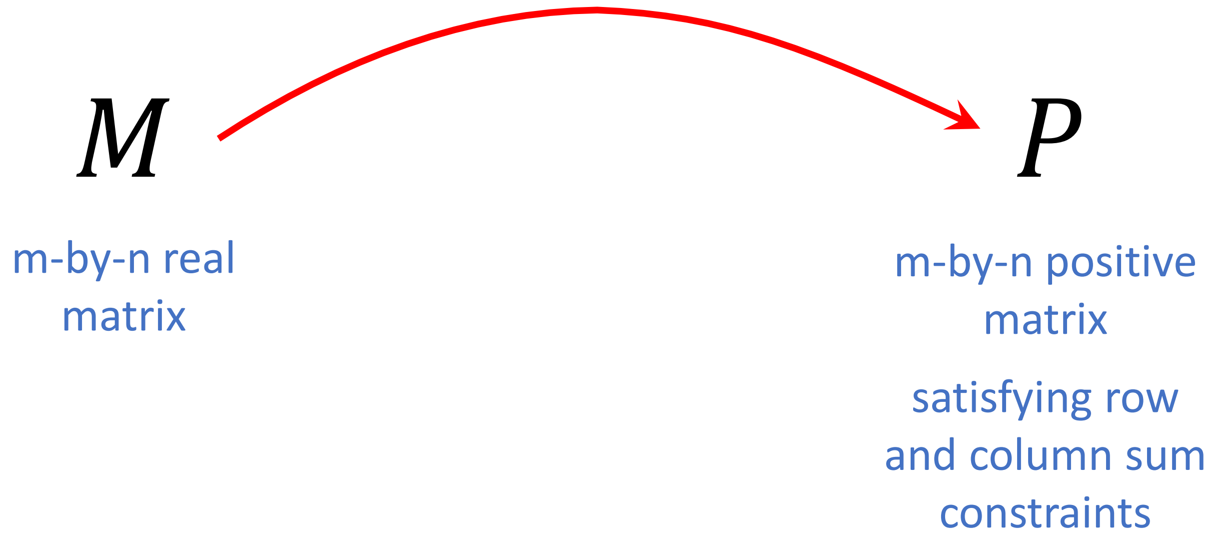
  for i in $1, \ldots, n$ **do**

    set β_j to the sum of the j-th column of P

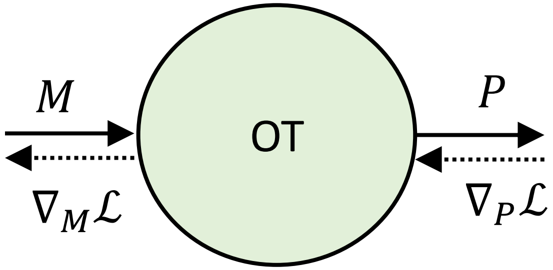
    scale the j-th column of P by $\frac{1}{\beta_j}$

return P

Function mapping M to P



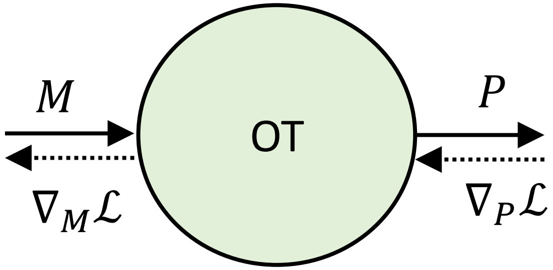
Backward pass for OT



$$\underbrace{\frac{d\mathcal{L}}{dM}}_{\text{m-by-n}} = \underbrace{\frac{d\mathcal{L}}{dP}}_{\text{m-by-n}} \overbrace{\frac{dP}{dM}}^{\text{m-by-n-by-m-by-n}}$$

$$v^T (H^{-1} A^T (A H^{-1} A^T)^{-1} A H^{-1} - H^{-1}) B$$

Backward pass for OT



$$\underbrace{\frac{d\mathcal{L}}{dM}}_{\text{1-by-mn}} = \underbrace{\frac{d\mathcal{L}}{dP}}_{\text{1-by-mn}} \overbrace{\frac{dP}{dM}}^{\text{mn-by-mn}}$$

$$v^T(H^{-1}A^T(AH^{-1}A^T)^{-1}AH^{-1} - H^{-1})B$$

Computing B

$$f(P, M) = \sum_{i=1}^{m} \sum_{j=1}^{n} M_{ij} P_{ij} + \frac{1}{\gamma} \sum_{i=1}^{m} \sum_{j=1}^{n} P_{ij} \log \frac{P_{ij}}{r_i c_j}$$

$$B_{ij,kl} = \frac{\partial^2 f}{\partial P_{ij} \partial M_{kl}} = \begin{cases} 1 & \text{if } ij = kl \\ 0 & \text{otherwise} \end{cases}$$

Computing H^{-1}

$$f(P, M) = \sum_{i=1}^{m} \sum_{j=1}^{n} M_{ij} P_{ij} + \frac{1}{\gamma} \sum_{i=1}^{m} \sum_{j=1}^{n} P_{ij} \log \frac{P_{ij}}{r_i c_j}$$

$$H^{-1} = \left(\left[\frac{\partial^2 f}{\partial P_{ij} \partial P_{kl}} \right]_{ij,kl} \right)^{-1} = \gamma \operatorname{diag}\big(\operatorname{vec}(P)\big)$$

Computing $AH^{-1}A^T$

row and column sum constraints

$$A = \begin{bmatrix} 0_n^T & 1_n^T & \cdots & 0_n^T \\ \vdots & \vdots & \ddots & \vdots \\ 0_n^T & 0_n^T & \cdots & 1_n^T \\ I_{n\times n} & I_{n\times n} & \cdots & I_{n\times n} \end{bmatrix}$$

$$AH^{-1}A^T = \gamma \begin{bmatrix} \text{diag}(r_{2:m}) & P_{2:m,1:n} \\ P_{2:m,1:n}^T & \text{diag}(c) \end{bmatrix}$$

Inverting $AH^{-1}A^T$

$$(AH^{-1}A^T)^{-1} = \frac{1}{\gamma}\begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix}$$

$$\Lambda_{11} = \left(\text{diag}(r_{2:m}) - P_{2:m,1:n}\text{diag}(c)^{-1}P_{2:m,1:n}^T\right)^{-1}$$
$$\Lambda_{12} = -\Lambda_{11}P_{2:m,1:n}\text{diag}(c)^{-1}$$
$$\Lambda_{22} = \text{diag}(c)^{-1} - \text{diag}(c)^{-1}P_{2:m,1:n}^T\Lambda_{12}$$

$$\gamma v^T \vec{P} \begin{bmatrix} A_1^T & A_2^T \end{bmatrix} \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

$$\gamma v^T \vec{P} \begin{bmatrix} A_1^T & A_2^T \end{bmatrix} \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

$$\gamma v^T \vec{P} [A_1^T \quad A_2^T] \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

$$\gamma v^T \vec{P} [A_1^T \quad A_2^T] \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

$$
\begin{aligned}
\Lambda_{11} &= \left(\operatorname{diag}(r_{2:m}) - P_{2:m,1:n}\operatorname{diag}(c)^{-1}P_{2:m,1:n}^{T}\right)^{-1} \\
&= (LL^{T})^{-1}
\end{aligned}
$$

```
block_11 = torch.cholesky(...)
```

$$
\begin{aligned}
\Lambda_{12} &= -\Lambda_{11}P_{2:m,1:n}\operatorname{diag}(c)^{-1} \\
&= -L^{-T}L^{-1}P_{2:m,1:n}\operatorname{diag}(c)^{-1}
\end{aligned}
$$

```
block_12 = torch.cholesky_solve(..., block_11)
```

$$\gamma v^T \vec{P} [A_1^T \quad A_2^T] \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

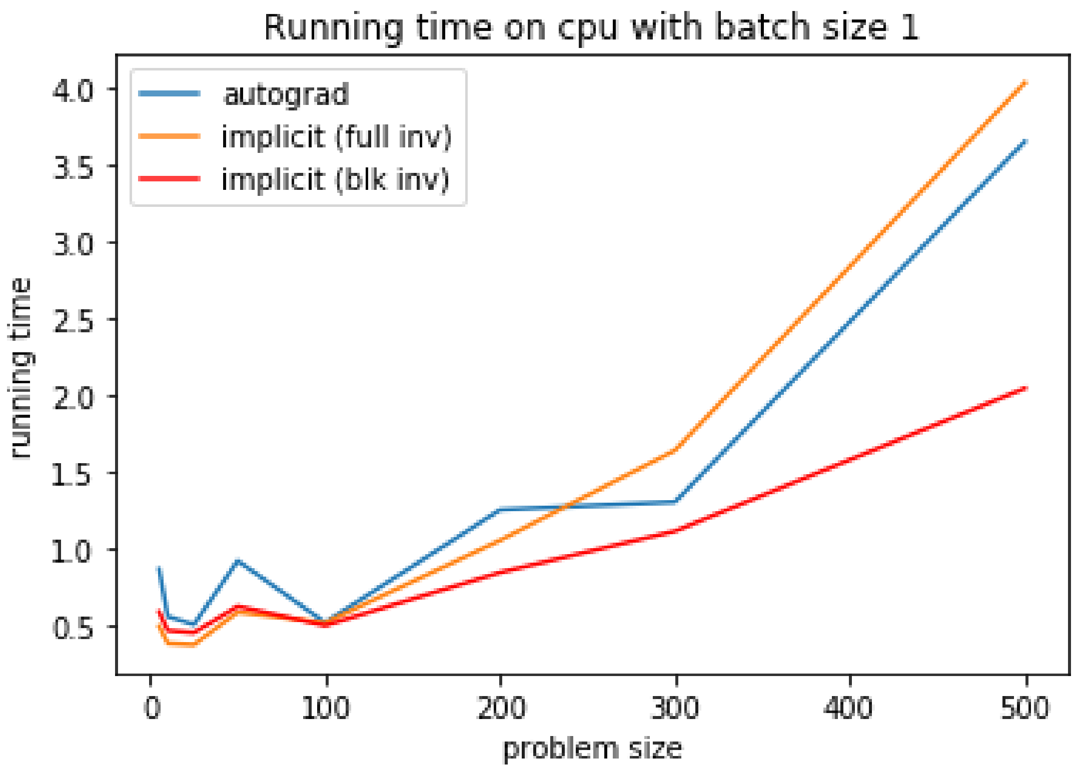
```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

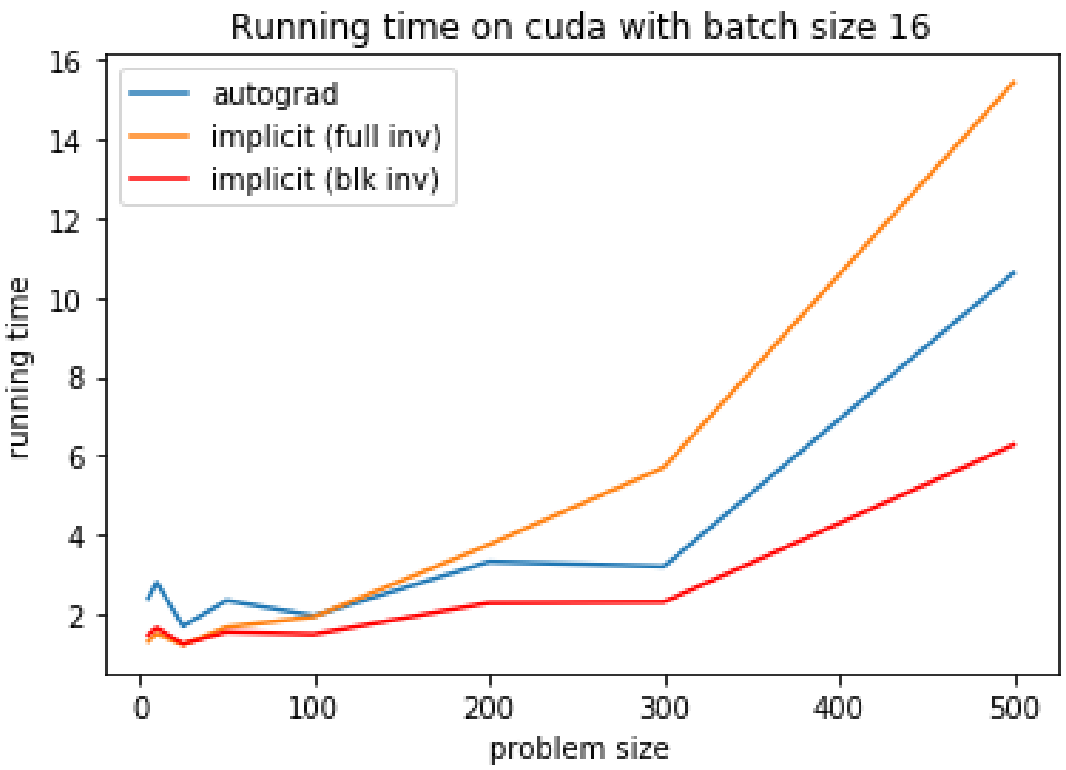
$$\gamma v^T \vec{P} [A_1^T \quad A_2^T] \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{12}^T & \Lambda_{22} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{P} - \gamma v^T \vec{P}$$

```
# initialize backward gradients (-v^T H^{-1} B)
dJdM = -1.0 * gamma * P * dJdP

# compute [vHAt1, vHAt2] = -v^T H^{-1} A^T
vHAt1 = torch.sum(dJdM[:, 1:m, 0:n], dim=2)
vHAt2 = torch.sum(dJdM, dim=1)

# compute [v1, v2] = -v^T H^{-1} A^T (A H^{-1] A^T)^{-1}
P_over_c = P[:, 1:m, 0:n] / c.view(batches, 1, n)
block_11 = torch.cholesky(torch.diag_embed(r[:, 1:m]) - torch.einsum("bij,bkj->bik", P[:, 1:m, 0:n], P_over_c))
block_12 = torch.cholesky_solve(P_over_c, block_11)
block_22 = torch.diag_embed(1.0 / c) + torch.einsum("bji,bjk->bik", block_12, P_over_c)

v1 = torch.cholesky_solve(vHAt1.view(batches, m-1, 1), block_11).view(batches, m-1) - torch.einsum("bi,bji->bj", vHAt2, block_12)
v2 = torch.einsum("bi,bij->bj", vHAt2, block_22) - torch.einsum("bi,bij->bj", vHAt1, block_12)

# compute v^T H^{-1} A^T (A H^{-1] A^T)^{-1} A H^{-1} B - v^T H^{-1} B
dJdM[:, 1:m, 0:n] -= v1.view(batches, m-1, 1) * P[:, 1:m, 0:n]
dJdM -= v2.view(batches, 1, n) * P
```

Unrolling vs implicit differentiation

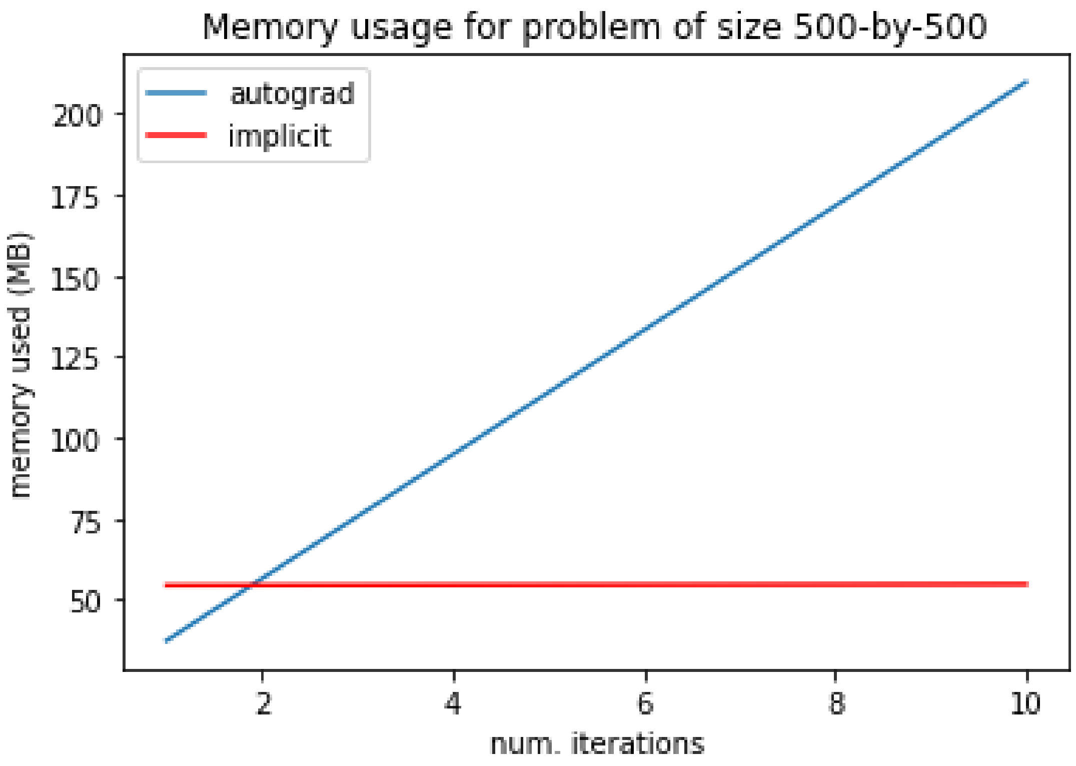
speed

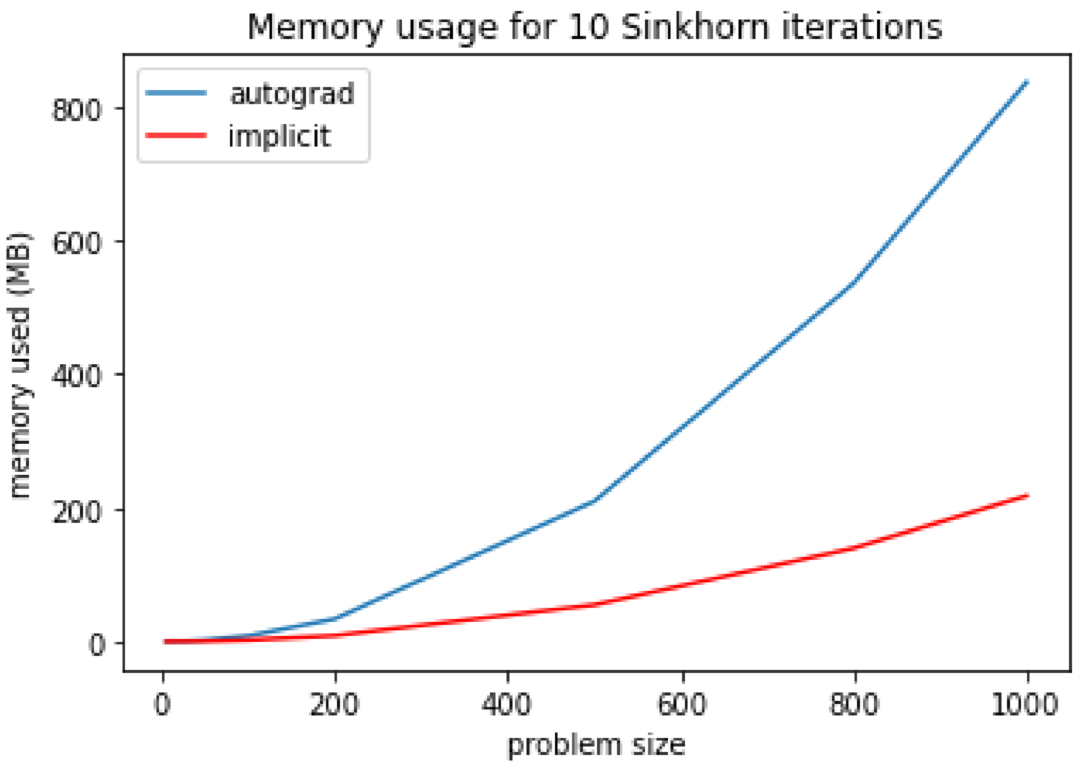




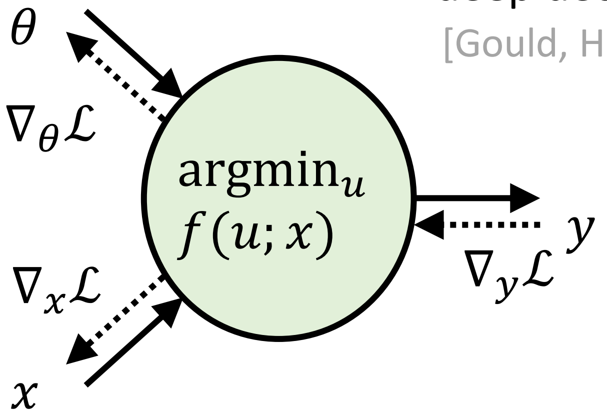
Unrolling vs implicit differentiation

memory



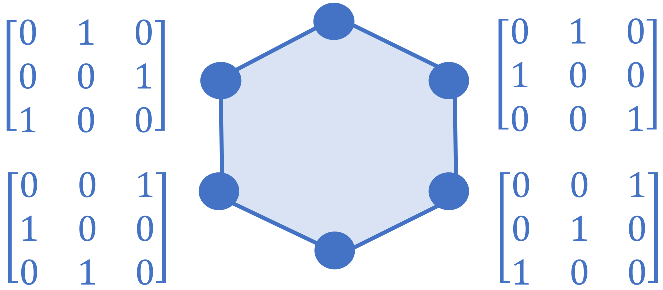


Summary & Questions



deep declarative networks
[Gould, Hartley and Campbell, PAMI 2021]

code and tutorials at
http://deepdeclarativenetworks.com



exploiting problem structure in DDNs: two case studies
[Gould, Campbell, Ben-Shabat, Koneputugodage
and Xu, OT-SDM@AAAI 2022]

collaborators