Deep Declarative Networks: A New Hope

A/Prof. Stephen Gould

Research School of Computer Science
The Australian National University
2019

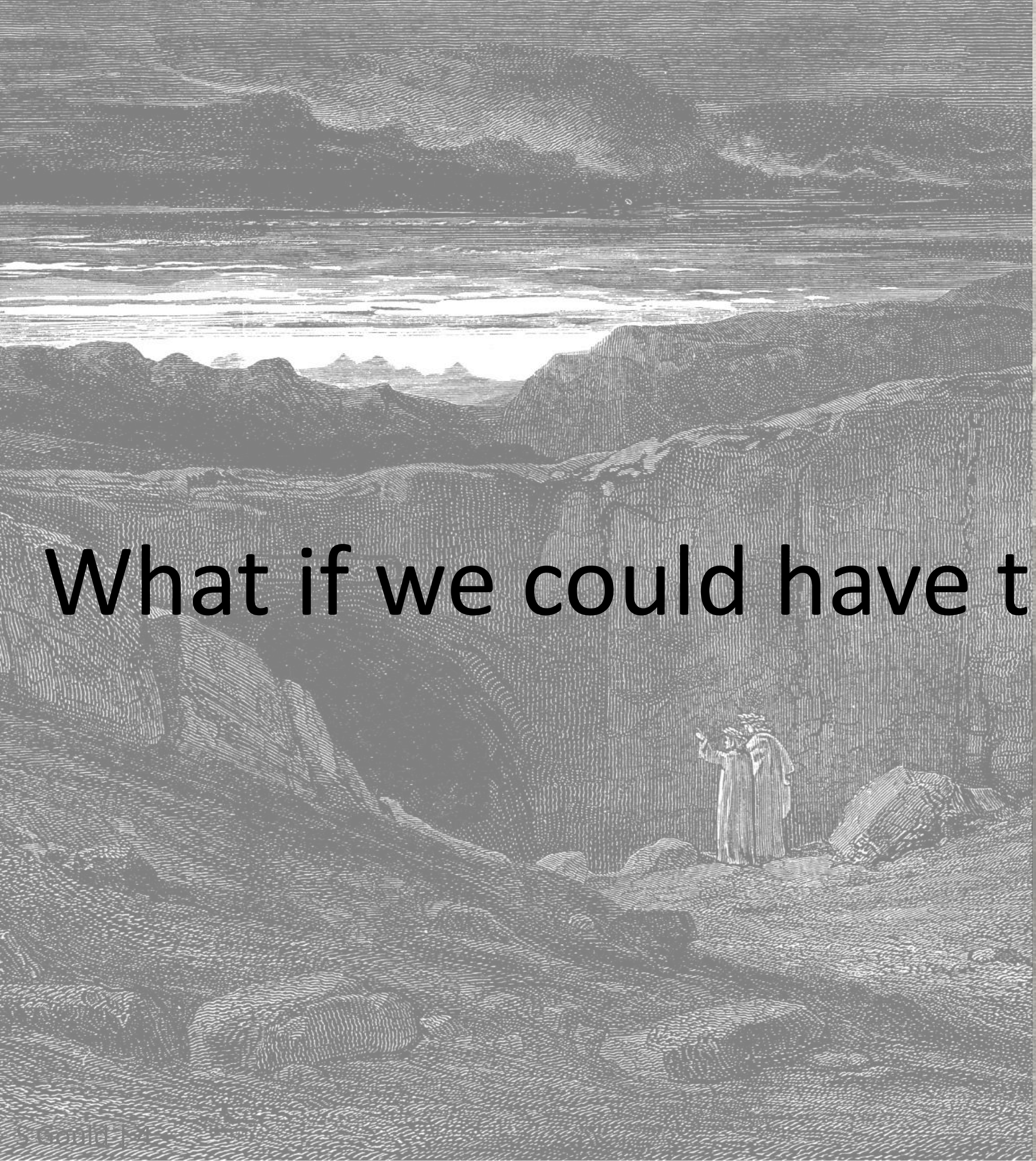

slide credit: Dylan Campbell

What did we gain?

- ✓ Better-than-human performance on closed-world classification tasks
- ✓ Very fast inference (with the help of GPU acceleration)
  - ✓ versus very slow iterative optimization procedures
- ✓ Common tools and software frameworks for sharing research code
- ✓ Robustness to variations in real-world data if training set is sufficiently large and diverse

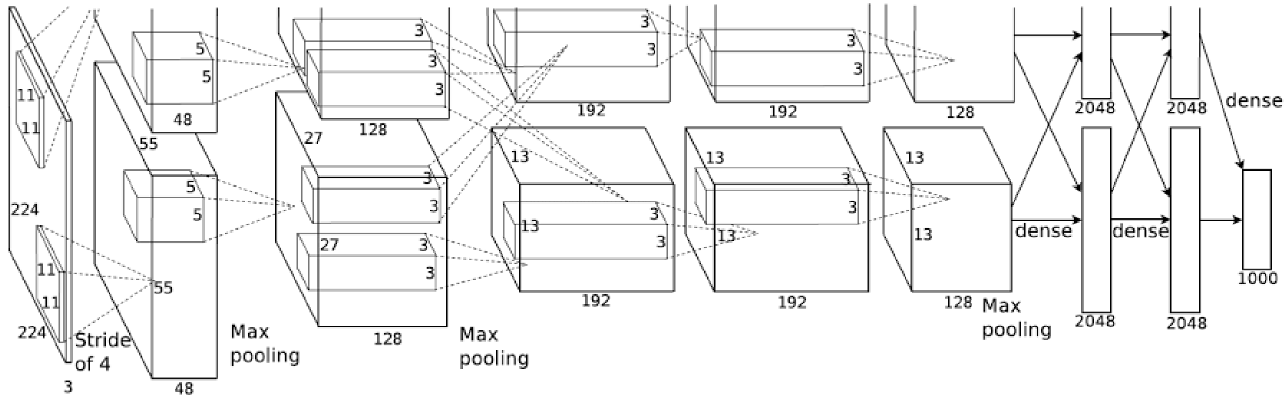
What did we lose?

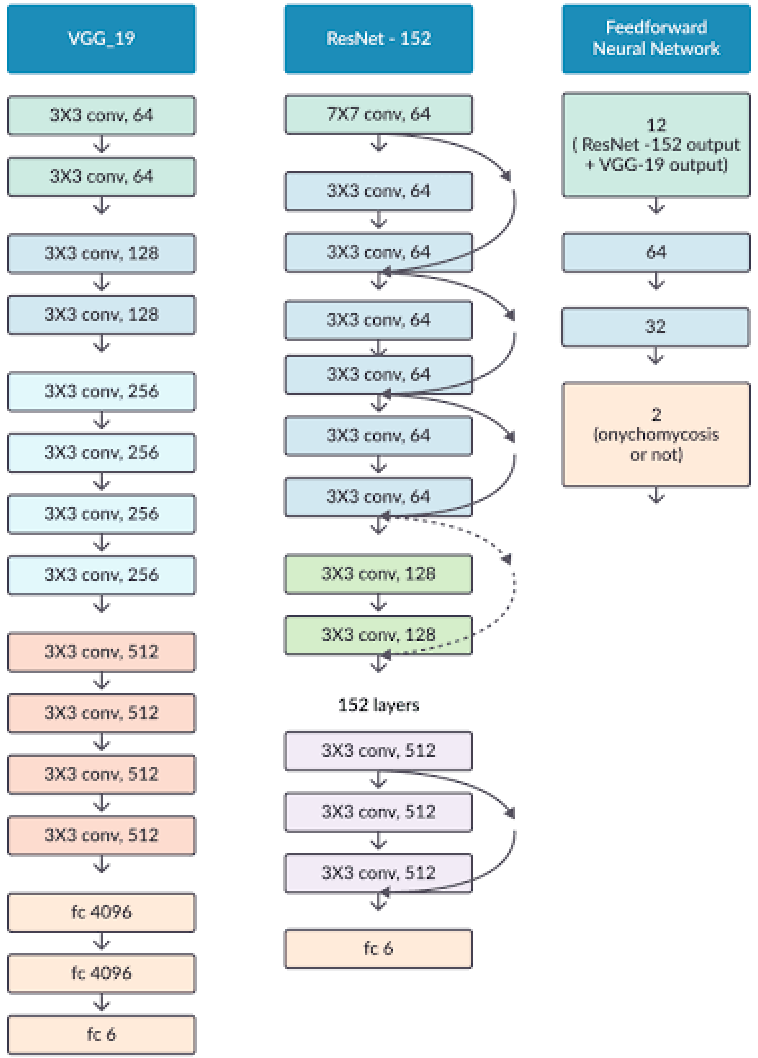
- ✗ Clear mathematical models; separation between algorithm and objective (loss function)
- ✗ Theoretical performance guarantees
- ✗ Interpretability and robustness to adversarial attacks
- ✗ Ability to enforce hard constraints
- ✗ Intuition guided by physical models
- ✗ Parsimony – capacity consumed learning what we already know



What if we could have the best of both worlds?

Deep learning models

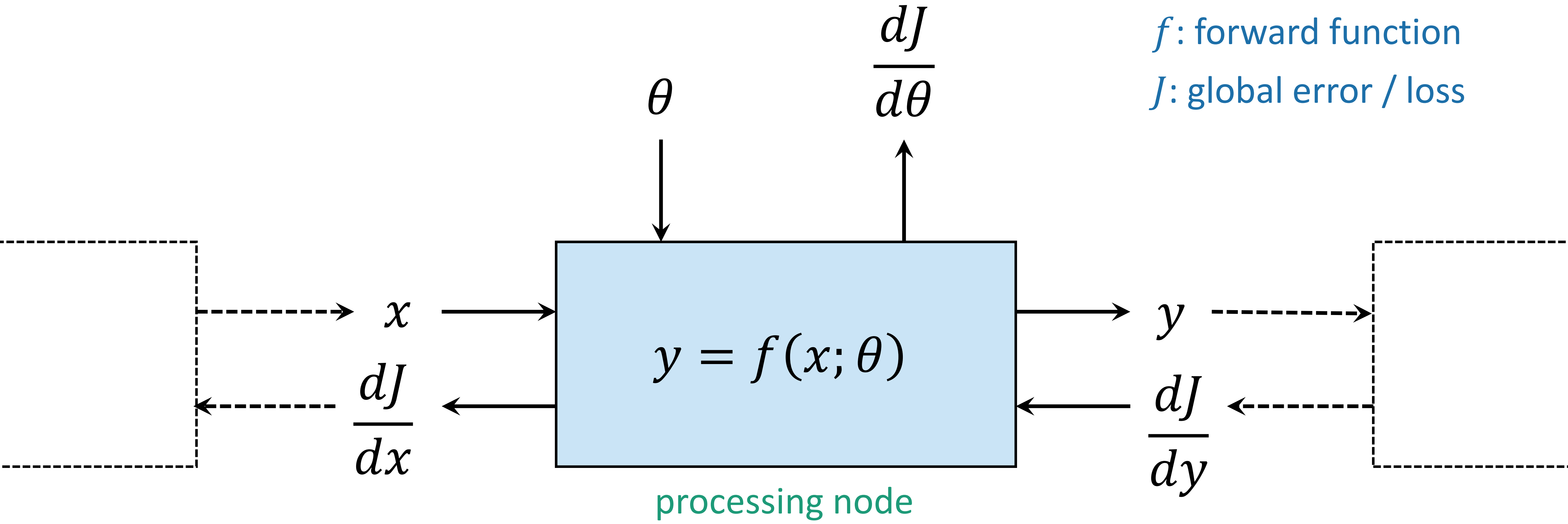




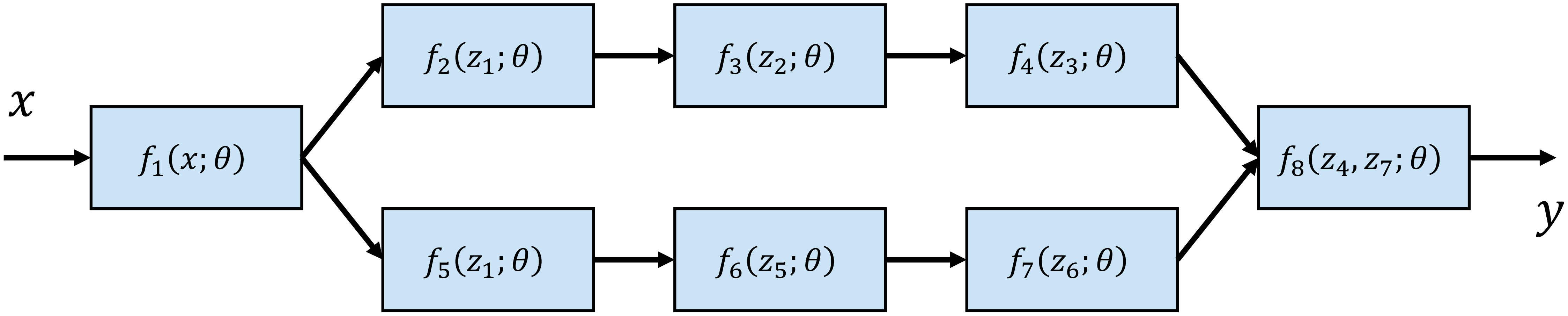
- Linear transforms (i.e., convolutions)
- Elementwise non-linear transforms
- Spatial/global pooling

Deep learning layer

x: input
y: output
θ: local parameters
f: forward function
J: global error / loss



End-to-end computation graph

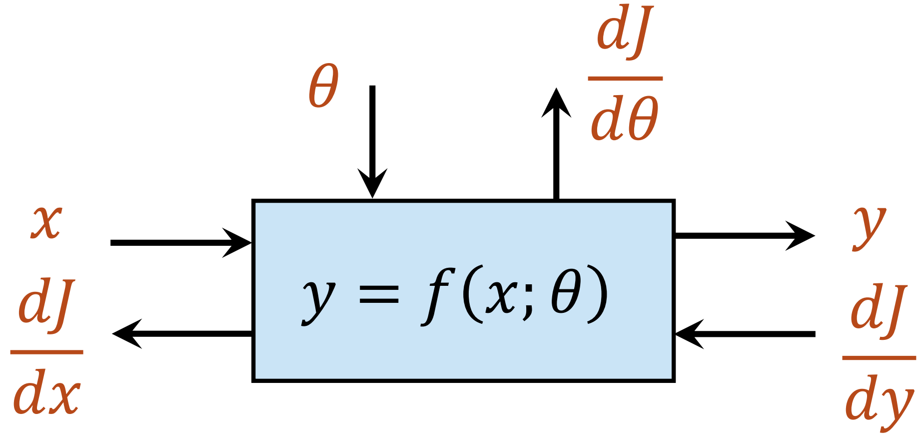


$$y = f_8\left(f_4\left(f_3\left(f_2\left(f_1(x)\right)\right)\right), f_7\left(f_6\left(f_5\left(f_1(x)\right)\right)\right)\right)$$

End-to-end learning

- Learning is about finding parameters that maximize performance,
$$\text{argmax}_\theta \quad \text{performance}(\text{model}(\theta))$$
- To do so we need to understand how the model output changes as a function of its input and parameters
- (Local based learning) incrementally updates parameters based on a signal back-propagated from the output of the network
- This requires calculation of gradients

$$\frac{dJ}{dx} = \frac{dJ}{dy}\frac{dy}{dx} \text{ and } \frac{dJ}{d\theta} = \frac{dJ}{dy}\frac{dy}{d\theta}$$



Example: Back-propagation through a node

Consider the following implementation of a node

```
fwd_fcn(x)
  y_0 = (1/2) x
  for t = 1, …, T do
      y_t ← (1/2)(y_{t−1} + x / y_{t−1})
  return y_T
```

$$y_0 = \frac{1}{2}x$$

$$y_t \leftarrow \frac{1}{2}\left(y_{t-1} + \frac{x}{y_{t-1}}\right)$$

We can back-propagate gradients as

$$\frac{\partial y_t}{\partial y_{t-1}} = \frac{1}{2}\left(1 - \frac{x}{y_{t-1}^2}\right)$$

$$\frac{\partial y_t}{\partial x} = \frac{1}{2y_{t-1}} + \frac{\partial y_t}{\partial y_{t-1}}\frac{\partial y_{t-1}}{\partial x}$$

It turns out that this node implements the Babylonian algorithm, which computes

$$y = \sqrt{x}$$

As such we can compute its derivative directly as

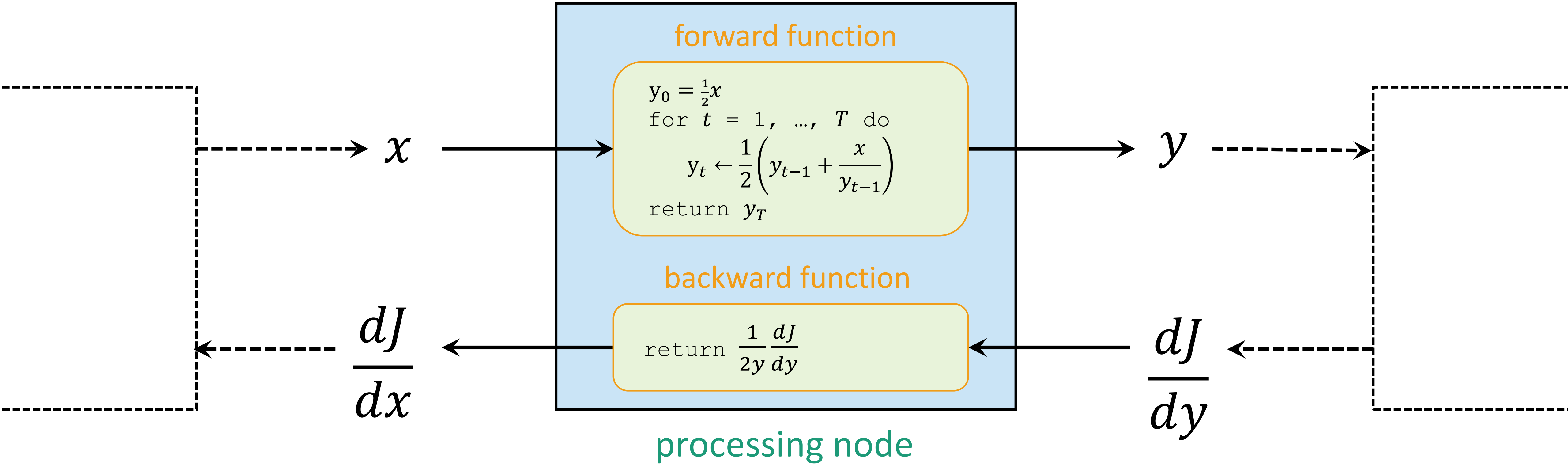
$$\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{x}}$$

$$= \frac{1}{2y}$$

```
bck_fcn(x, y)
  return 1/(2y)
```

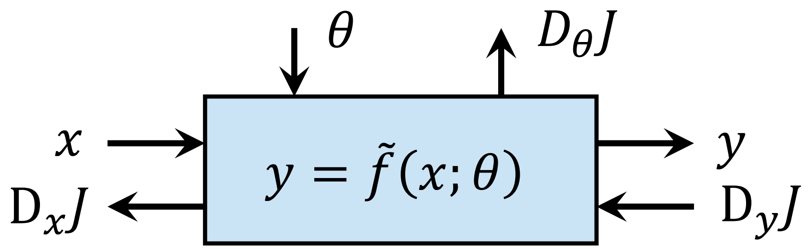
Chain rule gives $\frac{\partial J}{\partial x}$ from $\frac{\partial J}{\partial y}$ (input) and $\frac{\partial y}{\partial x}$ (computed)

Separate of forward and backward operations



[Gould et al., 2019]

Deep declarative networks (DDNs)



In an **imperative node** the implementation of the forward processing function $\tilde{f}$ is explicitly defined. The output is then

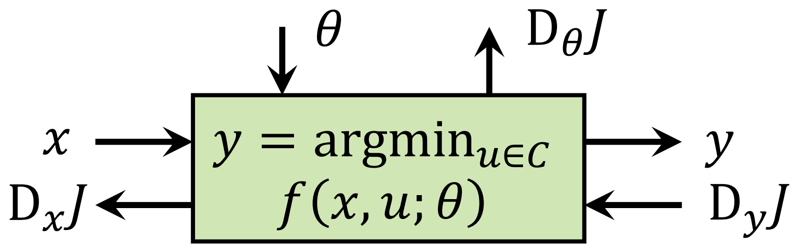
$$y = \tilde{f}(x; \theta)$$

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 vs. declarative node example: global average pooling

$$\{x_i \in \mathbb{R}^m \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$

Imperative specification:

$$y = \frac{1}{n}\sum_{i=1}^{n} x_i$$

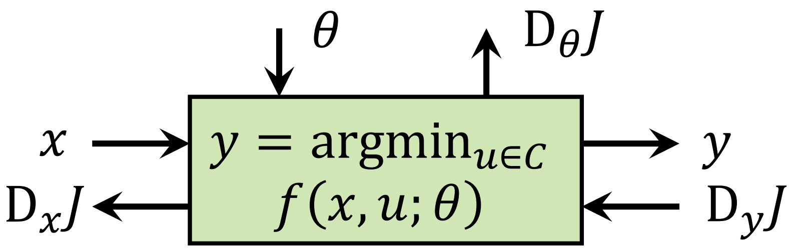
Declarative specification:

$$y = \operatorname*{argmin}_{u \in \mathbb{R}^m} \sum_{i=1}^{n} \|u - x_i\|^2$$

"the vector u that is the minimum distance to all input vectors x_i"

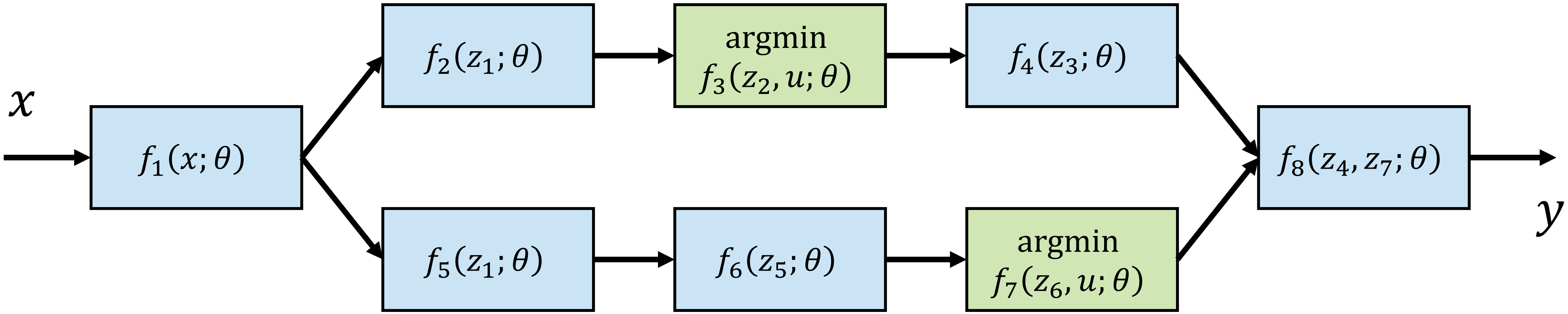
[Gould et al., 2019]

Deep declarative nodes: special cases



Unconstrained (e.g., robust pooling)	$y(x) \in \mathrm{argmin}_{u \in \mathbb{R}^m} f(x, u)$
Equality Constrained (e.g., projection onto L_p-sphere)	$y(x) \in \left\{ \begin{array}{c} \mathrm{argmin}_{u \in \mathbb{R}^m} f(x, u) \\ \text{subject to } h(x, u) = 0 \end{array} \right\}$
Inequality Constrained (e.g., projection onto L_p-ball)	$y(x) \in \left\{ \begin{array}{c} \mathrm{argmin}_{u \in \mathbb{R}^m} f(x, u) \\ \text{subject to } h(x, u) \le 0 \end{array} \right\}$

Imperative and declarative nodes can co-exist



$$y = f_8\left(f_4\left(\operatorname{argmin} f_3(f_2(f_1(x)),u)\right), \operatorname{argmin} f_7\left(f_6\left(f_5(f_1(x))\right),u\right)\right)$$

Learning as bi-level optimization

learning problem

minimize (over $\boldsymbol{x}$)	objective($\boldsymbol{x}$)
subject to	constraints($\boldsymbol{x}$)

bi-level learning problem

minimize (over $\boldsymbol{x}$)	objective($\boldsymbol{x}$, $\boldsymbol{y}$)
subject to	constraints($\boldsymbol{x}$)

declarative node problem

minimize (over $\boldsymbol{y}$)	objective($\boldsymbol{x}$, $\boldsymbol{y}$)
subject to	constraints($\boldsymbol{y}$)

A game theoretic perspective

- Consider two players, a **leader** and a **follower**
  - The market dictates the price its willing to pay for some goods based on supply, i.e., quantity produced by both players, $\boldsymbol{P(q_1 + q_2)}$
  - Each player has a cost structure associated with producing goods, $\boldsymbol{C_i(q_i)}$ and wants to maximize profits, $\boldsymbol{q_i P(q_1 + q_2) - C_i(q_i)}$
  - The **leader** picks a quantity of goods to produce knowing that the **follower** will respond optimally. In other words, the **leader** solves

$$
\begin{array}{ll}
\text{maximize}_{q_1} & q_1 P(q_1 + q_2) - C_1(q_1) \\
\text{subject to} & q_2 \in \text{argmax}_q \; qP(q_1 + q) - C_2(q)
\end{array}
$$

[Bard, 1998; Dempe and Franke, 2015]

Solving bi-level optimization problems

$$\begin{array}{ll} \text{minimize}_x & J(x, y) \\ \text{subject to} & y \in \text{argmin}_u f(x, u) \end{array}$$

- **Closed-form lower-level problem:** substitute for $\boldsymbol{y}$ in upper problem

$$\text{minimize}_x \quad J(x, y(x))$$

  - May result in a difficult (single-level) optimization problem

[Bard, 1998; Dempe and Franke, 2015]

Solving bi-level optimization problems

$$\begin{array}{ll} \text{minimize}_x & J(x, y) \\ \text{subject to} & y \in \text{argmin}_u f(x, u) \end{array}$$

- **Convex lower-level problem:** replace lower problem with sufficient conditions (e.g., KKT conditions)

$$\begin{array}{ll} \text{minimize}_{x,y} & J(x, y) \\ \text{subject to} & h(y) = 0 \end{array}$$

  - May result in non-convex problem if KKT conditions are not convex

[Bard, 1998; Dempe and Franke, 2015]

Solving bi-level optimization problems

$$\begin{array}{ll} \text{minimize}_x & J(x, y) \\ \text{subject to} & y \in \text{argmin}_u f(x, u) \end{array}$$

- **Gradient descent:** compute gradient with respect to $\boldsymbol{x}$

$$x \leftarrow x - \eta \left(\frac{\partial J(x, y)}{\partial x} + \frac{\partial J(x, y)}{\partial y} \frac{dy}{dx} \right)$$

  - But this requires computing the gradient of $\boldsymbol{y}$ (itself a function of $\boldsymbol{x}$)

[Bard, 1998; Dempe and Franke, 2015; Gould et al., 2019]

Algorithm for solving bi-level optimization

SolveBiLevelOptimization:

initialize x

repeat until convergence:

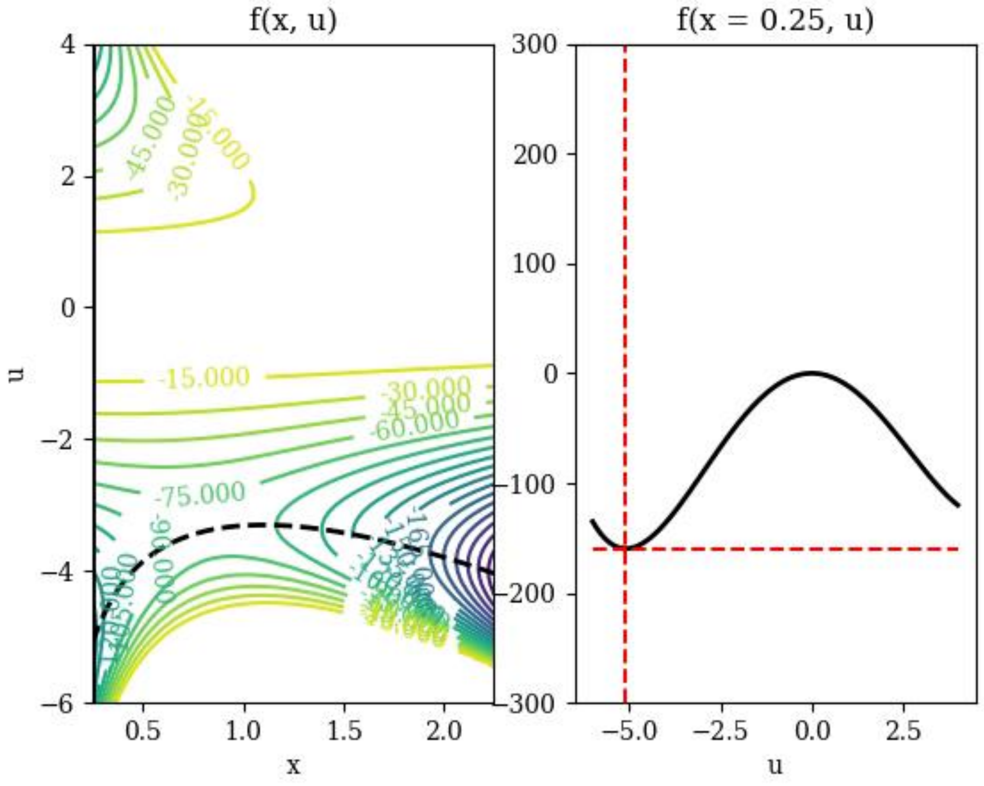
  solve $y \in \text{argmin}_u f(x, u)$

  compute $J(x, y)$

  compute $\frac{dJ}{dx} = \frac{\partial J(x,y)}{\partial x} + \frac{\partial J(x,y)}{\partial y}\frac{dy}{dx}$

  update $x \leftarrow x - \eta \frac{dJ}{dx}$

return x



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

[Krantz and Parks, 2013; Dontchev and Rockafellar, 2014; Gould et al., 2016; Gould et al., 2019]

Implicit differentiation

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

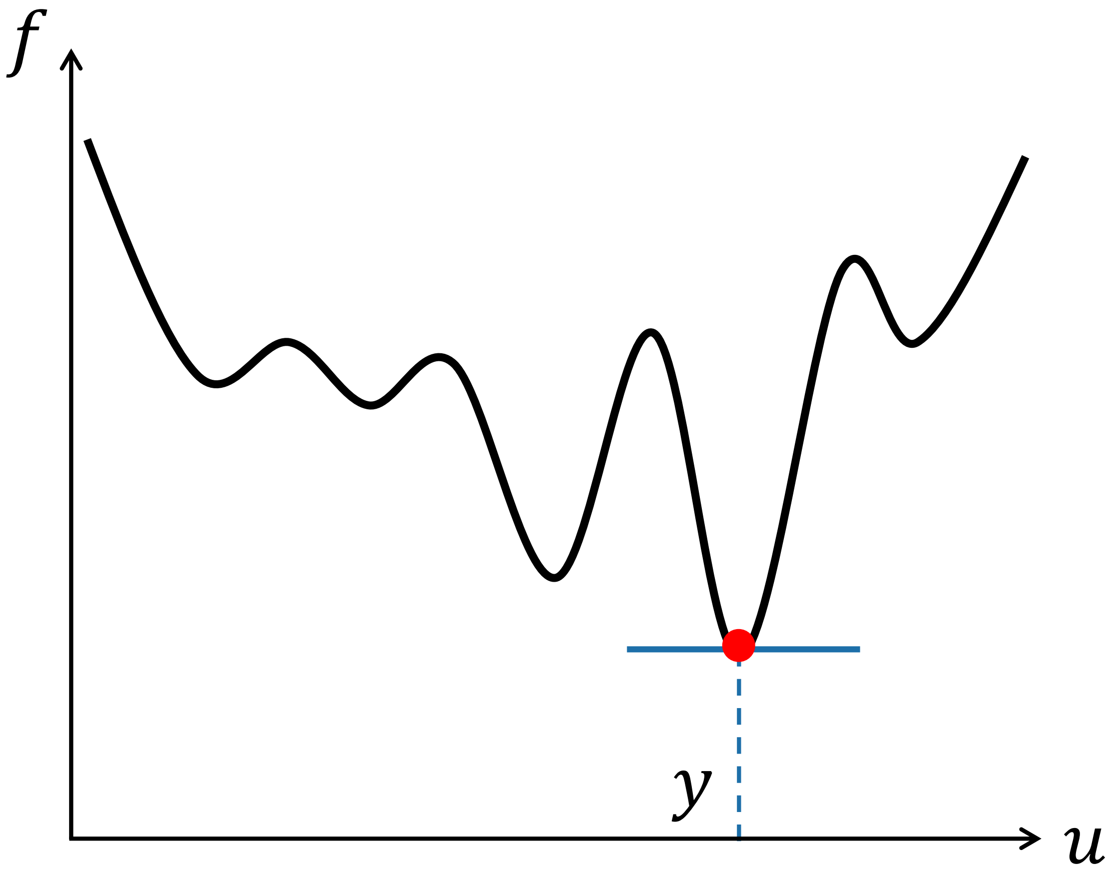
$$y(x) = \text{argmin}_u f(x, u)$$

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

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

The result extends to vector-valued functions, vector-argument functions and (equality) constrained problems. See [Gould et al., 2019].

Proof sketch



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

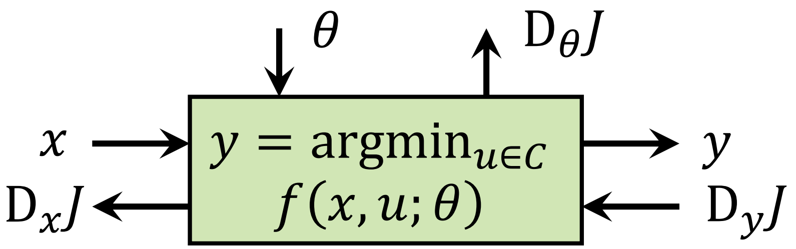
LHS: $$\frac{d}{dx}\frac{\partial f(x,y)}{\partial y} = \frac{\partial^2 f(x,y)}{\partial x \partial y} + \frac{\partial^2 f(x,y)}{\partial y^2}\frac{dy}{dx}$$

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

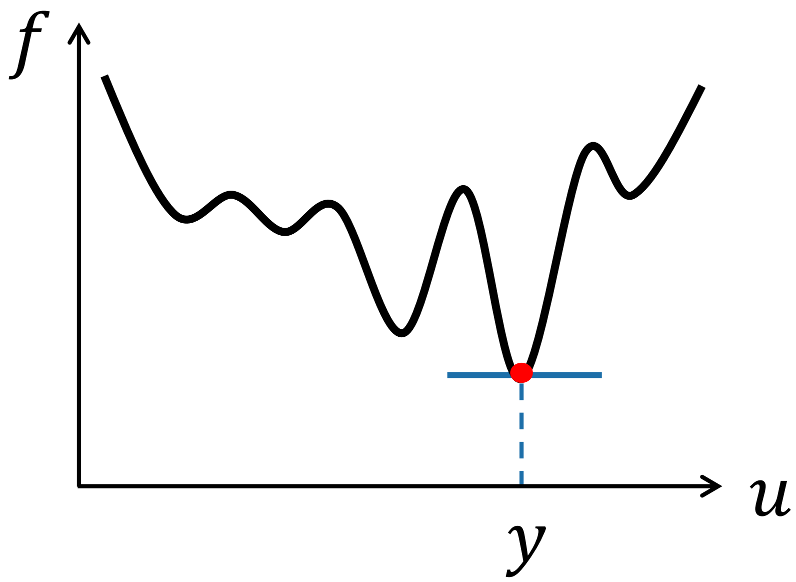
Rearranging gives $\frac{dy}{dx} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}$.

[Gould et al., 2019]

Deep declarative nodes: what do we need?



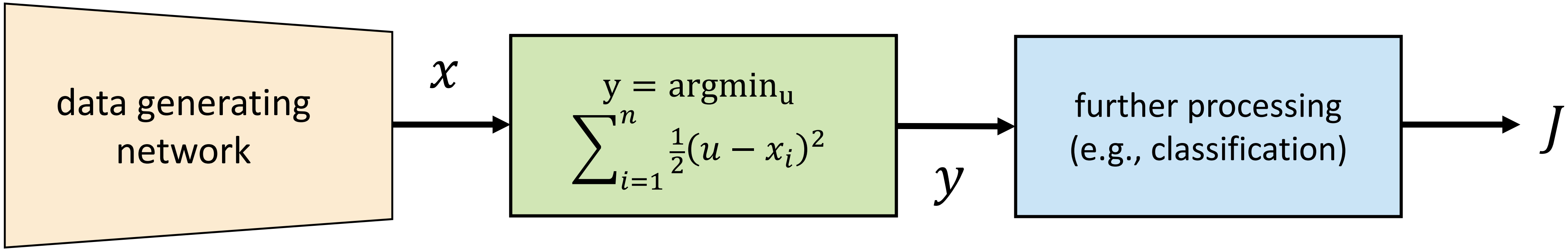
- **Forward pass**
  - A method to solve the optimization problem
- **Backward pass**
  - Specification of objective and constraints
  - (And cached result from the forward pass)
  - Do **not** need to know how the problem was solved



examples

Global average pooling

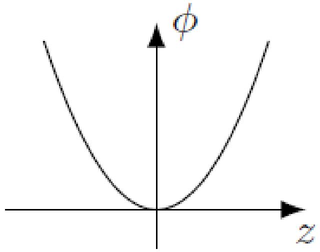
$$\{x_i \in \mathbb{R}^m \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$



Robust penalty functions, ϕ

Quadratic

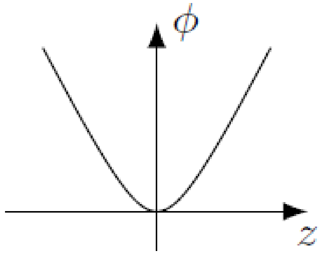
$$\tfrac{1}{2}z^2$$



closed-form, convex, smooth, unique solution

Pseudo-Huber

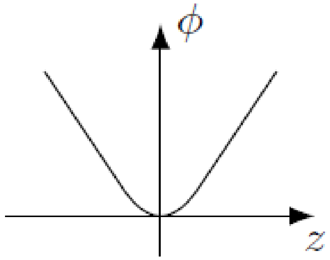
$$\sqrt{1 + \left(\frac{z}{\alpha}\right)^2} - 1$$



convex, smooth, unique solution

Huber

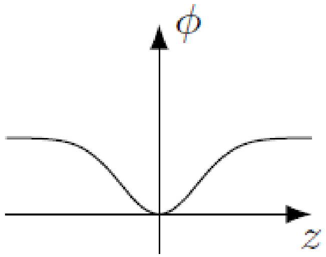
$$\begin{cases} \tfrac{1}{2}z^2 \text{ for } |z| \leq \alpha \\ \text{else } \alpha(|z| - \tfrac{1}{2}\alpha) \end{cases}$$



convex, non-smooth, non-isolated solutions

Welsch

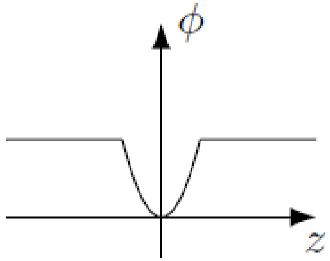
$$1 - \exp\left(\frac{-z^2}{2\alpha^2}\right)$$



non-convex, smooth, isolated solutions

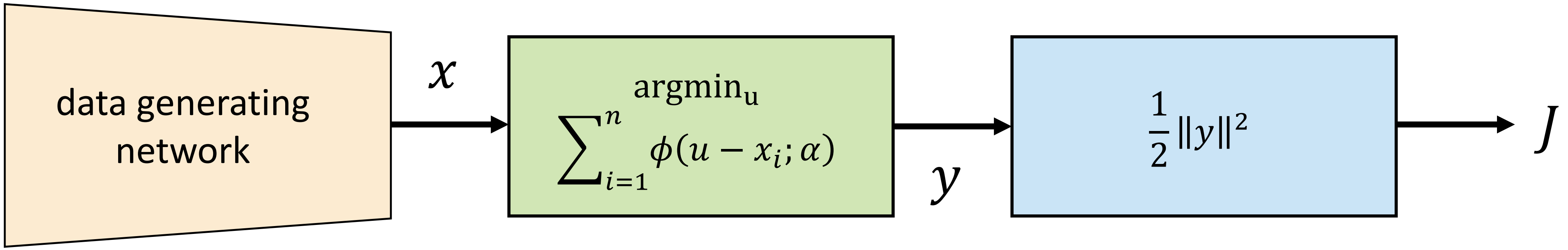
Truncated Quad.

$$\begin{cases} \tfrac{1}{2}z^2 & \text{for } |z| \leq \alpha \\ \tfrac{1}{2}\alpha^2 & \text{otherwise} \end{cases}$$

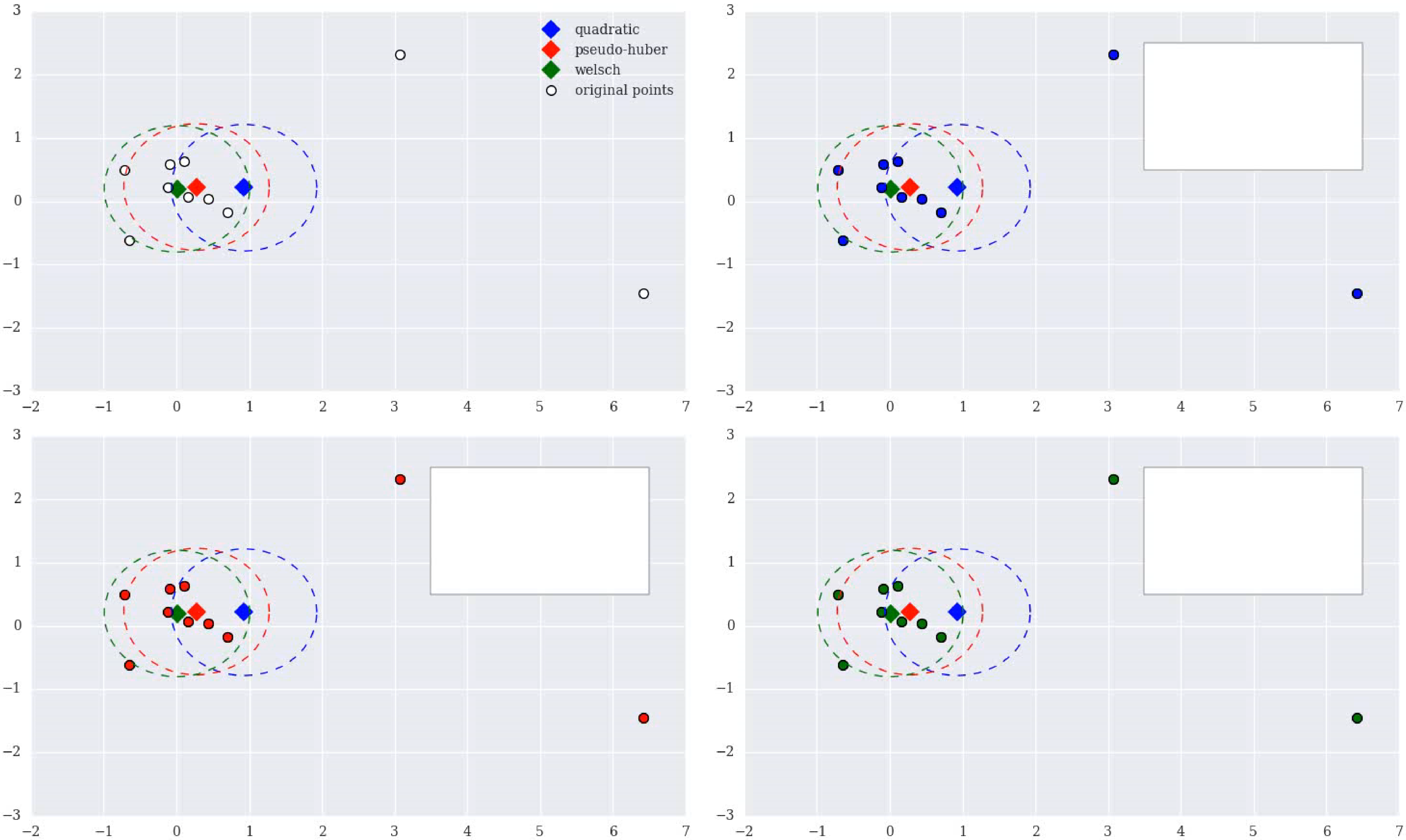


non-convex, non-smooth, isolated solutions

Example: robust pooling



minimize (over x)	$J(x, y) \triangleq \frac{1}{2}\|y\|^2$
subject to	$y \in \text{argmin}_u \sum_{i=1}^{n} \phi(u - x_i; \alpha)$

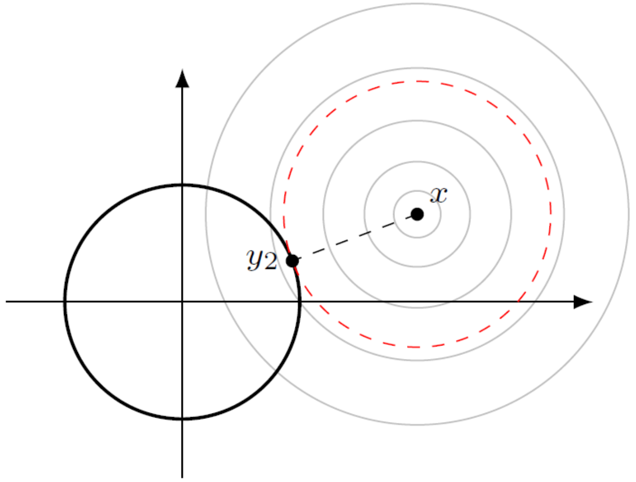


quadratic
pseudo-huber
welsch
original points

[Gould et al., 2019]

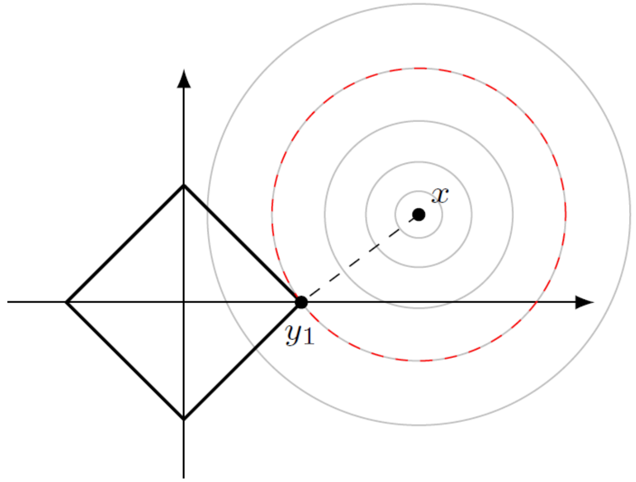
Example: Euclidean projection

L_2



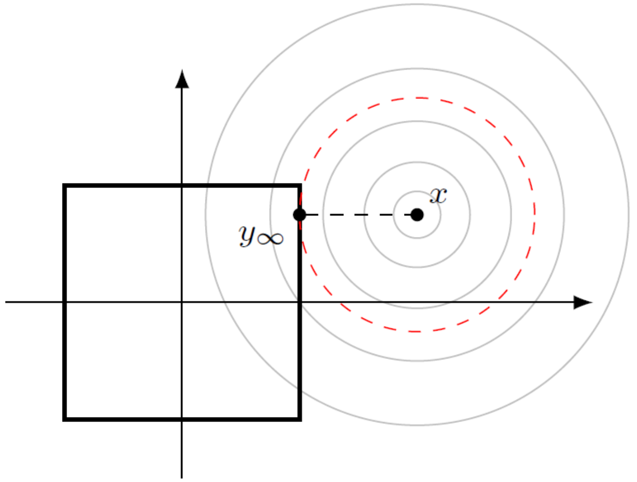
closed-form, smooth,
unique solution*

L_1



non-smooth, isolated solutions

L_∞



non-smooth,
isolated solutions

[Amos and Kolter, 2017]

Example: quadratic programs

$$
\begin{array}{ll}
\text{argmin}_{u \in \mathbb{R}^m} & \frac{1}{2} u^T P u + q^T u + r \\
\text{subject to} & Au = b \\
 & Gu \leq h
\end{array}
$$

Can be differentiated with respect to its parameters:

$$P \in \mathbb{R}^{m \times m}, \qquad q \in \mathbb{R}^m, \qquad A \in \mathbb{R}^{n \times m}, \qquad b \in \mathbb{R}^n$$

[Agrawal et al., 2019]

Example: convex programs

$$\begin{array}{ll} \text{argmin}_{u \in \mathbb{R}^m} & c^T u \\ \text{subject to} & b - Au \in K \end{array}$$

Can be differentiated with respect to its parameters:

$$A \in \mathbb{R}^{n \times m}, \qquad b \in \mathbb{R}^n, \qquad c \in \mathbb{R}^m$$

[Gould et al., 2019]

Implementing deep declarative nodes

- Need: objective and constraint functions, solver to obtain y
- Gradient by automatic differentiation

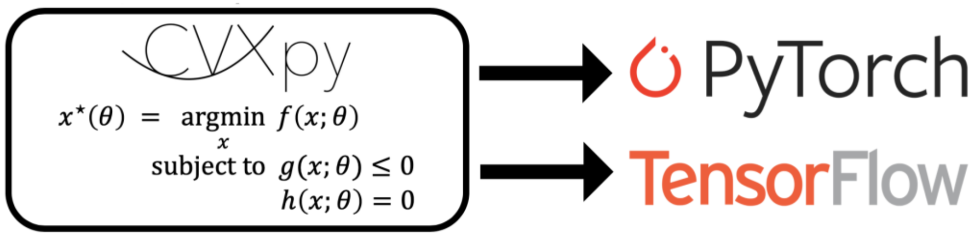
$$\frac{dy(x)}{dx} = -\left(\frac{\partial^2 f}{\partial y^2}\right)^{-1} \frac{\partial^2 f}{\partial x \partial y}$$

```
import autograd.numpy as np
from autograd import grad, jacobian

def gradient(x, y, f)
  fY = grad(f, 1)
  fYY = jacobian(fY, 1)
  fXY = jacobian(fY, 0)
  return -1.0 * np.linalg.solve(fYY(x,y), fXY(x,y))
```

cvxpylayers

- Disciplined convex optimization
  - Subset of optimization problems

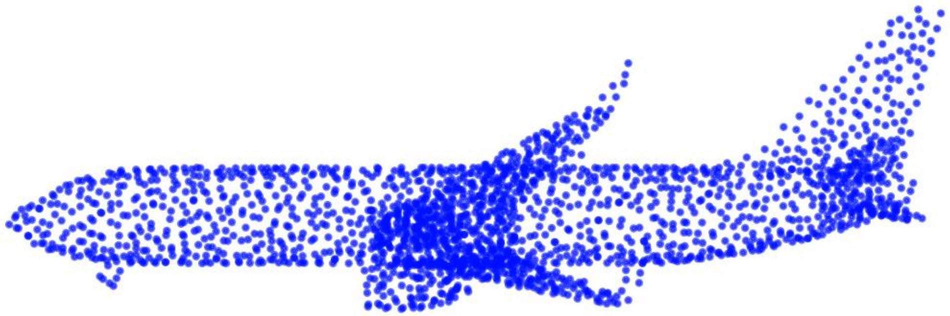


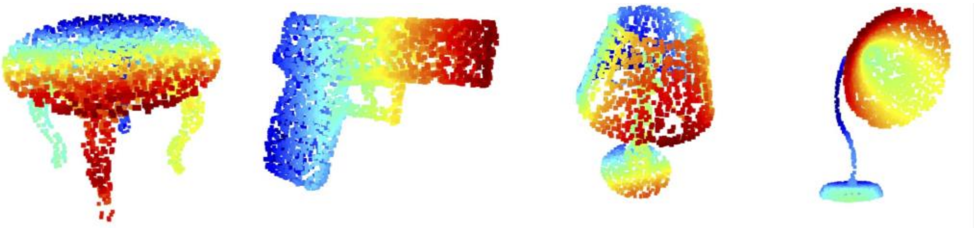
- Write problem using `cvx`
  - Solver and gradient computed automatically!

```
x = cp. Parameter(n)
y = cp. Variable(n)
obj = cp. Minimize(cp.sum_squares(x - y ))
cons = [ y >= 0]
prob = cp. Problem(obj, cons)
layer = CvxpyLayer(prob, parameters=[x], variables=[y])
```

applications

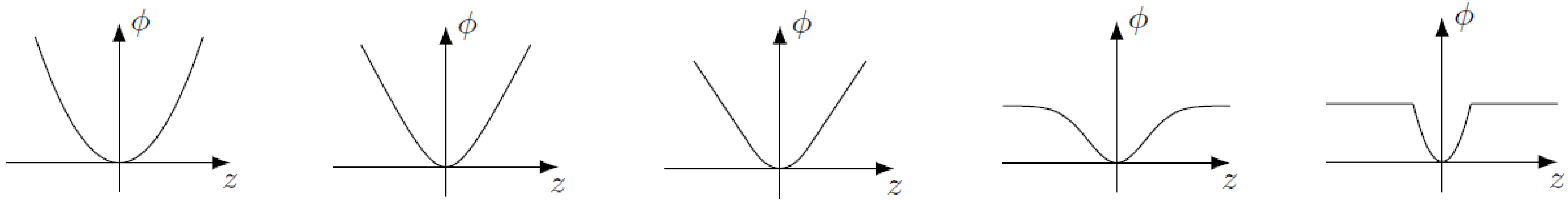
Robust point cloud classification



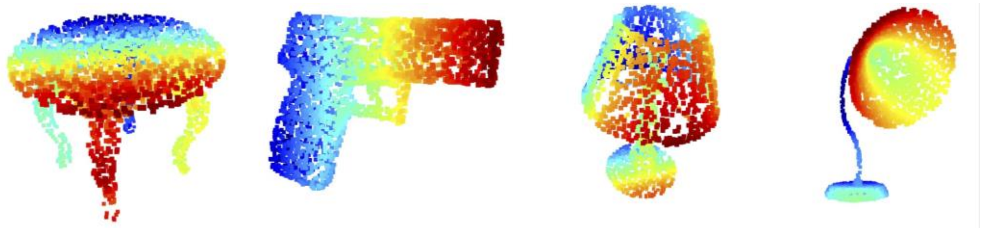


[Gould et al., 2019]

Robust point cloud classification



O	Top-1 Accuracy %						Mean Average Precision ×100					
%	[34]	Q	PH	H	W	TQ	[34]	Q	PH	H	W	TQ
0	88.4	84.7	84.7	86.3	86.1	85.4	**95.6**	93.8	95.0	95.4	95.0	93.8
10	79.4	84.3	85.6	85.5	**86.6**	85.5	89.4	94.3	94.6	**95.1**	94.6	94.7
20	76.2	84.8	84.8	85.2	**86.3**	85.5	87.8	94.8	95.0	**95.0**	94.8	95.0
50	72.0	84.0	83.1	83.9	**84.3**	83.9	83.3	93.8	93.5	94.3	94.8	**94.8**
90	29.7	61.7	63.4	63.1	**65.3**	61.8	38.9	76.8	78.7	78.5	**79.1**	76.6



O	Top-1 Accuracy %						Mean Average Precision ×100					
%	[34]	Q	PH	H	W	TQ	[34]	Q	PH	H	W	TQ
0	88.4	84.7	84.7	86.3	86.1	85.4	**95.6**	93.8	95.0	95.4	95.0	93.8
1	32.6	84.9	84.7	**86.4**	86.2	85.3	48.6	93.8	95.1	**95.3**	95.1	93.0
10	6.47	83.9	84.6	85.3	**86.0**	85.9	8.20	93.4	94.8	94.4	**94.9**	93.9
20	5.95	79.6	82.8	81.1	84.7	**84.9**	7.73	91.9	93.4	92.7	94.2	**94.6**
30	5.55	70.9	74.2	72.2	77.6	**83.2**	6.00	87.8	89.5	85.1	90.9	**92.8**
40	5.35	55.3	59.1	55.4	63.1	**75.6**	6.41	77.6	80.2	72.7	83.2	**90.6**
50	4.86	32.9	36.0	34.6	44.1	**57.9**	5.68	62.3	60.2	60.1	66.4	**85.3**
60	4.42	14.5	16.2	18.1	27.1	**30.6**	5.08	39.1	36.3	38.5	42.7	**68.5**
70	4.25	5.03	6.33	7.95	**14.1**	11.9	4.66	22.5	19.3	18.4	25.7	**47.9**
80	3.11	4.10	4.51	5.64	**8.88**	5.11	4.21	10.8	8.91	8.98	14.9	**26.7**
90	3.72	4.06	4.06	4.30	**5.68**	4.22	4.49	8.20	5.98	5.80	8.37	**9.78**

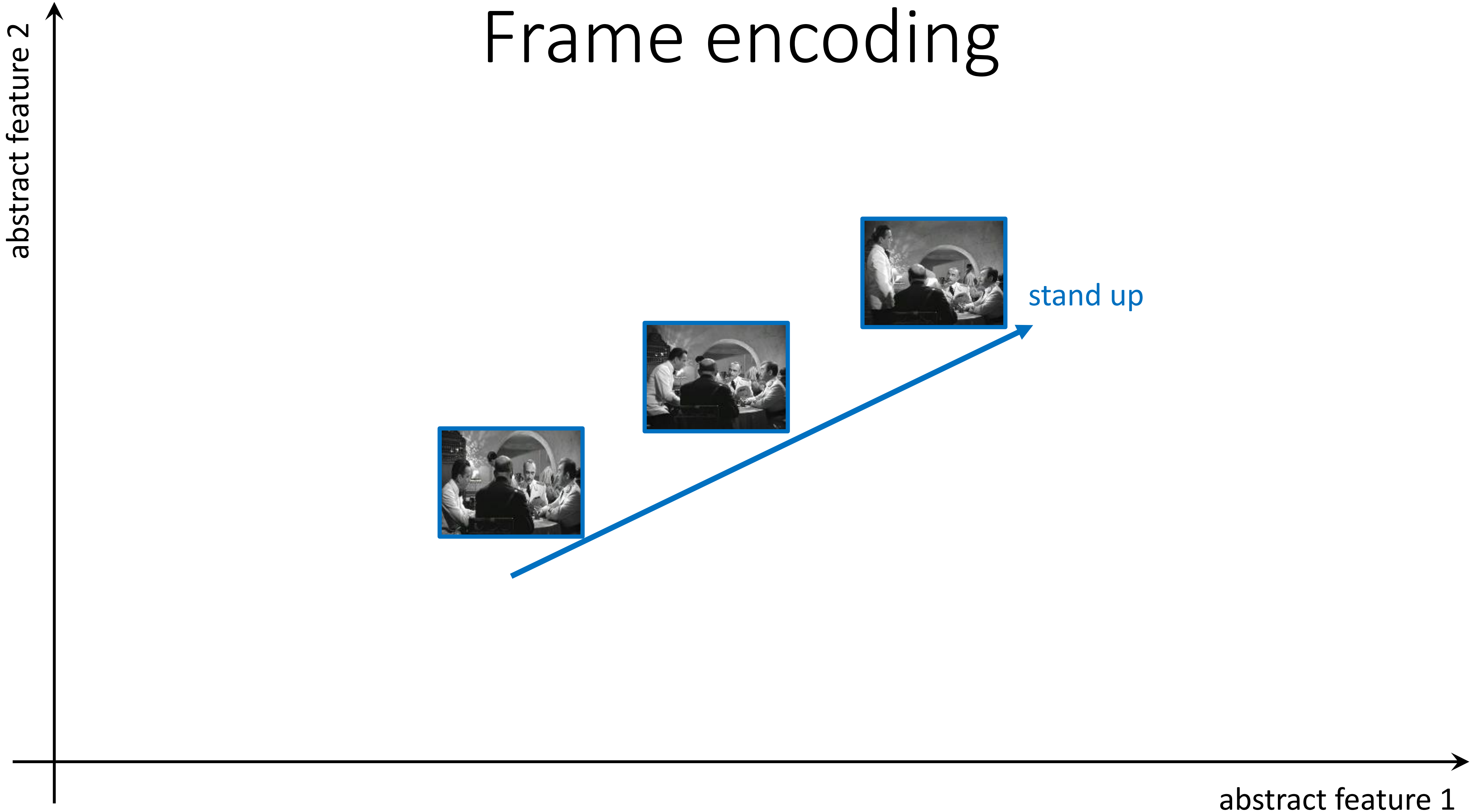
Video activity recognition

Stand Up

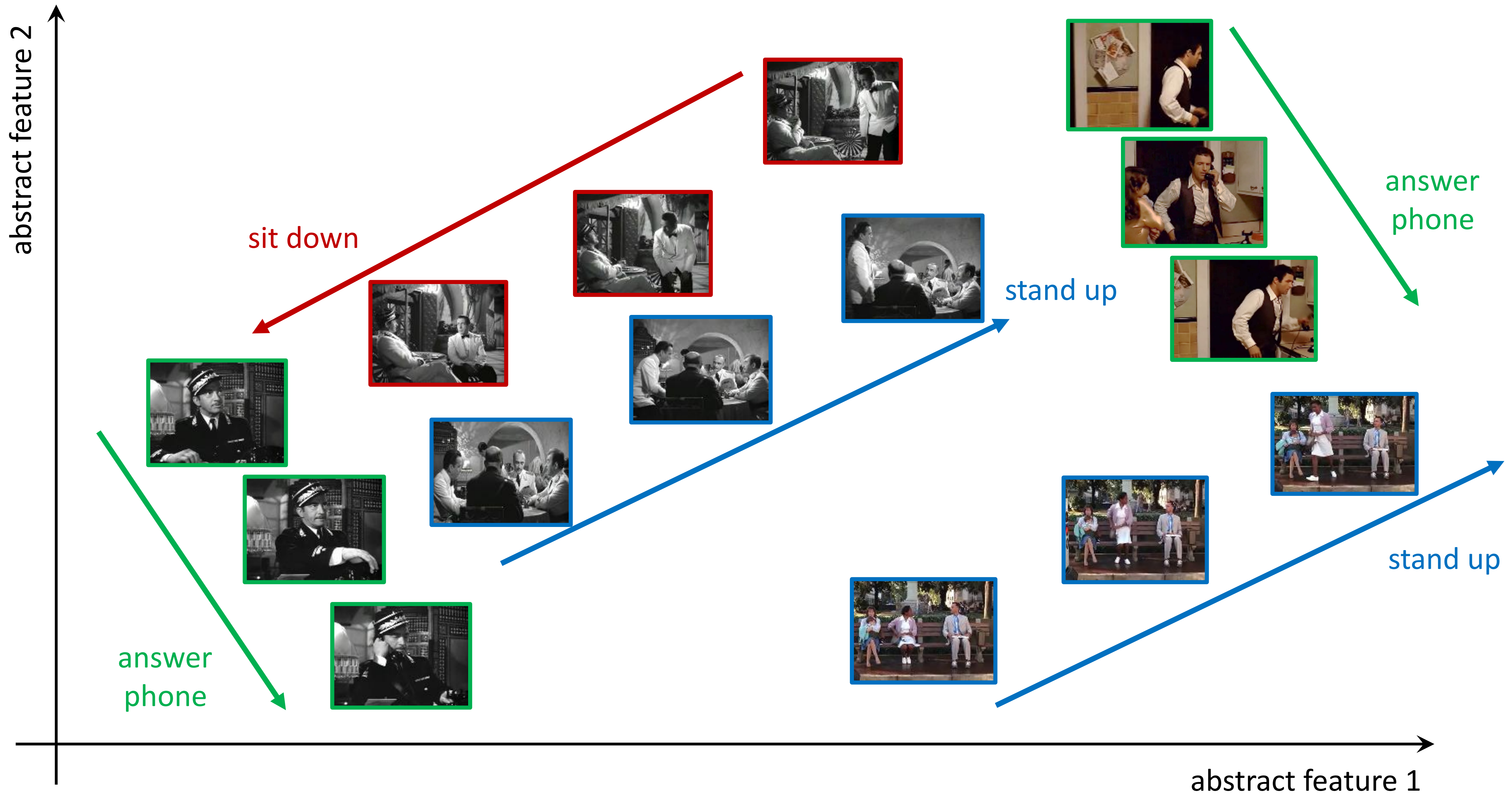
Sit Down

[Fernando and Gould, 2016]

Frame encoding



[Fernando and Gould, 2016]



Video clip classification pipeline

Temporal pooling

- Max/avg/robust pooling summarizes an **unstructured set** of objects

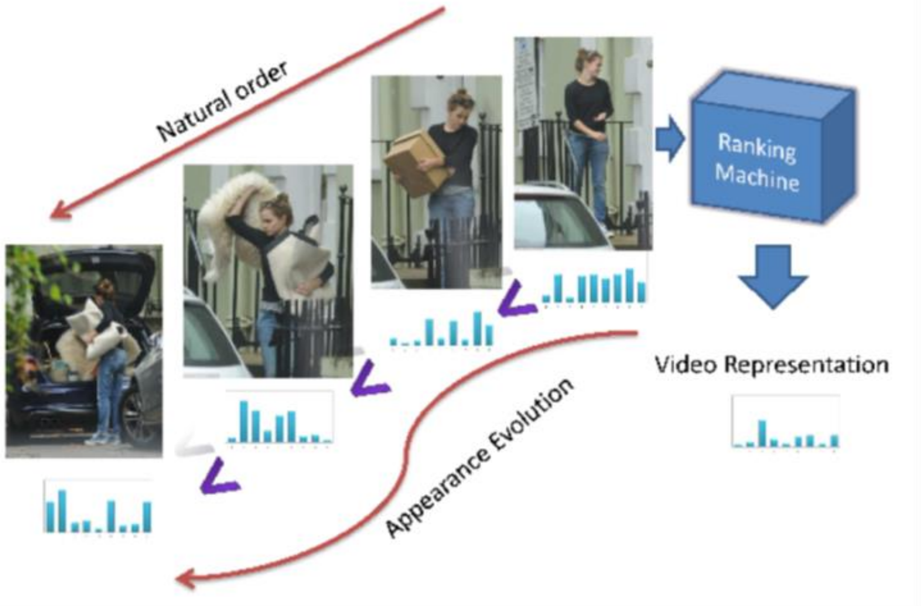
$$\{x_i \mid i = 1, \ldots, n\} \to \mathbb{R}^m$$

- Rank pooling summarizes a **structured sequence** of objects

$$\langle x_i \mid i = 1, \ldots, n \rangle \to \mathbb{R}^m$$

Rank Pooling

- Find a ranking function $r: \mathbb{R}^n \to \mathbb{R}$ such that $r(x_t) < r(x_s)$ for $t < s$
- In our case we assume that $r: x \mapsto u^T x$ is a linear function
- Use u as the representation



Experimental results

Method	Accuracy (%)
Max-Pool + SVM	66
Avg-Pool + SVM	67
Rank-Pool + SVM	66
Max-Pool-CNN (end-to-end)	71
Avg-Pool-CNN (end-to-end)	70
Rank-Pool-CNN (end-to-end)	**87**
Improved trajectory features + fisher vectors + rank-pooling	**87**

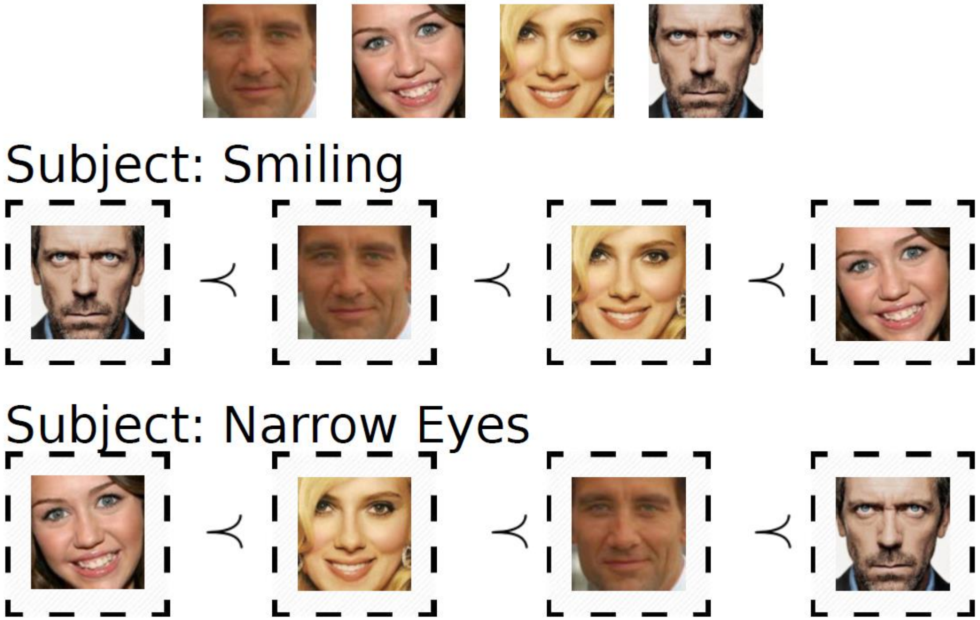
150 video clips from BBC and ESPN footage
10 sports actions

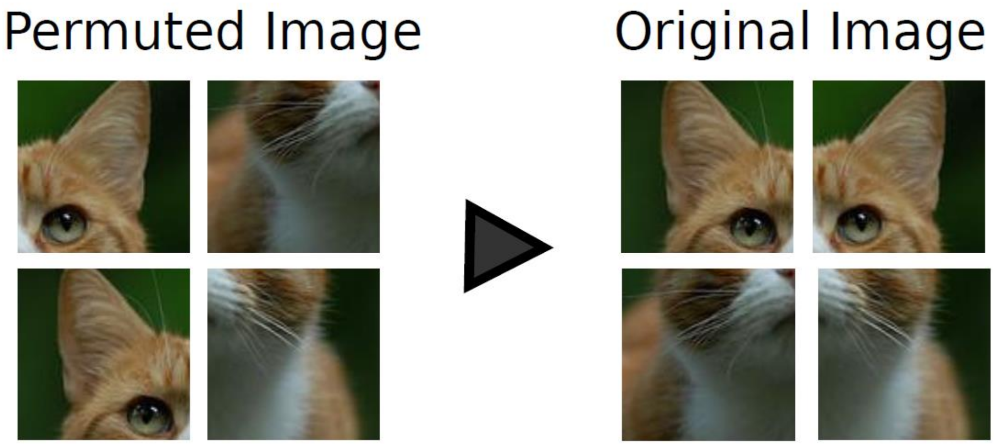
[Rodriguez et al., 2008]

21% improvement!

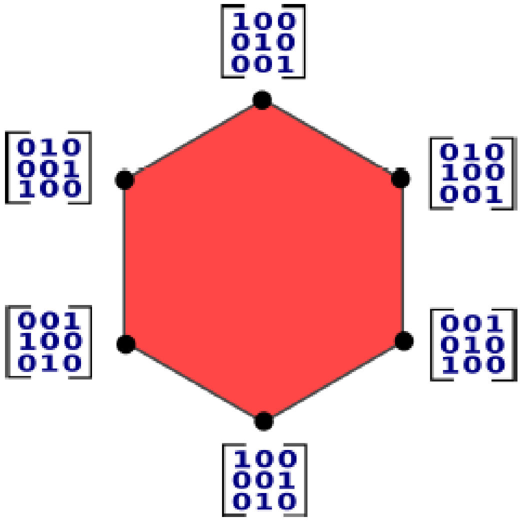
Visual attribute ranking

1. Order a collection of images according to a given attribute
2. Recover the original image from shuffled image patches





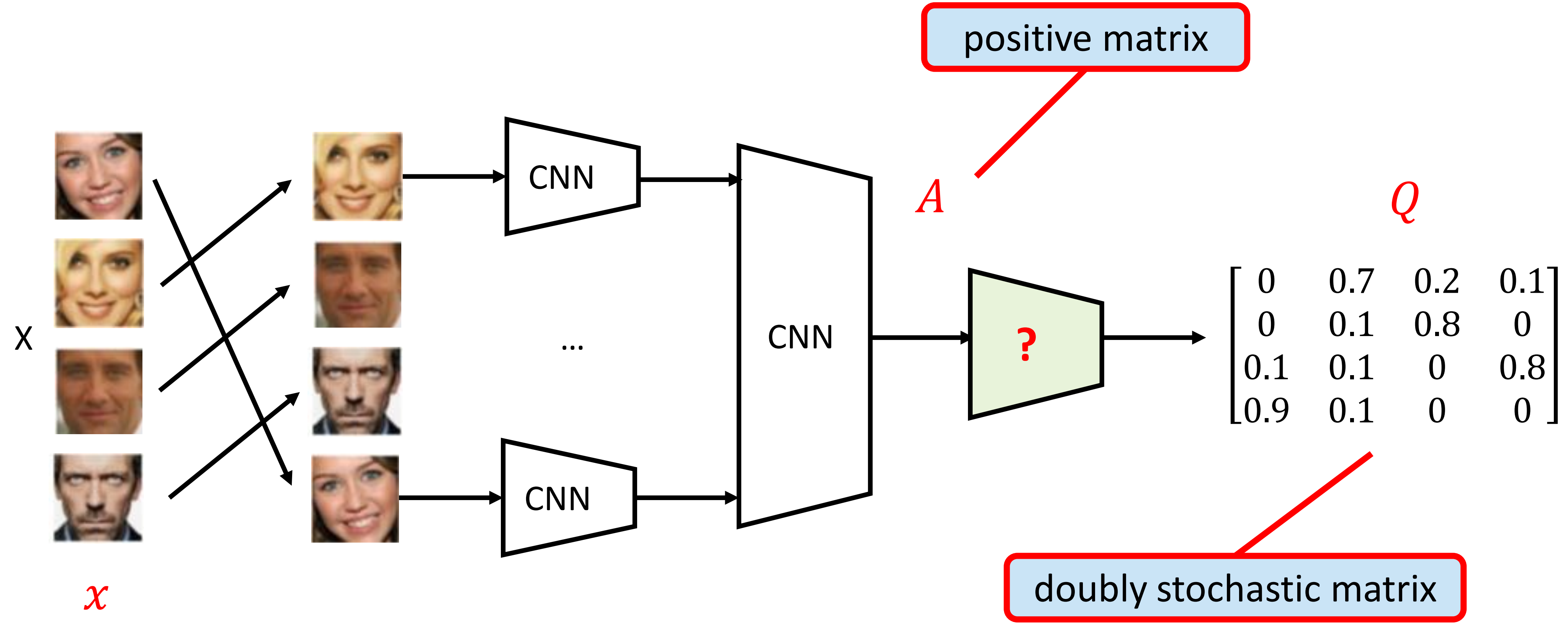
Birkhoff polytope



- Permutation matrices form discrete points in Euclidean space which imposes difficulties for gradient based optimizers
- The Birkhoff polytope is the convex hull for the set of $n \times n$ permutation matrices
- This coincides exactly with the set of $n \times n$ doubly stochastic matrices
- We relax our visual permutation learning problem over permutation matrices to a problem over doubly stochastic matrices

$$\{x_1, \dots, x_n\} \to B^n$$

End-to-end visual permutation learning



$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad Q = \begin{bmatrix} 0 & 0.7 & 0.2 & 0.1 \\ 0 & 0.1 & 0.8 & 0 \\ 0.1 & 0.1 & 0 & 0.8 \\ 0.9 & 0.1 & 0 & 0 \end{bmatrix}$$

Sinkhorn normalization or projection onto B^n

sinkhorn_fcn(A)

$Q = A$

for t = 1, …, T do

$$Q_{i,j} \leftarrow \frac{Q_{i,j}}{\sum_k Q_{i,k}}$$

$$Q_{i,j} \leftarrow \frac{Q_{i,j}}{\sum_k Q_{k,j}}$$

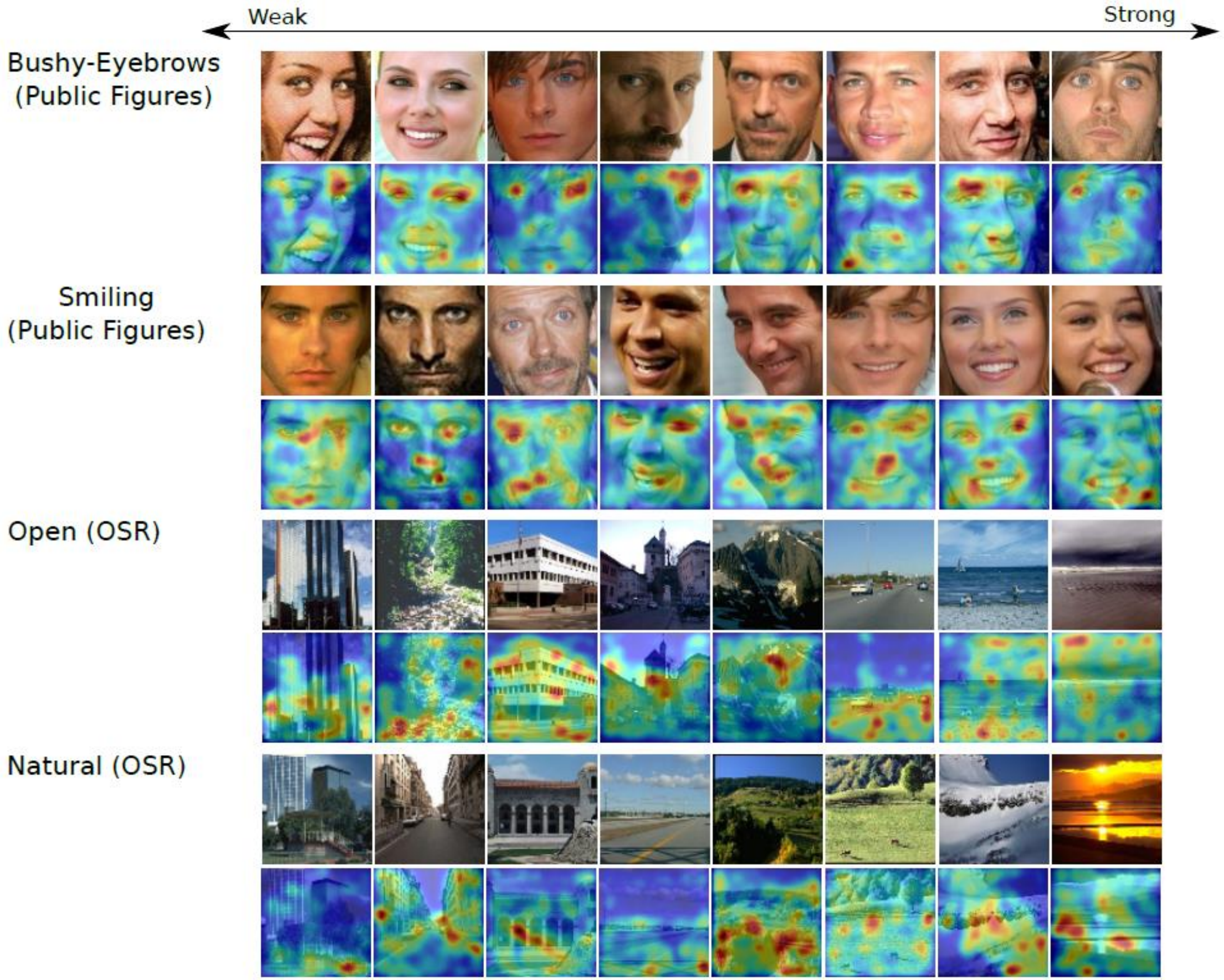
return Q

Alternatively, define a deep declarative module

$$\begin{array}{ll} \underset{Q \in \mathbb{R}_{+}^{n \times n}}{\text{minimize}} & \|Q - A\| \\ \text{subject to} & Q\,\mathbf{1} = \mathbf{1} \\ & Q^T\,\mathbf{1} = \mathbf{1} \end{array}$$

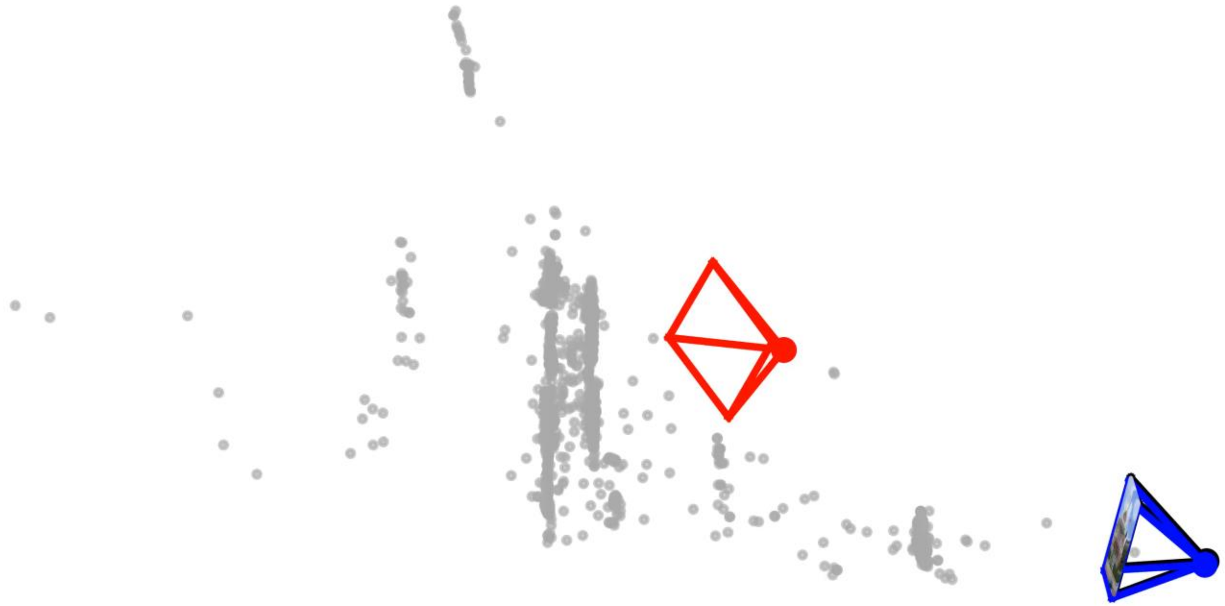
[Santa Cruz et al., 2017]

Visual attribute learning results



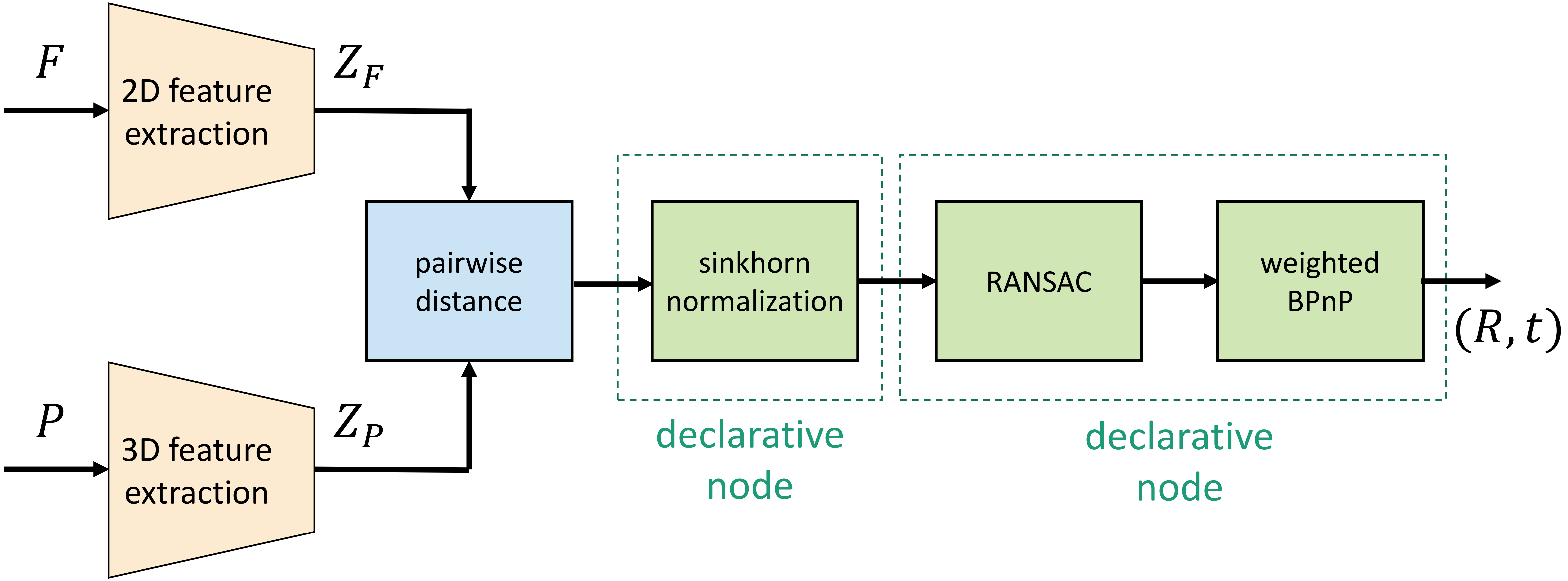
[Campbell et al., 2019 (unpublished)]

Blind perspective-n-point

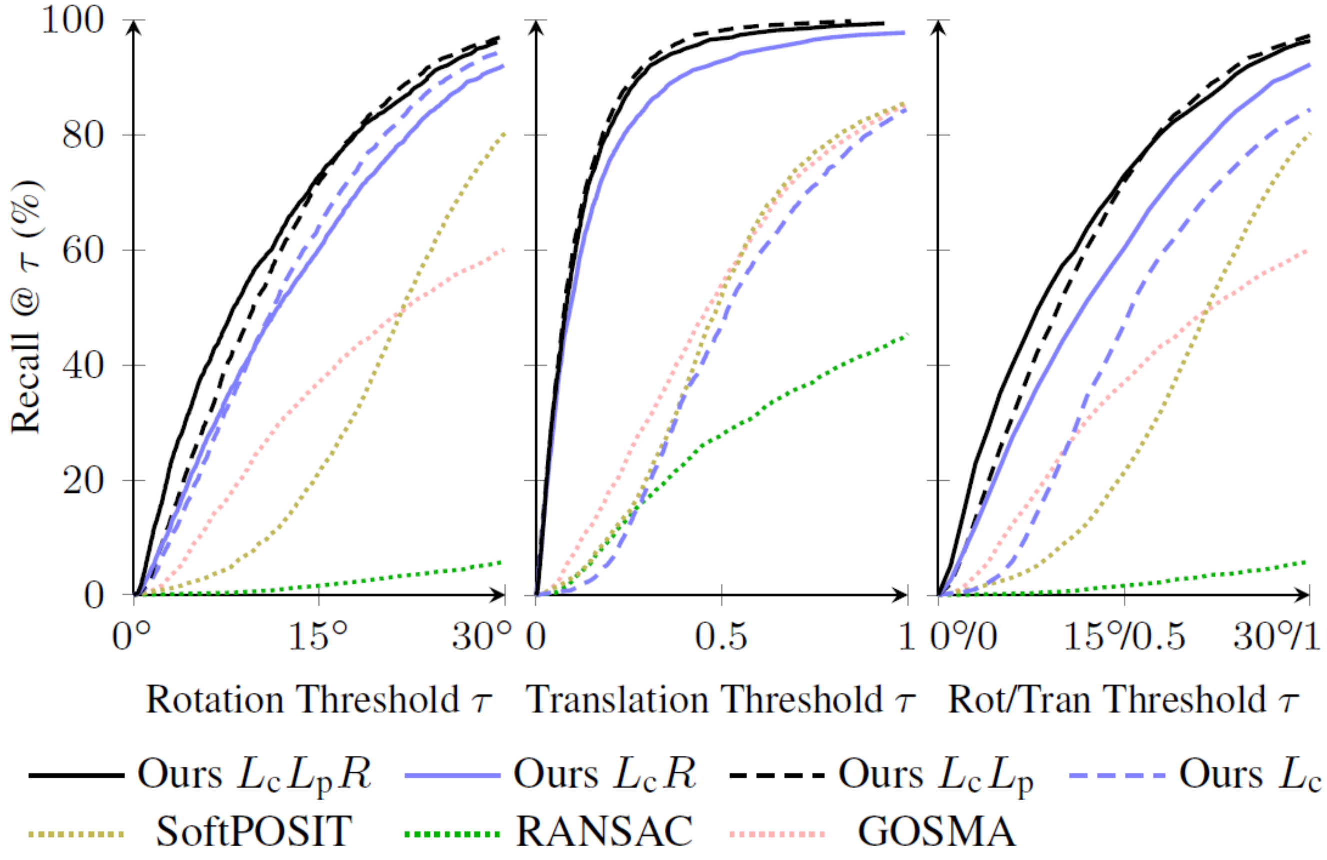


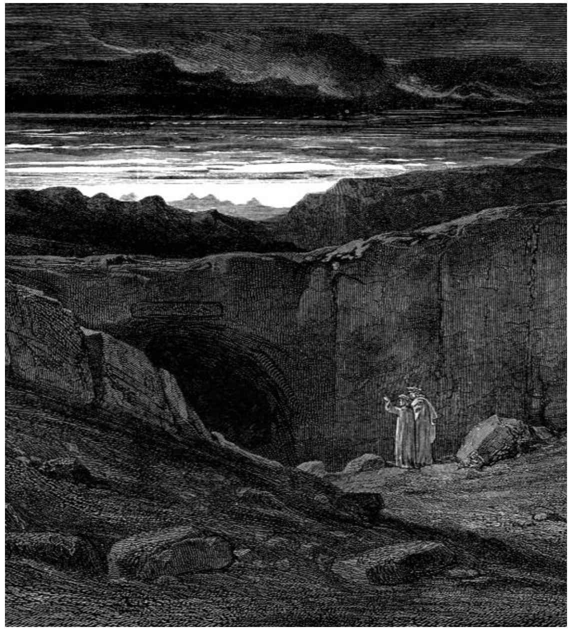
[Campbell et al., 2019 (unpublished)]

Blind perspective-n-point



Blind perspective-n-point





code and tutorials at http://deepdeclarativenetworks.com

CVPR 2020 Workshop (http://cvpr2020.deepdeclarativenetworks.com)