

MACHINE LEARNING IN BIOINFORMATICS

INVERTIBLE NEURAL NETWORKS

Philipp Benner
philipp.benner@bam.de

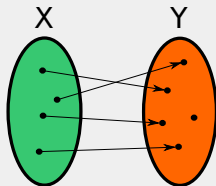
S.3 - eScience
Federal Institute for Materials Research and Testing (BAM)

June 25, 2023

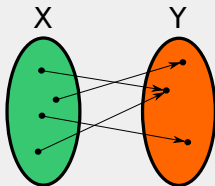
- Inverse problems
- Invertible Neural Networks (INNs) [Ardizzone et al., 2018]
- Normalizing Flows
- Invertible ResNets

INVERSE PROBLEMS

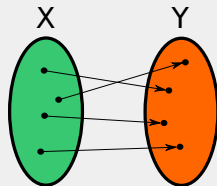
INJECTIVE, SURJECTIVE, BIJECTIVE



Injective



Surjective



Bijective

INVERSE PROBLEMS - LINEAR ALGEBRA

- Given a linear equation

$$y = Ax$$

where $A \in \mathbb{R}^{n \times p}$, $x \in \mathbb{R}^p$ and $y \in \mathbb{R}^n$

- We can compute y if we have x given (and A of course)
- A is *injective* iff $\text{rank}(A) = n$
- A is *surjective* iff $\text{rank}(A) = p$
- A is *bijective* iff $\text{rank}(A) = n = p$ ($\Rightarrow A$ is invertible)

$$x = A^{-1}y$$

- A *linear map* defined by

$$y = Ax$$

is invertible if A is a square matrix with full rank

- An *affine map* defined by

$$y = Ax + b$$

is invertible under the same condition

- Nonlinear functions are invertible iff they are strictly monotonic, but the inverse might be difficult to compute

INVERSE PROBLEMS - PROBABILITY

- Assume X and Y are random variables such that $X \rightarrow Y$
- The likelihood of an event $\{Y = y\}$ given $\{X = x\}$ is

$$\text{pr}(y | x)$$

- Bayes theorem tells us that

$$\text{pr}(x | y) = \frac{\text{pr}(y | x)\text{pr}(x)}{\text{pr}(y)}$$

- The posterior distribution $\text{pr}(x | y)$ is also called *inverse probability*
- It allows us to compute the probability of a *cause* (x) from a given or observed *effect* (y)¹

¹If $X \rightarrow Y$ is a causal relationship

INVERTIBLE NEURAL NETWORKS (INNs)

INVERTIBLE NEURAL NETWORKS (INNs)

Invertible neural network (INN)

A network f is bijective or invertible if it has an inverse network $g = f^{-1}$ such that $x = (g \circ f)(x)$ for all input values x

- There are multiple invertible architectures
- Invertible neural networks are constructed by concatenating invertible subnetworks called *coupling blocks*
- For a network to be invertible, all coupling blocks must be invertible
- There exist multiple architectures, e.g. GLOW, RNVP, NICE

INVERTIBLE NEURAL NETWORKS (INNs) - NICE

- Input x and output y are split into two halves, i.e.

$$x = [x_1, x_2], \quad y = [y_1, y_2]$$

- The NICE coupling block is defined by [Dinh et al., 2014]

$$y_1 = x_1$$

$$y_2 = x_2 + t(x_1)$$

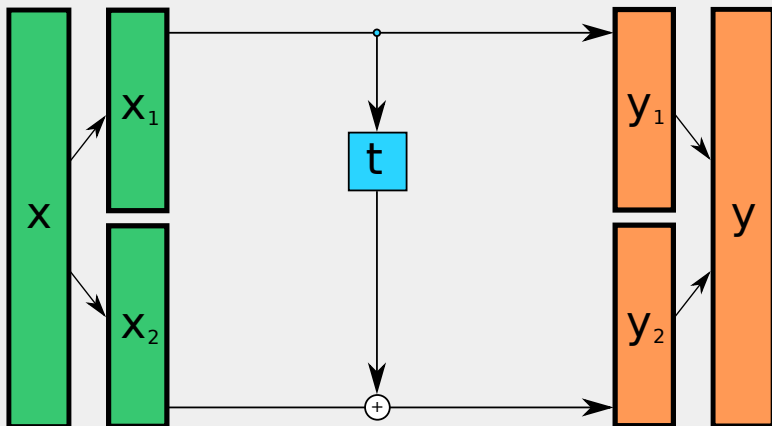
where t is an arbitrary function such as a neural network

- The inverse is given by

$$x_1 = y_1$$

$$x_2 = y_2 - t(x_1)$$

INVERTIBLE NEURAL NETWORKS (INNs) - NICE



INVERTIBLE NEURAL NETWORKS (INNs) - RNVP

- The RealNVP (RNVP) coupling block is defined by [Dinh et al., 2016]

$$y_1 = x_1 \odot \exp [s_2(x_2)] + t_2(x_2)$$

$$y_2 = x_2 \odot \exp [s_1(y_1)] + t_1(y_1)$$

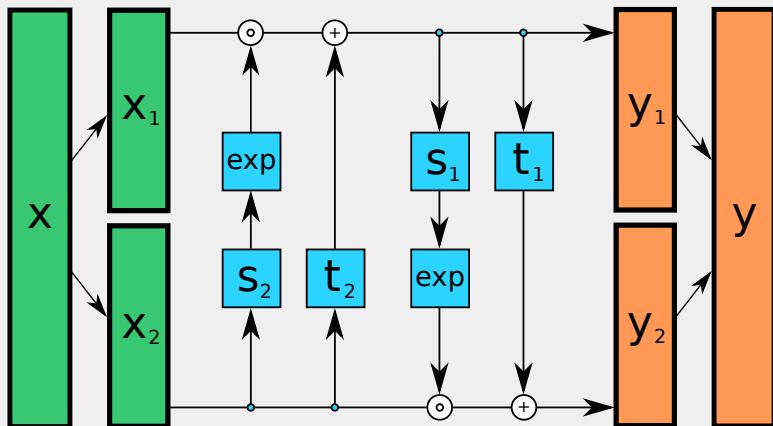
where \odot is the element-wise multiplication, input and output are split into two halves

$$x = [x_1, x_2], \quad y = [y_1, y_2]$$

and t_1, t_2, s_1, s_2 are arbitrary functions (e.g. dense neural networks)

- Notice that this architecture is an affine function, which can be easily inverted

INVERTIBLE NEURAL NETWORKS (INNs) - RNVP



INVERTIBLE NEURAL NETWORKS (INNs) - RNVP

- Inverting the neural network leads to

$$\begin{aligned}y_1 &= x_1 \odot \exp [s_2(x_2)] + t_2(x_2) \\ \Rightarrow y_1 - t_2(x_2) &= x_1 \odot \exp [s_2(x_2)] \\ \Rightarrow y_1 - t_2(x_2) \odot \exp [-s_2(x_2)] &= x_1\end{aligned}$$

where x_2 is obtained from

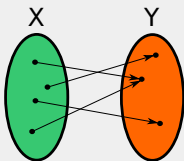
$$\begin{aligned}y_2 &= x_2 \odot \exp [s_1(y_1)] + t_1(y_1) \\ \Rightarrow y_2 - t_1(y_1) &= x_2 \odot \exp [s_1(y_1)] \\ \Rightarrow (y_2 - t_1(y_1)) \odot \exp [-s_1(y_1)] &= x_2\end{aligned}$$

- INNs typically stack many of these invertible blocks. The input components $(x^{(1)}, \dots, x^{(p)})$ of x are permuted after each block

INVERTIBLE NEURAL NETWORKS FOR SURJECTIVE PROBLEMS

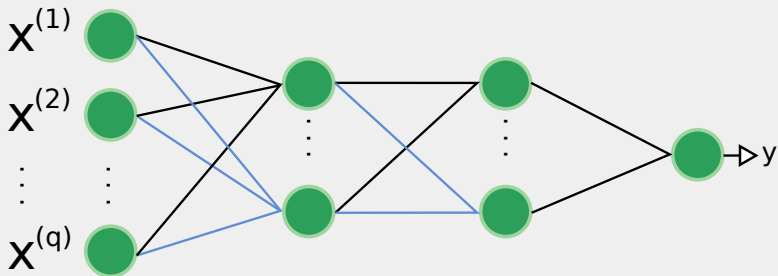
INVERTIBLE NEURAL NETWORKS (INNs)

- Most problems in machine learning are surjective



- Example: In object recognition there are typically many images that belong to the same classification

INVERTIBLE NEURAL NETWORKS (INNs)



INVERTIBLE NEURAL NETWORKS (INNs) - AD-HOC

- Let f be a trained neural network for predicting Y from some input variable X
- Given a fixed output value y , compute the inverse by optimizing the input, i.e.

$$\hat{x} = \arg \min_x \mathcal{L}(f(x), y)$$

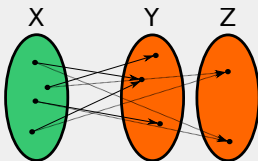
- The loss function \mathcal{L} should be the same as for learning the network weights
- Use gradient descent to invert the neural network

INVERTIBLE NEURAL NETWORKS (INNs) - AD-HOC

- For surjective problems the solution is not unique and depends on the initial condition
- By testing multiple initial conditions, we may collect many possible inverse solutions
- What initial conditions should we select?
- How can we be sure that we obtained all important solutions?
- Is there a better approach?

INVERTIBLE NEURAL NETWORKS (INNs)

- We extend the invertible network so that it generates (samples) all input values $\{x_i\}_i$ that correspond to a given output value y
- Idea: Augment y with additional values z



- Elements X that map to the same points in Y have to be mapped to different elements in Z

Augmented targets

The invertible neural network f computes

$$[y, z] = [f_y(x), f_z(x)] = f(x)$$

for an input x , where $y = f_y(x)$ and $z = f_z(x)$

- If both y and z are given, we can easily compute the inverse

$$x = g(y, z) = f^{-1}(y, z)$$

- The (intrinsic) dimension of $[y, z]$ must be greater or equal to the dimension of x

INVERTIBLE NEURAL NETWORKS (INNs)

- Given only the target value y , what z value should we select?
- z values that have never been observed during training will most likely result in unreasonable x values
- We must constrain/regularize z . We want z to follow a particular distribution, e.g.

$$z \sim \mathcal{N}(0, I)$$

where I is the identity matrix

- To obtain a possible inverse of y , we first draw z and compute

$$x = g([y, z]) = f^{-1}([y, z])$$

INVERTIBLE NEURAL NETWORKS (INNs) - LEARNING

- We have two training objectives:

- ▶ Given a set of training points $(x_i, y_i)_i$, $f_y(x_i)$ should match y_i with respect to some metric defined by the loss function
- ▶ For any pair of inputs (x_i, x_j) such that $f_y(x_i) = f_y(x_j)$ we want that $f_z(x_i) \neq f_z(x_j)$. In probabilistic terms we want that y and z are independent.

- Since we do not want to define a probability distribution for x and y , we will work with empirical distributions

$$\hat{p}(x), \hat{p}(y)$$

derived from our training data $(x_i, y_i)_i$

- For simplicity, we also assume that $\text{pr}(z)$ is given as empirical distribution $\hat{p}(z)$

INVERTIBLE NEURAL NETWORKS (INNs) - LEARNING

■ Formal definition of learning objectives

- ▶ Minimize $\mathcal{L}_y = \sum_i^n \|y_i - f_y(x_i)\|_2^2$ (or any other norm)
- ▶ Minimize $\mathcal{L}_{y,z}$ which measures the discrepancy between

$$\hat{p}(y)\hat{p}(z) \quad \text{and} \quad \hat{q}(y,z),$$

where $\hat{q}(y,z)$ is the empirical distribution estimated on

$$\{[\hat{y}_i, \hat{z}_i] = f(x_i) \mid i = 1, \dots, n\}$$

i.e. the set of points $(\hat{y}_i, \hat{z}_i)_i$ resulting from applying the neural network f to training points $(x_i)_i$

- We use the *maximum mean discrepancy (MMD)* to measure the discrepancy between two empirical distributions

INVERTIBLE NEURAL NETWORKS (INNs) - MMD

- Assume we have two random variables X and Y
- How can we measure the difference between their distributions?
- Example: Look at the difference between expectations, i.e.

$$\|\mathbb{E}_X X - \mathbb{E}_Y Y\|_2^2$$

- What if two different distributions have the same mean?
- We need to incorporate higher *moments*, i.e.

$$\left\| \mathbb{E}_X \begin{bmatrix} X \\ X^2 \end{bmatrix} - \mathbb{E}_Y \begin{bmatrix} Y \\ Y^2 \end{bmatrix} \right\|_2^2$$

INVERTIBLE NEURAL NETWORKS (INNs) - MMD

- How many moments do we need?
- Using a feature mapping ϕ we can incorporate as many as we want

Maximum mean discrepancy (MMD)

Given two random variables, X and Y , the MMD is defined as

$$\text{MMD}^2(X, Y) = \|\mathbb{E}_X \phi(X) - \mathbb{E}_Y \phi(Y)\|_{\mathcal{H}}^2$$

where ϕ is a mapping into feature space \mathcal{H} equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and the corresponding norm $\|x\|_{\mathcal{H}}^2 = \langle x, x \rangle_{\mathcal{H}}$

INVERTIBLE NEURAL NETWORKS (INNs) - MMD

- The *kernel trick* is used to efficiently compute the MMD
- Let $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$ denote the corresponding kernel function
- The MMD is expanded as follows

$$\begin{aligned}\text{MMD}^2(X, Y) &= \|\mathbb{E}_X \phi(X) - \mathbb{E}_Y \phi(Y)\|_{\mathcal{H}}^2 \\ &= \langle \mathbb{E}_X \phi(X), \mathbb{E}_{X'} \phi(X') \rangle_{\mathcal{H}} + \langle \mathbb{E}_Y \phi(Y), \mathbb{E}_{Y'} \phi(Y') \rangle_{\mathcal{H}} \\ &\quad - 2 \langle \mathbb{E}_X \phi(X), \mathbb{E}_Y \phi(Y) \rangle_{\mathcal{H}} \\ &= \mathbb{E}_{X, X'} \kappa(X, X') + \mathbb{E}_{Y, Y'} \kappa(Y, Y') - 2 \mathbb{E}_{X, Y} \kappa(X, Y)\end{aligned}$$

INVERTIBLE NEURAL NETWORKS (INNs) - MMD

- Empirical estimate of the MMD
- Assume we have n i.i.d. samples x_i from X and m i.i.d. samples y_i from Y

$$\mathbb{E}_{X, X'} \kappa(X, X') = \frac{1}{n(n-1)} \sum_i \sum_{j \neq i} \kappa(x_i, x_j)$$

where we have to exclude the case where $x_i = x_j$ because for continuous distributions this will happen with probability zero

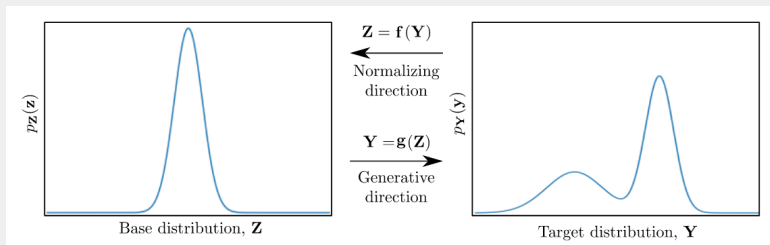
$$\mathbb{E}_{X, Y} \kappa(X, Y) = \frac{1}{nm} \sum_i \sum_j \kappa(x_i, y_j)$$

NORMALIZING FLOWS

NORMALIZING FLOWS

- Invertible neural networks are a special class of *normalizing flows*
- A *Normalizing Flow* is a transformation of a (simple) probability distribution into another (more complex) distribution [Kobyzev et al., 2020]
- The transformation is computed using a sequence of invertible and differentiable mappings
- Let Z be a random variable with a simple and tractable probability distribution, e.g. normal distribution
- Let Y be a random variable such that $Y = g(Z)$ where g is an invertible function, i.e. $g = f^{-1}$

NORMALIZING FLOWS



- Generative direction: To generate samples from Y we can sample from the simple distribution Z and use g to obtain Y
- Normalizing direction: If we have an observation $\{Y = y\}$ we can use f to compute the probability of y

- Using the *change of variables formula*, we obtain

$$\begin{aligned}\mathrm{pr}_Y(y) &= \mathrm{pr}_Z(f(y)) |\det Df(y)| \\ &= \mathrm{pr}_Z(f(y)) |\det Dg(f(y))|^{-1}\end{aligned}$$

where $Df(y)$ denotes the Jacobian of f evaluated at y

- If $f = f_1 \circ f_2 \circ \cdots \circ f_k$ is a sequence of invertible mappings f_i then

$$\det Df(y) = \prod_{i=1}^k \det Df_i(y^{(i)})$$

where $y^{(i+1)} = f_i(y^{(i)})$ and $y^{(1)} = y$

NORMALIZING FLOWS - TRAINING

- Given a set of n observations (y_1, \dots, y_n)
- Maximum likelihood approach: Maximize the probability

$$\begin{aligned}\text{pr}_Y(y_1, \dots, y_n) &= \sum_{i=1}^n \log \text{pr}_Y(y_i) \\ &= \sum_{i=1}^n p_Z(f(y_i)) + \log |\det Df(y_i)|\end{aligned}$$

with respect to parameters of f (i.e. weights of neural network)

- Use *maximum entropy approach* [Loaiza-Ganem et al., 2017]

INVERSES OF RESIDUAL NEURAL NETWORKS

INVERSE OF RESNETS

- Residual neural networks (ResNets) are a special case where the inverse can be computed without gradient descent (under some constraints) [Behrmann et al., 2019]
- Recall that a layer of a ResNet is defined as

$$x_{t+1} = x_t + g(x_t)$$

where g is a non-linear neural network layer

- The inverse is given by

$$\begin{aligned} x_t &= x_{t+1} - g(x_t) \\ &= f(x_t) \end{aligned}$$

where $f(x_t) = x_{t+1} - g(x_t)$ and x_{t+1} is treated as a parameter

Fixed point

For a function f a point x^* that satisfies $x^* = f(x^*)$ is called a *fixed point*

- x_t is a *fixed point* of f , which is also the *inverse* of the ResNet with layer g
- A fixed point x^* is (locally) stable if

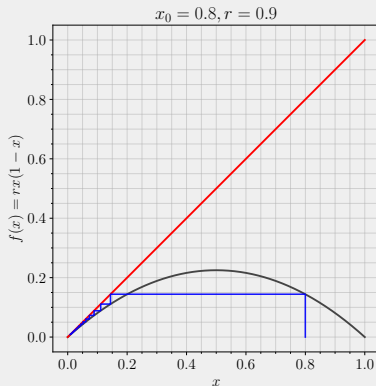
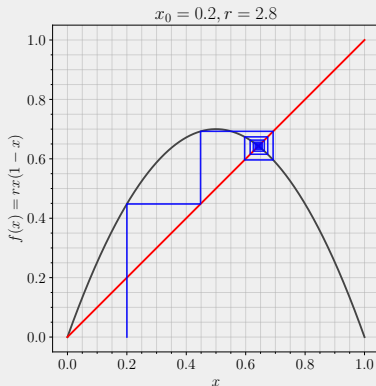
$$\left| \frac{d}{dx} f(x) \right|_{x=x^*} < 1$$

- A fixed point x^* is (locally) unstable if

$$\left| \frac{d}{dx} f(x) \right|_{x=x^*} > 1$$

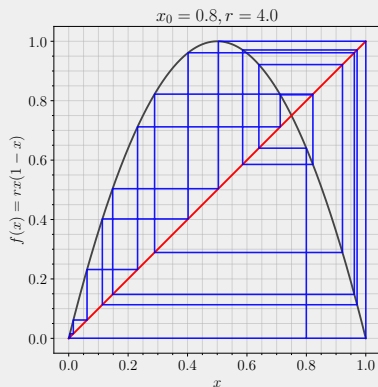
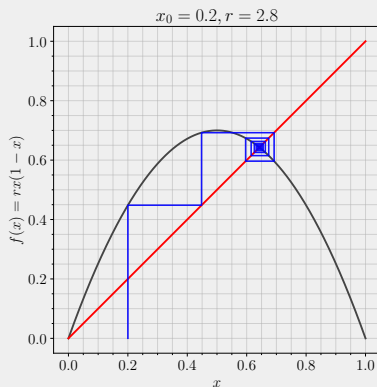
FIXED POINTS - COBWEB PLOTS

Cobweb plot of the logistic map $f(x) = rx(1 - x)$:



$x_{t+1} = f(x_t)$ [blue line], $x = y$ [red line]

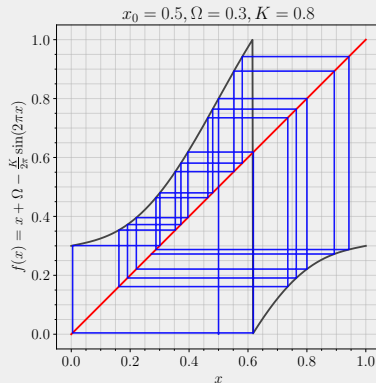
FIXED POINTS - STABILITY



Fixed point $x^* = f(x^*)$ stable (left) and unstable (right)

FIXED POINTS

- Some maps do not have fixed points
- One example is the *circle map* (for specific parameters)



Lipschitz constant

Let X, Y be two metric spaces with distance measures d_X and d_Y . A function $f : X \rightarrow Y$ is called *Lipschitz continuous* if there exists a constant c such that

$$d_Y(f(x_1), f(x_2)) \leq c d_X(x_1, x_2)$$

The smallest constant c is called the *Lipschitz constant*

- A special case is when $f : \mathbb{R} \rightarrow \mathbb{R}$ is differentiable
- In this case we have

$$c = \sup_{x^*} \left| \frac{d}{dx} f(x) \right|_{x=x^*}$$

- When $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is differentiable then

$$c = \sup_{x^*} \|Df(x^*)\|_o$$

where $Df(x^*)$ is the Jacobi matrix evaluated at x^* and $\|\cdot\|_o$ the operator norm

- Let $\lambda_1(x^*), \dots, \lambda_n(x^*)$ denote the n eigenvalues of the Jacobi matrix $Df(x^*)$, then

$$\|Df(x^*)\|_o = \max_k |\lambda_k(x^*)|$$

Banach fixed-point theorem

Let (X, d) be a metric space and $f : X \rightarrow X$ a mapping such that

$$d(f(x_1), f(x_2)) \leq cd(x_1, x_2)$$

with $c \in [0, 1)$, then f is called a *contraction* and it has a unique and stable fixed point $x^* = \lim_{t \rightarrow \infty} x_{t+1} = f(x_t)$

- The Lipschitz constant c is an upper bound on the absolute value of the slope of f
- For $c \in [0, 1)$ the function f must cross the main diagonal
- Therefore, it must have a single fixed-point x^*

COMPUTING RESNET INVERSES

- Assume that our ResNet layer f is sufficiently well behaving:
 - ▶ No discontinuities
 - ▶ Absolute value of the slope bounded everywhere by 1, i.e. $c \in [0, 1)$
 - ▶ This can be achieved by constraining the eigenvalues of the Jacobian during training





- In this case we should be able to iterate

$$x_t \leftarrow f(x_t) = x_{t+1} - g(x_t)$$

until arriving at a (stable) fixed point x^*

- The fixed point x^* is our inverse

REFERENCES I

-  ARDIZZONE, L., KRUSE, J., WIRKERT, S., RAHNER, D., PELLEGRINI, E. W., KLESSEN, R. S., MAIER-HEIN, L., ROTHER, C., AND KÖTHE, U. (2018).
ANALYZING INVERSE PROBLEMS WITH INVERTIBLE NEURAL NETWORKS.
arXiv preprint arXiv:1808.04730.
-  BEHRMANN, J., GRATHWOHL, W., CHEN, R. T., DUVENAUD, D., AND JACOBSEN, J.-H. (2019).
INVERTIBLE RESIDUAL NETWORKS.
In International Conference on Machine Learning, pages 573–582. PMLR.
-  DINH, L., KRUEGER, D., AND BENGIO, Y. (2014).
NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION.
arXiv preprint arXiv:1410.8516.
-  DINH, L., SOHL-DICKSTEIN, J., AND BENGIO, S. (2016).
DENSITY ESTIMATION USING REAL NVP.
arXiv preprint arXiv:1605.08803.

REFERENCES II



KOBYZEV, I., PRINCE, S. J., AND BRUBAKER, M. A. (2020).

NORMALIZING FLOWS: AN INTRODUCTION AND REVIEW OF CURRENT METHODS.

IEEE transactions on pattern analysis and machine intelligence, 43(11):3964–3979.



LOAIZA-GANEM, G., GAO, Y., AND CUNNINGHAM, J. P. (2017).

MAXIMUM ENTROPY FLOW NETWORKS.

arXiv preprint arXiv:1701.03504.