# 最適輸送入門

佐藤竜馬

@IBIS 2021 チュートリアル

KYOTO UNIVERSITY

# 最適輸送は確率分布を比較するツール

- このチュートリアルのトピック: 最適輸送

take home message

**最適輸送は確率分布と確率分布を比較するのに使えるツール**

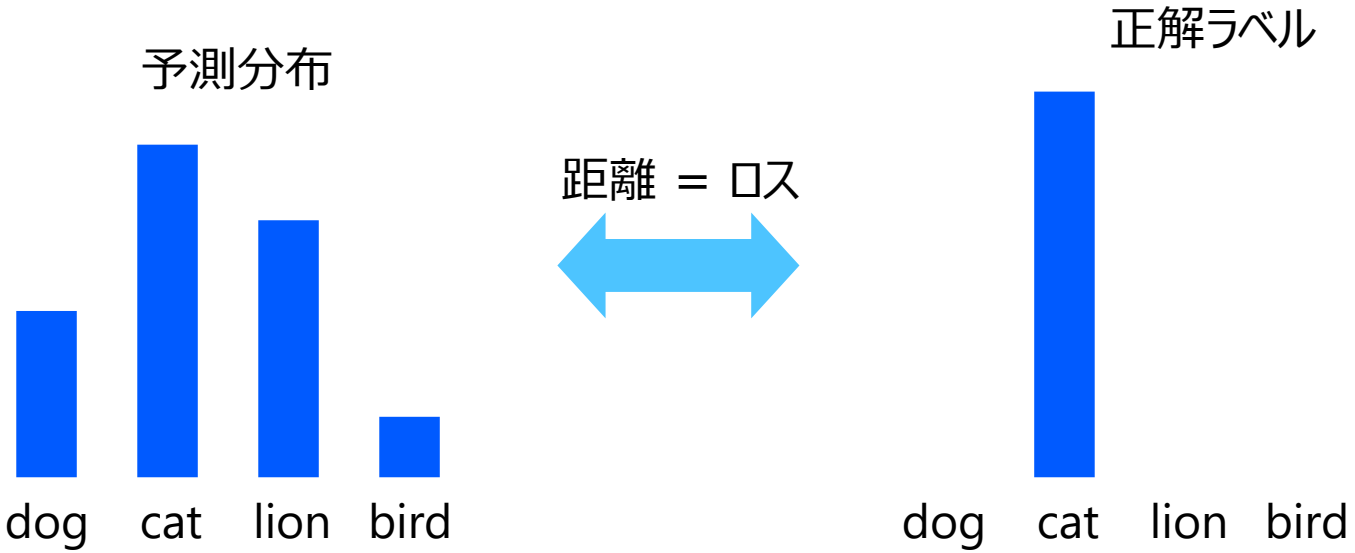
- 類似ツール: KLダイバージェンス, MMD
- このチュートリアルでは
  - KL ダイバージェンスに比べた最適輸送の優れた点を知る
  - 最適輸送のアルゴリズムを知る

  をめざします

# 最適輸送の直感的な導入とメリット

# 分布比較の例: カテゴリ予測

- 確率分布の比較は機械学習のあらゆる場面で登場する
- 例1: カテゴリ予測のロス関数




- クロスエントロピー（KL ダイバージェンス）が有名

# 分布比較の例: 生成モデル

- 確率分布の比較は機械学習のあらゆる場面で登場する
- 例2: 生成モデルのロス関数

生成サンプルの経験分布

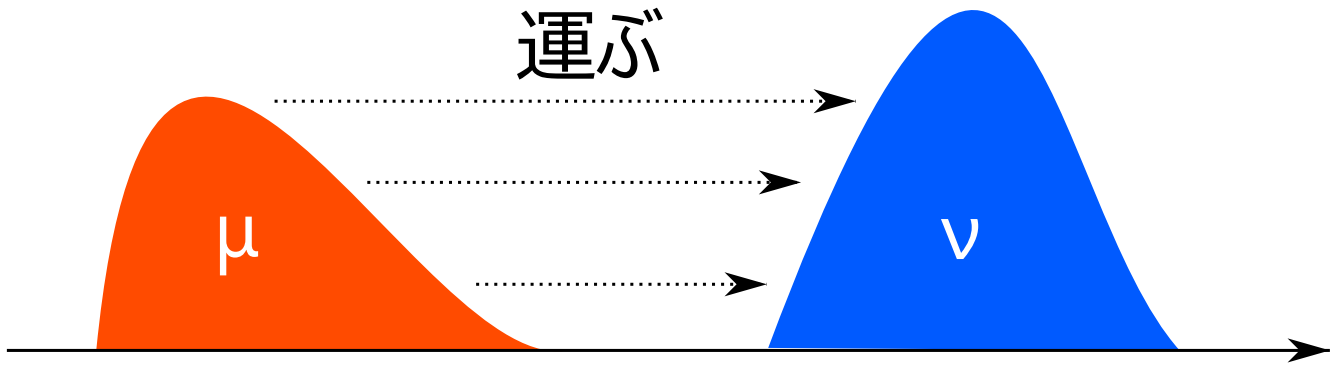
距離 = ロス

訓練サンプルの経験分布

Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR 2018.
https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

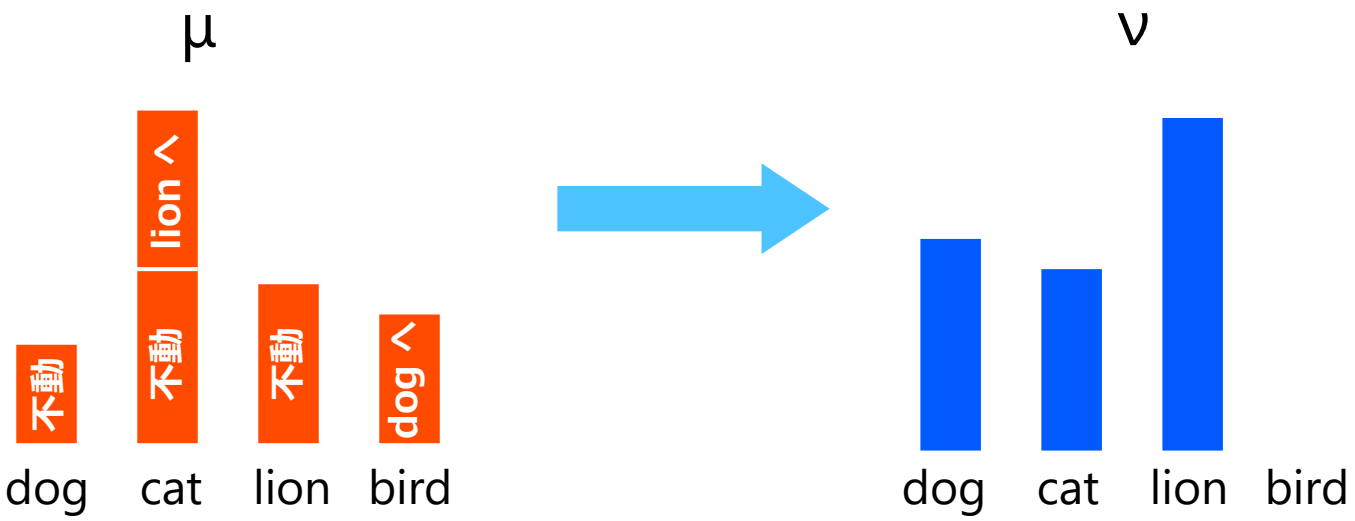
# 最適輸送の直感的な意味: 確率分布の「山」を輸送

- **確率分布の「山」を移動させて一致させるのにかかるコストが最適輸送の直感的な定義**
- 輸送という単語はここからきている
- 土塊を運ぶイメージから、Earth Mover's Distance とも呼ばれる



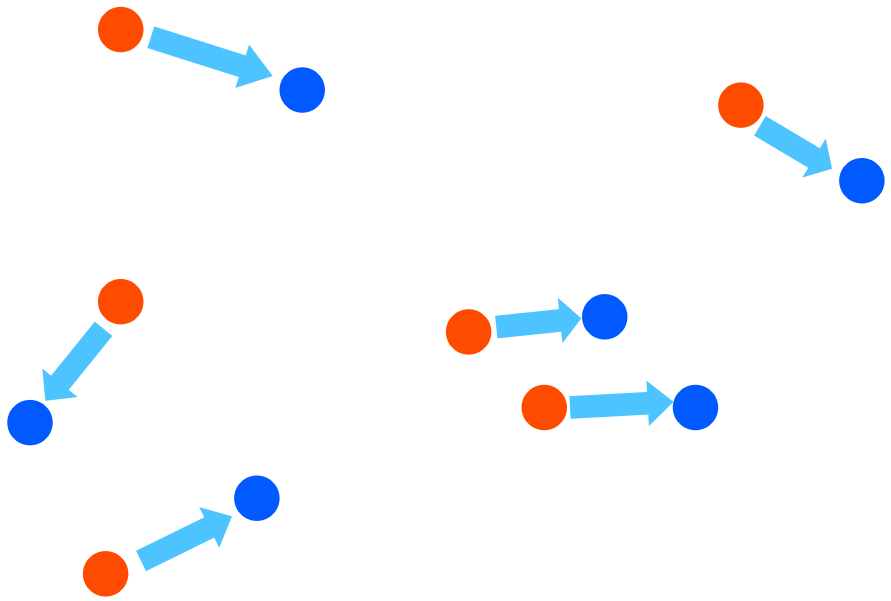
# ヒストグラム比較の場合の例

- クラス確率（ヒストグラム）比較の場合
- 各クラスから他のクラスへ山を移動させるコストを人手で定める
  cat → lion は cat → bird よりも低コストという知識を込めてもよい
- μ の山を移動させて ν に一致させるコストが最適輸送距離
  （ν の山を移動させて μ の山に移動させるとしてもコストは同じ）



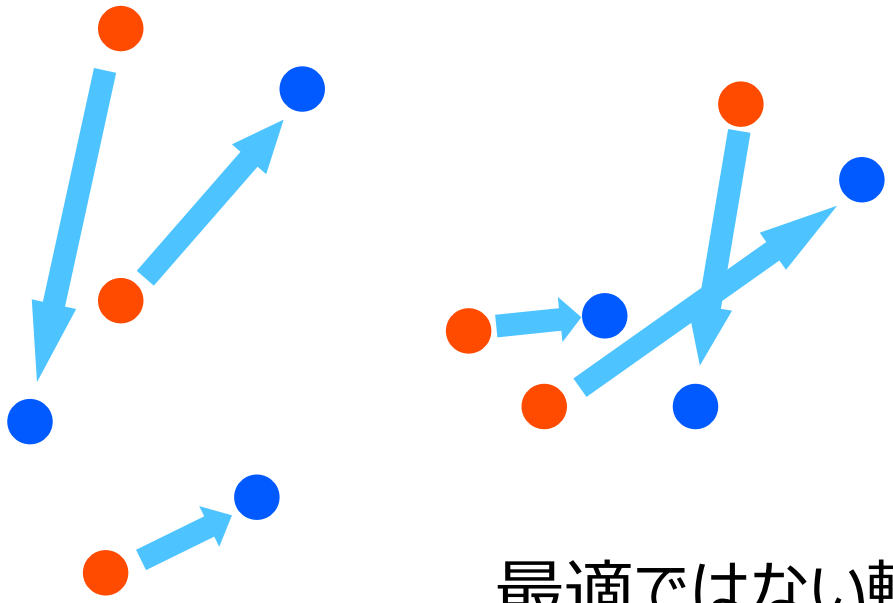
# 点群比較の場合の例

- 連続分布の経験分布（点群）比較の場合
- 各点に質量 1/n の砂山があると考えその輸送距離を考える



# 最適な輸送についてのコストを用いて距離を定義する

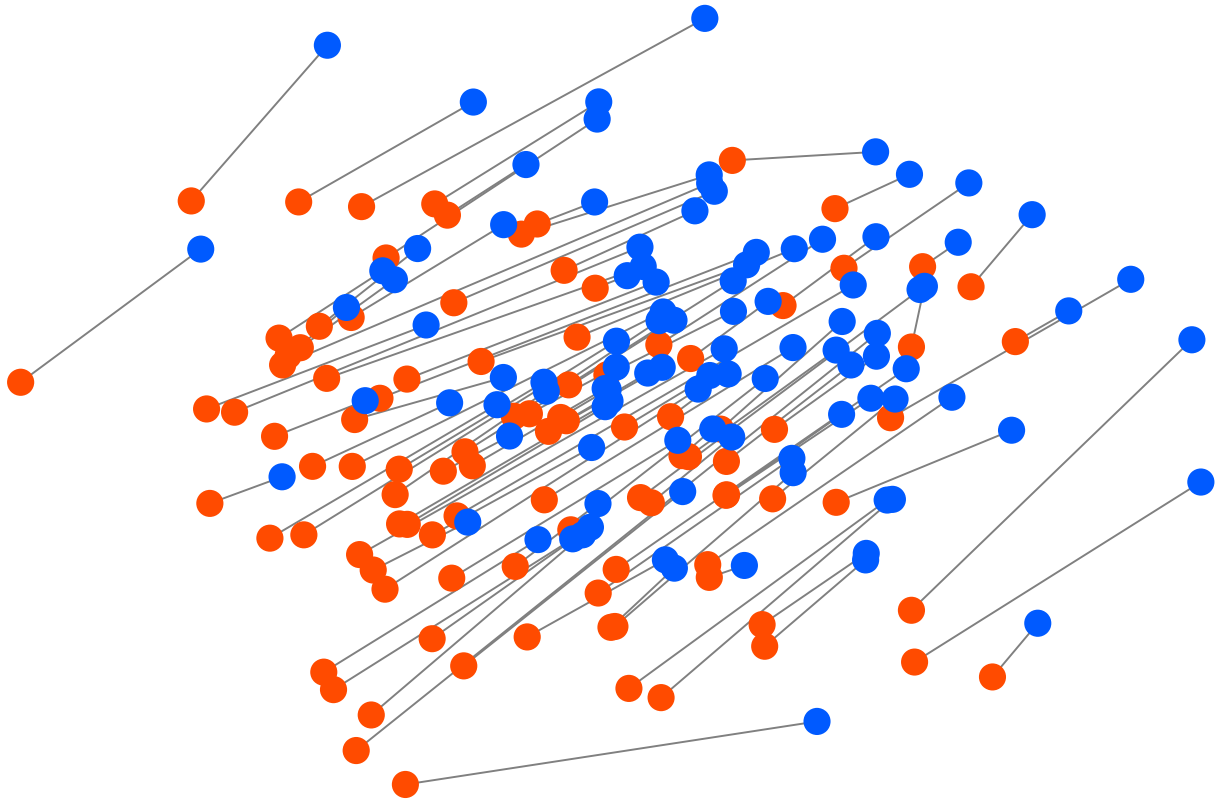
- 輸送の仕方は複数通りあるが、**最もコストが少ない**輸送方法を選ぶ
  ヒストグラムの場合も同様
- 「最適」という言葉はここからきている



最適ではない輸送の例

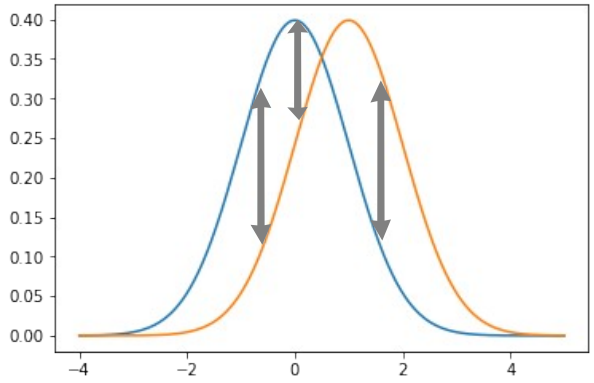
# もう少し規模が大きい場合の最適輸送の例

- もう少し規模の大きい点群の最適輸送



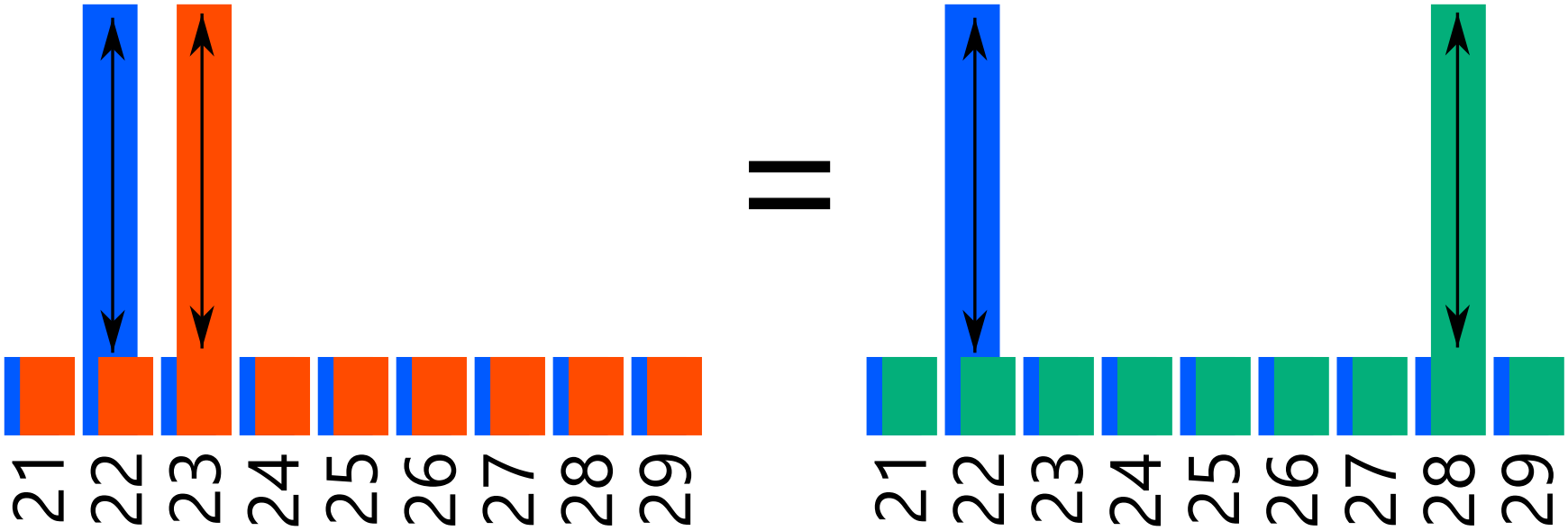
# KL ダイバージェンスは要素ごとの項の和

- KL ダイバージェンスとの比較を通して最適輸送の利点を確認する
- 離散分布は n 次元のベクトル p で表現できる
  $p_i$ は i 番目の要素の確率値
- KL ダイバージェンス: $\mathrm{KL}(\mathrm{p}\,\|q) = \sum_i p_i \log \frac{p_i}{q_i}$
- **要素ごとに独立に項を足し合わせているのがポイント**



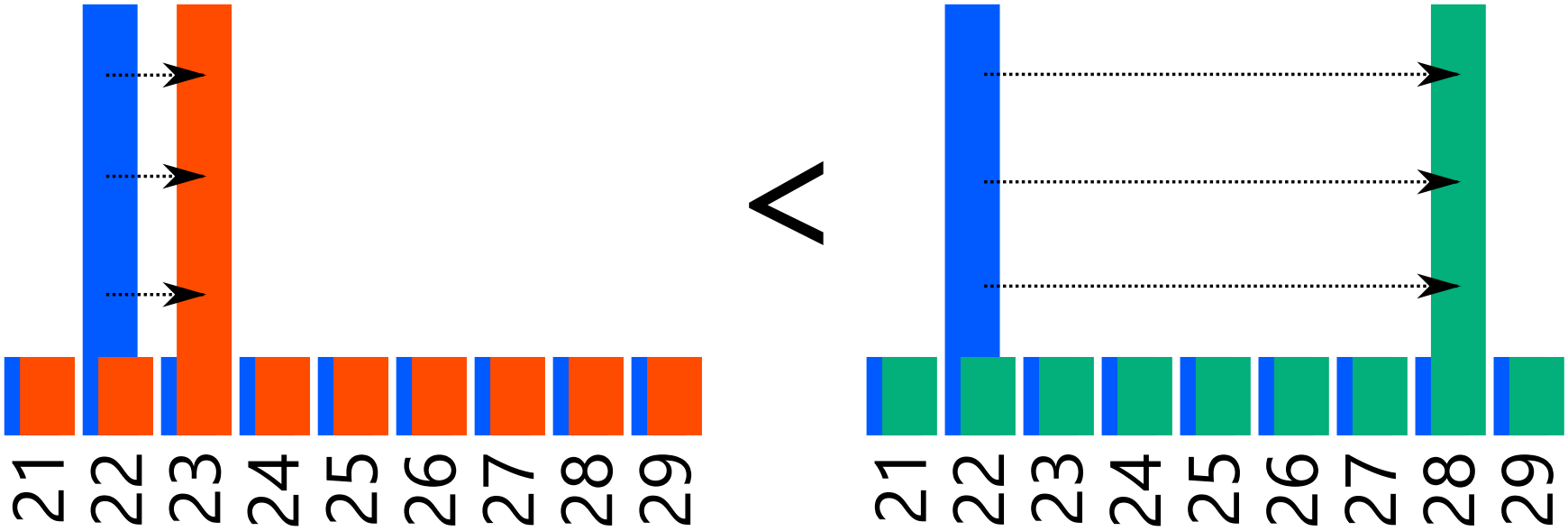
# KL ダイバージェンスは要素の距離を考慮できない

- 気温をクラス分類により予測する問題を考える
- KL ダイバージェンスを用いると、要素ごとの和となるので、
  青・赤の距離と青・緑の距離は同じ



# 最適輸送は要素どうしの距離を考慮できる

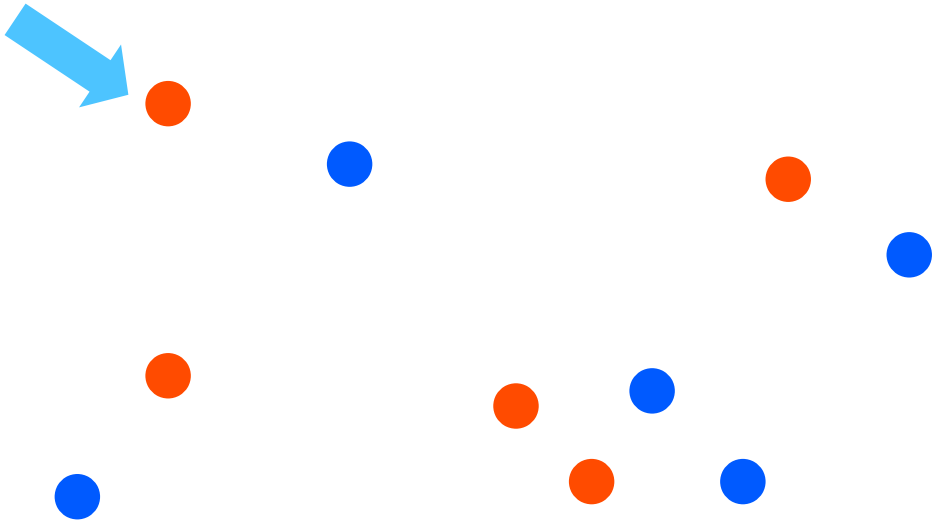
- 最適輸送を用いると、青・赤の方が輸送にかかるコストが少ないので距離が小さいと判定される
- 22 度を 23 度と間違えているだけなので、22 度を 28 度と間違えるより距離が小さいと判定されるのは直感に適合している



# KL はサポートが被っていないと使えない

- 連続分布の経験分布（点群）比較を考える

  経験分布においては、ちょうどこの点における
  青の確率はゼロと判断される → - log 0 の項が登場して KL = ∞

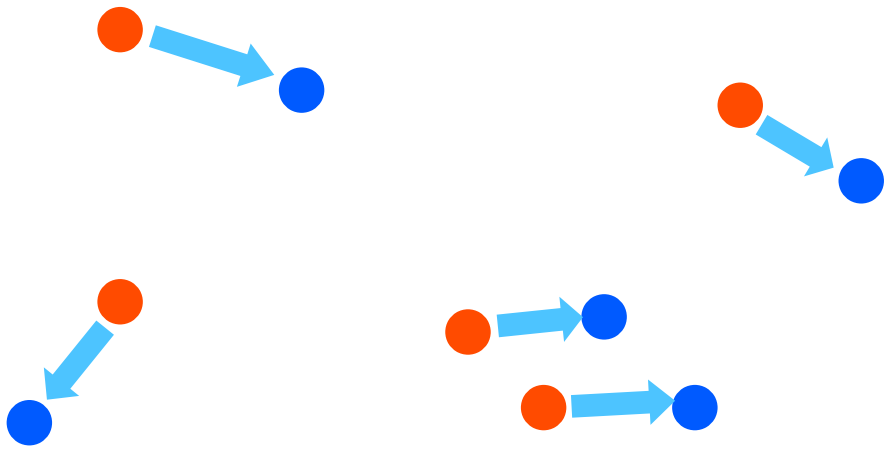


確率値が正を取る点集合

- KL はサポートが被っていないと距離が ∞ と判断される
  → 連続分布の経験分布の比較ができない

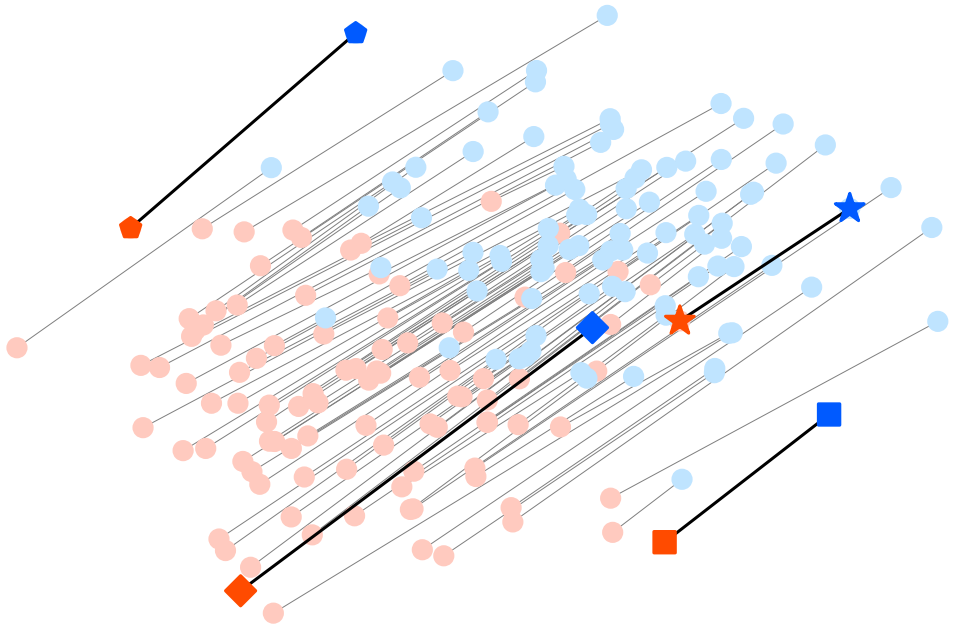
# 最適輸送はサポートが被っていなくても使える

- 最適輸送は輸送という概念のおかげでサポートが被っていないときにも利用できる



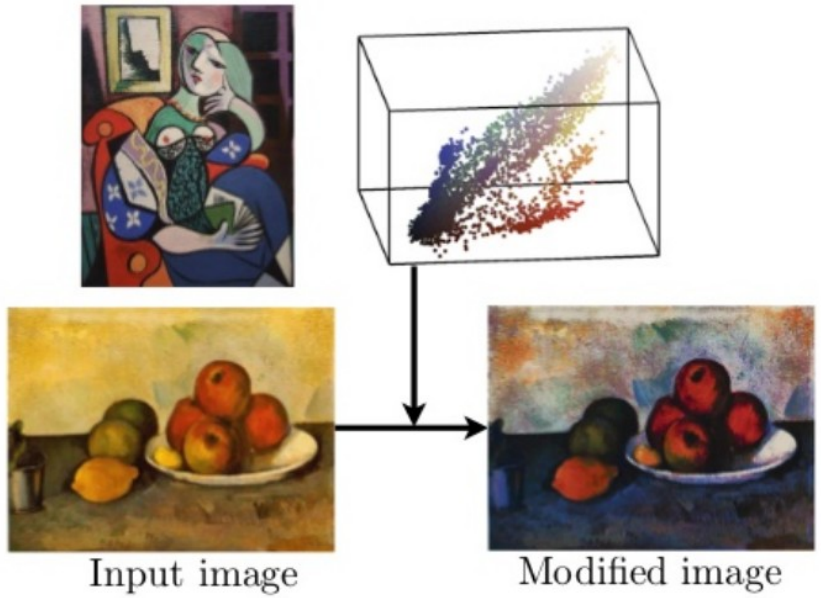
# 最適輸送は分布の対応関係を得ることができる

- 最適輸送は副産物として、分布の対応関係を得ることができる
- 正規分布の対応関係:
  分布がずれていても、ずれを修正するような対応関係が得られる



# 応用: 最適輸送は色相変換にも利用できる

- 入力: 色相の異なる二つの画像
- ピクセルを RGB 空間内におき、画像を RGB 空間内の点群とみる
  → 二つの点群の最適輸送を計算
  → 得られた対応関係をもとに、ピクセルの色を置換する
- 出力: 色相を入れ替えた二つの画像




Gabriel Peyré. Optimal Transport in Imaging Sciences. 2012.
https://www.slideshare.net/gpeyre/optimal-transport-in-imaging-sciences

# 最適輸送は KL の限界を克服できる

- 最適輸送まとめ:
  - 最適輸送は距離構造を利用できる
    （↔ KL はできない）
  - 最適輸送はサポートが被っていないときにも利用できる
    （↔ KL はできない）
  - 最適輸送は分布の対応関係を得ることができる
    （↔ KL はできない）
- 点どうしに自然な距離を導入できるときには最適輸送チャンス
  「猫」と「ライオン」は「猫」と「鳥」より近い（クラス分類）
  温度のような順序尺度
  ユークリッド空間上の点群

# KL を最適輸送に置き換えることを提案する論文たち

- 実際、KL を最適輸送に置き換える提案がさまざまな文脈で
  なされている
- Arjovsky et al. Wasserstein GAN. ICML 2017.
  GAN のロスを JS ダイバージェンスから最適輸送距離に
- Frogner et al. Learning with Wasserstein Loss.
  NeurIPS 2015.
  マルチクラス分類のロスをクロスエントロピーから最適輸送距離に
- Liu et al. Importance-Aware Semantic Segmentation
  in Self-Driving with Discrete Wasserstein Training.
  AAAI 2020.
  セグメンテーションのロスをクロスエントロピーから最適輸送距離に

# KL が現れたら最適輸送に置き換えられないか考えてみる

take home message

**最適輸送は KL の欠点を克服できる。**
**手法中に KL が現れたら最適輸送を考えてみましょう。**

- 少しズルいですが:
  卒論のテーマが思い浮かばないとき、KL を使っている手法を探して
  最適輸送に置き換えるだけで論文になります。

山ほどあります

- ただし:
  タスクによって最適輸送との相性あり。
  距離構造の定義の仕方は工夫のみせどころ。
  よいタスク・距離構造をうまく選べると非常によい研究になります。

# 最適輸送の定義と求め方

# ヒストグラムの最適輸送距離の定式化: 線形計画

- 入力: 比較するヒストグラム $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$
  各点の距離を表す行列 $\mathbf{C} \in \mathbb{R}^{n \times n}$
- 出力: ヒストグラムの距離 $\mathrm{OT}(\mathbf{a}, \mathbf{b}, \mathbf{C}) \in \mathbb{R}$
- 最適輸送距離を以下の最適化問題の最適値と定義する

$$\underset{P \in \mathbb{R}^{n \times n}}{\text{minimize}} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} P_{ij}$$ 👈 総コスト

$$\text{s.t.} \quad P_{ij} \geq 0 \quad \forall i, j$$ 👈 輸送量は非負

$$\sum_{j=1}^{n} P_{ij} = a_i \quad \forall i$$ 👈 余りなし

$$\sum_{i=1}^{n} P_{ij} = b_j \quad \forall j$$ 👈 不足なし

決定変数 $P_{ij}$ は点 i から点 j に輸送する量を表す

これは線形計画

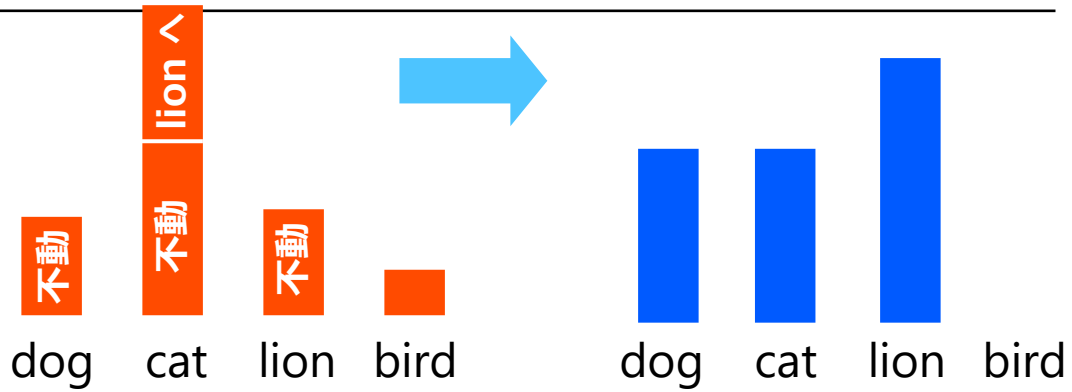
# ヒストグラム比較の例

- 入力:

dog, cat, lion, bird

$$\mathbf{a} = (0.2, 0.5, 0.2, 0.1)$$

$$\mathbf{b} = (0.3, 0.3, 0.4, 0.0)$$




dog, cat, lion, bird

$$\mathbf{C} = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 \\ 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \text{dog, cat, lion, bird} \end{matrix}$$

↑ cat と lion のミスはコストが低い

- 出力:

cat から lion へ
0.2 の輸送

$$\mathbf{P}^* = \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.3 & 0.2 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathrm{OT}(\mathbf{a}, \mathbf{b}, \mathbf{C}) = 0.4$$

# 点群の最適輸送距離の定式化: 線形計画

- 入力: 比較する点群 $\mu = \{x_1, \cdots, x_n\}, \nu = \{y_1, \cdots, y_m\} \subset \mathcal{X}$
  各点の距離を表す関数 $C\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- 出力: 点群の距離 $\mathrm{OT}(\mu, \nu, C) \in \mathbb{R}$
- 最適輸送距離を以下の最適化問題の最適値と定義する

$$\underset{P \in \mathbb{R}^{n \times m}}{\text{minimize}} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} C(x_i, y_j) P_{ij}$$ 👉 総コスト

$$\text{s.t.} \quad P_{ij} \geq 0 \quad \forall i, j$$ 👉 輸送量は非負

$$\sum_{j=1}^{m} P_{ij} = \frac{1}{n} \quad \forall i$$ 👉 余りなし

$$\sum_{i=1}^{n} P_{ij} = \frac{1}{m} \quad \forall j$$ 👉 不足なし

これは線形計画

決定変数 $P_{ij}$ は点 i から点 j に輸送する量を表す

# 連続分布比較の場合の最適輸送距離の定式化

- 入力: 比較する確率分布 $\mu, \nu$
  各点の距離を表す関数 $C\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- 出力: 分布の距離 $\mathrm{OT}(\mu, \nu, C) \in \mathbb{R}$
- 最適輸送距離を以下の最適化問題の最適値と定義する

$$\underset{\pi \in \mathcal{P}(\mathcal{X}\times\mathcal{X})}{\text{minimize}} \quad \int_{\mathcal{X}\times\mathcal{X}} C(x,y)d\pi(x,y)$$

👉 総コスト

$$\text{s.t.} \quad \pi(\mathcal{A}, \mathcal{B}) \geq 0 \quad \forall \mathcal{A}, \mathcal{B}$$

👉 輸送量は非負

$$\pi(\mathcal{A}, \mathcal{X}) = \mu(\mathcal{A}) \ \forall \mathcal{A}$$

👉 余りなし

$$\pi(\mathcal{X}, \mathcal{B}) = \nu(\mathcal{B}) \ \forall \mathcal{B}$$

👉 不足なし

ざっくり言うと
和を積分にして
連続にしている

# 離散最適輸送は線形計画、連続の場合は工夫が必要

- ヒストグラムの場合と点群の場合はほとんど同じ線形計画
  アルゴリズムを考える上ではこれらは同じ問題とみなされることが多い
- 線形計画なので、既存のソルバを用いて解くことができる
- 連続分布どうしの最適輸送は連続分布の最適化問題になるので
  解くのが難しい
  1. 連続分布からサンプリングを行い点群比較に帰着する、または
  2. このチュートリアルの後半で紹介する双対を使ったアプローチで
     直接解く
- 以下、主に離散最適輸送を考える

# ワッサースタイン距離は最適輸送の特殊ケース

- 最適輸送距離はコスト行列の設定次第で距離公理は満たさない
  すべてコストが 0 のとき、すべての分布の距離はゼロになってしまう
- ワッサースタイン距離は距離公理を満たす最適輸送の特殊ケース
- 定義: ワッサースタイン距離
  $\mathcal{X}$ 上の距離関数 $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ と実数 $p \geq 1$
  を用いてコスト行列を $C(x, y) = d(x, y)^p$ と定義する。
  $W_p(\mu, \nu) = \mathrm{OT}(\mu, \nu, C)^{1/p}$ を p-ワッサースタイン距離という。
- ユークリッド距離の 1-ワッサースタイン $C(x, y) = \|x - y\|_2$ や
  2-ワッサースタイン $C(x, y) = \|x - y\|_2^2$ がよく用いられる

# ワッサースタイン距離は距離公理を満たす

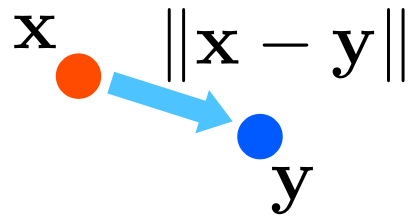
- 定理：ワッサースタイン距離は距離公理を満たす
  証明は下記文献など参照
- すなわち、点の距離として距離公理を満たすものを使えば、
  分布の距離として自動的に距離公理が満たされるようになる
- cf. KL は距離の公理を満たさない

Gabriel Peyré, Marco Cuturi. Computational Optimal Transport. 2019.

# ワッサースタイン距離の例

- $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ について、$\mathbf{x}$, $\mathbf{y}$ が確率 1 で出てくる確率分布 $\delta_{\mathbf{x}}, \delta_{\mathbf{y}}$ のワッサースタイン距離は

$$W_p(\delta_{\mathbf{x}}, \delta_{\mathbf{y}}) = \|\mathbf{x} - \mathbf{y}\|$$




- 一方、KL ダイバージェンスは

$$\mathrm{KL}(\delta_{\mathbf{x}} \| \delta_{\mathbf{y}}) = \begin{cases} 0 & (\mathbf{x} = \mathbf{y}) \\ \infty & (\mathbf{x} \neq \mathbf{y}) \end{cases}$$

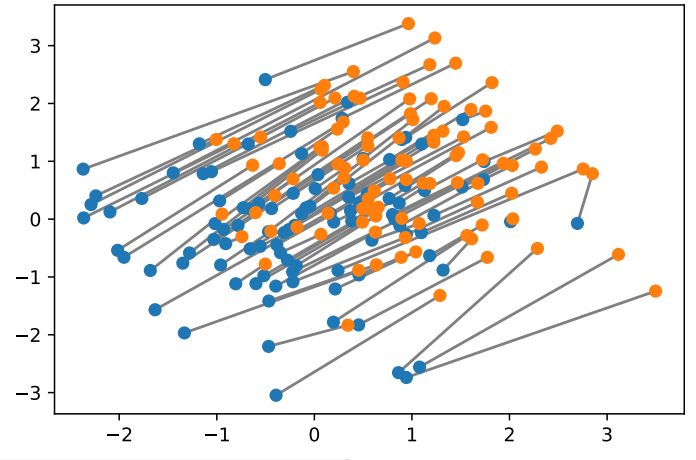
- ワッサースタイン距離の方がより細かく見分けているといえる

# ソルバの紹介: POT がオススメ

- 最適輸送を実際に使う際には様々なソルバが利用できる
- Python Optimal Transport (POT): オススメ
  `pip install pot` でインストールできる
  a: numpy array (n,)
  b: numpy array (m,)
  C: numpy array (n, m)
  `ot.emd(a, b, C)` を呼び出せば最適輸送行列 P* が返る
- Scipy:
  `scipy.optimize.linear_sum_assignment`
  → 1 対 1 対応の場合のみ
  `scipy.optimize.linprog` → 一般の線形計画

# POT の使用例: 簡単に使えます

- 数値例: 二つの正規分布からの点群の比較




```python
import numpy as np
import matplotlib.pyplot as plt
import ot # POT ライブラリ

n = 100 # 点群サイズ
mu = np.random.randn(n, 2)     # 入力分布 1
nu = np.random.randn(n, 2) + 1 # 入力分布 2
a = np.ones(n) / n # 質量ヒストグラム (1/n, ..., 1/n)
b = np.ones(n) / n # 質量ヒストグラム (1/n, ..., 1/n)
C = np.linalg.norm(nu[np.newaxis] - mu[:, np.newaxis], axis=2) # コスト行列

P = ot.emd(a, b, C) # 最適輸送距離の計算

plt.scatter(mu[:, 0], mu[:, 1]) # mu の散布図描写
plt.scatter(nu[:, 0], nu[:, 1]) # nu の散布図描写
for i in range(n):
    j = P[i].argmax() # i の対応相手: 最もたくさん輸送している先
    plt.plot([mu[i, 0], nu[j, 0]], [mu[i, 1], nu[j, 1]], c='grey', zorder=-1)
```

↑ **コピペで試せます**

# 最適輸送は線形計画。ソルバで簡単に解ける。

take home message

**最適輸送距離は線形計画として定式化される**
**ワッサースタイン距離は距離公理を満たす特殊ケース**
**ソルバを利用すると簡単に最適輸送を計算できる**

# シンクホーンアルゴリズム

# 高速でホワイトボックスなアルゴリズムが欲しい

- 前章での議論:
  最適輸送は線形計画 → 線形計画ソルバに投げると解ける
- 欠点:
  汎用ソルバ・厳密ソルバは遅い 😫
  ブラックボックスなので、自分の手法に有機的に組込みづらい 😫
- これから紹介するシンクホーンアルゴリズム:
  高速 😆
  シンプル 😆 → さまざまな手法と組み合わせやすい 😆
  ただし、厳密な最適輸送は求まらない 😫

# エントロピー正則化つき最適輸送問題を考える

- 以下のエントロピー正則化つき最適輸送問題を考える

$$\begin{aligned}
\underset{P \in \mathbb{R}^{n \times m}}{\text{minimize}} \quad & \left( \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} P_{ij} \right) - \varepsilon \sum_{i=1}^{n} \sum_{j=1}^{m} P_{ij} (1 - \log P_{ij}) \\
\text{s.t.} \quad & P_{ij} \geq 0 \quad \forall i, j \\
& \sum_{j=1}^{m} P_{ij} = a_i \quad \forall i \\
& \sum_{i=1}^{n} P_{ij} = b_j \quad \forall j
\end{aligned}$$

エントロピー項 = H(P)

- $\varepsilon > 0$ は正則化係数（ハイパーパラメータ）
  オリジナルの最適輸送とは別問題。ただし $\varepsilon \to 0$ で元問題に。

# エントロピー正則化つき問題は強凸

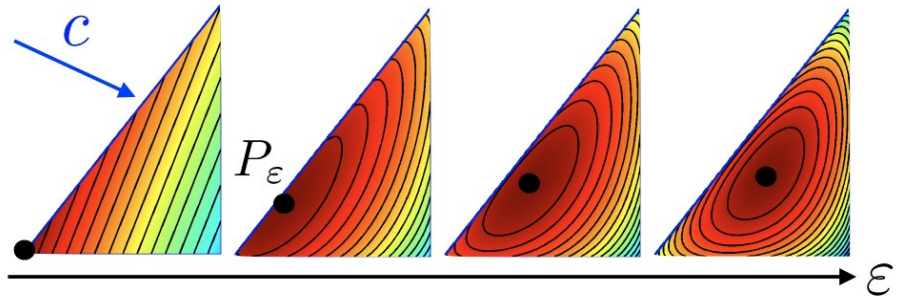
- 嬉しさ：エントロピー正則化つき問題の目的関数は強凸 😆

線形　　エントロピーは強凹

$$\left(\sum_{i=1}^{n}\sum_{j=1}^{m} C_{ij}P_{ij}\right) - \varepsilon H(P)$$

→ 全体で強凸
→ 最適化しやすい

- 制約が線形制約だけということは実行可能領域は凸なので
  実行可能領域凸 ＋ 目的関数強凸 → **最適解は一意に定まる**
  ↔ 線形計画は面全体で最適をとることがあり、一意に定まらない




Gabriel Peyré, Marco Cuturi. Computational Optimal Transport. 2019.

# 双対問題は制約なし最大化問題

- エントロピー正則化つき最適輸送の（凸計画としての）双対問題は

$$\underset{f\in\mathbb{R}^n, g\in\mathbb{R}^m}{\text{maximize}} \sum_{i=1}^{n} a_i f_i + \sum_{j=1}^{m} b_j g_j - \varepsilon \sum_{i=1}^{n}\sum_{j=1}^{m} \exp((f_i + g_j - C_{ij})/\varepsilon)$$

→ 制約**なし**最大化問題 😆

- 双対問題を解くことをめざす
- 双対問題の導出は下記文献などを参照

Gabriel Peyré, Marco Cuturi. Computational Optimal Transport. 2019.
最適輸送の解き方 https://www.slideshare.net/joisino/ss-249394573

# 双対問題の目的関数の勾配は f 内 g 内で絡みなし

- 双対問題の目的関数を D をおく

$$D(f,g) = \sum_{i=1}^{n} a_i f_i + \sum_{j=1}^{m} b_j g_j - \varepsilon \sum_{i=1}^{n} \sum_{j=1}^{m} \exp((f_i + g_j - C_{ij})/\varepsilon)$$

- D の f と g についての勾配は以下の通り

$$\frac{\partial D}{\partial f_i} = a_i - \sum_j \exp((f_i + g_j - C_{ij})/\varepsilon) \qquad \frac{\partial D}{\partial g_j} = b_j - \sum_i \exp((f_i + g_j - C_{ij})/\varepsilon)$$

- $f_i$ の勾配の中に $f_k$ ($k \neq i$), $g_j$ の勾配の中に $g_k$ ($k \neq j$) は **出てこない**（絡みなし）

# f ごと、g ごとに最適値が厳密に求まる

- シンクホーンアルゴリズムの基本的な考えは座標向上法
- f を固定したときの g の最適値は勾配イコールゼロとおいて

$$g_j^* = \varepsilon \log b_j - \varepsilon \log \sum_{i=1}^{n} \exp((f_i - C_{ij})/\varepsilon)$$

g を固定したときの f の最適値も同様に

$$f_i^* = \varepsilon \log a_i - \varepsilon \log \sum_{j=m}^{n} \exp((g_j - C_{ij})/\varepsilon)$$

- **シンクホーンアルゴリズムは f 固定での g の厳密最適化 → g 固定での f の厳密最適化を交互に繰り返す**

# シンクホーンアルゴリズムは f と g を交互に最適化する

- 対数領域でのシンクホーンアルゴリズム
- ステップ 1: $f^{(1)}$ を適当に初期化（例えばゼロベクトル), t = 1 に
- ステップ 2: $$g_j^{(t+1)} = \varepsilon \log b_j - \varepsilon \log \sum_{i=1}^{n} \exp((f_i^{(t)} - C_{ij})/\varepsilon)$$
- ステップ 3: $$f_i^{(t+1)} = \varepsilon \log a_i - \varepsilon \log \sum_{j=m}^{n} \exp((g_j^{(t+1)} - C_{ij})/\varepsilon)$$
- ステップ 4: f と g が収束するまで t ← t + 1 でステップ 2 へ
- 目的関数が微分可能で各ステップ唯一解なので**大域最適に収束**

# 指数関数を使って変数変換するとシンプルに書ける

- 先程のイテレーションでは log や exp がたくさん出てくる

$$g_j^{(t+1)} = \varepsilon \log b_j - \varepsilon \log \sum_{i=1}^{n} \exp((f_i^{(t)} - C_{ij})/\varepsilon)$$

$$f_i^{(t+1)} = \varepsilon \log a_i - \varepsilon \log \sum_{j=m}^{n} \exp((g_j^{(t+1)} - C_{ij})/\varepsilon)$$

- $u \leftarrow \exp(f/\varepsilon), v \leftarrow \exp(g/\varepsilon)$ と変数変換するとシンプルに

$$v_j^{(t+1)} = b_j \left( \sum_{i=1}^{n} \exp(-C_{ij}/\varepsilon) u_i^{(t)} \right)^{-1}$$

$$u_i^{(t+1)} = a_i \left( \sum_{j=1}^{m} \exp(-C_{ij}/\varepsilon) v_j^{(t+1)} \right)^{-1}$$

# ギブスカーネルの定義

- さらに $K \in R^{n \times m}$ を以下で定める

exp は単調

コストに負号 → 類似度

$$K_{ij} = \exp(-C_{ij}/\varepsilon)$$

- K を**ギブス (Gibbs) カーネル**という
- i と j の類似度を表している
- 例えば C がユークリッド距離の自乗（2-ワッサースタイン）のときガウスカーネルになる

# シンクホーンのイテレーションは行列積でかける

- ギブスカーネル行列を使うとさらにシンプルに

$$v^{(t+1)} = \frac{b}{K^\top u^{(t)}}$$

$$u^{(t+1)} = \frac{a}{K v^{(t+1)}}$$

ふつうシンクホーンというと
この形の更新アルゴリズムを指す

- ただし $K^T$ は転置行列、割り算は要素ごと、分母は行列ベクトル積
- **非常に単純に実装できる**
- 行列ベクトル積がメインなので **GPU で高速計算可能**

# シンクホーン変数から元の変数への戻し方

- シンクホーンにより (u, v) が求まると、変数変換の式より元の双対変数は

$$f \leftarrow \varepsilon \log(u), g \leftarrow \varepsilon \log(v)$$

- 主問題の変数は双対最適解と主最適解の対応関係より

$$P_{ij} = \exp((f_i + g_j - C_{ij})/\varepsilon) = u_i v_j K_{ij}$$

f, g が完璧に収束しない限り、求めた $P_{ij}$ は厳密には実行可能とは限らないことに注意（実用上は多少の誤差は問題ないことも多い）

- この $P_{ij}$ から違反分をいい感じに分配して実行可能解を計算するアルゴリズムも提案されている [Altschuler+ 2017]

Jason Altschuler, Jonathan Weed, Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. NeurIPS 2017.

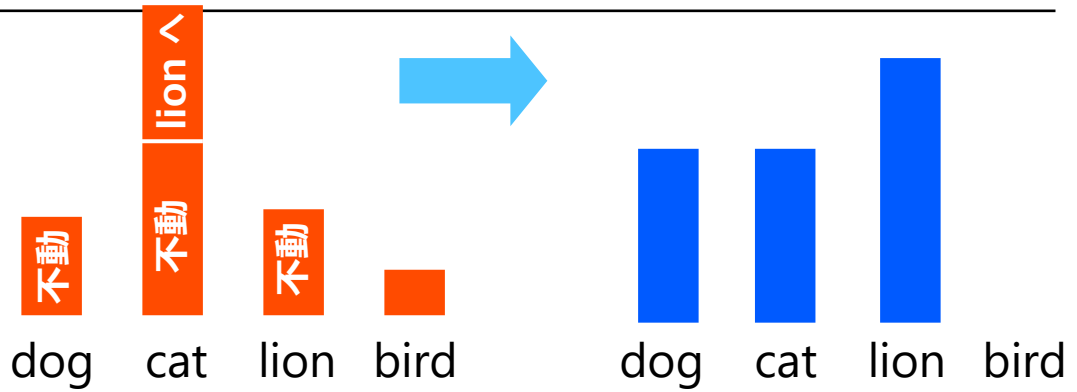
# シンクホーンアルゴリズム: 例

- 例（以前使ったもの）：

dog, cat, lion, bird

$$\mathbf{a} = (0.2, 0.5, 0.2, 0.1)$$

$$\mathbf{b} = (0.3, 0.3, 0.4, 0.0)$$




dog, cat, lion, bird

$$\mathbf{C} = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 \\ 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \text{dog, cat, lion, bird}$$

↑ cat と lion のミスはコストが低い

- エントロピー正則化なしの場合の出力：

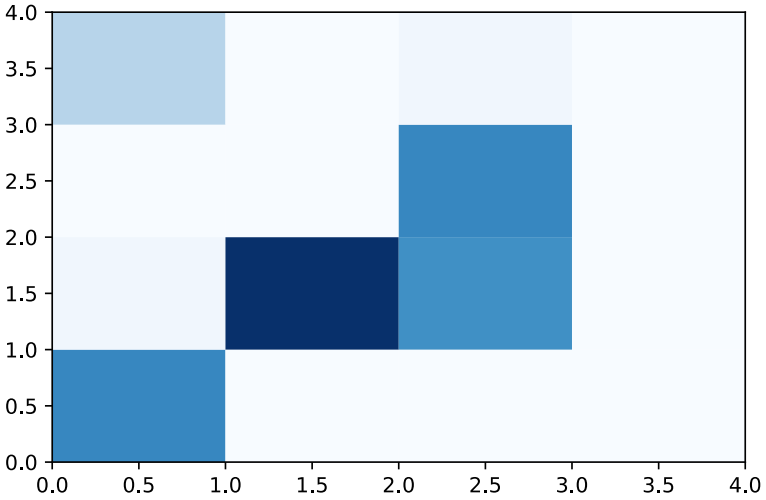
$$\mathbf{P}^* = \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.3 & 0.2 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathrm{OT}(\mathbf{a}, \mathbf{b}, \mathbf{C}) = 0.4$$

# シンクホーンアルゴリズム: 例

外部ライブラリに頼らない

```python
import numpy as np
import matplotlib.pyplot as plt

n, m = 4, 4
C = np.array([
    [0, 2, 2, 2],
    [2, 0, 1, 2],
    [2, 1, 0, 2],
    [2, 2, 2, 0]])
a = np.array([0.2, 0.5, 0.2, 0.1])
b = np.array([0.3, 0.3, 0.4, 0.0])

eps = 0.2 # 大きいと高速に、小さいと厳密に
K = np.exp(- C / eps) # ギブスカーネルの計算

u = np.ones(n) # すべて 1 で初期化
for i in range(100):
    v = b / (K.T @ u) # ステップ (2)
    u = a / (K @ v)   # ステップ (3)

f = eps * np.log(u + 1e-9) # 対数領域に戻す
g = eps * np.log(v + 1e-9) # 対数領域に戻す

P = u.reshape(n, 1) * K * v.reshape(1, m) # 主解

plt.pcolor(P, cmap=plt.cm.Blues) # 解の可視化
```




↑ エントロピーなしの解とほぼ一致
（左下が (1, 1) であることに注意）

メインロジックはわずか 3 行

← **コピペで試せます**

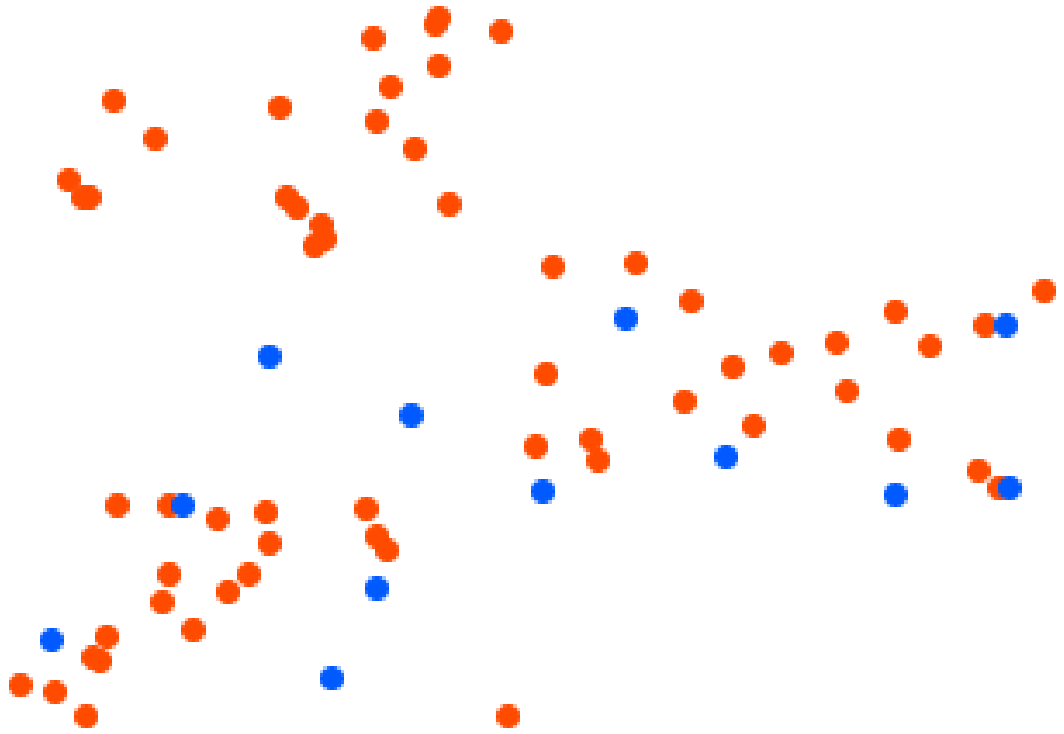
# シンクホーンは自動微分できる

- 計算が行列演算だけからなるので、自動微分ライブラリにより微分が求まる
- たとえば以下のように numpy の代わりに torch とする

```
import torch

K = torch.exp(- C / eps) # ギブスカーネルの計算

u = torch.ones(n) # すべて 1 で初期化
for i in range(100):
    v = b / (K.T @ u) # ステップ (2)
    u = a / (K @ v)   # ステップ (3)

f = eps * torch.log(u + 1e-9) # 対数領域に戻す
g = eps * torch.log(v + 1e-9) # 対数領域に戻す

P = u.reshape(n, 1) * K * v.reshape(1, m) # 主解
loss = (P * C).sum()
loss.backward()
```

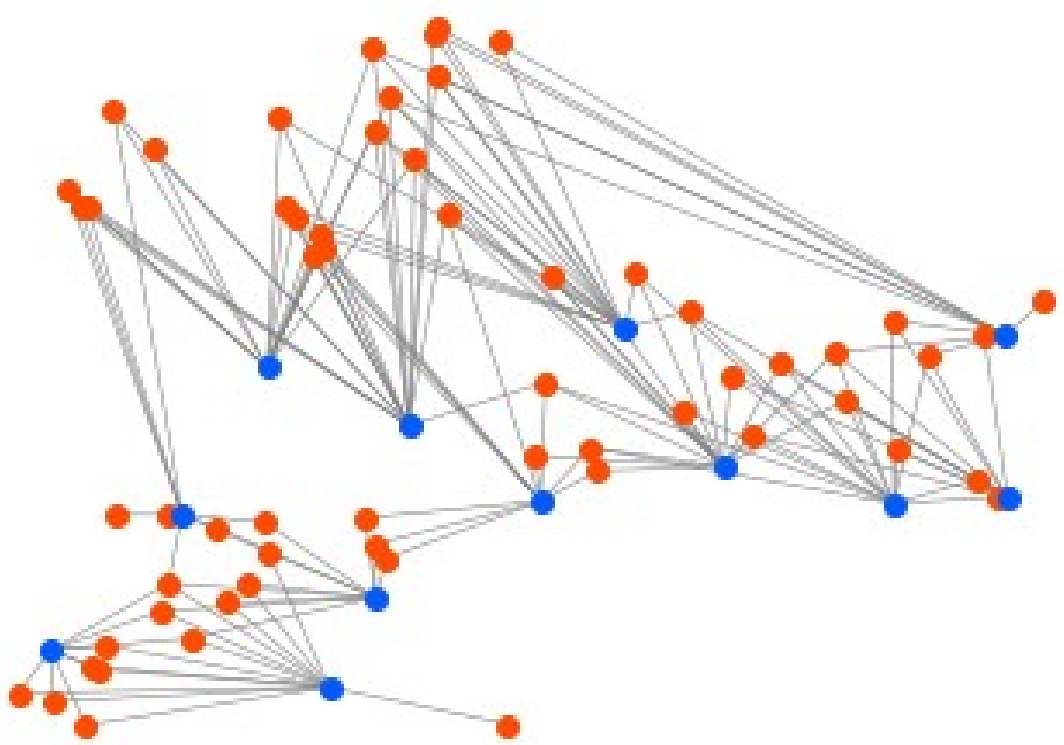
# 自動微分の例

- 例: 以下の点群を考える。赤は固定、青の位置がパラメータ。
  Sinkhorn により計算された値をロスとして勾配法で最適化。



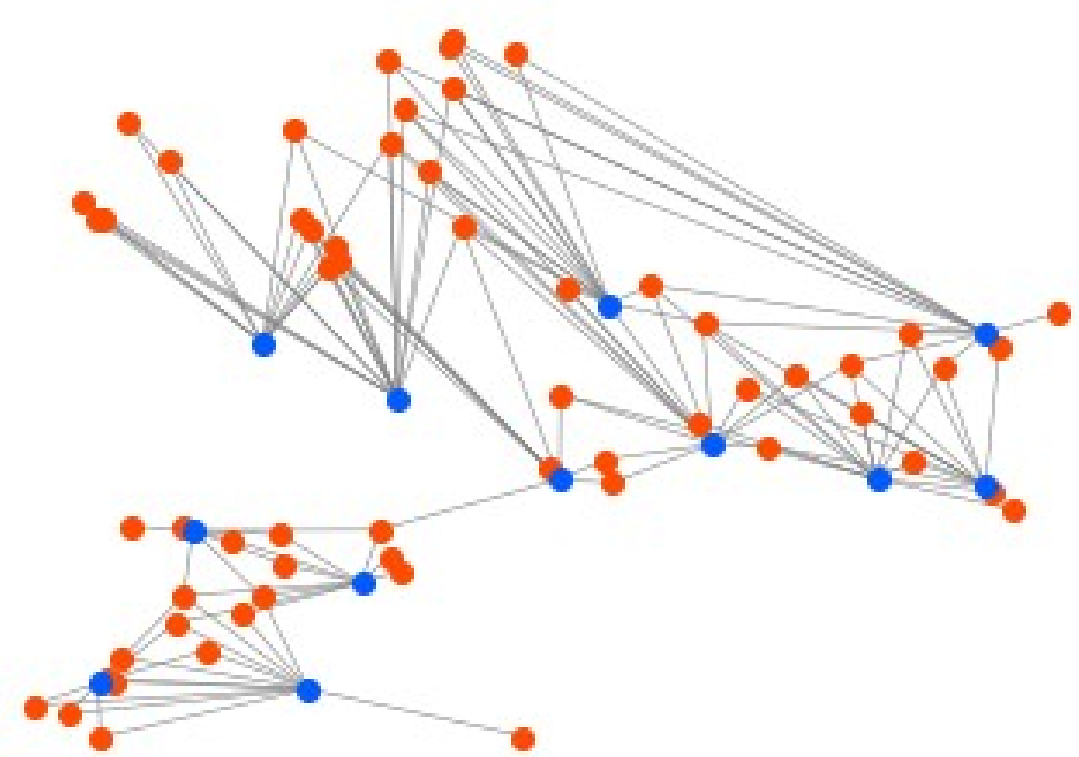
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



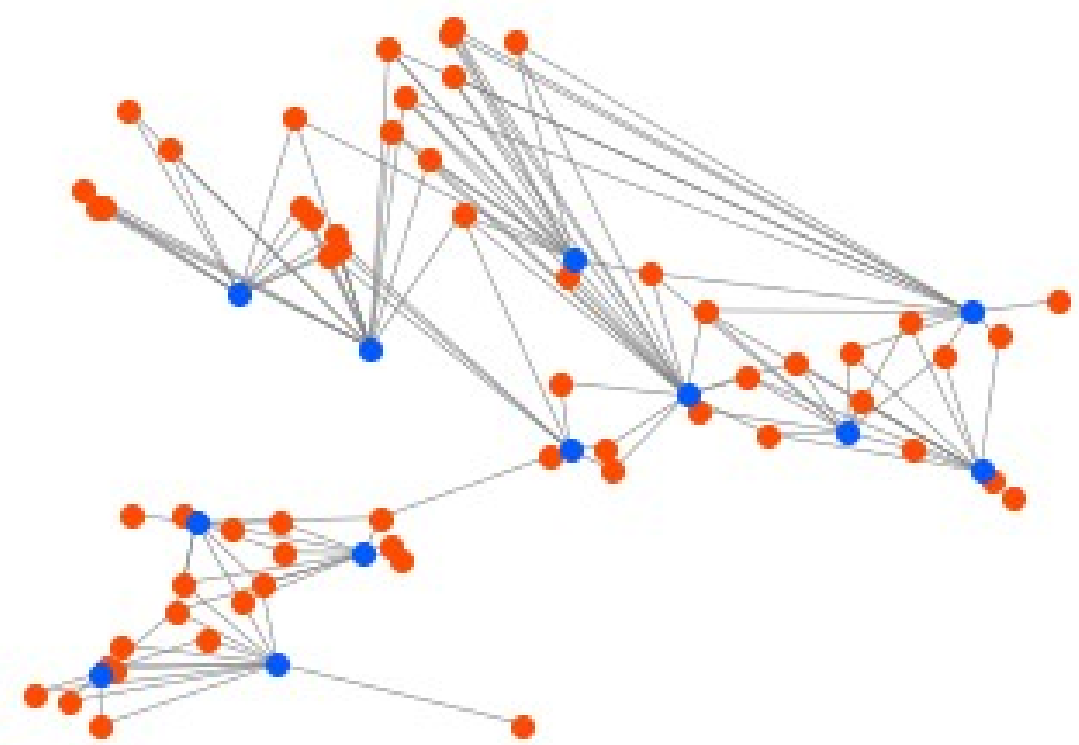
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



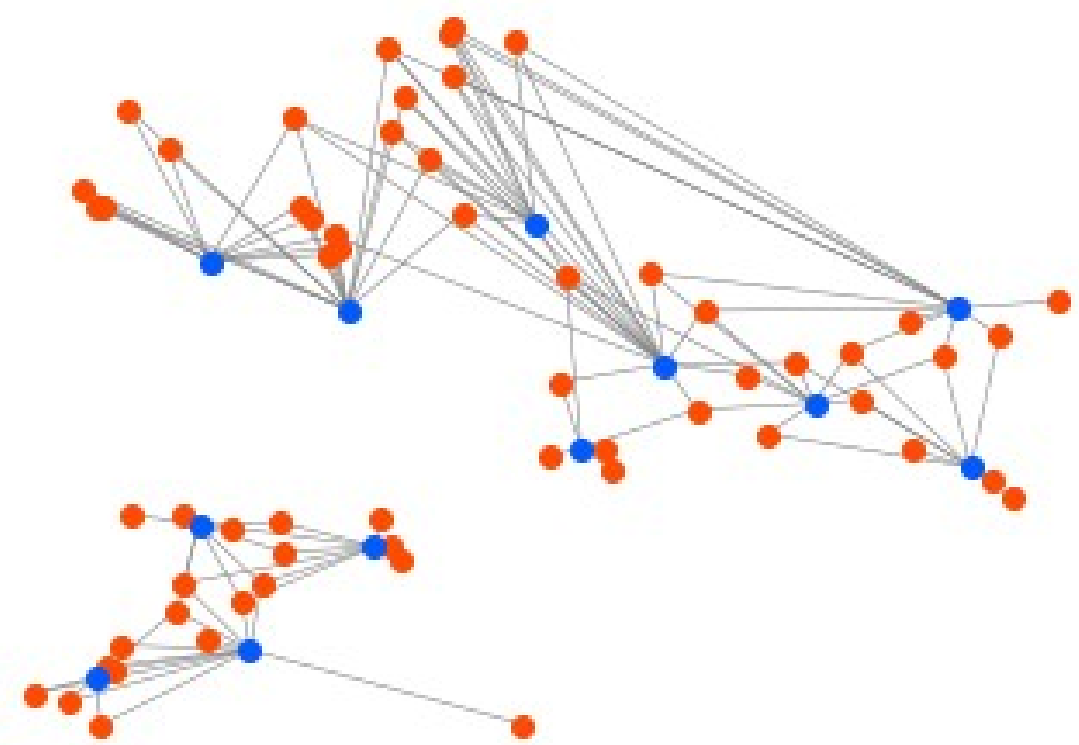
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



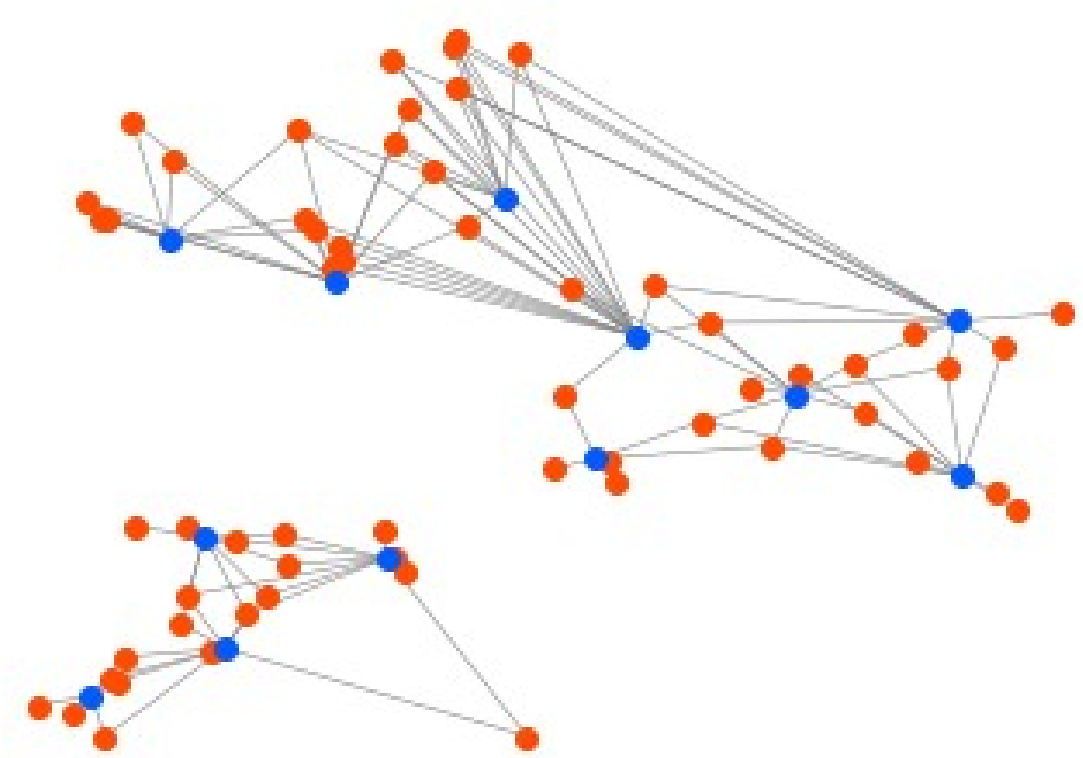
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



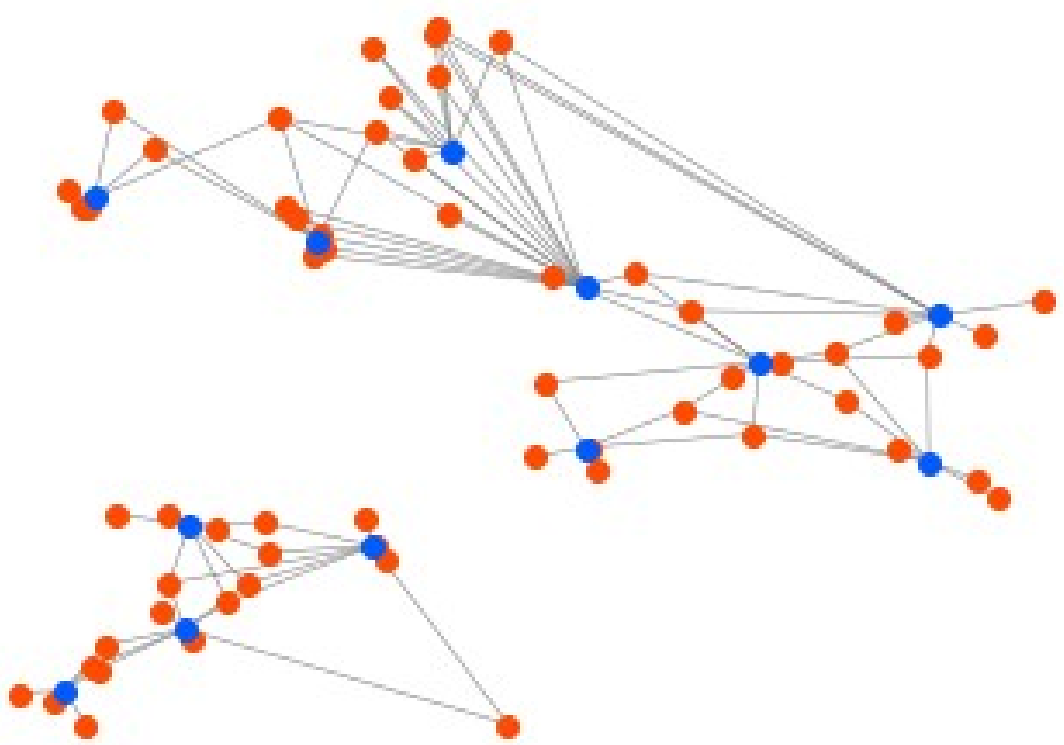
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



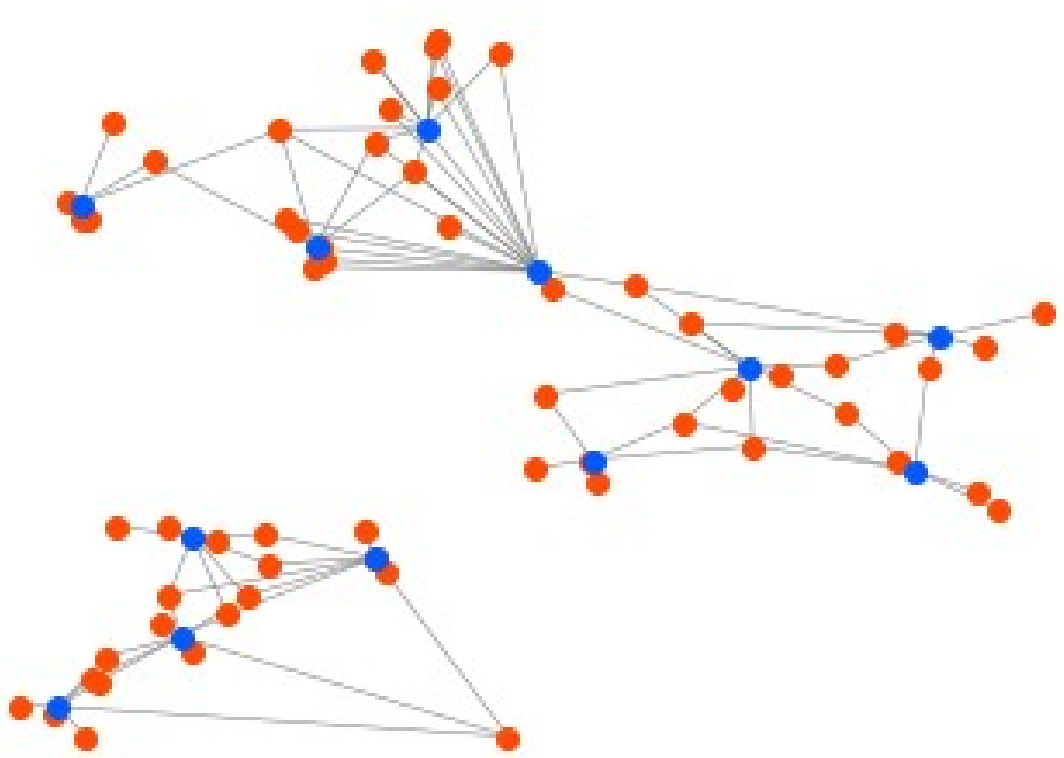
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



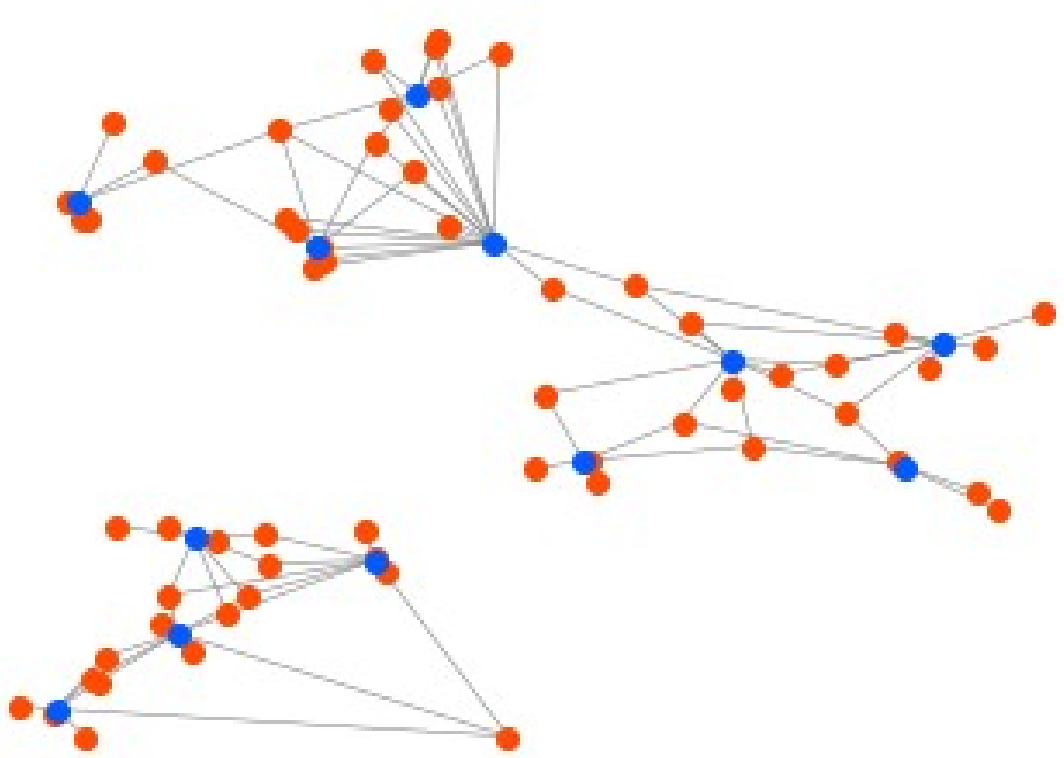
# 自動微分の例

- Sinkhorn により求まった輸送行列を参考に記している



# 自動微分の例

- 輸送距離の小さい青の配置が求まった



# ソースコード全貌

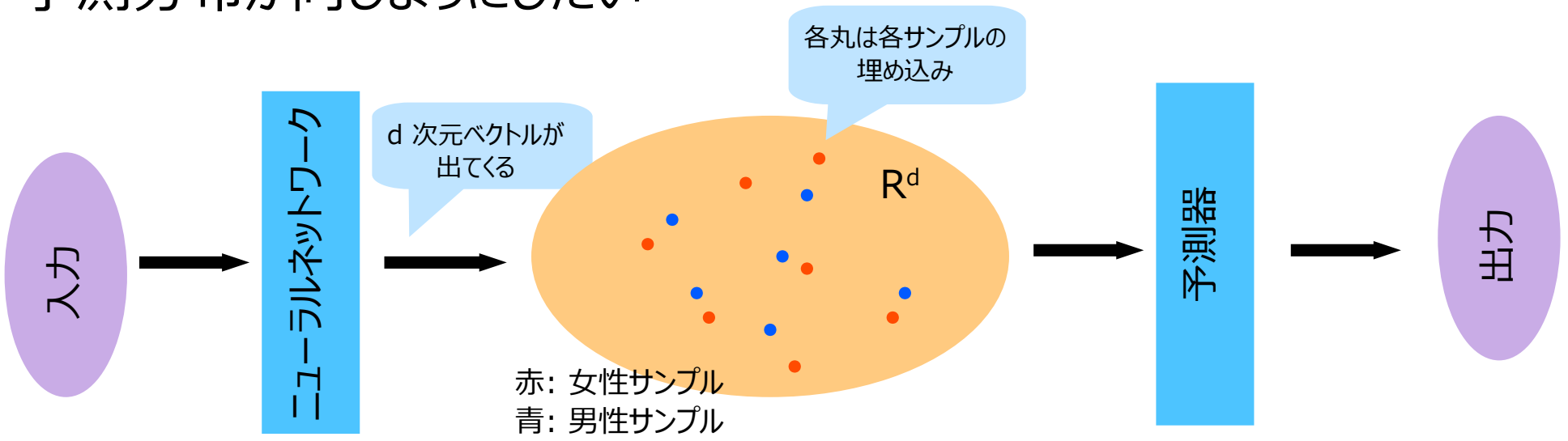
```
import matplotlib.pyplot as plt
import torch
import torch.optim
import torch.nn

# データ生成
torch.manual_seed(0)
x = torch.rand(20, 2)
y = torch.rand(20, 2) + \
      torch.FloatTensor([0, 2])
z = torch.rand(20, 2) + \
      torch.FloatTensor([1, 1])
mu = torch.cat([x, y, z])
nu = torch.rand(12, 2) * 2
nu = torch.nn.parameter.Parameter(nu)
n, m = len(mu), len(nu)
a = torch.ones(n) / n
b = torch.ones(m) / m
```

**コピペで試せます**

```
optimizer = torch.optim.SGD([nu], lr=1.0)
for it in range(100):
    eps = 0.1
    D = torch.linalg.norm(mu.reshape(n, 1, 2) - \
        nu.reshape(1, m, 2), axis=2)
    K = torch.exp(- D / eps) # ギブスカーネルの計算

    u = torch.ones(n) # すべて 1 で初期化
    for i in range(100):
        v = b / (K.T @ u) # ステップ (2)
        u = a / (K @ v)   # ステップ (3)

    f = eps * torch.log(u + 1e-9) # 対数領域に戻す
    g = eps * torch.log(v + 1e-9) # 対数領域に戻す

    P = u.reshape(n, 1) * K * v.reshape(1, m) # 主解
    loss = (P * D).sum()
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    plt.clf()
    plt.scatter(mu[:, 0], mu[:, 1])
    plt.scatter(nu.data[:, 0], nu.data[:, 1])
    plt.show()
```

# 最新の手法でも公平予測問題にシンクホーンが使われる

- 他にも、二つの集合の距離を近づけるための微分可ロスとして使われる
- 例えば、公平性の担保のため、男性についての予測と女性についての予測分布が同じようにしたい




- 予測誤差 ＋ 赤と青の最適輸送距離を最小化 [Oneto+ NeurIPS 2020]

Luca Oneto, Michele Donini, Giulia Luise, Carlo Ciliberto, Andreas Maurer, Massimiliano Pontil. Exploiting MMD and Sinkhorn Divergences for Fair and Transferable Representation Learning. NeurIPS 2020.

# まとめ: シンクホーンはシンプルかつ高い柔軟性を誇る

take home message

**エントロピーを導入すると最適化が簡単に**
**シンクホーンは行列演算だけでできるシンプル最適化法**
**簡単にニューラルネットワークに組み込むことができる**

# Wasserstein GAN
# （連続分布についての最適輸送）

# 連続分布の最適輸送を直接解くアプローチを考える

- 連続分布についての最適輸送を考える

$$\underset{\pi \in \mathcal{P}(\mathcal{X}\times\mathcal{X})}{\text{minimize}} \quad \int_{\mathcal{X}\times\mathcal{X}} C(x,y)d\pi(x,y)$$

👉 総コスト

$$\text{s.t.} \quad \pi(\mathcal{A},\mathcal{B}) \geq 0 \quad \forall \mathcal{A},\mathcal{B}$$

👉 輸送量は非負

$$\pi(\mathcal{A},\mathcal{X}) = \mu(\mathcal{A}) \ \forall \mathcal{A}$$

👉 余りなし

$$\pi(\mathcal{X},\mathcal{B}) = \nu(\mathcal{B}) \ \forall \mathcal{B}$$

👉 不足なし

# 双対を取ると 2 つの関数の最適化に -> 難しい

- 双対をとると

$$\begin{aligned} \underset{f,g:\ \mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad & \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[g(y)] \\ \text{s.t.} \quad & f(x) - g(y) \le C(x,y) \quad \forall x, y \in \mathcal{X} \end{aligned}$$

この問題の最適値 = 最適輸送距離（強双対性）
双対の導出は下記文献などを参照

- 連続関数 f, g を最適化する問題
  → 依然難しい

最適輸送の解き方 https://www.slideshare.net/joisino/ss-249394573

# 1-ワッサースタインのときには 1 つの関数の最適化に

- 命題: コスト関数 C が距離関数のとき（1-ワッサースタイン）、
  最適解において f = g

  証明は下記文献など参照

- 以降 1-ワッサースタインのみを考える。これにより変数 g を削除できる

$$\begin{aligned} \underset{f:\,\mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad & \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)] \\ \text{s.t.} \quad & f(x) - f(y) \le C(x,y) \quad \forall x, y \in \mathcal{X} \end{aligned}$$

- 連続関数 f を最適化する問題

最適輸送の解き方 https://www.slideshare.net/joisino/ss-249394573

# 1-ワッサースタインの双対の条件はリプシッツと等価

$$\underset{f:\,\mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)]$$

$$\text{s.t.} \quad f(x) - f(y) \le C(x,y) \quad \forall x, y \in \mathcal{X}$$

C が距離関数のとき、
これは f が 1-リプシッツということ

# 以降、リプシッツ連続関数を最適化する問題を考える

$$\underset{f:\,\mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)]$$

$$\text{s.t.} \qquad f \text{ is 1-Lipschitz}$$

- リプシッツ性は局所的な条件（各点での勾配）で表せるので嬉しい
- 双対や f = g の議論は込み入っているので、短時間では詳細まで追えませんでした。気になる人は下記文献をあとで参照してください。
- 以降は、上記の問題さえ解ければ最適輸送が解けるということを前提に、上記の問題を解くことに集中します。

最適輸送の解き方 https://www.slideshare.net/joisino/ss-249394573

# μ のサンプルを上げ、ν のサンプルを下げるのが目的

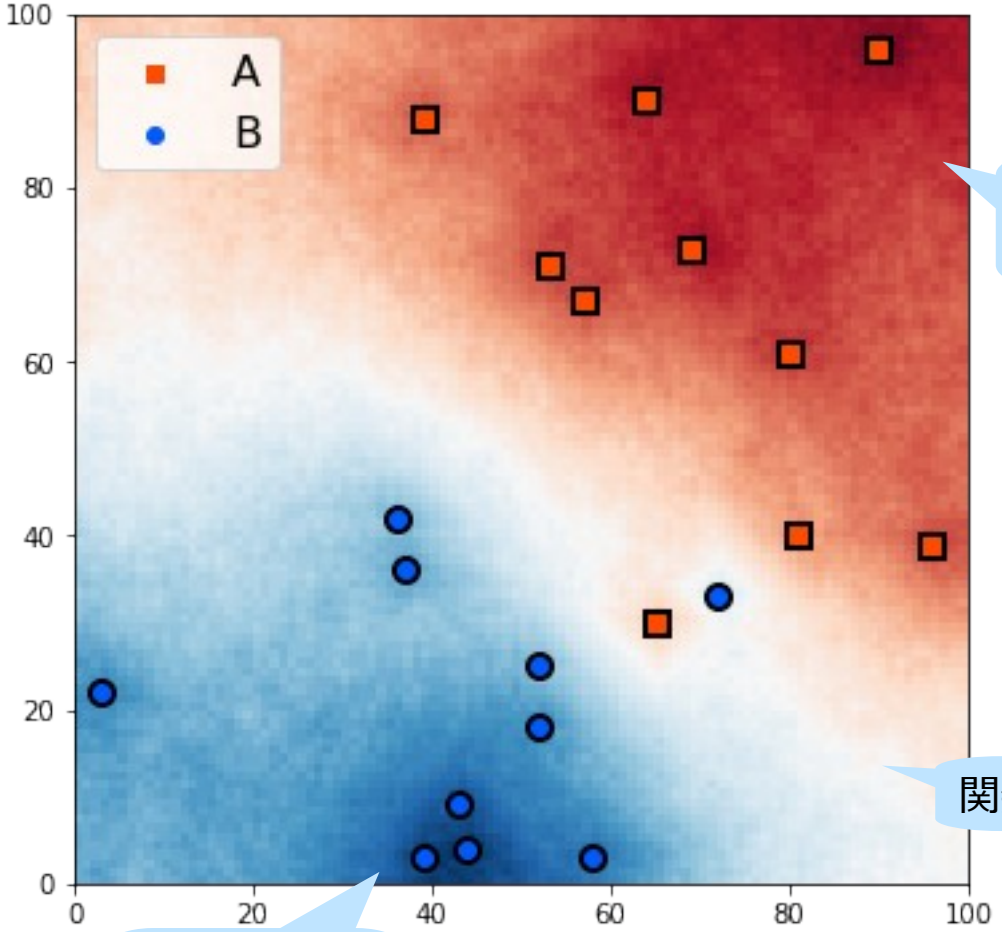
$$\underset{f:\,\mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)]$$

$$\text{s.t.} \qquad\qquad f \text{ is 1-Lipschitz}$$

- f は μ からのサンプル上で大きければ大きいほど
  ν からのサンプル上で小さければ小さいほどよい
- ただし、f は「滑らか」（リプシッツ連続）でなければならない

# 関数最適化のイメージ

- 例: 赤点は μ からのサンプル、青点は ν からのサンプルに対応



# 双対問題は 2 クラス分類問題と見ることができる

- このように見ると、μ からのサンプルを正例、ν からのサンプルを負例としたときの **2 クラス分類問題と見ることができる**
- 激しく変化する分類器は制約違反
  「正則化」としてリプシッツ関数の条件

$$\begin{aligned} \underset{f:\,\mathcal{X}\to\mathbb{R}}{\text{maximize}} \quad & \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)] \\ \text{s.t.} \quad & f \text{ is 1-Lipschitz} \end{aligned}$$

# 分類問題をニューラルネットワークに任せる

- 「分類器」f をニューラルネットワークでモデリングすることを考える
- ロス関数はシンプルに

$$\mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)]$$

- これを経験誤差最小化 $\mathcal{L} = \sum_{y\sim\nu} f(y) - \sum_{x\sim\mu} f(x)$
- f が 1-リプシッツ連続であることを課すのは難しい
- ここでは素朴な解決策を 2 つ紹介

# パラメータの値を無理やり制限してリプシッツ性を課す

- 解決策 1: weight clipping [Arjovsky+ 2017]
- 訓練の際ニューラルネットワークの**各パラメータの絶対値が定数 γ > 0 を越えるたびに絶対値 γ にクリップする**（γ:ハイパラ）
- こうすると、ニューラルネットワーク f の表現できる関数は制限され急激な変化をする関数は表せなくなる
- γ の設定次第で、何らかの k > 0 について k-リプシッツであることが保証できる
- k-リプシッツな関数全てを表現できる訳ではないが、ニューラルネットワークの柔軟性よりそれなりに豊富な関数が表現できることが期待できる

Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN. ICML 2017

# 各点での勾配が 1 から離れるとペナルティ

- 解決策 2: gradient penalty [Gulrajani+ 2017]
- f が微分可のとき各点での勾配のノルムが 1 以下 ↔ 1-リプシッツ
- 色んな点で**勾配を評価して 1 から離れていたらペナルティを課す**

$$\sum_{y\sim\nu} f(y) - \sum_{x\sim\mu} f(x) + \lambda \sum_{x} (\|\nabla_x f(x)\|_2 - 1)^2$$

ハイパラ正則化係数

勾配を 1 に近づける

をロスとして最適化

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, Aaron C. Courville.. Improved Training of Wasserstein GANs. NeurIPS 2017

# ニューラルネットワークの定式化は GAN に用いられる

- f をニューラルネットワークで表すと f(x) は x について微分可能
  → 最適輸送コストを下げるには x をどう動かせばよいかが分かる
- ニューラルネットワークによる定式化は GAN（生成モデル）の訓練によく用いられる: **Wasserstein GAN**
- 自然サンプルの集合を $X \subset \mathbb{R}^d$ とする。例えば画像の集合
- GAN により、X に似たサンプル集合を生成したい
- GAN により生成したサンプルの集合を $Y \subset \mathbb{R}^d$ とし、X と Y の距離を最適輸送コストで測る
- これを最小化するように GAN を訓練する

# ニューラルネットワークの推定は大規模データと相性よし

- 典型的な X は非常に大きい（数百万枚の画像など）
- 生成モデルの生成結果 Y の候補は無限にある
- シンクホーンアルゴリズムでは全ての対を一度に解かなければならず、非常に大きいデータには適していない
- 「分類器」f を分類するという考え方に立てば、X と Y から**一部のデータをミニバッチとして取り出し f を徐々に訓練していけばよい**
  → ニューラルネットワークによる推定は**大規模（or 無限）のデータについて解くのと相性がよい**

# Wasserstein GAN: 生成モデルと f の交互訓練

- 1. 生成モデル G: $R^r \to R^d$ をランダムに初期化
  これはノイズ $z \in R^r$ を受け取り画像 $G(z) \in R^d$ を返す
  最適輸送問題を解くニューラルネットワーク f をランダムに初期化
- 2. G を使って画像 $Y = \{G(z_1), G(z_2), \ldots, G(z_m)\}$ を生成
- 3. f が X と Y を分類できるようにミニバッチで訓練して
  X と Y の（ミニバッチの）最適輸送コストを計算
- 4. f を使った最適輸送コストが小さくなるように G をアップデート
- 5. 2 へ戻る。このとき Y は変化するが f を一から訓練するのではなく、
  少し G が変わったくらいでは最適な f もほぼ同じなので
  **以前のパラメータから訓練をスタートする**（ウォームスタート）

# 深層畳み込みネットワークを使うことでリアルな画像生成

- Wasserstein GAN の論文や後続の論文では G や f として
  深層畳み込みニューラルネットワークを使う

- 上は X を顔写真集合として訓練したてできた生成画像
  非常にリアルな顔写真が生成できている
  上の画像は段階的学習など別のテクニックも駆使して訓練されている

Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR 2018.

# 連続分布の最適輸送は分類問題として解ける

take home message

**連続分布の最適輸送の双対問題は分類問題とみなせる**
**「分類器」f をニューラルネットワークでモデリング**
**WGAN は最適輸送距離が小さくなるようにサンプルを動かす**

# まとめ

# テイクホームメッセージ 1

take home message

**最適輸送は確率分布と確率分布を比較するのに使えるツール**

# テイクホームメッセージ 2

take home message

**最適輸送は KL の欠点を克服できる。**
**手法中に KL が現れたら最適輸送を考えてみましょう。**

# テイクホームメッセージ 3

take home message

**最適輸送距離は線形計画として定式化される**
**ワッサースタイン距離は距離公理を満たす特殊ケース**
**ソルバを利用すると簡単に最適輸送を計算できる**

# テイクホームメッセージ 4

take home message

**エントロピーを導入すると最適化が簡単に**
**シンクホーンは行列演算だけでできるシンプル最適化法**
**簡単にニューラルネットワークに組み込むことができる**

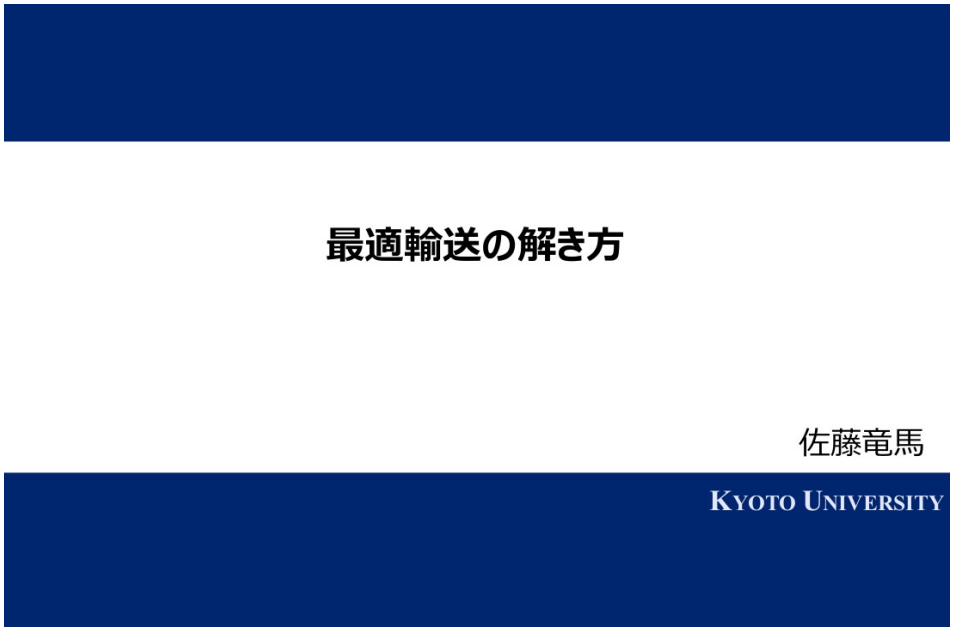
# テイクホームメッセージ 5

take home message

**連続分布の最適輸送の双対問題は分類問題とみなせる**
**「分類器」f をニューラルネットワークでモデリング**
**WGAN は最適輸送距離が小さくなるようにサンプルを動かす**

# アルゴリズムを勉強したい方におすすめの資料

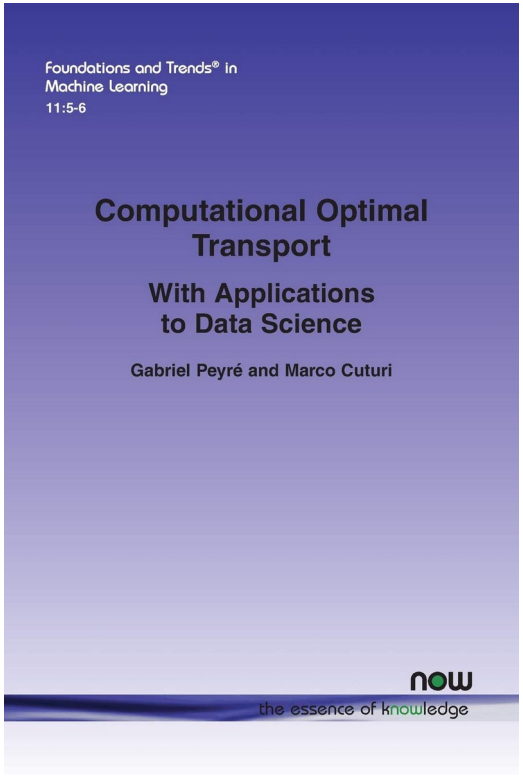
- 最適輸送のアルゴリズム面についてもっと詳しく知りたい方は:
- アルゴリズム面について以前行ったセミナーの資料が利用できるのでぜひご覧ください




- 最適輸送の解き方
  https://www.slideshare.net/joisino/ss-249394573

# さらに勉強したい方におすすめの本

- 最適輸送全般についてもっと詳しく知りたい方は:
- Gabriel Peyré and Macro Cuturi. Computational Optimal Transport with Applications to Data Science.
- arXiv で電子版が公開されています
  https://arxiv.org/abs/1803.00567



# 最適輸送の学習にうってつけの本（宣伝）

- 機械学習プロフェッショナルシリーズより最適輸送本が刊行予定です（来年末ごろ予定）
- 発売された暁にはぜひご覧ください

# 参考文献一覧

# 参考文献一覧


- Jason Altschuler, Jonathan Weed, Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. NeurIPS 2017.
- Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN. ICML 2017.
- Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, Tomaso A. Poggio. Learning with a Wasserstein Loss. NIPS 2015.
- Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, Aaron C. Courville.. Improved Training of Wasserstein GANs. NeurIPS 2017.
- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR 2018.

# 参考文献一覧


- Xiaofeng Liu, Yuzhuo Han, Song Bai, Yi Ge, Tianxing Wang, Xu Han, Site Li, Jane You, Jun Lu. Importance-Aware Semantic Segmentation in Self-Driving with Discrete Wasserstein Training. AAAI 2020.
- Luca Oneto, Michele Donini, Giulia Luise, Carlo Ciliberto, Andreas Maurer, Massimiliano Pontil. Exploiting MMD and Sinkhorn Divergences for Fair and Transferable Representation Learning. NeurIPS 2020.
- Gabriel Peyré. Optimal Transport in Imaging Sciences. 2012.
- https://www.slideshare.net/gpeyre/optimal-transport-in-imaging-sciences
- Gabriel Peyré, Marco Cuturi. Computational Optimal Transport. 2019.
- 佐藤竜馬. 最適輸送の解き方. https://www.slideshare.net/joisino/ss-249394573