

Introduction to Computational Optimal Transport

Josh Nguyen

School of Computing
Australian National University

COMP4680/8650 Advanced Topics in Machine Learning
Semester 2, 2022

“Distance” between Two Probability Distributions

In machine learning, we often need to approximate an underlying distribution by minimising some “distance”.

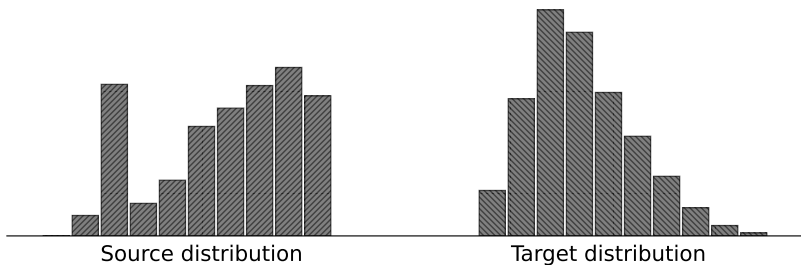
The *Kullback-Leibler divergence* between p and q is defined as

$$\mathbf{KL}(p \parallel q) := \int_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p} \left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})} \right].$$

- ▶ Measures the relative entropy between p and q .
- ▶ Examples: maximum likelihood, variational inference, etc.
- ▶ Always non-negative: $\forall p, q, \mathbf{KL}(p, q) \geq 0$.
- ▶ Zero iff p and q are equal.
- ▶ But not symmetric in general: $\mathbf{KL}(p \parallel q) \neq \mathbf{KL}(q \parallel p)$.
- ▶ Similar to Bregman, KL divergence is *not* a distance metric.

Consider each (discrete) probability distribution as a pile of sand.

- ▶ Similar to a histogram.
- ▶ No matter what pile, its total mass is 1.



The question is, how do we move one pile to another *efficiently*?

- ▶ Move some mass from a bin in p to a bin in q .
- ▶ Cost proportional to amount of mass moved.

Suppose \mathbf{r} and \mathbf{c} are n -dimensional probability vectors. That is, they belong to the $(n - 1)$ -dimensional simplex Δ_{n-1} .

Let $\mathbf{C} \in \mathbb{R}_+^{n \times n}$ be the cost matrix.

► $\mathbf{C}_{i,j}$ = cost of moving a unit of mass from \mathbf{r}_i to \mathbf{c}_j .

Let $\mathbf{X} \in \mathbb{R}^{n \times n}$ be the transport matrix.

► $\mathbf{X}_{i,j}$ = amount of mass transported from \mathbf{r}_i to \mathbf{c}_j .

Therefore, the total cost is

$$\sum_{i=1}^n \sum_{j=1}^n \mathbf{C}_{i,j} \cdot \mathbf{X}_{i,j} = \langle \mathbf{C}, \mathbf{X} \rangle.$$

We will aim to minimise this cost.

It looks like this is a linear program. But do we have any constraints?

Remember $\mathbf{X}_{i,j}$ is the mass transported *from* \mathbf{r}_i *to* \mathbf{c}_j .

- ▶ Must flow in one direction: $\mathbf{X}_{i,j} \geq 0 \quad \forall i, j$.
- ▶ All masses must flow out of \mathbf{r} (row constraints):

$$\sum_{j=1}^n \mathbf{X}_{i,j} = \mathbf{r}_i \quad \forall i \iff \mathbf{X} \mathbf{1} = \mathbf{r}.$$

- ▶ All masses flowing into bin j of \mathbf{c} must equal \mathbf{c}_j (column constraints):

$$\sum_{i=1}^n \mathbf{X}_{i,j} = \mathbf{c}_j \quad \forall j \iff \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

Such an \mathbf{X} satisfying these constraints is called a *coupling* of \mathbf{r} and \mathbf{c} . That is, \mathbf{X} is a joint distribution with marginals \mathbf{r} and \mathbf{c} .

To get the optimal cost, we solve the linear program:

$$\begin{array}{ll} \text{minimise} & \langle \mathbf{C}, \mathbf{X} \rangle \\ \text{subject to} & \mathbf{X} \geq \mathbf{0} \\ & \mathbf{X} \mathbf{1} = \mathbf{r} \\ & \mathbf{X}^\top \mathbf{1} = \mathbf{c}. \end{array}$$

We can extend this problem for (continuous) distributions p and q :

$$\min_{\pi \in \Pi(p, q)} \mathbb{E}_{(\mathbf{x}, \mathbf{x}') \sim \pi} [c(\mathbf{x}, \mathbf{x}')],$$

where π is a coupling of p and q , and $c(\cdot, \cdot)$ is a cost function.

In fact, if c is a distance metric, we have the *p-Wasserstein distance*:

$$W_p(p, q) = \inf_{\pi \in \Pi(p, q)} \left(\mathbb{E}_{(\mathbf{x}, \mathbf{x}') \sim \pi} [c(\mathbf{x}, \mathbf{x}')^p] \right)^{1/p}.$$

We will focus on solving for the optimal transport plan \mathbf{X} in the discrete case.

Word2Vec is a common *word embedding* in natural language processing.

- ▶ Each word in the vocabulary is mapped to a d -dimensional vector.
- ▶ The embedding is trained to accurately predict a word given its context.
 - ▶ Given a context $c = [\text{the, prime, } \square, \text{of, Australia}]$
 - ▶ We should have $p(\square = \text{minister} \mid c) > p(\square = \text{governor} \mid c)$.

The Word2Vec space also encodes interesting *semantic relationships*.

- ▶ $\text{vec}(\text{Canberra}) - \text{vec}(\text{Australia}) \approx \text{vec}(\text{Stockholm}) - \text{vec}(\text{Sweden})$.
- ▶ It's much cheaper to “move” from Canberra to Australia than from Stockholm to Australia.

Idea: To “move” one document to another, we simply need to move their corresponding words. Two questions:

1. How do we represent a document?
2. How do we define the cost between moving from word i to word j ?

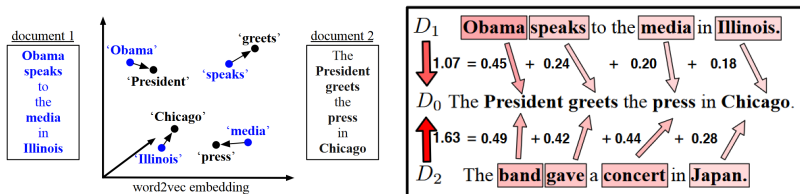
1. Document as a distribution over vocabulary.
 - ▶ Count the occurrences of each word in a document.
 - ▶ Then normalise so the frequencies add up to 1.
 - ▶ Each document is now a histogram. Very sparse: most bins are 0.
2. Moving mass from word i to word j .
 - ▶ Let $\mathbf{v}_i, \mathbf{v}_j$ denote the embeddings of these words.
 - ▶ Then $\mathbf{C}_{i,j} = \|\mathbf{v}_i - \mathbf{v}_j\|_2$. Cost is proportional to distance.

So, given two documents \mathbf{r} and \mathbf{c} , both as probability distributions over the vocabulary, the distance between them is the solution to the same problem:

$$\min_{\mathbf{X}} \langle \mathbf{C}, \mathbf{X} \rangle \text{ s.t. } \mathbf{X} \geq \mathbf{0}, \mathbf{X}\mathbf{1} = \mathbf{r}, \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

The optimal value is called the *word mover's distance*.

Example: Word Mover's Distance 3



(a) Moving words between two documents

(b) Word mover's distance between D_0 and two other sentences D_1, D_2 .

Figure: Visual illustration of the word mover's distance. Source: Kusner et al.

What can we do after calculating document distances?

- ▶ k -nearest neighbour prediction.
- ▶ Agglomerative clustering.
- ▶ etc.

Solving Optimal Transport: Linear Program

Given the linear program above, we can solve it using widely implemented methods like the *simplex* algorithm. Here's an example using *cvxpy*.

```
1 x = cp.Variable(n * n)
2 objective = cp.Minimize(C.flatten() @ x)
3
4 constraints = [x >= 0]
5
6 for i in range(n):
7     constraints.append(cp.sum(x[n * i:n * i + n:]) == r[i])
8
9 for j in range(n):
10    constraints.append(cp.sum(x[j::n]) == c[j])
11
12 prob = cp.Problem(objective=objective, constraints=constraints)
13 prob.solve()
```

However, this is undesirable because of

- ▶ high complexity with respect to the dimensionality;
- ▶ quadratic number of variables.

We will solve problem numerically by designing two techniques, *regularisation* and *duality*.

Define a slightly modified objective function:

$$f(\mathbf{X}) := \langle \mathbf{C}, \mathbf{X} \rangle - \gamma \mathbf{H}(\mathbf{X}) = \langle \mathbf{C}, \mathbf{X} \rangle + \gamma \sum_{i,j} \mathbf{X}_{i,j} \log \mathbf{X}_{i,j}.$$

- ▶ $\mathbf{H}(\mathbf{X}) := -\sum_{i,j} \mathbf{X}_{i,j} \log \mathbf{X}_{i,j}$ is the *discrete entropy* of a coupling matrix.
 - ▶ Forces all $\mathbf{X}_{i,j}$ to be positive.
 - ▶ By convention, $\mathbf{H} = -\infty$ if any component is 0.
- ▶ $\gamma > 0$ is the regularisation strength.

Why add this term?

- ▶ The entropy is 1-strongly concave, which makes f γ -strongly convex w.r.t $\|\cdot\|_1$. Therefore, it accepts a unique minimiser.
- ▶ We can implicitly remove the constraint $\mathbf{X} \geq \mathbf{0}$.
- ▶ (Other motivations are traced back to modelling in transport theory.)

We have the entropic regularised problem as follows:

$$\min_{\mathbf{X}} \{f(\mathbf{X}) := \langle \mathbf{C}, \mathbf{X} \rangle + \gamma \langle \mathbf{X}, \log \mathbf{X} \rangle\} \quad \text{s.t. } \mathbf{X}\mathbf{1} = \mathbf{r} \text{ and } \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

What is the relationship between the original and regularised problems? The following result ([2, Proposition 4.1]) describes it.

Theorem

Let L and L_γ be the optimal objective values for the original linear program and the regularised problem, respectively. Then, as $\gamma \rightarrow 0$, $L_\gamma \rightarrow L$.

In other words, we recover the original solution as regularisation vanishes.

$$\min_{\mathbf{X}} \{f(\mathbf{X}) := \langle \mathbf{C}, \mathbf{X} \rangle + \gamma \langle \mathbf{X}, \log \mathbf{X} \rangle\} \quad \text{s.t.} \quad \mathbf{X}\mathbf{1} = \mathbf{r} \quad \text{and} \quad \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

This problem has

- ▶ n^2 variables: $\mathbf{X}_{i,j}$ ($i, j = 1, \dots, n$).
- ▶ $2n$ equality constraints: n rows, n columns.

Therefore, solving the dual problem might be easier, as it will have $2n$ variables only.

Let $\mathbf{g}, \mathbf{h} \in \mathbb{R}^n$ be the dual variables associated with the two equality constraints. The Lagrangian is

$$\mathcal{L}(\mathbf{X}, \mathbf{g}, \mathbf{h}) = \langle \mathbf{C}, \mathbf{X} \rangle + \gamma \langle \mathbf{X}, \log \mathbf{X} \rangle + \langle \mathbf{g}, \mathbf{X}\mathbf{1} - \mathbf{r} \rangle + \langle \mathbf{h}, \mathbf{X}^\top \mathbf{1} - \mathbf{c} \rangle.$$

At optimality, the gradient of the Lagrangian w.r.t. \mathbf{X} is zero. This helps us establish the relationship between \mathbf{X} and the dual variables \mathbf{g} and \mathbf{h} .

Since $\mathcal{L}(\mathbf{X}, \mathbf{g}, \mathbf{h})$ is jointly strongly convex w.r.t. \mathbf{X} , we solve $\nabla_{\mathbf{X}_{i,j}} \mathcal{L}(\mathbf{X}, \mathbf{g}, \mathbf{h}) = 0$ as follows.

The components of \mathcal{L} containing \mathbf{X}_{ij} are

$$\mathcal{L}_{i,j}(\mathbf{X}, \mathbf{g}, \mathbf{h}) = \mathbf{C}_{i,j} \cdot \mathbf{X}_{i,j} + \gamma \cdot \mathbf{X}_{i,j} \cdot \log \mathbf{X}_{i,j} + \mathbf{g}_i \cdot \mathbf{X}_{i,j} + \mathbf{h}_j \cdot \mathbf{X}_{i,j}.$$

Setting the gradient to zero:

$$\frac{\partial \mathcal{L}_{i,j}(\mathbf{X}, \mathbf{g}, \mathbf{h})}{\partial \mathbf{X}_{i,j}} = \mathbf{C}_{i,j} + \gamma \log \mathbf{X}_{i,j} + \gamma + \mathbf{g}_i + \mathbf{h}_j = 0,$$

which gives

$$\mathbf{X}_{i,j} = \exp \left\{ \frac{1}{\gamma} (-\mathbf{C}_{i,j} - \mathbf{g}_i - \mathbf{h}_j) - 1 \right\}.$$

In vectorised format, at optimality we have:

$$\mathbf{X} = e^{-1} e^{-\mathbf{g}/\gamma} e^{-\mathbf{C}/\gamma} e^{-\mathbf{h}/\gamma}.$$

Note that $e^{-\mathbf{C}/\gamma}$ is constant. The matrix $\mathbf{K} := e^{-\mathbf{C}/\gamma}$ is called the *Gibbs kernel*.

Further, if we let $\mathbf{u} = e^{-\mathbf{g}/\gamma-1/2}$ and $\mathbf{v} = e^{-\mathbf{h}/\gamma-1/2}$, then

$$\mathbf{X} = \text{diag}(\mathbf{u})\mathbf{K}\text{diag}(\mathbf{v}).$$

The solution \mathbf{X} only depends on two (unknown) scaling variables $\mathbf{u}, \mathbf{v} \in \mathbb{R}_{++}^n$.

Therefore, we can recover an n^2 -dimensional solution using only $2n$ variables!

The question now is, how do we find \mathbf{u} and \mathbf{v} ?

Iterative algorithm based on a simple idea.

If \mathbf{X} is the solution, it must satisfy

$$\begin{aligned}\mathbf{X}\mathbf{1} &= \mathbf{r} && \text{(Marginal constraints for rows)} \\ \text{diag}(\mathbf{u})\mathbf{K}\text{diag}(\mathbf{v})\mathbf{1} &= \mathbf{r} && \text{(First-order condition)} \\ \text{diag}(\mathbf{u})(\mathbf{K}\mathbf{v}) &= \mathbf{r} && \text{(Since } \text{diag}(\mathbf{v})\mathbf{1} = \mathbf{v} \text{)} \\ \mathbf{u} \odot (\mathbf{K}\mathbf{v}) &= \mathbf{r}. && \text{(Element-wise multiplication)}\end{aligned}$$

This suggests us that, *if \mathbf{v} is kept fixed*, \mathbf{u} should be updated as

$$\mathbf{u} = \mathbf{r} \oslash (\mathbf{K}\mathbf{v}). \quad \text{(Element-wise division)}$$

Similar for \mathbf{v} .

Sinkhorn: start with positive vectors $\mathbf{u}_0, \mathbf{v}_0$. Alternatingly update \mathbf{u} and \mathbf{v} as

$$\begin{aligned}\mathbf{u}_{k+1} &= \mathbf{r} \oslash (\mathbf{K}\mathbf{v}_k), \\ \mathbf{v}_{k+1} &= \mathbf{c} \oslash (\mathbf{K}^\top \mathbf{u}_{k+1}).\end{aligned}$$

Here's a simple implementation in numpy.

```
1 K = np.exp(-C / gamma)
2 u = np.random.rand(n)
3 v = np.random.rand(n)
4
5 for i in range(num_iters):
6     if i % 2 == 0:
7         u = r / (K @ v)
8     else:
9         v = c / (K.T @ u)
10
11 X = np.diag(u) @ K @ np.diag(v)
```

How should we set `num_iters`? Given a tolerance ϵ :

- ▶ Row constraints violation: $\|\mathbf{X}\mathbf{1} - \mathbf{r}\| < \epsilon$.
- ▶ Column constraints violation: $\|\mathbf{X}^\top \mathbf{1} - \mathbf{c}\| < \epsilon$.
- ▶ A combination of them.

It turns out, these conditions relate to the convergence of $(\mathbf{u}_k, \mathbf{v}_k)$ to $(\mathbf{u}^*, \mathbf{v}^*)$ (details omitted).

Why do the Sinkhorn iterates $(\mathbf{u}_k, \mathbf{v}_k)$ converge to $(\mathbf{u}^*, \mathbf{v}^*)$? A sketch

- ▶ The objective function $f(\mathbf{X})$ can be written as a KL divergence:¹

$$\begin{aligned} f(\mathbf{X}) &= \langle \mathbf{X}, \mathbf{C} + \gamma \log \mathbf{X} \rangle \\ &= \gamma \left\langle \mathbf{X}, -\log \underbrace{\left(e^{-\mathbf{C}/\gamma} \right)}_{=\mathbf{K}} + \log \mathbf{X} \right\rangle \\ &= \gamma \left\langle \mathbf{X}, \log \frac{\mathbf{X}}{\mathbf{K}} \right\rangle \\ &= \gamma \mathbf{KL}(\mathbf{X} \parallel \mathbf{K}). \end{aligned}$$

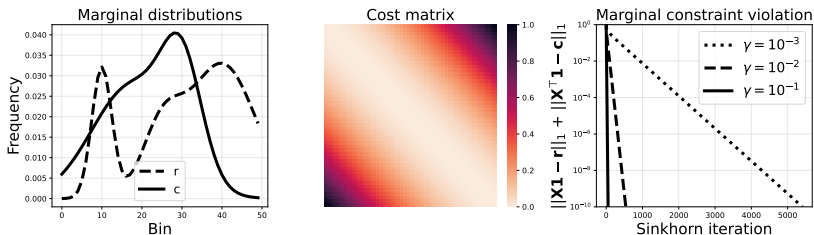
- ▶ The operation $\mathbf{u}_{k+1} = \mathbf{r} \oslash (\mathbf{K} \mathbf{v}_k)$ is equivalent to a *Bregman projection* of \mathbf{X} onto the affine set $\{\mathbf{X}: \mathbf{X} \mathbf{1} = \mathbf{r}\}$. Similar for \mathbf{v}_{k+1} .
- ▶ These iterative projections are known to converge to \mathbf{X} that minimises the KL divergence above.

We omit the convergence rate analysis in this lecture.

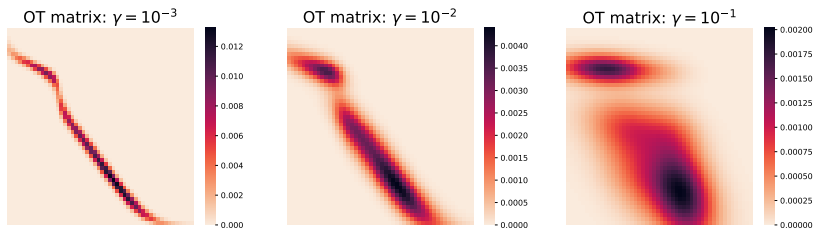
¹KL is also defined for “non-probability” matrix \mathbf{K} . It is still non-negative and is zero iff the two¹⁸ matrices equal.

Solving OT: The Sinkhorn Algorithm 4

Here's an example of running Sinkhorn between two Gaussian mixtures.



Effect of regularisation strength γ on the sparseness of solution.



Many open problems relating to optimal transport:

- ▶ Other regularisers than entropy: ℓ_1 , ℓ_2 , etc.
- ▶ Unbalanced sources: what if \mathbf{r} and \mathbf{c} don't have the same total mass? We could only transport a portion of them.
 - ▶ Unbalanced OT: using more KL regularisers in the objective.
 - ▶ Partial OT: constrain the amount of transported mass to some value.
- ▶ Multi-marginal OT: when you have more than 2 sources of mass.
 - ▶ High-dimensional problem. Requires tensor operations.
- ▶ Any other algorithms than Sinkhorn?
 - ▶ OT is a case of $\min_{\mathbf{x}} f(\mathbf{x})$ s.t. $\mathbf{Ax} = \mathbf{b}$, where f is strongly convex.
 - ▶ Therefore, the dual problem $\min_{\mathbf{y}} g(\mathbf{y})$ has a smooth and differentiable objective.
 - ▶ This makes (accelerated) gradient methods particularly useful.
- ▶ Applications of OT: generative modeling (diffusion models), etc.
- ▶ And many more...

- [1] Stephen Gould, Dylan Campbell, Itzik Ben-Shabat, Chamin Hewa Koneputugodage, and Zhiwei Xu. “Exploiting Problem Structure in Deep Declarative Networks: Two Case Studies”. In: *OT-SDM 2022: The 1st International Workshop on Optimal Transport and Structured Data Modeling* (2022).
- [2] Gabriel Peyré and Marco Cuturi. “Computational Optimal Transport: With Applications to Data Science”. In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607 (cit. on p. 12).
- [3] Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. “Near-Linear Time Approximation Algorithms for Optimal Transport via Sinkhorn Iteration”. In: *Advances in Neural Information Processing Systems*. 2017.
- [4] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. “From Word Embeddings to Document Distances”. In: *International Conference on Machine Learning*. 2015, pp. 957–966 (cit. on p. 9).