Introduction to Computational Optimal Transport

Josh Nguyen

School of Computing
Australian National University

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

In machine learning, we often need to approximate a *true* underlying probability distribution using some *family* of simpler, tractable distributions.

In machine learning, we often need to approximate a *true* underlying probability distribution using some *family* of simpler, tractable distributions.

To do so, we minimise the *Kullback-Leibler divergence* between p and q:

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

In machine learning, we often need to approximate a *true* underlying probability distribution using some *family* of simpler, tractable distributions.

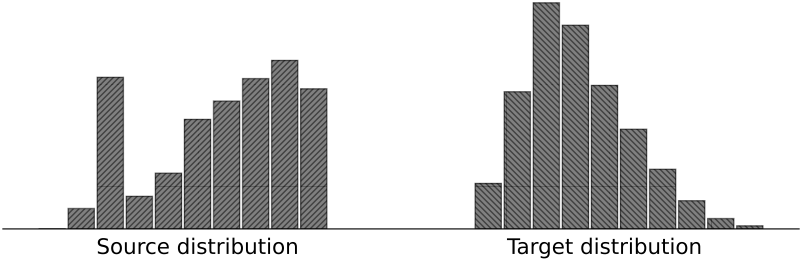
To do so, we minimise the *Kullback-Leibler divergence* between p and q:

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

- Measures the relative entropy between p and q.
- Examples: maximum likelihood, variational inference, etc.
- Always non-negative: $\forall p, q, \ \mathbf{KL}(p \parallel q) \geq 0$.
- 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.

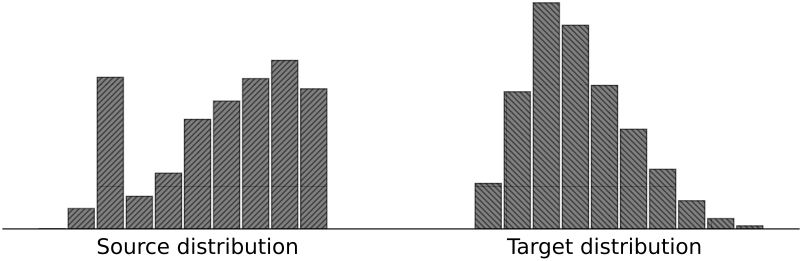
Consider each (discrete) probability distribution as a pile of sand.

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



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}.

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

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 $\mathbf{c}_j$.

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

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

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 $\mathbf{c}_j$.

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

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

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 $\mathbf{c}_j$.

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?

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* $\mathbf{c}_j$.

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: $\mathbf{X}_{i,j} \geq 0 \quad \forall i, j$.

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):

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

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):

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

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:

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

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

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

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 ρ-*Wasserstein distance*:

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

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 $c = [\texttt{the}, \texttt{Prime}, \square, \texttt{of}, \texttt{Australia}]$
  - We should have $p(\square = \texttt{Minister} \mid c) > p(\square = \texttt{Governor} \mid c)$.

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 $c = [\texttt{the}, \texttt{Prime}, \square, \texttt{of}, \texttt{Australia}]$
  - We should have $p(\square = \texttt{Minister} \mid c) > p(\square = \texttt{Governor} \mid c)$.

The Word2Vec space also encodes interesting *semantic relationships*.

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

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 $c = [\texttt{the, Prime, } \square\texttt{, of, Australia}]$
  - We should have $p(\square = \texttt{Minister} \mid c) > p(\square = \texttt{Governor} \mid c)$.

The Word2Vec space also encodes interesting *semantic relationships*.

- $\text{vec}(\texttt{Canberra}) - \text{vec}(\texttt{Australia}) \approx \text{vec}(\texttt{Stockholm}) - \text{vec}(\text{Sweden})$.
- 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.

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 $c = [\texttt{the}, \texttt{Prime}, \square, \texttt{of}, \texttt{Australia}]$
  - We should have $p(\square = \texttt{Minister} \mid c) > p(\square = \texttt{Governor} \mid c)$.

The Word2Vec space also encodes interesting *semantic relationships*.

- $\text{vec}(\texttt{Canberra}) - \text{vec}(\texttt{Australia}) \approx \text{vec}(\texttt{Stockholm}) - \text{vec}(\texttt{Sweden})$.
- 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.

1. How do we represent a document?

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 $c = [\texttt{the}, \texttt{Prime}, \square, \texttt{of}, \texttt{Australia}]$
  - We should have $p(\square = \texttt{Minister} \mid c) > p(\square = \texttt{Governor} \mid c)$.

The Word2Vec space also encodes interesting *semantic relationships*.

- $\text{vec}(\texttt{Canberra}) - \text{vec}(\texttt{Australia}) \approx \text{vec}(\texttt{Stockholm}) - \text{vec}(\texttt{Sweden})$.
- 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.

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

Example: Word Mover's Distance 2

1. Document as a *bag of words*.
   - 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.

1. Document as a *bag of words*.
   - 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.

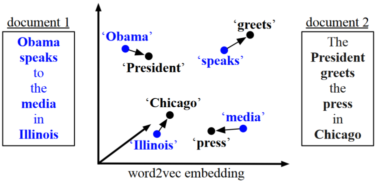
1. Document as a *bag of words*.
   - Count the occurrences of each word in a document.
   - Then normalise so the frequencies add up to 1.
   - Each document is now a histogram. Very sparse: most bins are 0.

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.

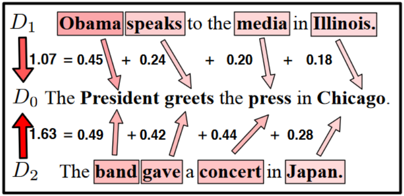
So, given two documents $\mathbf{r}$ and $\mathbf{c}$, both as probability distributions over the vocabulary, the distance between them is the solution to the same problem:

$$\min_{\mathbf{X}} \langle \mathbf{C}, \mathbf{X} \rangle \text{ s.t. } \mathbf{X} \geq \mathbf{0}, \mathbf{X1} = \mathbf{r}, \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

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



(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.

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

```
x = cp.Variable(n * n)
objective = cp.Minimize(C.flatten() @ x)

constraints = [x >= 0]

for i in range(n):
  constraints.append(cp.sum(x[n * i:n * i + n:]) == r[i])

for j in range(n):
  constraints.append(cp.sum(x[j::n]) == c[j])

prob = cp.Problem(objective=objective, constraints=constraints)
prob.solve()
```

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

```
x = cp.Variable(n * n)
objective = cp.Minimize(C.flatten() @ x)

constraints = [x >= 0]

for i in range(n):
  constraints.append(cp.sum(x[n * i:n * i + n:]) == r[i])

for j in range(n):
  constraints.append(cp.sum(x[j::n]) == c[j])

prob = cp.Problem(objective=objective, constraints=constraints)
prob.solve()
```

However, this is undesirable because of

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

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.

```python
x = cp.Variable(n * n)
objective = cp.Minimize(C.flatten() @ x)

constraints = [x >= 0]

for i in range(n):
  constraints.append(cp.sum(x[n * i:n * i + n:]) == r[i])

for j in range(n):
  constraints.append(cp.sum(x[j::n]) == c[j])

prob = cp.Problem(objective=objective, constraints=constraints)
prob.solve()
```

However, this is undesirable because of

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

We will solve the 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.

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:

$$\min_{\mathbf{X}} \ \{f(\mathbf{X}) := \langle \mathbf{C}, \mathbf{X} \rangle + \gamma \langle \mathbf{X}, \log \mathbf{X} \rangle \} \quad \text{s.t.} \quad \mathbf{X1} = \mathbf{r} \quad \text{and} \quad \mathbf{X}^\top \mathbf{1} = \mathbf{c}.$$

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 $\gamma \to 0$, $L_\gamma \to L$.

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

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

This problem has

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

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

This problem has

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

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

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

This problem has

- n^2 variables: $\mathbf{X}_{i,j}$ $(i, j = 1, \ldots, n)$.
- $2n$ 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

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

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

This problem has

- n^2 variables: $\mathbf{X}_{i,j}$ $(i, j = 1, \ldots, n)$.
- $2n$ 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

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

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 $\mathbf{X}_{i,j}$ 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

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

We can rewrite $\mathbf{X}$ as:

$$\mathbf{X}_{i,j} = \exp\left\{-\frac{\mathbf{g}_i}{\gamma} - \frac{1}{2}\right\} \cdot \exp\left\{-\frac{\mathbf{C}_{i,j}}{\gamma}\right\} \cdot \exp\left\{-\frac{\mathbf{h}_j}{\gamma} - \frac{1}{2}\right\}.$$

Note that $\exp\left\{-\frac{\mathbf{C}_{i,j}}{\gamma}\right\}$ is constant. The matrix $\mathbf{K} = \exp\left\{-\frac{\mathbf{C}}{\gamma}\right\}$ is called the *Gibbs kernel*.

Further, if we let $\mathbf{u} = \exp\left\{-\frac{\mathbf{g}}{\gamma} - \frac{1}{2}\right\}$ and $\mathbf{v} = \exp\left\{-\frac{\mathbf{h}}{\gamma} - \frac{1}{2}\right\}$, then

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

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!

We can rewrite $\mathbf{X}$ as:

$$\mathbf{X}_{i,j} = \exp\left\{-\frac{\mathbf{g}_i}{\gamma} - \frac{1}{2}\right\} \cdot \exp\left\{-\frac{\mathbf{C}_{i,j}}{\gamma}\right\} \cdot \exp\left\{-\frac{\mathbf{h}_j}{\gamma} - \frac{1}{2}\right\}.$$

Note that $\exp\left\{-\frac{\mathbf{C}_{i,j}}{\gamma}\right\}$ is constant. The matrix $\mathbf{K} = \exp\left\{-\frac{\mathbf{C}}{\gamma}\right\}$ is called the *Gibbs kernel*.

Further, if we let $\mathbf{u} = \exp\left\{-\frac{\mathbf{g}}{\gamma} - \frac{1}{2}\right\}$ and $\mathbf{v} = \exp\left\{-\frac{\mathbf{h}}{\gamma} - \frac{1}{2}\right\}$, then

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

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

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

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

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

Similar for $\mathbf{v}$.

Iterative algorithm based on a simple idea.

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

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

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

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

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`.

```
K = np.exp(-C / gamma)
u = np.random.rand(n)
v = np.random.rand(n)

for i in range(num_iters):
  if i % 2 == 0:
      u = r / (K @ v)
  else:
      v = c / (K.T @ u)

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: $\left\|\mathbf{X}^\top\mathbf{1} - \mathbf{c}\right\| < \epsilon$.
- 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]

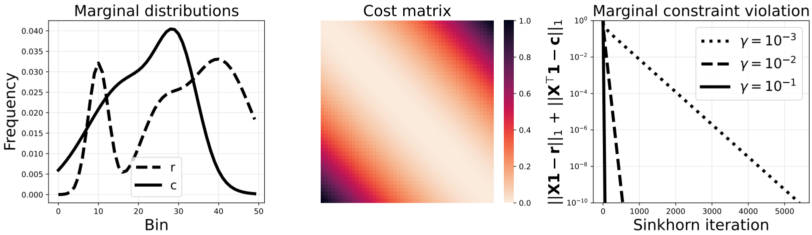
$$\begin{aligned} f(\mathbf{X}) &= \langle \mathbf{X}, \mathbf{C} + \gamma \log \mathbf{X} \rangle \\ &= \gamma \left\langle \mathbf{X}, -\log \underbrace{\left(e^{-\mathbf{C}/\gamma}\right)}_{=\mathbf{K}} + \log \mathbf{X}) \right\rangle \\ &= \gamma \left\langle \mathbf{X}, \log \frac{\mathbf{X}}{\mathbf{K}}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

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

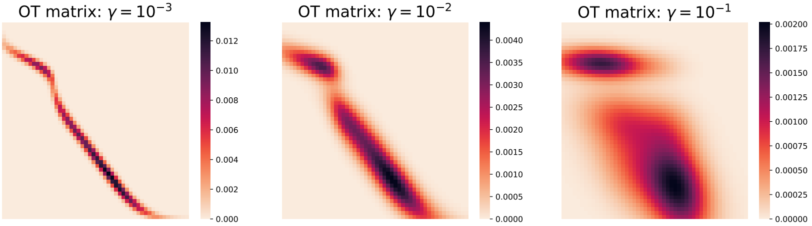
We omit the convergence rate analysis in this lecture.

[1] KL is also defined for "non-probability" matrix $\mathbf{K}$. It is still non-negative and is zero iff the two matrices equal.

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, ℓ_∞, 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 modelling (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) (cit. on p. 49).

[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. 33).

[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. 27).

$$\mathbf{x} \longrightarrow \boxed{f(\cdot;\boldsymbol{\theta})} \longrightarrow \mathbf{M} \longrightarrow \boxed{\texttt{Sinkhorn}} \longrightarrow \mathbf{P} \longrightarrow \boxed{J(\cdot)}$$

Problem: $\min_{\boldsymbol{\theta}} J$, where, $\mathbf{P}$ is the solution to the entropic regularised OT.

Our goal is to find $\mathrm{D}J(\boldsymbol{\theta}) = \mathrm{D}J(\mathbf{P}) \cdot \mathrm{D}\texttt{Sinkhorn}(\mathbf{M}) \cdot \mathrm{D}f(\mathbf{M};\boldsymbol{\theta})$.

- $\mathrm{D}J(\mathbf{P})$ and $\mathrm{D}f(\mathbf{M};\boldsymbol{\theta})$ can be found normally as in a imperative node.
- However, $\mathrm{D}\texttt{Sinkhorn}(\mathbf{M})$ requires unrolling all Sinkhorn steps, which is a huge memory requirement.

This fits directly into the deep declarative network setting

$$\begin{array}{rl} \text{minimise} & J(\mathbf{y}(\mathbf{x})) \\ \text{where} & \mathbf{y}(\mathbf{x}) \in \arg\min_{\mathbf{u}} f(\mathbf{x},\mathbf{u}) \text{ s.t. } h_i(\mathbf{x},\mathbf{u}) = 0, i = 1,\ldots,p, \end{array}$$

where $\mathbf{y}$ acts as $\mathbf{P}$ and $\mathbf{x}$ as $\mathbf{M}$.

We learned that the expression for $D\mathbf{y}(\mathbf{x})$ is

$$\mathbf{H}^{-1}\mathbf{A}^{\top}(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^{\top})^{-1}(\mathbf{A}\mathbf{H}^{-1}\mathbf{B}-\mathbf{C})-\mathbf{H}^{-1}\mathbf{B}.$$

The most expensive operation is computing $(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^{\top})^{-1}$. Luckily, this is a *block* matrix, containing only non-zero blocks along the diagonal. Therefore, we can invert it more efficiently.

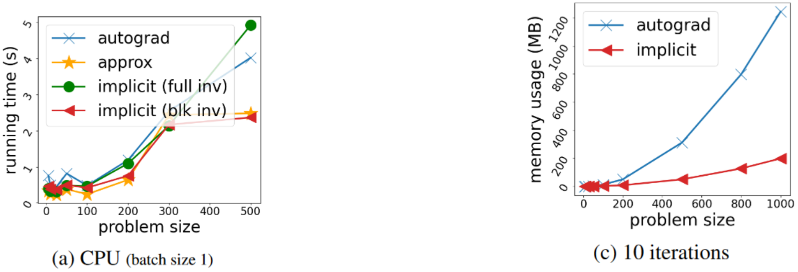


Figure: Time and memory comparison for autograd and implicit differentiation. Source: Gould et al.