Learning Portfolio: Deeper dive into optimal transport

Lily Lin

A quick google search indicates that optimal transport is more than just a linear programming problem about moving resources, and has many far reaching applications

https://kantorovich.org/post/ot_intro/

```
This basic problem has a wealth of applications within mathematics (in the theory of partial differential
```

```
The Kanotorovich Initiative is dedicated towards research and dissemination of modern OT mathematics towar
```

What is so special about optimal transport?

Questions - Why is this specific case of network flows so valued for study? - What makes optimal transport so widely applicable? - What are some unique problems that can be solved by optimal transport? We saw image recognition in class but what else can be done? - What methods are there to solve optimal transport besides linear programming?

What makes optimal transport special?

The key insight that seems to make OT special is the idea of moving probability distributions

1. Many many many things can be modelled as a probability distribution over a space
2. OT can be used to solve how one probability distribution can optimally be "transformed" into another space

The way OT provides it's solution gives a number of other applications as well

3. It can say how "hard" it is to move one distribution to another, giving the ability to compare the "similarity" of probability distributions
4. It gives a continuous set of "inbetween" distributions, essentially interpolating the two distributions by moving the points in their optimal direction but not all the way

So from what I've read, it's these 4 benefits of OT that make it such a fruitful area of research. Point 1 makes it extremely flexible and applicable. Point 3 and 4 allow you to mathematically compute interesting things about probability distributions that you can't (or can you, I don't know) by other methods

There's also an additional point that I think applies

5. It can be formulated as a linear programming problem, which has many highly optimized solvers like `linprog` or `pulp`

What are some stuff optimal transport can do?

The image recognition project is a clear example of using that 3rd property, we can compare a given image (a distribution) with a bunch of known images (also distributions) and determine which one is the most similar, giving us some rudimentary image recognition

This blog post has many interesting examples

https://towardsdatascience.com/optimal-transport-a-hidden-gem-that-empowers-todays-machine-learning-2609bbf67e59

Looking at the color transfer example, what I think is happening is - Pixels are mapped into points on the RGB R^3 space - The sunset image pixels are mapped to the distribution of the daytime image - So black-ish sunset colors are mapped to black-ish daytime colors - The interesting part is that light-ish sunset pixels are also mapped to

light-ish daytime pixels, and everything inbetwee as well. So the distribution of reddish pixels are mapped onto the daytime distribution of blue-ish pixels. - The bright parts of the image stay "bright", but they're now "bright" in the daytime image's distribution of colors. Really cool stuff - The core idea is essentially still that idea of mapping one probability distribution to another, but seeing this physical example is really cool

https://arxiv.org/pdf/1809.00013.pdf

I tried to learn a bit about this method for automated translation using OT... it's complicated

The basic idea (I think) is: - We have a distribution for language A - We have a distribution for language B - We want to use OT to make points of one language to another, but a concept of "distance" between the two distributions is complicated. What's the distance between "cat" and "katze"? They're just two points on two different distributions of words for their respective languages - The authors instead opt to use a Gromov-Wasserstein distance - This metric focuses on trying to map out the overall structure of pairs in the different distributions, rather than comparing one point in distribution A and one point in distribution B - How does it work? It measures how much it would make sense for a pair of words in language A to be mapped to a pair of words in language B by defining a distance between distances. If the distance between words a_1 and a_2 in language A is similar to that of b_1 and b_2 in language B, then it makes sense to pair a_1 to b_1 and a_2 to b_2. - The authors use OT to optimize for this pairing of words across languages, giving a mapping of words between languages, which can be used for things like translation - One major benefit of this method is that we never need to have any knowledge of how language A relates to language B, we only need to know the pairwise relationships of words *within* each language, making this method extensible to other languages quickly (I think?) - We also only need to learn the relationships between the points in the distribution for language A, not the *actual positions* of the distriubtion for language A - We only need to model how similar "dog" and "cat" and "rainbow" are, not place them exactly onto a R^d distribution - This is also a problem that machine learning is good at (I think? Oh yea it says most modern word embedding is done by machine learning? `word2vec`?) - Anyway that was a fun little hour long adventure into an application of optimal transport

https://www.jmlr.org/papers/volume22/20-451/20-451.pdf

```
Examples where this approach is useful in machine learning (ML)
are ubiquitous and include, for instance, prominent tasks such as training generative adversarial networks
the vanishing gradient problem, and domain adaptation (Courty et al., 2016) where its capacity to align tw
```

```
Other examples showing the versatility of OT are given by the averaging of population
data in neuroscience (Gramfort et al., 2015), shape reconstruction (Bonneel et al., 2016)
and learning word embeddings in natural language processing (Kusner et al., 2015).
```

I'm not exactly sure what these mean but the key point seems to be

```
Optimal Transport (OT) is a field of mathematics which studies the geometry of probability spaces. Among i
considered metric space.
```

Compare -> determine cost of moving one distribution to another Align -> Smoothly interpolate from one distribution to another

"Taking into account the underlying geometry of the metric space" is an interesting point though. Since we can define these distributions over really whatever space we want, we can also define our own ideas of distance (ie defining distance between images by rgb value instead of something else)

Anyway what I wanted to learn more about was algorithms for solving OT problems. In class we reduced them to LP problems but what about specialized OT solvers?

```
The library provides recent state-of-the-art solvers for various optimal transport problems
related to statistics and machine learning. As mentioned earlier, POT has sub-modules
dedicated to different problems. In this section, we give a short description of the provided
OT solvers.
```

Ah, as expected the library defines a bunch of specialized solvers for OT problems, going one step further by having specizlied solvers for specific *kinds* of OT problems

```
Documentation. POT provides detailed documentation2
for each function and class.
```

`For each function, the documentation includes the mathematical problem that the function solves, parameter descriptions, returned elements, examples, and bibliographical references. All the papers related to the available solvers are listed in the documentation.`

Wow that's rare

Let's look at that documentation

https://pythonot.github.io/quickstart.html#optimal-transport-and-wasserstein-distance

`The method implemented for solving the OT problem is the network simplex.`

Common OT is implemented using the simplex method, oh but wait no it says "network simplex"

https://en.wikipedia.org/wiki/Network_simplex_algorithm

`In mathematical optimization, the network simplex algorithm is a graph theoretic specialization of the sim`

Woah.

Reading the article the method does something like this (I think) 1. Start with a solution, which in this case is a spanning tree of the graph. I guess because it's a spanning tree it's equivalent solution in the normal LP problem is on a vertex of the polytope? 2. Attempt to pivot to a better solution by forming a cycle in the tree (entering variable), and then removing the "weakest link" (the one with the highest cost, leaving variable). This results in a new spanning tree (new solution on a vertex), with a higher value. 3. Continue to pivot from vertex to vertex until the solution can't find a new entering variable (new edge to introduce that would increase the objective function)

Huh that's cool, I wonder what exactly makes this faster than simplex. I guess invertiing the matricies is harder than solving the inherently more "local" problem of considering only the cycle that gets added onto the spanning tree? The matrix always has to contain all the variables since entering and leaving variables affect all the other values, and the spanning tree edge in/edge out only affects the local area that becomes a cycle.

Anyway that's enough OT for me I think, learned some cool stuff though.