

# Dorian Goldman Research Summary

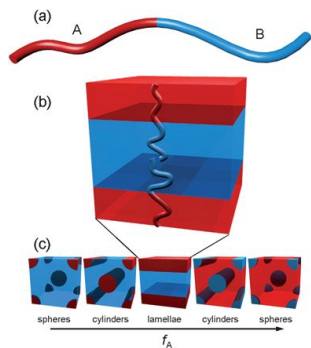
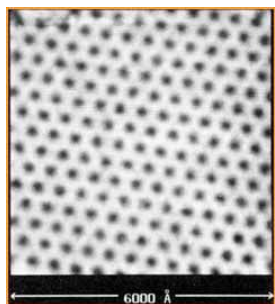
---

Research summary and links to recent advances and new directions

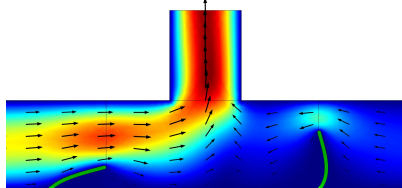
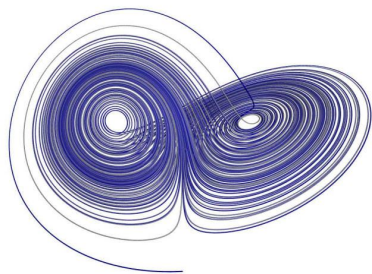
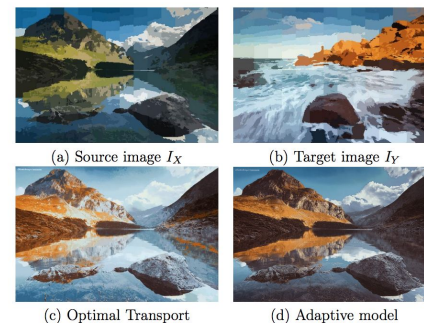
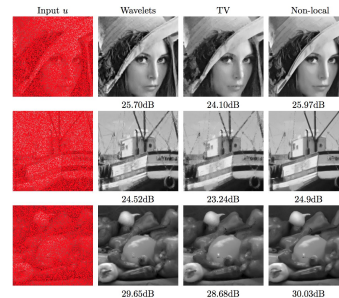
Dorian Goldman, Lyft

# Methods from physics are being used in AI

## Physics



## Artificial Intelligence



# Outline

- Energy Driven Pattern Formation and non-local isoperimetric problems.
  - Previous work on a non-local Ginzburg-Landau energy
  - Applications AI: Image segmentation, reconstruction and noise removal.
- Optimal Transportation and Dynamical Systems.
  - Relevance in AI: Transfer learning, image segmentation, color transfer.
  - Previous work related to chaotic dynamics in weather models using OT.
- Reinforcement learning and offline policy selection.
  - Effective email campaigns via offline policy discovery.
  - Python code and simulations.

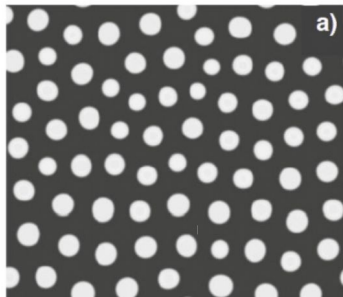
# Energy Driven Pattern Formation

---

Asymptotics of two-phase energies



# Modeling phase transitions

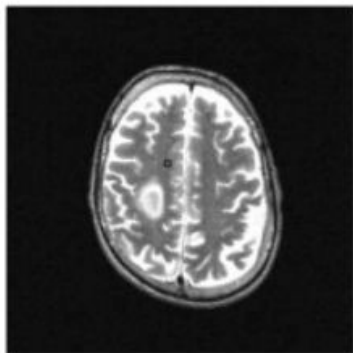


$$\mathcal{E}^\epsilon(u) := \int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2$$

$\epsilon$  • The scale of the phase transition.

$u \in H^1(\Omega)$  • Represents the two phases of the material or image.

- Either a mass constraint or boundary conditions are imposed.



# How do we understand it?

We can compute the **Euler Lagrange equation**

$$\frac{d}{ds} \Big|_{s=0} \mathcal{E}^\epsilon(u^\epsilon + vs) = 0$$

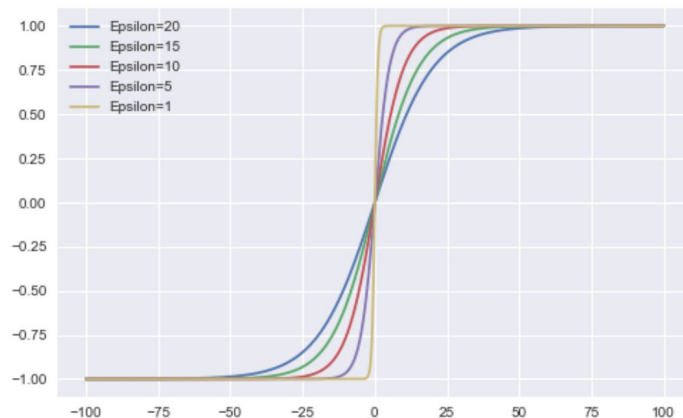
→ 
$$-\epsilon^2 \Delta u = u(u^2 - 1)$$

In one dimension:

$$u(x) \sim \tanh \left( \frac{x - x_0}{\epsilon} \right)$$

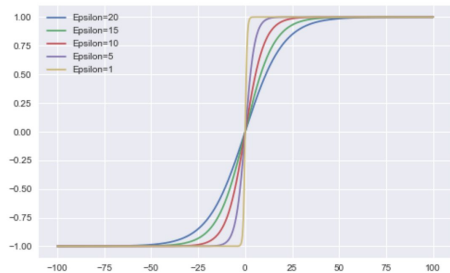
- Thus we have a phase transition of order epsilon which occurs between the two states.
- The size of the edges shrinks as epsilon shrinks.
- Can we understand this asymptotically?

```
In [29]: eps = [20,15,10,5,1]
x0=0
omega = np.linspace(-100,100,1000)
plt.figure(figsize=(8,5))
for e in eps:
    u = [math.tanh((x-x0)/e) for x in omega ]
    plt.plot(omega,u,label="Epsilon="+str(e))
plt.legend()
```



# Understanding limiting behavior

```
In [29]: eps = [20,15,10,5,1]
x0=0
omega = np.linspace(-100,100,1000)
plt.figure(figsize=(8,5))
for e in eps:
    u = [math.tanh((x-x0)/e) for x in omega ]
    plt.plot(omega,u,label="Epsilon="+str(e))
plt.legend()
```



$$\mathcal{E}^\epsilon(u) := \int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2$$

$$-\epsilon^2 \Delta u = u(u^2 - 1)$$

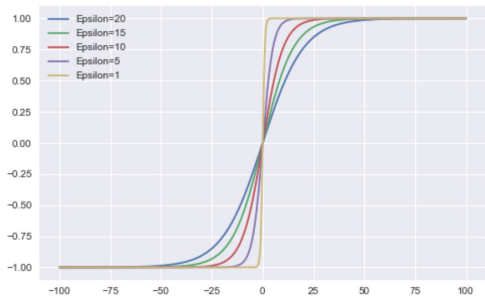
Can we understand the limiting behavior by a simpler asymptotic functional?

$$\mathcal{E}^\epsilon(u^\epsilon) = F_0(u) + o_\epsilon(1)$$

$u \in BV(\{+1, -1\})$  Sets of finite perimeter.

# An optimal lower bound

```
In [29]: eps = [20,15,10,5,1]
x0=0
omega = np.linspace(-100,100,1000)
plt.figure(figsize=(8,5))
for e in eps:
    u = [math.tanh((x-x0)/e) for x in omega ]
    plt.plot(omega,u,label="Epsilon="+str(e))
plt.legend()
```



$$\mathcal{E}^\epsilon(u) := \int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2$$

$$\int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2 \geq \int_{\Omega} \frac{\epsilon}{\sqrt{2}} \int |\nabla u| |u^2 - 1|$$

**Assume:**  $\limsup_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \mathcal{E}^\epsilon(u^\epsilon) < +\infty$

**Then:**

$$u^\epsilon \rightarrow u \in \{-1, 1\}$$

# An optimal lower bound

[Modica and S. Mortola., 1977]

$$\int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2 \geq \int_{\Omega} \frac{\epsilon}{\sqrt{2}} \int |\nabla u| |u^2 - 1| \quad \text{Cauchy Schwarz}$$

$$\int_{\Omega} |\nabla u| |u^2 - 1| = \int_{-\infty}^{+\infty} \int_{u^{-1}(s)} |s^2 - 1| d\mathcal{H}^{n-1}(s) ds \quad \text{Coarea formula}$$

$|\nabla u| dx dy \sim \frac{du}{d\hat{n}} d\hat{n} dS$

$$\int_{-\infty}^{+\infty} \int_{u^{-1}(s)} |s^2 - 1| d\mathcal{H}^{n-1}(s) ds = \int_{-\infty}^{+\infty} |s^2 - 1| \int_{u^{-1}(s)} d\mathcal{H}^{n-1}(s) ds$$

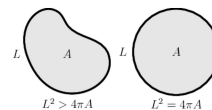
$$u^\epsilon \rightarrow u \in \{-1, 1\} \longrightarrow \sim \text{Per}(u(x) = 1) \int_{-1}^1 |s^2 - 1| ds = \frac{4}{3} \text{Per}(u(x) = 1)$$

Surface Energy

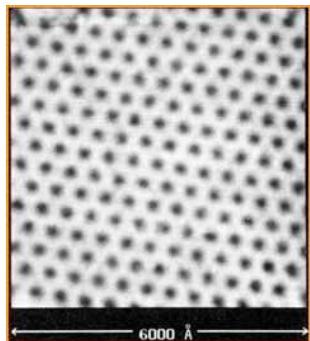
# The local term - Modica Mortola

$$\int_{u^{-1}(s)} \int_{-\infty}^{+\infty} |s^2 - 1| ds d\mathcal{H}^{n-1}(s) = \frac{4}{3} \int_{u^{-1}} d\mathcal{H}^{n-1}(s) = \boxed{\frac{4}{3} \int_{\Omega} |\nabla u|}$$

Minimizer is the ball

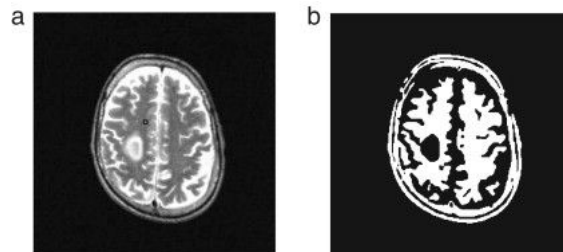


**Physics**



But is this lower bound too low?

**Machine Learning**



**Goal:** We can understand pattern formation of the physical system by the geometry of this functional?

**Goal:** Can we remove small oscillations, segment images or classify them?

# Gamma Convergence

$$\mathcal{E}^\epsilon \xrightarrow{\Gamma} \frac{4}{3} \int |\nabla u|$$

## Lower Bound

For every  $u^\epsilon$  there is a  $u$  such that

$$u^\epsilon \rightarrow u \in \{-1, 1\}$$

$$\liminf_{\epsilon \rightarrow 0} \mathcal{E}^\epsilon(u^\epsilon) \geq \frac{4}{3} \int_{\Omega} |\nabla u|$$

## Upper Bound

For each  $u \in BV(\{+1, -1\})$  there exists a *recovery sequence*,  
ie.

$$\exists u^\epsilon \in H^1(\Omega)$$

$$\text{s.t. } u^\epsilon \rightarrow u \text{ and } \lim_{\epsilon \rightarrow 0} \mathcal{E}^\epsilon(u^\epsilon) = \frac{4}{3} \int_{\Omega} |\nabla u|$$

[E. De Giorgi and T. Franzioni, 1975]

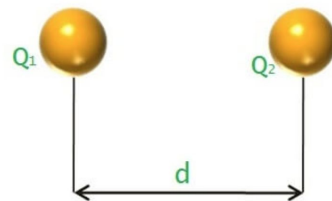
**Almost minimizers  
converge to minimizers:**

$$\operatorname{argmin}_u \mathcal{E}^\epsilon + o_\epsilon(1) \rightarrow \operatorname{argmin}_u \frac{4}{3} \operatorname{Per}(u)$$

- **Perimeter minimizing is not convex.** But we are now have a way of finding a minimum!

# Adding Coulombic Repulsion



$$\mathcal{E}^\epsilon(u) := \int_{\Omega} \epsilon^2 |\nabla u|^2 + \frac{1}{2} (u^2 - 1)^2 + \|u - \bar{u}\|_{H^{-1}(\Omega)}^2$$



- Competing non-local term added to surface tension (ie. charged phases).
- Now there is a complex interaction between wanting to minimize surface area and separating the phases. Two **phases want to separate**.
- Do we still obtain minimal surfaces? How do they separate?



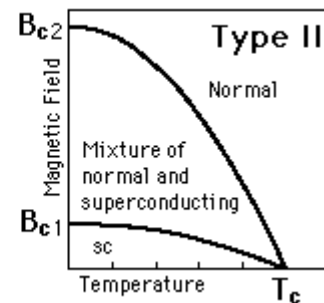
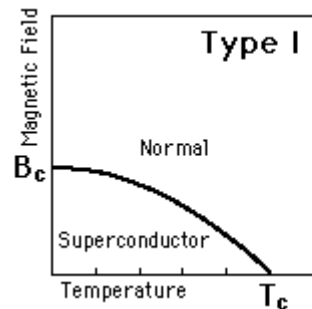
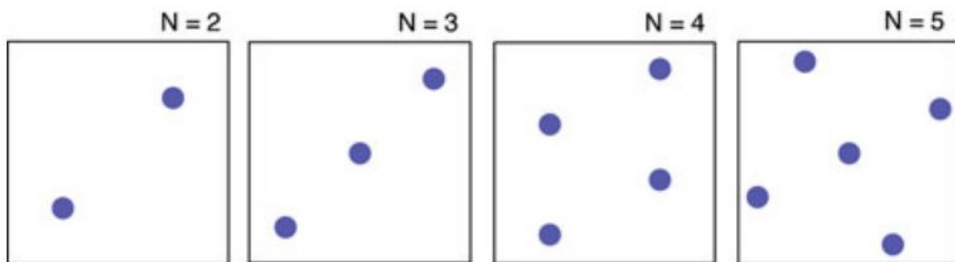
# Context

- We study a regime where one phase is very small compared to the other one
- Pure phase  $u \equiv +1$  
- Pure phase  $u \equiv -1$  
- What happens in between? Let's start from  $u \equiv -1$  and increase second phase slightly.



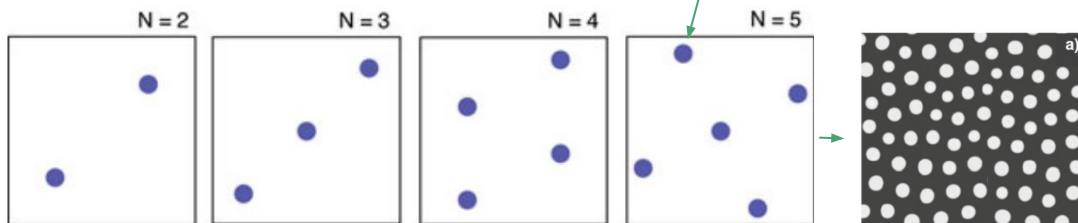
$$\Omega = \mathbb{T}^2$$

$$\|u - \bar{u}\|_{H^{-1}} = \int_{\mathbb{T}^2 \times \mathbb{T}^2} G(|x - y|)(u(x) - \bar{u})(u(y) - \bar{u}) dx dy$$



# Re-scaling “droplet density”

$$\frac{1}{2} \epsilon^{-\frac{2}{3}} |\log \epsilon|^{-\frac{1}{3}} (1 + u^\epsilon) = \frac{1}{N} \sum_{i=1}^N A_i \delta_{x_i}$$



## Physical/Numerical observations

- The droplets seem to be uniformly distributed.
- They all seem to have the same size and be spheres.
- There seems to be a lattice structure forming.

$$\bar{u}^\epsilon = -1 + \epsilon^{2/3} |\log \epsilon|^{1/3} \bar{\delta}$$

# Main Result I: First order limit

$$\epsilon^{-4/3} |\log \epsilon|^{-1/3} E^\epsilon \xrightarrow{\Gamma} E^0(\mu) := \frac{\bar{\delta}^2}{2\kappa^2} + \left( 3^{2/3} - \frac{2\bar{\delta}}{\kappa^2} \right) \int d\mu + \iint_{\mathbb{T}^2 \times \mathbb{T}^2} G(x-y) d\mu(x) d\mu(y),$$



$$\min_{\mu} E^0 = \bar{\mu}$$

Constant density so  
minimizers are  
equidistributed!

[Goldman et. al, 2012]

$$\frac{1}{N} \sum_{i=1}^N A_i \delta_{x_i} \xrightarrow{*} \mu \text{ in } C^*(\mathbb{T}^2)$$

$$\limsup_{\epsilon \rightarrow 0} \frac{1}{|\log \epsilon|} \sum_{i \in I_\gamma} |A_i - \pi 3^{2/3}|^2 = 0 \quad \limsup_{\epsilon \rightarrow 0} \frac{1}{|\log \epsilon|} \sum_i P_i - \sqrt{4\pi A_i} = 0$$

$$-\Delta G + \kappa^2 G = \mu$$

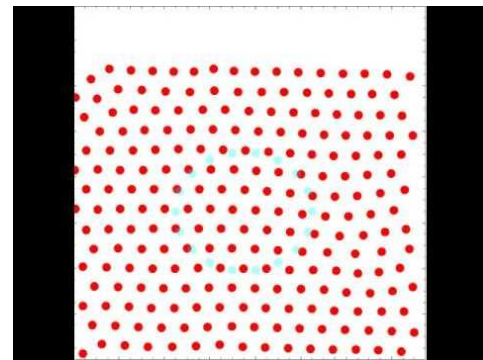
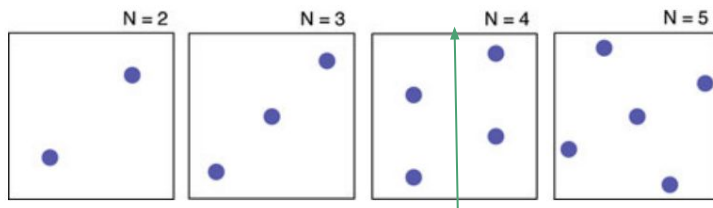
$$\bar{v}^\epsilon = -1 + \epsilon^{2/3} |\log \epsilon|^{1/3} \bar{\delta}$$

**Conclusion:** All droplets are **approximately round** and have the **same volume**.  
Moreover the energy has a **constant density minimizer** - equal distribution!

# Main Result II: Second order correction

$$\epsilon^{-4/3} |\ln \epsilon|^{-2/3} \min \mathcal{E}^\epsilon = E^0(\bar{\mu}) + |\ln \epsilon|^{-1} \left( 3^{4/3} \min W + \frac{3}{4} \bar{\mu} + o_\epsilon(1) \right)$$

Derive a second order correction which governs the location of each droplet

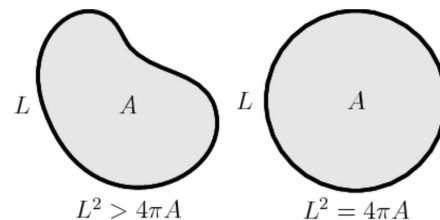


$$W(x_1, \dots, x_n) = \sum_{i=1}^N \sum_{j=1}^N G(x_i - x_j)$$

- Prove that over all possible lattice structures, the Abrikosov lattice has minimal energy **[Serfaty, 2010]**
  - This is the first rigorous result in the direction proving this observation.

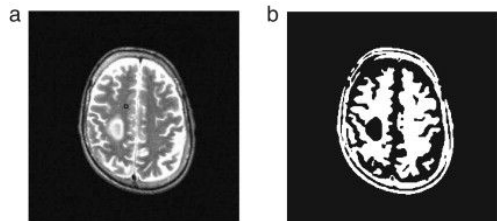
# General Critical Points

- [Goldman et. al, 2013] We show that in all dimensions, non-minimizing critical points converge to the uniform distribution of droplets.
- [Goldman et. al, 2013] We provide the first rigorous proof that non-minimizing critical points have an asymptotically smooth boundary up to a small set.
- [Goldman et al, 2012] Any connected planar set converges exponentially fast to the circle when the non-local term is sufficiently small.
- This generalizes the well known theorem of Gage [Gage, 1983] to the non-local energy.

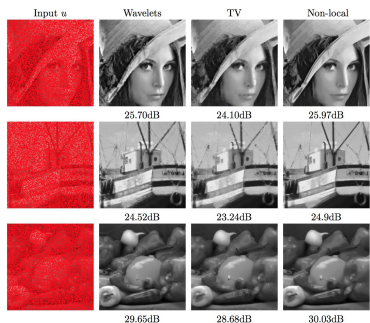


$$F(u) = \int_{\Omega} |\nabla u| + \nu \int_{\Omega \times \Omega} G(|x - y|) dx dy$$

# Applications to AI



MRI Segmentation using Gamma Convergence [Yoon Mo Jung, 2007]



- **Perimeter is not strictly convex**, so we can't guarantee we find a global minimum.
- The **diffuse energy IS** however and we now know that minimizers **converge in a stable way!**

## Applications to image processing include:

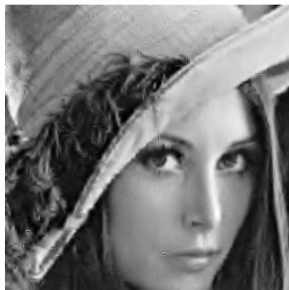
- Denoising, Segmentation, Reconstruction.
- Basic idea is to use the knowledge of Gamma convergence to find a minimizer of a surface energy which filters out some of the oscillations (but not all).
- [Zhang, 2009] uses the non-local term as a form of regularization (ie. it favors small oscillations).
- [Peyré, 2011] Image restoration using non-local sharp interface version of energy.

$$J_\epsilon(f) = \int_{\Omega} \left[ \epsilon^2 |\nabla f|^2 + \frac{1}{\epsilon} W(f) \right] dx + \int_{\Omega} (f(x) - I(x))^2 dx$$

Input  $u$

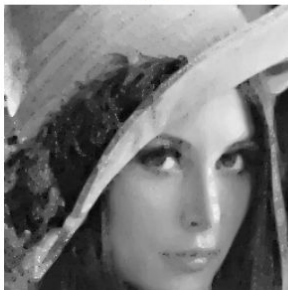


Wavelets



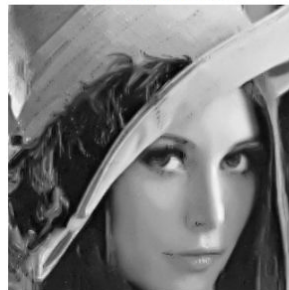
25.70dB

TV



24.10dB

Non-local



25.97dB



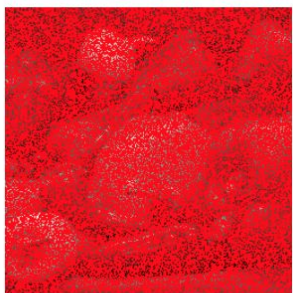
24.52dB



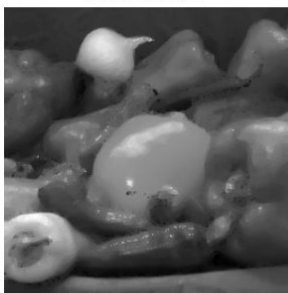
23.24dB



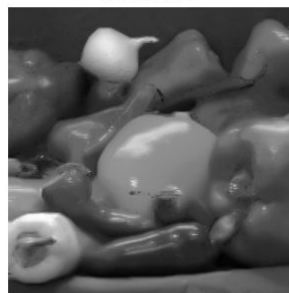
24.9dB



29.65dB



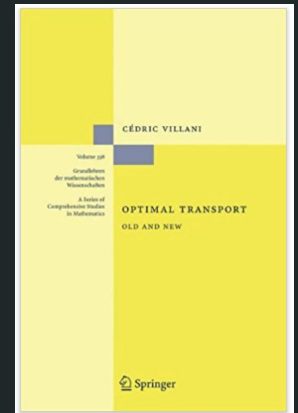
28.68dB



30.03dB

# Optimal Transportation and Dynamical Systems

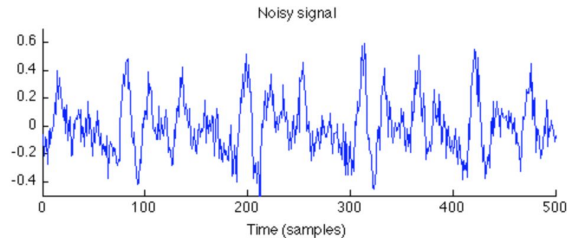
Connections to domain transfer, image Processing and results in fluid mechanics.





# Motivation

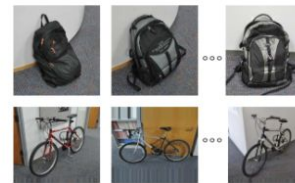
- In machine learning, we usually model data as a **joint probability distribution**. In many cases we need to adapt our models to account for changes such as **lighting, noise, color**, etc.
- Optimal Transport is the perfect tool for **comparing empirical distributions**.
- Image segmentation.
- Image classification and object detection.
- Color transfer
- Sound processing.



Amazon



DLIR



# Some examples

---

## Learning with a Wasserstein Loss

---

Charlie Frogner\* Chiyuan Zhang\*  
Center for Brains, Minds and Machines  
Massachusetts Institute of Technology  
frogner@mit.edu, chiyuan@mit.edu

Hossein Mobahi  
CSAIL  
Massachusetts Institute of Technology  
hmobahi@csail.mit.edu

Mauricio Araya-Polo  
Shell International E & P, Inc.  
Mauricio.Araya@shell.com

Tomaso Poggio  
Center for Brains, Minds and Machines  
Massachusetts Institute of Technology  
tp@ai.mit.edu

2015

## Optimal Transport for Domain Adaptation

Nicolas Courty, Rémi Flamary, Devis Tuia, *Senior Member, IEEE*,  
Alain Rakotomamonjy, *Member, IEEE*

2016

---

## Optimal Transport for Deep Joint Transfer Learning

---

Ying Lu Liming Chen Alexandre Saidi  
Ecole Centrale de Lyon, France  
{ying.lu, liming.chen, alexandre.saidi}@ec-lyon.fr

2017

### Wasserstein Dictionary Learning: Optimal Transport-based unsupervised non-linear dictionary learning

Morgan A. Schmitz\*, Matthieu Heitz†, Nicolas Bonneel†, Fred Ngolé‡, David Coeurjolly†,  
Marco Cuturi§, Gabriel Peyré¶, and Jean-Luc Starck\*

---

**Abstract.** This article introduces a new non-linear dictionary learning method for histograms in the probability simplex. The method leverages optimal transport theory, in the sense that our aim is to reconstruct histograms using so-called displacement interpolations (a.k.a. Wasserstein barycenters) between dictionary atoms; such atoms are themselves synthetic histograms in the probability simplex. Our method simultaneously estimates such atoms, and, for each datapoint, the vector of weights that can optimally reconstruct it as an optimal transport barycenter of such atoms. Our method is computationally tractable thanks to the addition of an entropic regularization to the usual optimal transportation problem, leading to an approximation scheme that is efficient, parallel and simple to differentiate. Both atoms and weights are learned using a gradient-based descent method. Gradients are obtained by automatic differentiation of the generalized Sinkhorn iterations that yield barycenters with entropic smoothing. Because of its formulation relying on Wasserstein barycenters instead of the usual matrix product between dictionary and codes, our method allows for non-linear relationships between atoms and the reconstruction of input data. We illustrate its application in several different image processing settings.

2018

---

### Sinkhorn Distances: Lightspeed Computation of Optimal Transport

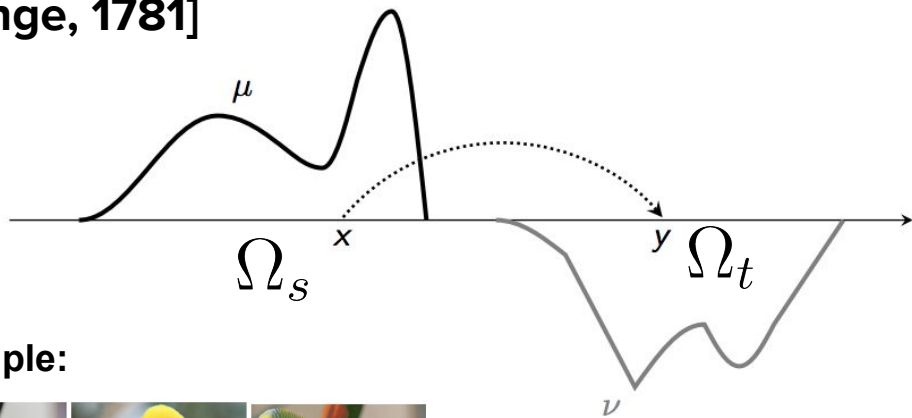
---

Marco Cuturi  
Graduate School of Informatics, Kyoto University  
mcuturi@i.kyoto-u.ac.jp

2013

# The Optimal Transport Problem

[Monge, 1781]



$$\mathcal{W}(\mu, \nu) = \inf_{T \# \mu = \nu} \int_{\Omega_s} c(\mathbf{x}_s, T(\mathbf{x}_s)) \mu(\mathbf{x}) d\mathbf{x}$$

$$T : \Omega_s \rightarrow \Omega_t$$

$$\nu(A) = \mu(T^{-1}(A))$$

$$c(\mathbf{x}_s, \mathbf{x}_t) = \|\mathbf{x}_s - \mathbf{x}_t\|^p$$

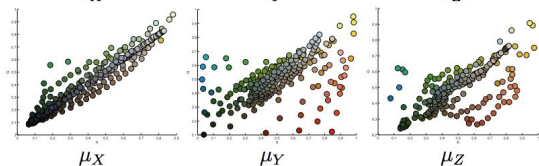
Example:



$I_X$

$I_Y$

$I_Z$

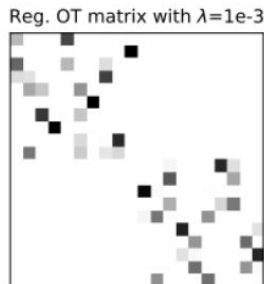
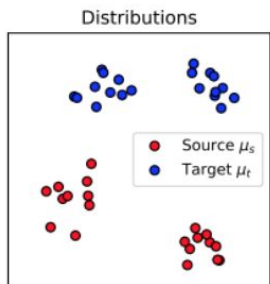


How do we transfer one mass to another in a way which minimizes transport cost?

**Problem:** The constraint on T is not convex!

# The Optimal Transport Relaxed

[Kantorovich, 1942]

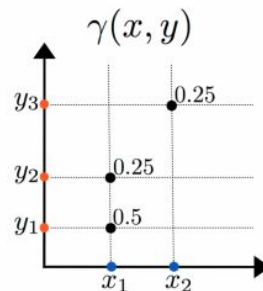
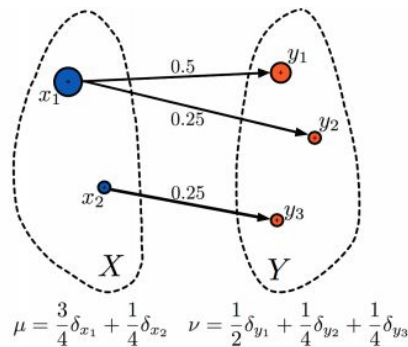
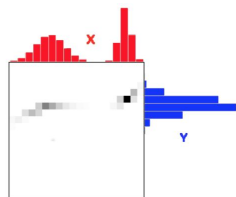


$$\mathcal{W}(\mu, \nu) := \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{\Omega_s \times \Omega_t} c(x, y) d\gamma(\mathbf{x}_s, \mathbf{x}_t)$$

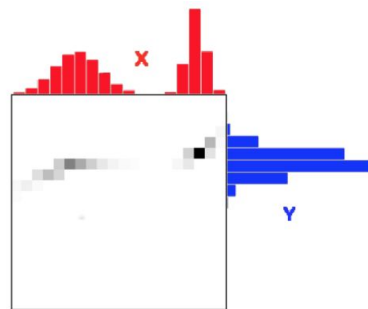
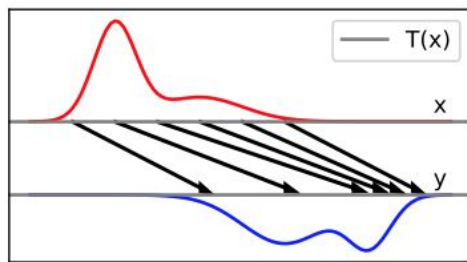
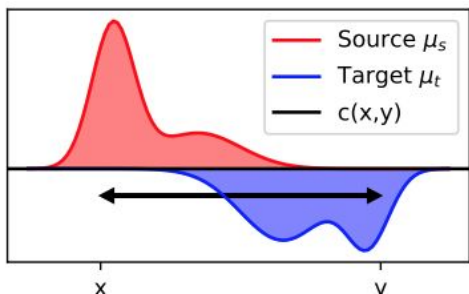
- This relaxed problem is now **convex** and therefore solvable (although may be degenerate).
- Take the example of (mass splits):

$\Gamma(\mu, \nu)$

- Is the set of all probability measures with marginals
- Also known as the set of **transportation plans**.



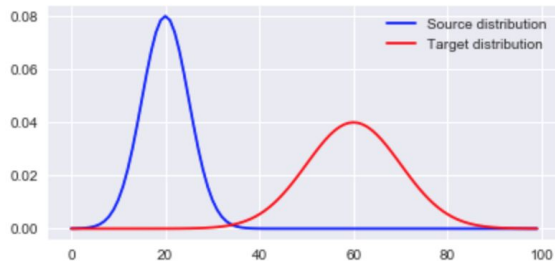
# Discrete optimal transport



**[Brenier, 95]** If the measures are continuous, the optimal joint probability is supported on the graph of a convex function  $T$ . (ie. solves original problem)

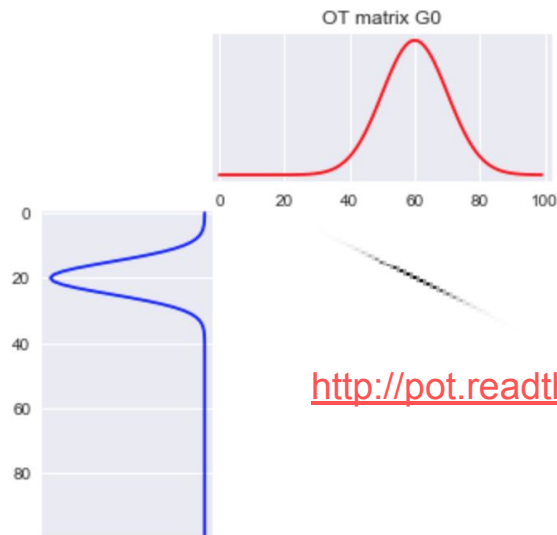
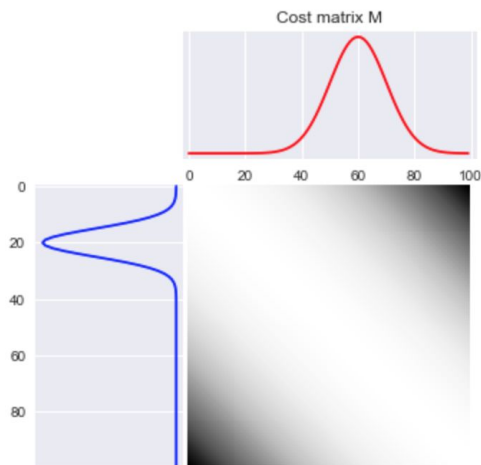
$$\begin{aligned} T(\mathbf{x}_s) &= \nabla \Psi(\mathbf{x}_s) \\ &\stackrel{\mathbf{1D}}{=} F_\nu^{-1} \circ F_\mu(\mathbf{x}_s) \end{aligned}$$

# Python Simulation of 1D case



In [11]:

```
### EMD  
  
G0 = ot.emd(a, b, M)  
  
pl.figure(3, figsize=(5, 5))  
ot.plot.plot1D_mat(a, b, G0, 'OT matrix G0')
```



<http://pot.readthedocs.io/en/stable/>

# Fluid Mechanics Formulation

$$\mathcal{W}(\mu, \nu) := \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{\Omega_x \times \Omega_y} c(x, y) d\gamma(x, y)$$

$$\partial_t \rho + \nabla \cdot (\rho v) = 0 \quad \rho(0, \cdot) = \rho_0 \quad \rho(T, \cdot) = \rho_T$$

$$K(\rho, v) = T \int_{\mathbb{R}^d} \int_0^T \rho(t, x) |v(t, x)|^2 dx dt$$

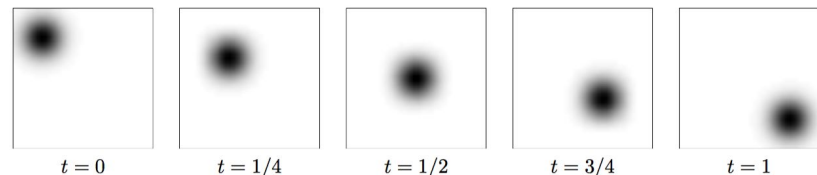
$$X(0, x) = x \quad \partial_t X(t, x) = v(t, X(t, x))$$

$$X(t, x) = x + \frac{t}{T} (\nabla \Phi(x) - x)$$

This formulation helps us formulate a well known meteorological model in the framework of optimal transport.

- Key observation is that the PDE preserves the structure of the transport map through a **Lagrangian flow**.
- This was used to improve time complexity by restricting class of solutions [**Brenier, Benamou, 2000**].

Exploits famous “displacement interpolation” by [**McCann, 1985**]



# Semi-geostrophic approximation

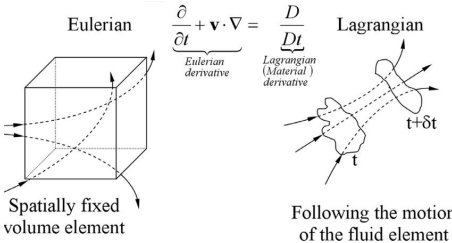
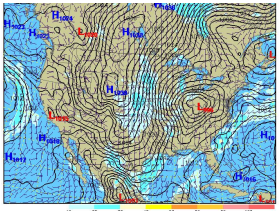
Recall the 2D incompressible Euler equations on a bounded domain  $\Omega \subset \mathbb{R}^2$  with Coriolis parameter  $f_0 = 1$  can be written as,

$$\frac{D\mathbf{u}}{Dt} + \mathbf{J} \cdot \mathbf{u} = -\nabla\phi \text{ where } \mathbf{J} := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (2)$$

Here the *advection operator* is defined as

$$\frac{D}{Dt} = \partial_t + \mathbf{u} \cdot \nabla. \quad (3)$$



Making a first order approximation to (1) we ignore the acceleration terms and get the **geostrophic velocity**

$$\mathbf{u}_g := \mathbf{J}\nabla\phi. \quad (4)$$

This yields

$$\frac{D\mathbf{u}_g}{Dt} + \mathbf{J}\mathbf{u} = -\nabla\phi \quad (5)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (6)$$

Equations (5)–(6) are known as the *2D Semigeostrophic equations*.

Note that (6) allows one to write  $\mathbf{u} = \mathbf{J}\nabla\psi$  for a stream function  $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}$ .

Flow can be recast as an optimal transport problem in “dual” variables

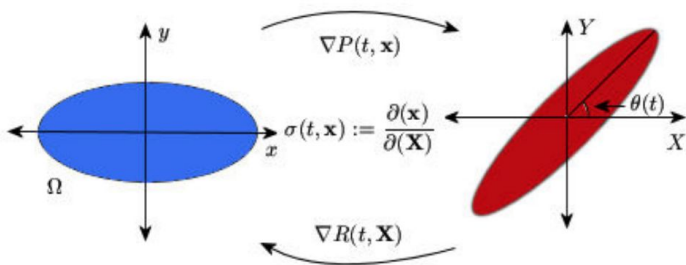


# Optimal Transport Formulation

The equations have a canonical Hamiltonian structure and can be written as,

$$\frac{d}{dt}(r, \theta) = J \nabla H_{SG}^{\sigma, s}(r, \theta), \quad (10)$$

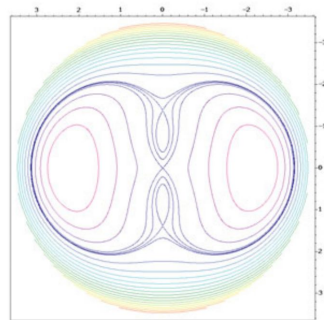
$$\frac{H_{SG}^{\sigma, s}}{2}(r, \theta) = \sigma^2 s + r - \sigma \left( 2 + 2rs + 2 \cos \theta \sqrt{(r^2 - 1)(s^2 - 1)} \right)^{1/2}, \quad (11)$$



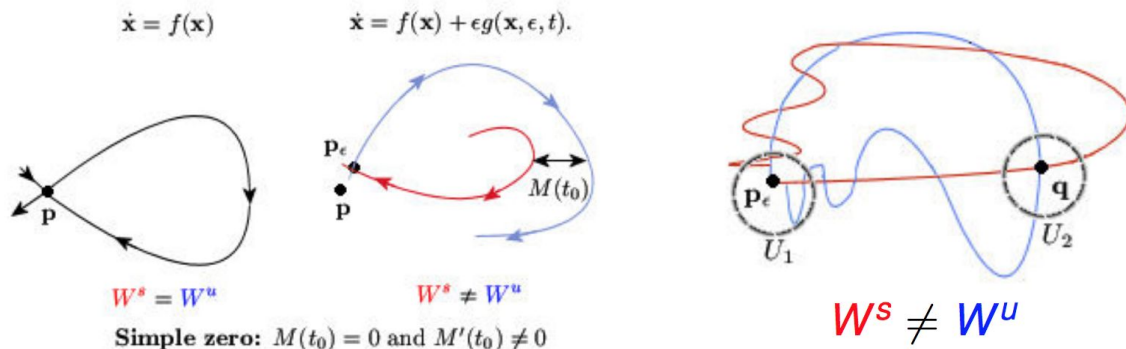
$$\det(D^2 \Psi) \rho_1(\nabla \Psi(\mathbf{x})) = \rho_0(\mathbf{x}).$$

- When restricted to an elliptical domain, the dynamics remain on a 2D ellipse for all time and have a Hamiltonian structure **[McCann, 2007]**
- Problem is an optimal transport between the two ellipses which stay ellipses (constrained to lie on the finite dimensional submanifold.)

Phase space  
**[McCann, 2017]**



# Periodic perturbations



- Integrating  $M$  along all points gives a measure of “distance” between the manifolds.
- This is known as the **Melnikov Method** [Melnikov, 1983].

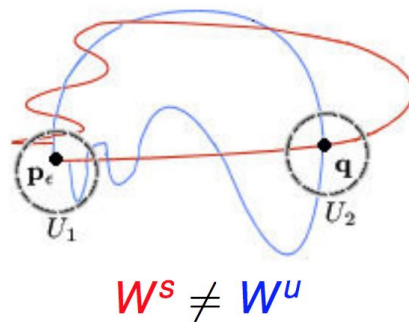
We consider a time periodic perturbation of the physical domain eccentricity in our Hamiltonian:

$$\frac{d}{dt}(r, \theta) = J \nabla H_{SG}^{\sigma, s}(r, \theta) + \epsilon \cos(kt + kt_0) J \nabla \frac{\partial H_{SG}^{\sigma, s}}{\partial s} + O(\epsilon^2).$$

# Main Result III: Chaotic Dynamics

[Goldman, McCann, 2008]

We consider a time periodic perturbation of the physical domain eccentricity in our Hamiltonian:

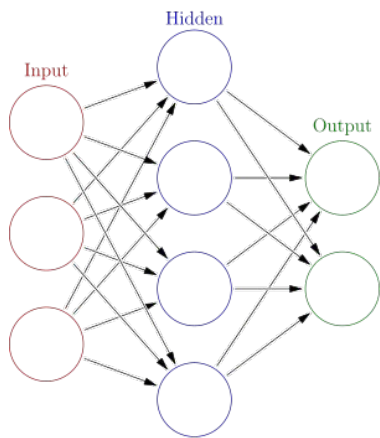


$$\frac{d}{dt}(r, \theta) = J \nabla H_{SG}^{\sigma, s}(r, \theta) + \epsilon \cos(kt + kt_0) J \nabla \frac{\partial H_{SG}^{\sigma, s}}{\partial s} + O(\epsilon^2).$$

For all but countably many frequencies, the dynamics are chaotic.

How is this relevant?

# Applications to AI: Stable Image Classification



$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j)$$

$$\sigma(\mathbf{Y}) = \tanh(\mathbf{Y})$$



$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}^T(t)\mathbf{y}(t) + b(t))$$

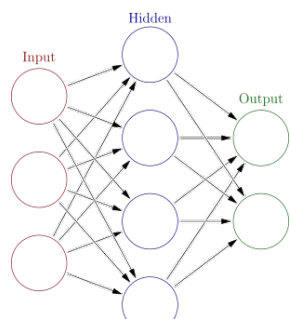
[Haber et al., 2017]

$$\mathbf{h}(\mathbf{X}) = \exp(\mathbf{X}) ./ (\exp(\mathbf{X})\mathbf{e}_m).$$

- Constructs a symmetric variant of the forward propagation which can be recast as a Hamiltonian system (which therefore preserves data).
- Uses explicit estimates on the eigenvalues to guarantee well posedness
- Says nothing about general stability of the phase space.



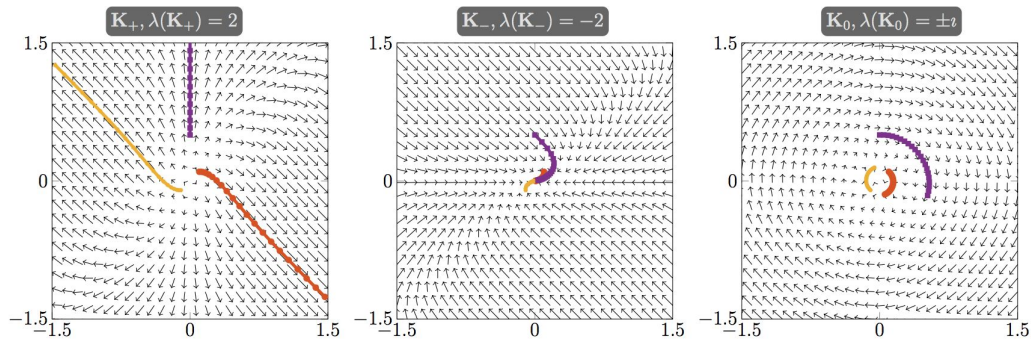
# Applications to AI: Stable Image Classification



$$\dot{\mathbf{y}}(t) = -\nabla_{\mathbf{z}} H(\mathbf{y}, \mathbf{z}, t)$$

$$\dot{\mathbf{z}}(t) = \nabla_{\mathbf{y}} H(\mathbf{y}, \mathbf{z}, t)$$

$$H(\mathbf{y}, \mathbf{z}) = \frac{1}{2} \mathbf{z}^T \mathbf{z} + f(\mathbf{y})$$



Unstable

Stable

- By constraining the form of  $f$ , they prove well posedness of the forward propagation scheme.
- It may be interesting to consider **similar variations of image noise** and determine when such perturbations will lead to **divergence of the models**.
- Interesting to see how this compares to [Goodfellow et al., 2017] which use adversarial training.

What about optimal transport?

# Applications to AI: Domain Adaptation and Transfer Learning

## Reweighting schemes [Sugiyama et al., 2008]

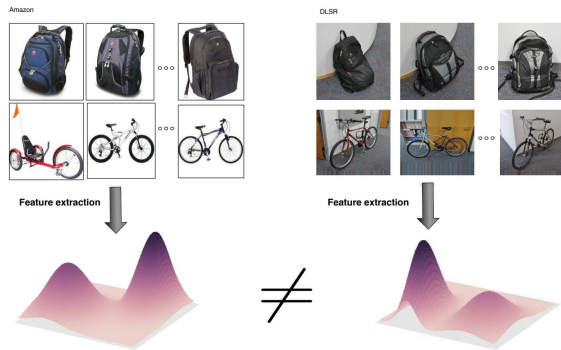
- Distribution change between domains.
- Reweight samples to compensate this change

## Subspace methods

- Data is invariant in a common latent subspace.
- Minimization of a divergence between the projected domains [Si et al., 2010].
- Use additional label information [Long et al., 2014].

## Gradual alignment

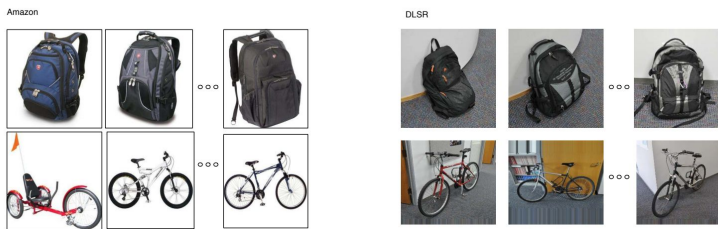
- Alignment along the geodesic between source and target subspace [R. Gopalan and Chellappa, 2014].
- Geodesic flow kernel [Gong et al., 2012].



All methods assume:

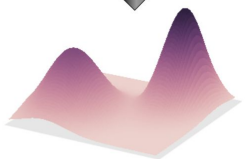
- You can transport entire domain to the other one (eg. PCA)
- Some very specific relationship between the distributions (same conditional distributions).

# Applications to AI: Transfer Learning via OT

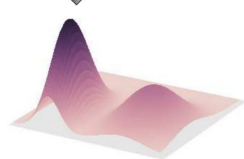


Feature extraction

Feature extraction



$\neq$

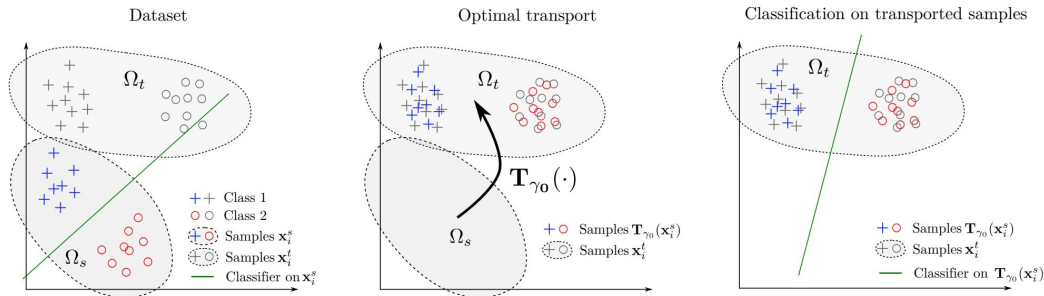


$$\mu_s = \sum_{i=1}^{n_s} p_i^s \delta_{\mathbf{x}_i^s} \quad \mu_t = \sum_{i=1}^{n_t} p_i^t \delta_{\mathbf{x}_i^t} \quad \begin{matrix} p_i^t = 1/n_t \\ p_i^s = 1/n_s \end{matrix}$$

$$\mathcal{W}(\mu_s, \mu_t) := \inf_{\gamma \in \Gamma(\mu_s, \mu_t)} \int_{\Omega_s \times \Omega_t} c(x, y) d\gamma(\mathbf{x}_s, \mathbf{x}_t)$$

**Courty et al. [2016]**

1. Train a classifier on  $\Omega_s$
2. Find optimal transportation plan between  $\mu_s$  and  $\mu_t$  (note only need marginals which are expectation over  $y$ ).
3. Train classifier on transported samples with labels.

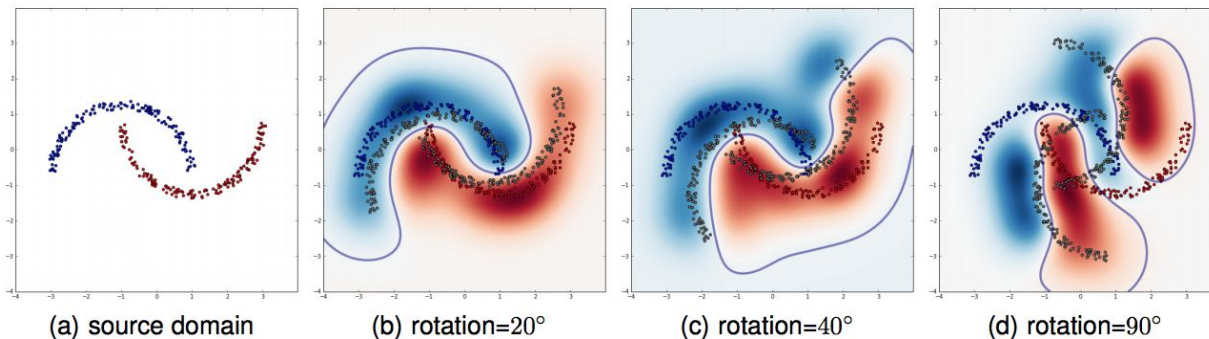


$$\hat{\mathbf{x}}_i^s = \operatorname{argmin}_{\mathbf{x} \in \Omega_T} \sum_{j=1}^n t \gamma_0(i, j) c(\mathbf{x}, \mathbf{x}_j^t)$$



# Performance Comparison - Two Moon Problem

Courty et al.  
[2016]



L2 cost  
between  
domains

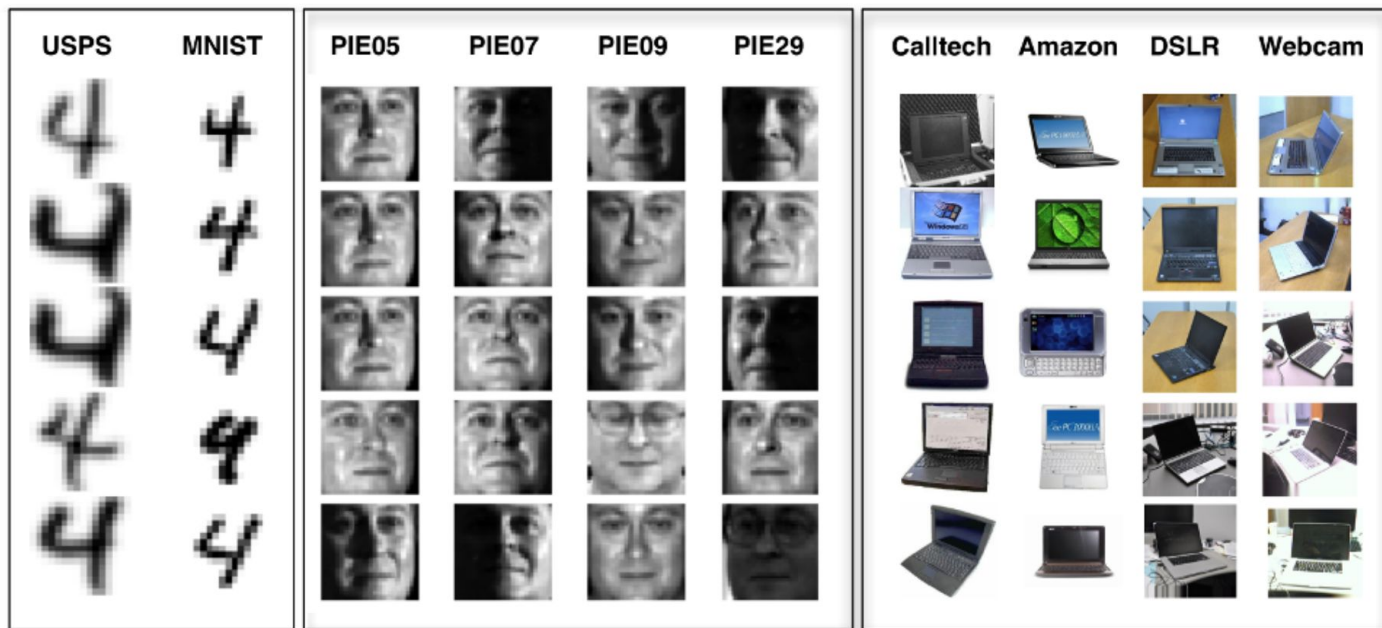
PAC-Bayesian  
Perspective

Different  
regularizations

	10°	20°	30°	40°	50°	70°	90°
SVM (no adapt.)	0	0.104	0.24	0.312	0.4	0.764	0.828
DASVM	0	<b>0</b>	0.259	0.284	0.334	0.747	0.820
PBDA	0	0.094	0.103	0.225	0.412	0.626	0.687
<b>OT-exact</b>	0	0.028	0.065	0.109	0.206	0.394	<b>0.507</b>
<b>OT-IT</b>	0	0.007	0.054	0.102	0.221	0.398	<b>0.508</b>
<b>OT-GL</b>	0	<b>0</b>	<b>0</b>	<b>0.013</b>	<b>0.196</b>	<b>0.378</b>	<b>0.508</b>
<b>OT-Lap</b>	0	<b>0</b>	0.004	0.062	0.201	0.402	0.524



# Examples used for evaluation



Courty et al. [2016]

Office Caltech Dataset

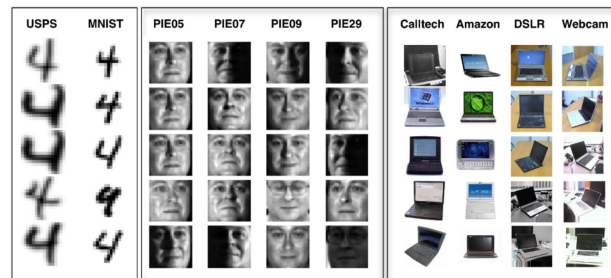
<http://www.vision.caltech.edu/archive.html>

# Results on Caltech dataset

TABLE 3: Overall recognition accuracies in % obtained over all domains pairs using the SURF features. Maximum values for each pair is indicated in bold font.

Domains	1NN	PCA	GFK	TSL	JDA	OT-exact	OT-IT	OT-Laplace	OT-LpLq	OT-GL
U→M	39.00	37.83	44.16	40.66	54.52	50.67	53.66	57.42	<b>60.15</b>	57.85
M→U	58.33	48.05	60.96	53.79	60.09	49.26	64.73	64.72	68.07	<b>69.96</b>
mean	48.66	42.94	52.56	47.22	57.30	49.96	59.20	61.07	<b>64.11</b>	63.90
P1→P2	23.79	32.61	22.83	34.29	<b>67.15</b>	52.27	57.73	58.92	59.28	<b>58.41</b>
P1→P3	23.50	38.96	23.24	33.53	56.96	51.36	57.43	57.62	58.49	<b>58.73</b>
P1→P4	15.69	30.82	16.73	26.85	40.44	40.53	47.21	47.54	47.29	<b>48.36</b>
P2→P1	24.27	35.69	24.18	33.73	<b>63.73</b>	56.05	60.21	62.74	62.61	<b>61.31</b>
P2→P3	44.45	40.87	44.03	38.35	<b>68.42</b>	59.15	63.24	64.29	62.71	64.36
P2→P4	25.86	29.83	25.49	26.21	49.85	46.73	51.48	<b>53.52</b>	50.42	52.68
P3→P1	20.95	32.01	20.79	39.79	<b>60.88</b>	54.24	57.50	57.87	58.96	57.91
P3→P2	40.17	38.09	40.70	39.17	65.07	59.08	63.61	<b>65.75</b>	64.04	64.67
P3→P4	26.16	36.65	25.91	36.88	52.44	48.25	52.33	<b>54.02</b>	52.81	52.83
P4→P1	18.14	29.82	20.11	40.81	<b>46.91</b>	43.21	45.15	45.67	46.51	45.73
P4→P2	24.37	29.47	23.34	37.50	<b>55.12</b>	46.76	50.71	52.50	50.90	51.31
P4→P3	27.30	39.74	26.42	46.14	<b>53.33</b>	48.05	52.10	52.71	51.37	52.60
mean	26.22	34.55	26.15	36.10	<b>56.69</b>	50.47	54.89	56.10	55.45	55.88
C→A	20.54	35.17	35.29	45.25	40.73	30.54	37.75	38.96	<b>48.21</b>	44.17
C→W	18.94	28.48	31.72	37.35	33.44	23.77	31.32	31.13	38.61	<b>38.94</b>
C→D	19.62	33.75	35.62	39.25	39.75	26.62	34.50	36.88	39.62	<b>44.50</b>
A→C	22.25	32.78	32.87	<b>38.46</b>	33.99	29.43	31.65	33.12	35.99	34.57
A→W	23.51	29.34	32.05	35.70	36.03	25.56	30.40	30.33	35.63	<b>37.02</b>
A→D	20.38	26.88	30.12	32.62	32.62	25.50	27.88	27.75	36.38	<b>38.88</b>
W→C	19.29	26.95	27.75	29.02	31.81	25.87	31.63	31.37	33.44	<b>35.98</b>
W→A	23.19	28.92	33.35	34.94	31.48	27.40	37.79	37.17	37.33	<b>39.35</b>
W→D	53.62	79.75	79.25	80.50	<b>84.25</b>	76.50	80.00	80.62	81.38	84.00
D→C	23.97	29.72	29.50	31.03	29.84	27.30	29.88	31.10	31.65	<b>32.38</b>
D→A	27.10	30.67	32.98	36.67	32.85	29.08	32.77	33.06	37.06	<b>37.17</b>
D→W	51.26	71.79	69.67	77.48	80.00	65.70	72.52	72.16	74.97	<b>81.06</b>
mean	28.47	37.98	39.21	42.97	44.34	36.69	42.30	43.20	46.42	<b>47.70</b>

- PCA, which consists in applying a projection on the first principal components of the joint source/target distribution (estimated from the concatenation of source and target samples);
- GFK, Geodesic Flow Kernel [23];
- TSL, Transfer Subspace Learning [44], which operates by minimizing the Bregman divergence between the domains embedded in lower dimensional spaces;
- JDA, Joint Distribution Adaptation [34], which extends the Transfer Component Analysis algorithm [38];



Model used was 1NN.

# Deep Neural Nets and Transfer Learning

[Ying Lu et al., Sep 2017]

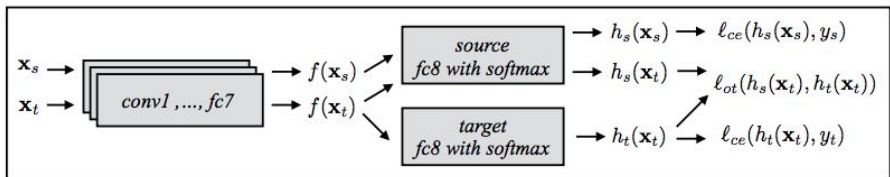


Figure 1: The structure and data flow of a Joint Transfer Learning Network based on Alexnet

- We have some very small training sample.
- We have a large collection of labeled related source data.
- Can we use relationships between source and target to help the final classifier?

$$h_t(\mathbf{x}_i) = \sigma(\mathbf{W}_t \cdot f(\mathbf{x}_i) + \mathbf{b}_t)$$

$$h_s(\mathbf{x}_i) = \sigma(\mathbf{W}_s \cdot f(\mathbf{x}_i) + \mathbf{b}_s)$$

$$\min_{\Theta} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}_{ce}(h_t(\mathbf{x}_i^t), y_i^t) + \frac{\lambda_s}{n_s} \sum_{i=1}^{n_s} \mathcal{L}_{ce}(h_s(\mathbf{x}_i^s), y_i^s)$$



**Training:** Boeing manufacturer



**Source:** All others

# Deep Neural Nets and Transfer Learning

[Ying Lu et al., Sep 2017]

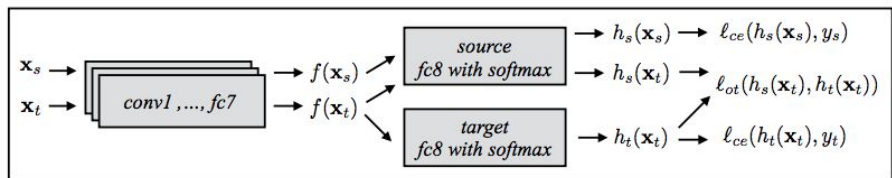


Figure 1: The structure and data flow of a Joint Transfer Learning Network based on Alexnet

$$\min_{\Theta} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}_{cc}(h_t(\mathbf{x}_i^t), y_i^t) + \frac{\lambda_s}{n_s} \sum_{i=1}^{n_s} \mathcal{L}_{cc}(h_s(\mathbf{x}_i^s), y_i^s) + \lambda_t \mathcal{W}(\mu_s, \mu_t)$$

+

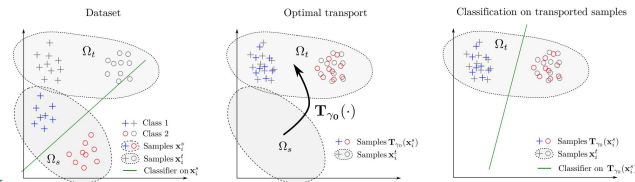


Table 1: ITL Datasets with FGVC-Aircraft

Dataset properties	Boeing	Airbus
N° of target categories	22	13
N° of target training images	1466	867
N° of target testing images	734	433
N° of source categories	78	87
N° of source images	7800	8700

Table 2: Experimental Results on the ITL Datasets (results are multi-class classification accuracy)

Methods	Boeing	Airbus
Finetuning on target	0.4796	0.4965
Consecutive finetuning on source+target	0.5286	0.545
Joint finetuning on source+target	0.5395	0.5497
JTLN (fc7MKMMD)	0.5422	<b>0.5982</b>
JTLN (fc7OT)	<b>0.5436</b>	0.5704

# Solving the transportation problem

$$\mathcal{B} = \{\gamma \in (\mathbb{R}^+)^{n_s \times n_t} \mid \gamma \mathbf{1}_{n_t} = \mu_s, \gamma^T \mathbf{1}_{n_s} = \mu_t\}$$

$$\mu_s = \sum_{i=1}^{n_s} p_i^s \delta_{\mathbf{x}_i^s}$$

$$\mu_t = \sum_{i=1}^{n_t} p_i^t \delta_{\mathbf{x}_i^t}$$

$$\gamma_0 = \operatorname{argmin}_{\gamma \in \mathcal{B}} \langle \gamma, \mathbf{C} \rangle_F$$

- This is all great, but is this problem tractable?
- The above problem is a linear program with convex constraints. This has **polynomial time complexity to solve** ( $O(n^3)$  in worst case).
- **[Curti, 2013]** - what is the **maximum entropy solution**? Obviously when we **spread mass evenly across all points**.
- There is only one way to do this so solution is trivial. So can we restrict to a neighborhood around this trivial solution?

## Maximum Entropy Solution

$$r = [p_1^s, p_2^s, \dots, p_n^s]$$

$$c = [p_1^t, p_2^t, \dots, p_n^t]$$

$$\gamma = r c^T = [r_i c_j]_{i,j}$$

# Shrinking the admissible space

## [Curti, 2013] Entropic Regularization

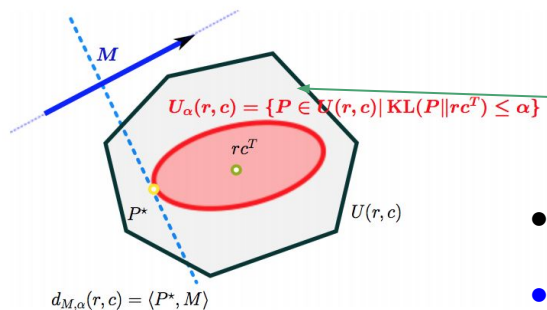
$$\gamma_0^\lambda = \operatorname{argmin}_{\gamma \in \mathcal{B}} \langle \gamma, \mathbf{C} \rangle_F + \lambda \Omega_s(\gamma)$$

$$\Omega_s(\gamma) = \sum_{i,j} \gamma_{i,j} \log \gamma(i,j)$$

### Dual version:

$$\gamma_0^\alpha = \operatorname{argmin}_{\gamma \in B_\alpha} \langle \gamma, \mathbf{C} \rangle \quad r = [p_1^s, p_2^s, \dots, p_n^s] \quad c = [p_1^t, p_2^t, \dots, p_n^t]$$

$$B_\alpha = \{ \gamma \in B \mid \Omega(\gamma) - \Omega(r) - \Omega(c) \geq \alpha \}$$

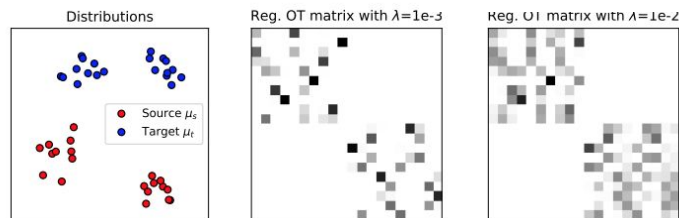
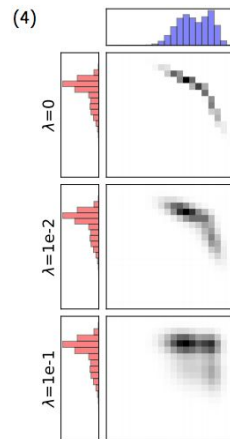


Trivial solution

- Makes problem strictly convex and allows one to solve using **Sinkhorn Knopp fixed point method**, which has linear complexity.
- **Makes OT usable for deep learning applications!**

$$\mu_s = \sum_{i=1}^{n_s} p_i^s \delta_{\mathbf{x}_i^s}$$

$$\mu_t = \sum_{i=1}^{n_t} p_i^t \delta_{\mathbf{x}_i^t}$$

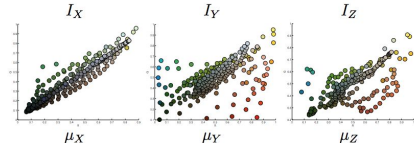




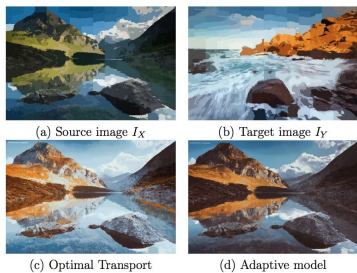
# More Applications to AI



Image editing [Perrot, 2017]



Color transfer, [J Rabin, 2014]



Siberian husky



Eskimo dog



Flickr : street, parade, dragon  
Prediction : people, protest, parade

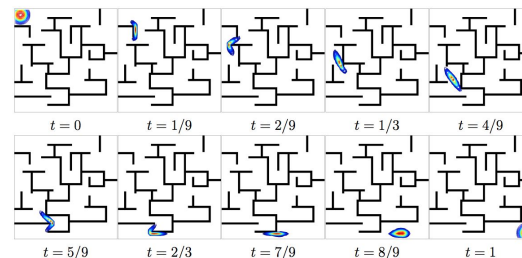


Flickr : water, boat, ref ection, sun-shine  
Prediction : water, river, lake, summer;

## Image tagging and segmentation [Frognier et al, 2015]



## Solving mazes



# Where did the transport map go?

## Drawbacks of Kantorovich formulation:

- Doesn't naturally extend to out of sample predictions. i.e. it can only be used on the samples given.
- This is a severe limitation for most applications where the generalization unseen data is essential.

[**Courty et al. 2016**] propose approximating the transport map by the joint distribution.

[**Brenier, Benamou, 2000**], Complexity of solution improved by Fluid Mechanics Formulation.

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.7.6791&rep=rep1&type=pdf>

The time complexity of optimal transport is still an issue facing large scale use. However it is mentioned in [**Papadakis, 2017**]:

*“In this context, the fluid dynamic formulation of the Optimal Transport problem introduced in [**Brenier, Benamou, 2000**] is an interesting approach for dealing with higher dimensions. The entropic regularization proposed in [**Courty, 2013**] has also offered new perspectives as faster algorithms based on Sinkhorn distances can be designed for computing approximate Optimal Transport in larger dimensions.”*



# Future Directions

1. Stability of image classification under realistic perturbations to pixels.
2. Image classification under fluid flow: complexity vs performance.
3. Voice recognition via optimal transport (ie. Echo?)
4. Translation?
5. Build robust libraries for general use related to reinforcement learning, image processing, etc.

Thank you!

# Reinforcement Learning

---

Choosing optimal policies from historical data

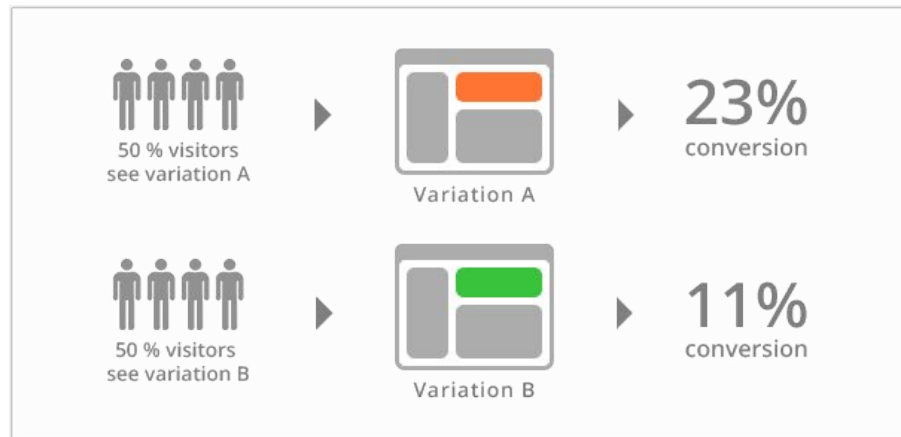
# Uplift Modeling vs AB Testing

## Uplift Modeling



Which variation is better for each individual?

## AB Testing



Which variation is better?

# What are we trying to maximize? Total Reward

$$\mathbb{E}_{\Omega}(y) := \int y(x)\Omega(x)dx$$

$$\Omega(y, a, x) = R(y|a, x)\Pi(a|x)w(x);$$

$\Omega$  Distribution of actions and rewards (ie. clicks).

$Q$  Distribution of logged data (outcome, action, user attributes)

- Reward  $y$  for action  $a$  and attributes  $x$ .
- Policy - what action does person with attributes  $x$  get? Can be stochastic or deterministic.
- Distribution of population.

# How do we test a policy?

$$\mathbb{E}_{\Omega}(y) := \int y(x)\Omega(x)dx$$

$$\Omega(y, a, x) = R(y|a, x)\Pi(a|x)w(x);$$

$$Q(y, a, x) = R(y|a, x)B(a|x)w(x)$$

**Importance sampling** is roughly the idea of writing the expectation in terms of the desired distribution, using the known one.

- Reward  $y$  for action  $a$  and attributes  $x$ .
- Policy (unknown) - what action does person with attributes  $x$  get?
- Distribution of population (known).
- Observed action from actual data - this is what we actually know (known).

## Connecting to the population distribution

$$\mathbb{E}_{\Omega}(y) := \int y(x)\Omega(x)dx$$

$$\Omega(y, a, x) = R(y|a, x)\Pi(a|x)w(x);$$

$$\frac{\Omega}{Q} = \frac{\Pi(a|x)}{B(a|x)},$$

We can estimate the expectation now from our observed data.

$$\mathbb{E}_{\Omega}(y) := \int y(x)\Omega(x)dx = \sum_{y,a,x} \frac{\Pi(a|x)}{B(a|x)} y Q(y, a, x)$$

# Empirical Distribution Examples

$$B(a|x) = \frac{1}{N} \sum_{i=1}^N \delta(a - a_i).$$

$a_i \in \{0 = \text{untreated}, 1 = \text{treated}\}$ , then  $B(a_i) \in \{B(0), B(1)\}$

$$B(0) = B(1) = 1/2$$

$$Q(y, a, x) \sim \frac{1}{N} \sum_{i=1}^N \delta(y - y_i) \delta(a - a_i) \delta(x - x_i).$$

$$Q_i(y, a, x) \in \{(1, 1, x_i), (1, 0, x_i), (0, 1, x_i), (0, 0, x_i)\}$$

$$Q_i = \frac{1}{4}$$

- Half the values are 0 and half are 1 for a randomized controlled trial

- In the special case that both outcomes and actions are uniformly distributed across the population,  $Q_i = \frac{1}{4}$



## Estimating Expected Reward

$$\mathbb{E}_{\Omega}(y) = \mathbb{E}_Q \left( \frac{\Pi}{B} y \right) \sim \sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i =: V_h(y)$$

$$V_h(y) \sim \hat{V}_h(y) = \sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i$$

*This is known as  
importance  
sampling.*

# Policy Function

$$\Pi(a_i|x_i) = \mathbf{1}(a_i = h(x_i)) \leftarrow \text{Each user gets some specific deterministic action, determined by } h.$$

$$h : \mathbf{X} \rightarrow a$$

$$\hat{V}_h(y) = \frac{1}{N} \sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i = \frac{1}{N} \sum_{i=1}^N y_i \frac{\mathbf{1}(a_i = h(x_i))}{B(a_i|x_i)}$$

# 1D Example - Should we send an email?

$$\Pi(a_i|x_i) = \mathbf{1}(a_i = h(x_i))$$

$h : \mathbf{X} \rightarrow a$  Policy based on age

$x_i$  Age of user  $i$

Simulated Data (eg. AB test)

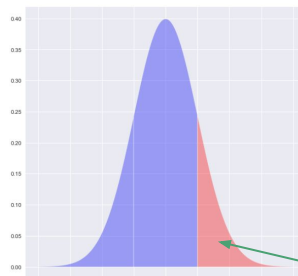
$a_i = 1$  and  $x_i > 30$  then 1

$a_i = 1$  and  $x_i < 30$  then - 1

otherwise 0

$$\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}$$

Age dist.



Only people over 30 responded well.

**What maximizes expectation here?**

**Answer:** We want  $h$  to predict the action right when we had a positive outcome.

# 1D Example - Should we send an email?

$$\Pi(a_i|x_i) = \mathbf{1}(a_i = h(x_i))$$

$h : \mathbf{X} \rightarrow a$  Policy based on age

$x_i$  Age of user  $i$

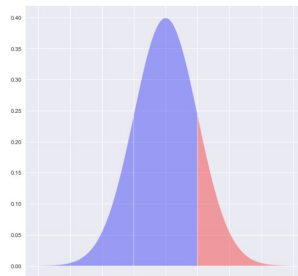
Simulated Data (eg. AB test)

$a_i = 1$  and  $x_i > 30$  then 1

$a_i = 1$  and  $x_i < 30$  then -1

otherwise 0

$$\frac{\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}{}$$



$$h(x) = \begin{cases} 1, & \text{if } x \geq 30 \\ 0, & \text{otherwise} \end{cases}$$

- We want to only send to people over 30 in this case.
- Derivation is by inspection so far but will address this.

# Policy Function

$$\Pi(a_i|x_i) = \mathbf{1}(a_i = h(x_i))$$

$$h : \mathbf{X} \rightarrow a$$

**Example:**

$$h(x) = \begin{cases} 1, & \text{if } x \geq 30 \\ 0, & \text{otherwise} \end{cases}$$

$a_i = 1$  and  $x_i < 30$  then 0

$$\hat{V}_h(y) = \frac{\sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i}{\sum_{i=1}^N \frac{\Pi_i}{B_i} Q_i} = \frac{\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}{\sum_{i=1}^N \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}$$

**The normalization accounts for when the outcome is highly correlated with the action. I.e. Only people who received the action responded positively.**

# Solving? Classification in Disguise!

$$\hat{V}_h(y) = \frac{\sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i}{\sum_{i=1}^N \frac{\Pi_i}{B_i} Q_i} = \frac{\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}{\sum_{i=1}^N \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}$$

Set a deterministic policy for the possible actions.

We just need to solve a weighted classification problem now and train a model to classify  $a_i$  where  $y$  is large.

- Our way of dealing with the denominator. Above problem isn't well posed.

$$\mathcal{L}_\lambda^w(h) = \sum_{i=1}^N w_i e(a_i, h(x_i)) + \lambda \sum_{i=1}^N \frac{\mathbf{1}(a_i \neq h(x_i))}{B(a_i|x_i)}$$

# Sanity Checks

1. Everyone purchases.
2. Nobody purchases.
3. You can only choose one action, independent of  $x$ .

$$\hat{V}_h(y) = \frac{\sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i}{\sum_{i=1}^N \frac{\Pi_i}{B_i} Q_i}$$
$$= \frac{\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}{\sum_{i=1}^N \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}$$

# Sanity Checks

1. Everyone purchases.  
Doesn't matter. No learnings.
2. Nobody purchases.  
Doesn't matter. No learnings.
3. You can only choose one action, independent of  $x$ .

The action which maximizes conversion rate.

$$\hat{V}_h(y) = \frac{\sum_{i=1}^N \frac{\Pi_i}{B_i} y_i Q_i}{\sum_{i=1}^N \frac{\Pi_i}{B_i} Q_i} = \frac{\sum_{i=1}^N y_i \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}{\sum_{i=1}^N \frac{\mathbf{1}(a_i=h(x_i))}{B(a_i|x_i)}}$$



# Optimizing the Policy

$$\mathcal{L}^w(h) = \sum_{i=1}^N w_i e(a_i, h(x_i))$$

$$e(a_i, h(x_i)) = \frac{\mathbf{1}(a_i \neq h(x_i))}{B(a_i|x_i)}$$

$$w_i = y_i$$

- Thus the expectation maximization can be understood as minimizing a weighted classification loss.
- We can solve the problem by trying out various priors such as Bernoulli (Logistic), piecewise constant (Decision Tree).
- Then we find the best policy by find the maximum likelihood estimator.

# Python Package for Uplift

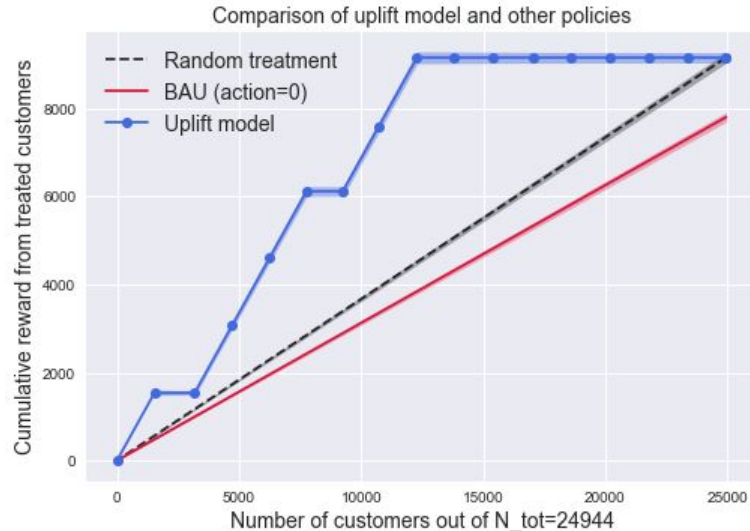
```
Parameters
X_train : pandas dataframe
    Full feature set for training.
action_train : pandas series or numpy array
    Actions to be fitted for (a=0, 1 for binary actions; a=0, 1, 2, ...
    for multiclass classification).
outcome_train : pandas series or numpy array
    Outcomes for X_train and action_train, used for sample weights.
lambda_ : float, optional, default = None
    Hyperparameter for reward. Acts like a regularizer.
    -infinity < lambda < min(reward)
mu_ : dict, optional, default=0
    Pre-defined cost to reward for each action. Only need to be passed
    at class initialization or 'fit()'.
clf : object, optional
    Classifier object to be fitted
set_best_model : boolean, optional, default=False
    Set the current fitted model as attribute 'best_model'.
plot_weight_hist : boolean, optional
    Make diagnostic plot of sample weights

Return
self : object
    Returns self.
...
# check classifier
try:
    if (not hasattr(self, 'clf')) and (not clf):
        raise NameError
    elif clf:
        self.clf = clf(**kwargs) # assign clf or overwrite previous version
except NameError:
    print('fit: Must provide a classifier object to fit!')
self._check_LR_solver()

# check and validate data
action_train = self._validate_narray(action_train)
outcome_train = self._validate_narray(outcome_train)
self._check_length(X_train, action_train)
self._check_length(X_train, outcome_train)
self._check_mu_(mu_) # check mu_ and assign to self.mu_
self._check_lambda_(lambda_)
print('fit: lambda_={0}, mu_={1}'.format(self.lambda_, self.mu_))

self.get_bias(action_train) # calculate bias and store in attributes to be called later
weights = self.get_weights(action_train, outcome_train, self.lambda_, mu=self.mu_[0])
fitted = self.clf.fit(X_train, action_train, weights)
if set_best_model:
    self.update_best_model(fitted)
if plot_weight_hist:
    plt.figure()
    plt.hist(weights)
    plt.title('UpliftModel.fit(): Sample Weights for model fitting')
return fitted
```

<https://github.com/doriang102/foraws>



# Next Steps: Sequences of policies

$$\hat{\mathbb{E}} \left[ \sum_{t=1}^T r_t | \pi_p \right] = \sum_{t=1}^T \frac{1}{m} \sum_{i=1}^m \frac{\prod \pi_p(a_{i,1}, \dots, a_{i,t} | h_{t-1,i})}{\prod \pi_q(a_{i,1}, \dots, a_{i,t} | h_{t-1,i})} r_{t,i}$$

$$\pi(a_{i,1}, \dots, a_{i,t} | h_{t-1,i}) = \prod_{t=1}^T \pi(a_i | h_{t-1,i})$$

$$\pi(a_i | h_{t-1,i}) = \frac{\exp(-\theta_i^T \cdot \psi_t)}{\sum_{a_j} \exp(-\theta_j^T \cdot \psi_T)}$$

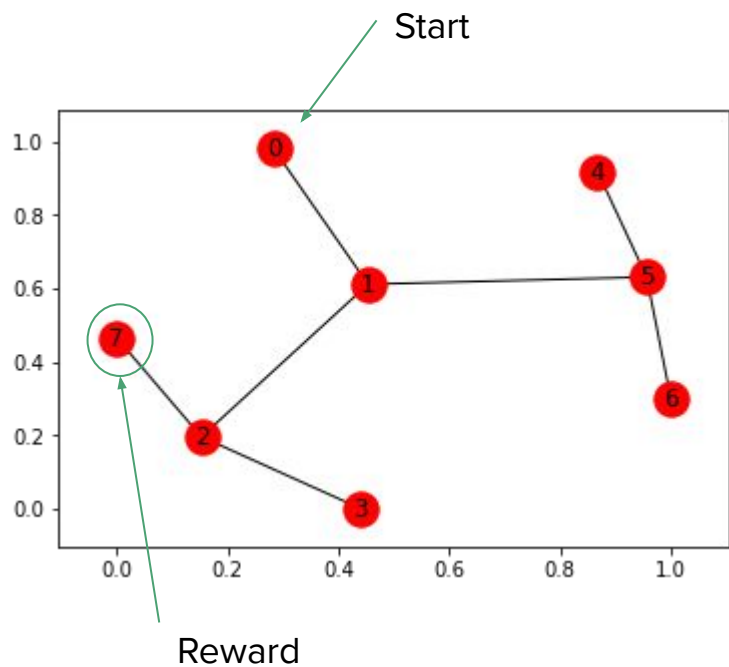
$h_{t-1}$  History up to time t.

$\psi_t$  Feature vector - x, history.

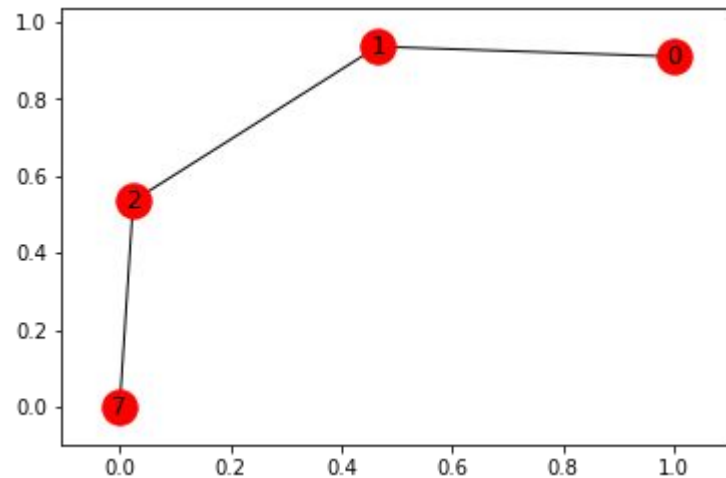
$\theta_i$  Action weight.

<https://homes.cs.washington.edu/~zoran/orderingpaperExtended.pdf>

# MDP via Policy Gradients



Best path



<https://github.com/Columbia-Intro-Data-Science/APMAE4990-/blob/master/notebooks/Markov%20Decision%20Processes%20via%20Policy%20Gradients.ipynb>

## Policy to maximize Reward

$$\mathbb{E}[R|\pi] = \int p(\tau|\pi)R(\tau)$$

$\tau = (a_0, s_0, r_0, a_1, s_1, r_1, \dots, a_t, s_t, r_t)$

$$p(\tau|\pi) = \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \quad \text{Markov property}$$

$$\pi(a_t|s_t) = \frac{e^{-\theta_{s_t, s_{t-1}}}}{\sum_k e^{-\theta_{s_t, s_k}}}$$

# Policy Gradient Trick

$$\begin{aligned}\nabla_{\theta} \mathbb{E}[R|\pi] &= \int \nabla_{\theta} p(\tau|\pi) R(\tau) \\ &= \int \frac{\nabla_{\theta} p(\tau|\pi)}{p(\tau|\pi)} p(\tau|\pi) R(\tau) \\ &= \mathbb{E}_p [\nabla_{\theta} \log p(\tau|\pi) R(\tau)]\end{aligned}$$

Now we can use our samples again to estimate this.

## Some magic

$$\begin{aligned}\log p(\tau|\pi) &= \log \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \\ &= \sum_{t=1}^{T-1} \log p(s_{t+1}|s_t, a_t) + \sum \log \pi(a_t|s_t)\end{aligned}$$

## A nudge in the right direction

$$\theta_{km}^t = \theta_{km}^{t-1} - R(t) \nabla_{\theta} \log \pi(k|m)$$

$$\log \pi(k|m) = -\theta_{km} - \log \left( \sum_l e^{-\theta_{lm}} \right).$$

$$\nabla_{\theta_{km}} \log \pi(k|m) = \pi(k|m) - 1,$$

A stochastic gradient descent based on the training example updates the weight in the “right” direction, weighted by the observed reward.

```
In [22]: def gradient(k,m,alpha=0.1):  
          #denom = sum([np.exp(-theta[k,l]) for l in range(8)])  
          return R[k,m]*(W[k,m]-1)
```



# Initialize Weights

```
: s_current = 0
# Initialize Weights
W = np.matrix(np.ones(shape=(8,8)))
#W *= 1/8

# Initialize Weights
theta = np.matrix(np.zeros(shape=(8,8)))

# Initialize Weights to be learned
for k in range(8):
    W[k]=softmax(theta[k].squeeze())
```

# Iterate and find shortest path

```
best_path=[]
best_length=1000
found_after = 0
for simulation in range(100):
    s_current = 0
    # Initialize Weights
    theta = np.matrix(np.zeros(shape=(8,8)))

    # Initialize Weights to be learned
    for k in range(8):
        W[k]=softmax(theta[k].squeeze())
    path=[]
    for t in range(10):
        reward=-1
        while reward < 0:
            s_next = np.where(np.random.multinomial(1, np.array(W[s_current,:]).squeeze(), size=1)==1)[1][0]
            reward = R[s_current,s_next]

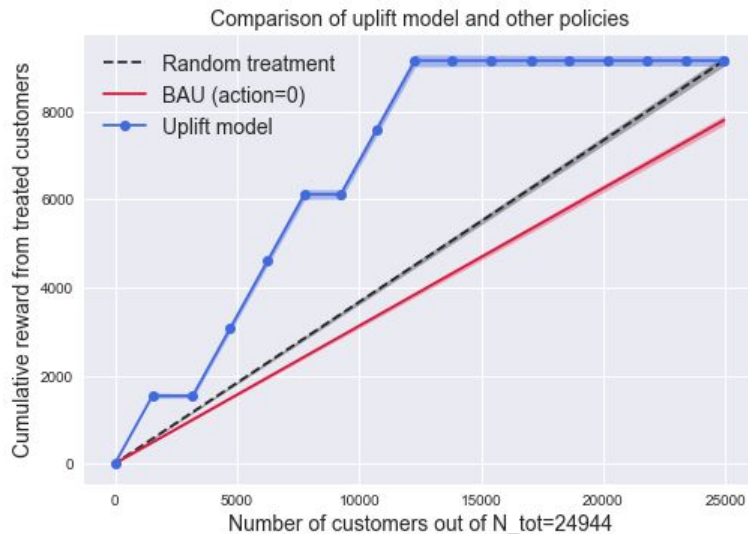
        path.append((s_current,s_next))
        if reward == 100:
            if len(path) < best_length:
                best_path = path
                best_length=len(best_path)
                found_after = simulation
                break
        #print (s_current,s_next,R[s_current,s_next])

    reward_next = R[s_current,s_next]
    #print (s_next)
    for m in range(8):
        theta[s_current,m] = theta[s_current,m] + gradient(s_current,m,alpha=0.01)
    for k in range(8):
        W[k]=softmax(theta[k].squeeze())
    s_current = s_next

print ("Found best path " + str(best_path) + " after " + str(found_after) + " simulations")
```

$$\theta_{km}^t = \theta_{km}^{t-1} - R(t) \nabla_{\theta} \log \pi(k|m)$$

# Some fake data simulated



## 1. Features:

- Feature1 : can\_afford product
- Feature2 : knows\_product
- Feature3 : likes\_product
- Feature4 : current subscriber

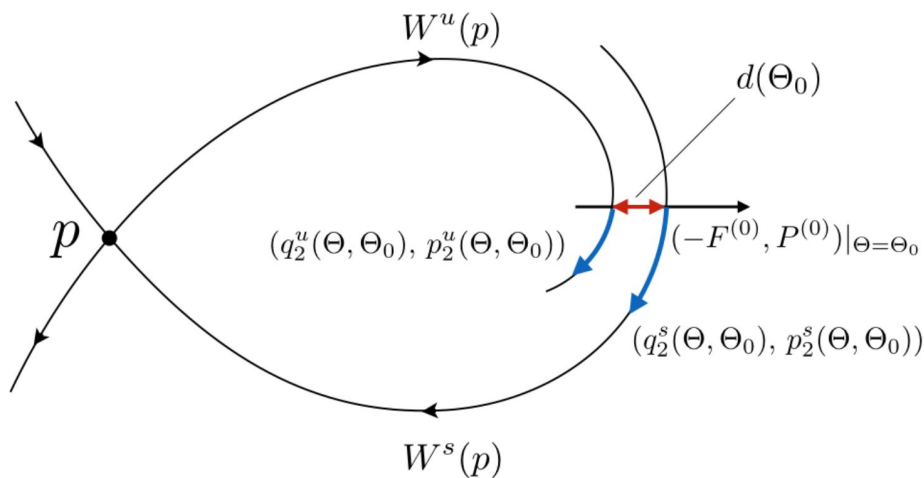
## 2. Outcome (aka reward, response) = 1, if:

- can\_afford==1 & knows\_prod ==1 & likes\_prod ==1, regardless of sub and action
  - can\_afford==1 & knows\_prod ==0 & likes\_prod ==1 & sub==0 or 1 & action ==1
  - can\_afford==1 & knows\_prod ==1 & likes\_prod ==0 & sub==1 & action ==0
  - can\_afford==1 & knows\_prod ==0 & likes\_prod ==0 & sub==1 & action ==0
  - can\_afford==0 & knows\_prod ==0 & likes\_prod ==1 & sub==1 & action ==1
  - can\_afford==0 & knows\_prod ==1 & likes\_prod ==1 & sub==1 & action ==1
  - can\_afford==0 & knows\_prod ==0 & likes\_prod ==0 & sub==1 & action ==0
- else outcome = 0

## Package Simulation:

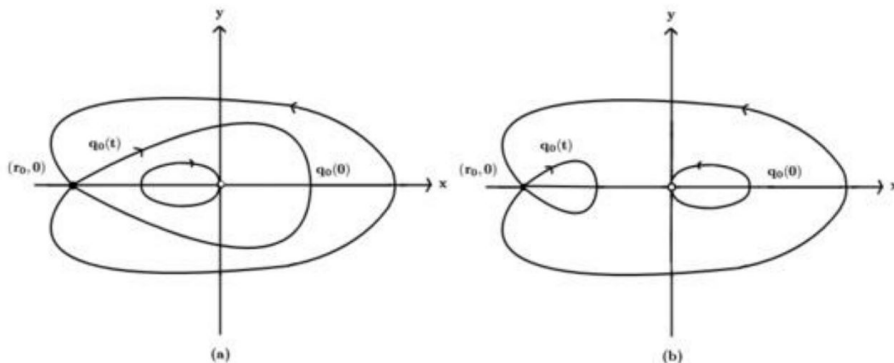
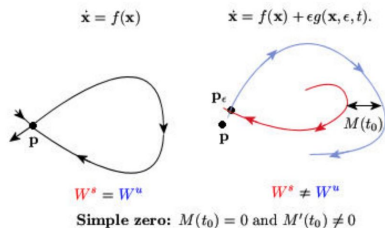
<https://github.com/CondeNast/personalization/blob/master/uplift/Uplift%20Modeling%20Simulation.ipynb>

# Melnikov Method



- To leading order, when the stable and unstable manifolds are close, the distance at any particular point can be approximated by the distance along the normal to the original orbit.
- Integrating along all points gives a measure of “distance” between the manifolds.
- This is known as the **Melnikov Method** [Melnikov, 1983].

# Method of Melnikov



Phase space shows existence of homoclinic orbits.

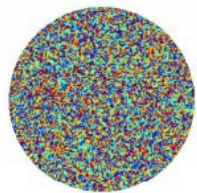
We seek to evaluate

$$M(t_0) = \int_{-\infty}^{\infty} \mathbf{J} \nabla H_{SG}^{\sigma, s}(\mathbf{q}_0(t)) \wedge \mathbf{J} \nabla \left. \frac{\partial H_{SG}^{\sigma, s}(\mathbf{q}_0(t))}{\partial s} \right|_{s_0} \cos(kt + kt_0) dt.$$

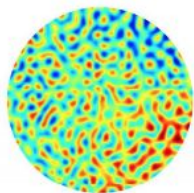
$$= - \int_{-\infty}^{\infty} \left( \frac{\partial H_{SG}^{\sigma, s}}{\partial \theta} \frac{\partial^2 H_{SG}^{\sigma, s}}{\partial s \partial r} + \frac{\partial H_{SG}^{\sigma, s}}{\partial r} \frac{\partial H_{SG}^{\sigma, s}}{\partial s \partial \theta} \right) (\mathbf{q}_0(t)) \cos(kt + kt_0) dt.$$

Thank you!

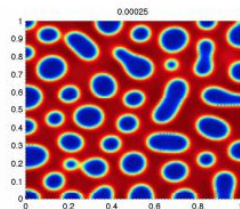
# Following Gradient Flow



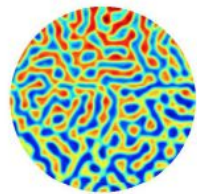
(a)  $t=0$



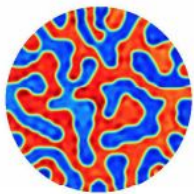
(b)  $t=5e-5$



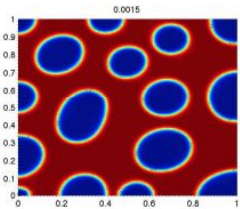
(b)  $t=2.5e-4$



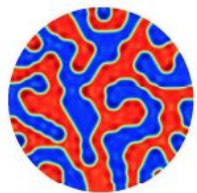
(c)  $t=1e-4$



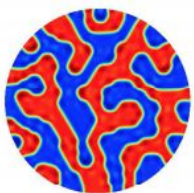
(d)  $t=1e-3$



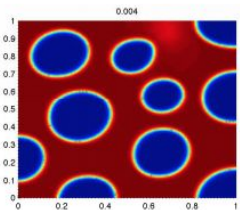
(d)  $t=1.5e-3$



(e)  $t=5e-3$



(f)  $t=1e-2$



(f)  $t=4.0e-3$

# Dual Formulation

McCann and Oberman (2004) make the *quadratic ansatz* that  $\psi$  and  $\phi$  can be written as

$$\psi(t, \mathbf{x}) = \mathbf{x}^T \cdot \Psi(t) \cdot \mathbf{x} + \psi(t) \cdot \mathbf{x} + \psi(t)$$

$$\phi(t, \mathbf{x}) = \mathbf{x}^T \cdot \Phi(t) \cdot \mathbf{x}.$$

Hoskins introduced the change of variables  $\mathbf{X} = \mathbf{x} + \nabla\phi$  called **dual variables**. Defining the *geopotential*

$$P(t, \mathbf{x}) = \frac{1}{2} \mathbf{x} \cdot \mathbf{x} + \phi,$$

note  $\mathbf{X} = \nabla P(t, \mathbf{x})$ .

Assuming  $P(t, \cdot)$  is convex we define the *Legendre transform*,

$$R(t, \mathbf{X}) := P^*(t, \mathbf{X}) = \sup_{\mathbf{y} \in \Omega} \mathbf{X} \cdot \mathbf{y} - P(t, \mathbf{y}), \quad (9)$$

so  $\mathbf{x} = \nabla R(t, \mathbf{X})$ .

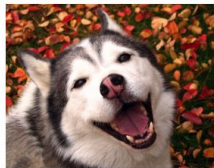
We now seek to model the SG evolution in dual variables



# Applications to AI: Image tagging



Siberian husky



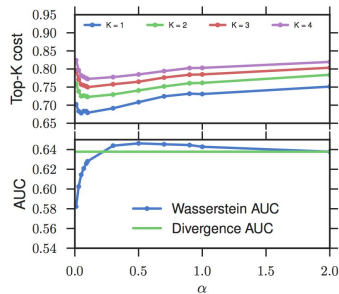
Eskimo dog



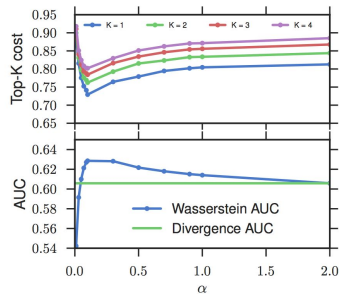
Flickr : street, parade, dragon  
Prediction : people, protest, parade



Flickr : water, boat, reflection, sun-shine  
Prediction : water, river, lake, summer;



(a) Original Flickr tags dataset.



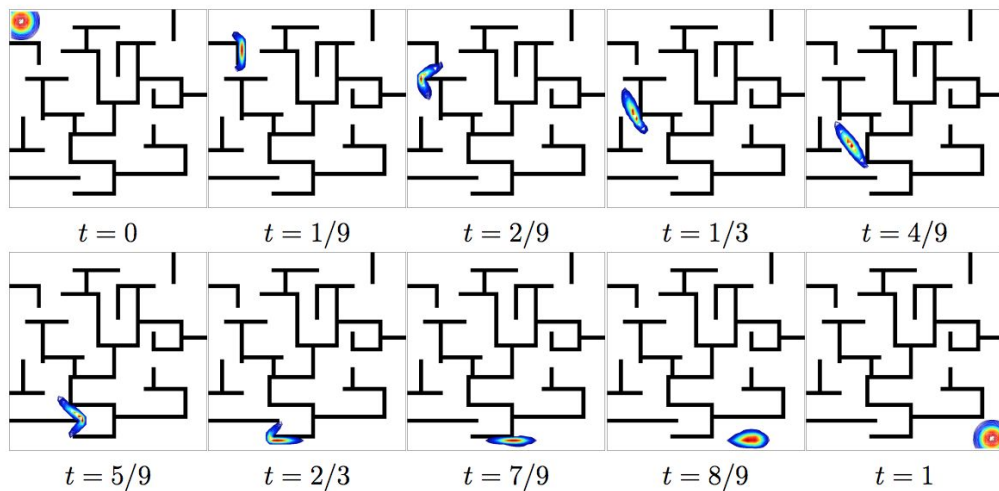
(b) Reduced-redundancy Flickr tags dataset.

$$W_p^p + \alpha \text{KL}$$

KL divergence is combined with a Wasserstein loss to improve performance

<https://papers.nips.cc/paper/5679-learning-with-a-wasserstein-loss.pdf>

# Applications to AI: OT with barriers



Uses optimal transport  
on a riemannian  
manifold.

# Applications to AI: Color transfer



(a) Source image  $I_X$



(b) Target image  $I_Y$



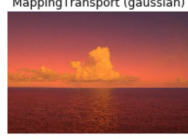
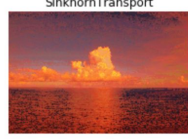
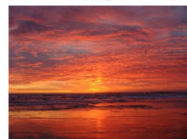
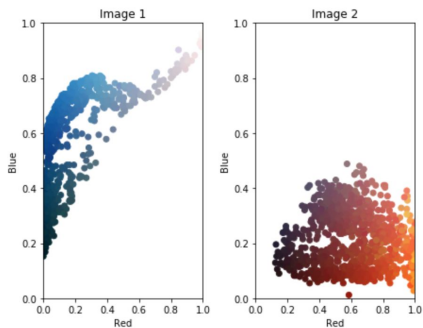
(c) Optimal Transport



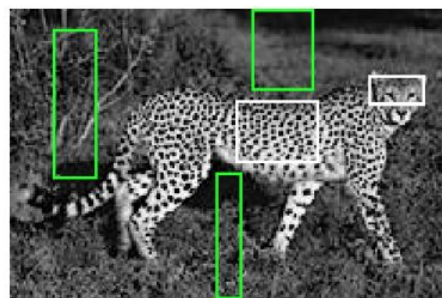
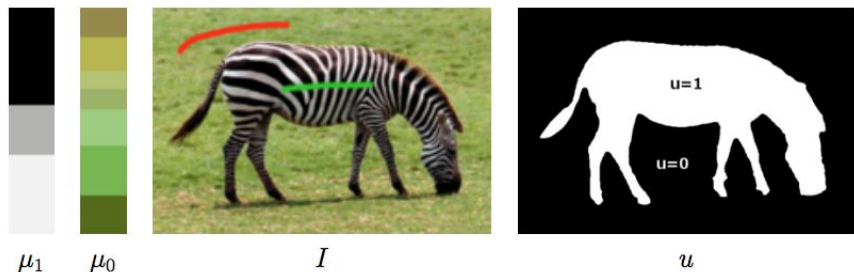
(d) Adaptive model

1. Define clusters for the colors using neighborhood metric.
2. Compute OT mapping between source and target images based on color histograms.
3. Compute color transfer.

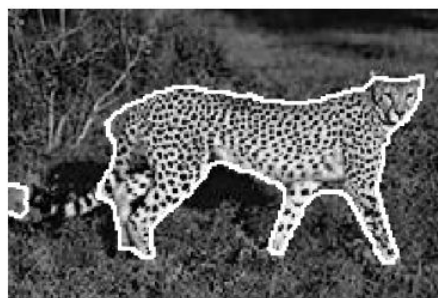
$$T\#\mu(x, U) = \sum_{i \in I_X} \mu_i \delta_{x_i} \delta_{\mathcal{T}(U_i)}$$



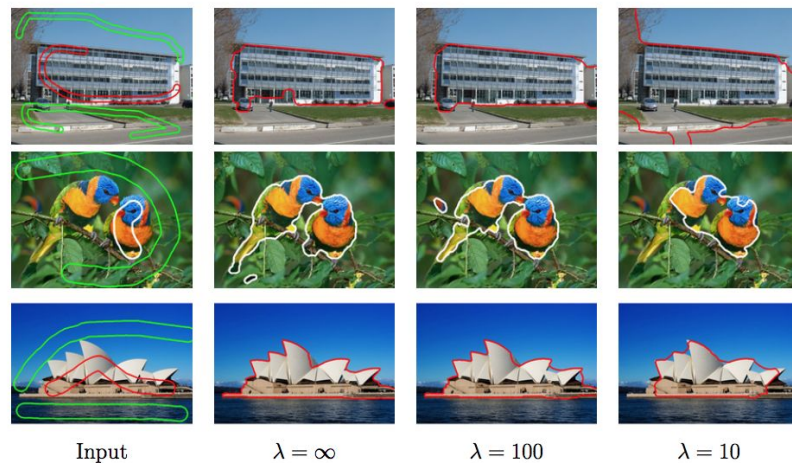
# Applications to AI: Image segmentation



(a) Initial image



(b) Segmentation



Input

$\lambda = \infty$

$\lambda = 100$

$\lambda = 10$

# Monte Carlo Estimator

## True reward

```
In [156]: h_true = 1 - stats.norm(loc=20, scale=3).cdf(30)
          h_true
```

```
Out[156]: 0.0004290603
```

## Sampling from distribution

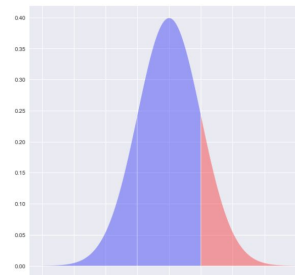
```
In [170]: n = 10000
          y = stats.norm(loc=20, scale=3).rvs(n)
          r = [1 if y_val > 30 else -1 for y_val in y]
          h_mc = 1.0/n * np.sum(y>30)
          # estimate and relative error
          h_mc, np.abs(h_mc - h_true)/h_true
```

```
Out[170]: (0.0001000000, 0.7669325448)
```

## Importance sampling

```
In [171]: n = 10000
          y = stats.expon(loc=30).rvs(n)
          h_is = 1.0/n * np.sum(stats.norm(loc=20, scale=3).pdf(y) / stats.expon(loc=30).pdf(y))
          # estimate and relative error
          h_is, np.abs(h_is - h_true)/h_true
```

```
Out[171]: (0.0004298905, 0.0019349490)
```







(a) Input image



(b)  $\lambda = 10$



(c)  $\lambda = 5$



(d)  $\lambda = 2$



(e)  $\lambda = 20$



(f)  $\lambda = 10$



(g)  $\lambda = 5$

Figure 5: Application to binary image segmentation for different data weights  $\lambda$ : The first row shows the input image, the second row shows the results for pure length-based regularization ( $a = 0$ ), and the third row shows the results for elastica-based regularization ( $a = 10$ ). Note that for smaller values of the data weight, elastica-based regularization leads to a better preservation of elongated structures.



