“Differentiable Economics”

Agenda

- **Mechanism design**
  - What is it, why do we care, what are the problems?
- Paper 1: RegretNet
- Paper 2: Differentiable Optimization
- Extensions of RegretNet
  - Strategyproof differentiable kidney exchange optimization problems

Why “differentiable economics”?

- *Differentiable programming* — term coined by Yann LeCun
    - “OK, Deep Learning has outlived its usefulness as a buzz-phrase. Deep Learning est mort. Vive Differentiable Programming! …Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques…”
    - “But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization. “
- *Differentiable economics* — term coined in a similar spirit by David Parkes
    - Combines the same basic building blocks to create *mechanisms* which are differentiable, allowing them to be optimized using gradient descent

Mechanism Design

- Mechanism design is the design of economic mechanisms — a mechanism is run by a central coordinator, who asks agents to report their preferences and then aggregates them to make some kind of resource allocation.
- Common goals of mechanism design
    - Maximize global welfare
    - Ensure some notion of fairness
    - Maximize revenue to the central agent
- Agents generally have private information about preferences (their *type*), and can choose to lie about them to the coordinator, but often the distribution over types is assumed to be public common knowledge.

Mechanism Design and Equilibria

- Common assumptions of mechanism design
  - Agents are rational utility maximizers (rational in a very strong sense)
    - Will play a (Bayes-)Nash equilibrium
- Designing a mechanism that will have a good equilibrium may be hard. Common approach: require *strategyproofness*, simplifying agent behavior; then get the best mechanism you can.

Strategyproofness

- Strategyproofness (aka incentive compatibility, truthfulness) means that agents cannot improve their utility by lying about their type.
- Two versions
  - Dominant-strategy incentive compatible (DSIC): no matter what anyone else does, you should tell the truth. (we will focus on this one)
  - Bayes-Nash incentive compatible: in expectation over possible opponent types, everybody's best choice is to tell the truth assuming everybody else does.
- If strategyproofness holds, then rational agents will tell the truth. No more worrying about figuring out equilibrium behavior!

Agenda

- Mechanism design
  - What is it, why do we care, what are the problems?
- **Paper 1: RegretNet**
- Paper 2: Differentiable Optimization
- Extensions of RegretNet
  - Strategyproof differentiable kidney exchange optimization problems

RegretNet Paper

- The ideal auction mechanism: strategyproof while *maximizing revenue*

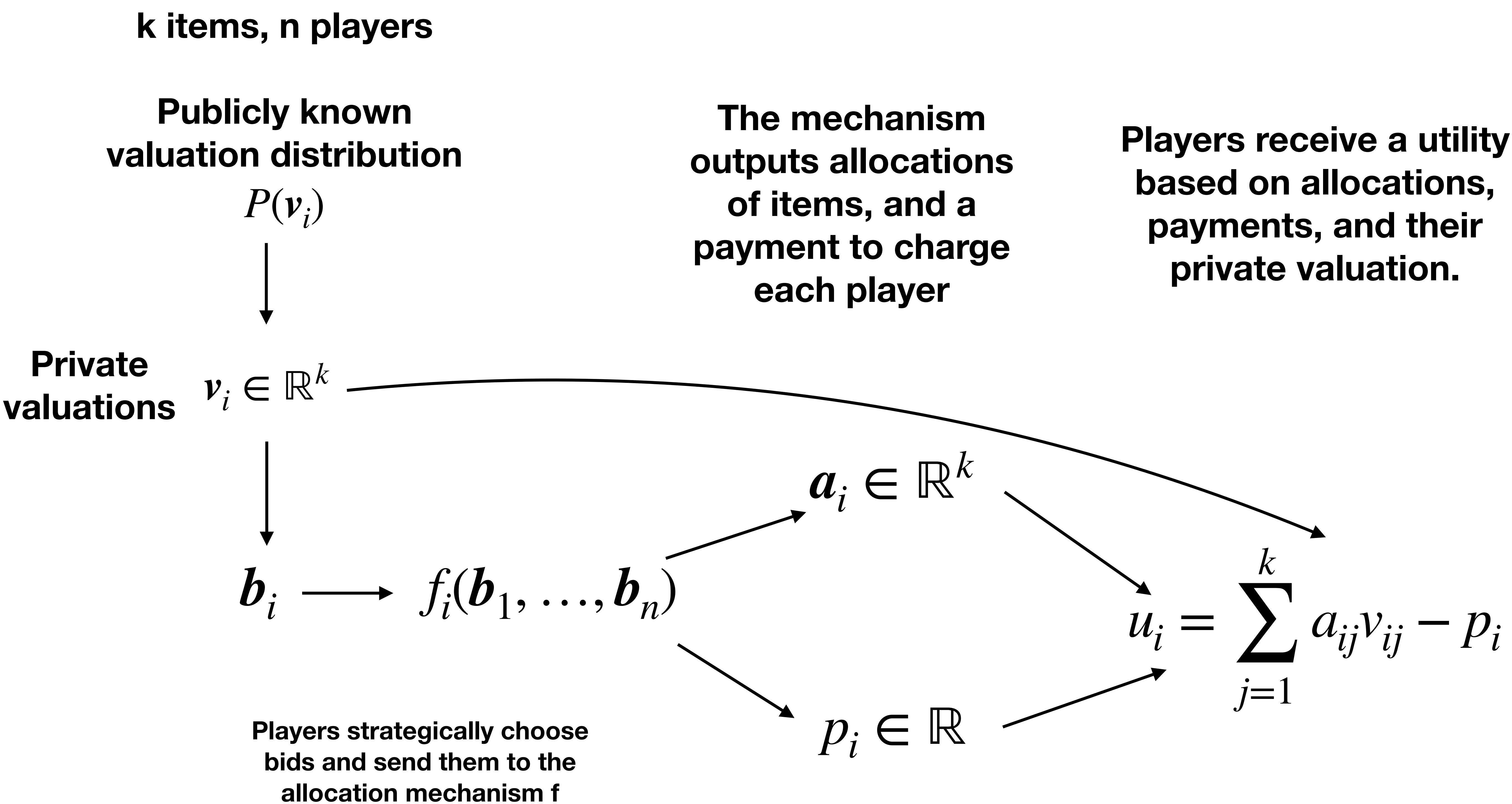
- Nobody knows how to do this except in limited cases, even though a lot of really smart people have been working hard for 30+ years

- Dütting et al, "Optimal Auctions Through Deep Learning": parameterize auction mechanism (function from bids to winners/ payments) as deep neural network:

  - Maximize revenue => have a revenue term in the loss function

  - Strategyproof => compute strategic inputs via gradient ascent, train on these to reduce how much strategyproofness is violated

Auction process



Desirable properties of auctions

- *Individual rationality (IR)*: nobody who is truthful ever pays more than their expected value for the allocation

- *Dominant-strategy incentive compatible (DSIC)*: it is always optimal to bid your true valuation, no matter what anyone else does:

$$\forall \boldsymbol{v}_{-i} : \mathbf{rgt}_i = \max_{\boldsymbol{b}_i} u_i(\boldsymbol{b}_i, \boldsymbol{v}_{-i}) - u_i(\boldsymbol{v}_i, \boldsymbol{v}_{-i}) = 0$$

- *Revenue maximization*: we want $\sum_i p_i$ to be as large as we can get away with

Network Architecture

- Network architecture of f will ensure allocations make sense, and also enforce individual rationality
- Feedforward, with output activations depending on utility structure:
  - Additive utilities: softmax to ensure $\sum_i a_{ij} \leq 1$ for all items j.
  - Unit-demand utilities: optimal allocation is one item to one agent, so take the min of row-wise and column-wise softmax to ensure $\sum_i a_{ij} \leq 1$ and $\sum_j a_{ij} \leq 1$
  - Combinatorial utilities: complicated thing with even more softmax
- Enforce IR: first compute allocation, then compute expected utility of allocation, then final payments are a fraction of expected utility.

Estimating regret

How to estimate $\max_{\boldsymbol{b}_i} u_i(\boldsymbol{b}_i, \boldsymbol{v}_{-i}) - u_i(\boldsymbol{v}_i, \boldsymbol{v}_{-i})$?

Just do gradient ascent on utility

Networks and inputs relatively small, so can do many steps (25 train time, 1000 test time)

(Easily implemented in PyTorch by just setting requires_grad=True on input tensors)

$\approx \arg\max_{\boldsymbol{b}_i} \mathbf{rgt}_i(\boldsymbol{v}_i) \longleftarrow \nabla_{\boldsymbol{b}_i} u_i(\boldsymbol{b}_i, \boldsymbol{v}_{-i})$

Learning procedure for auctions

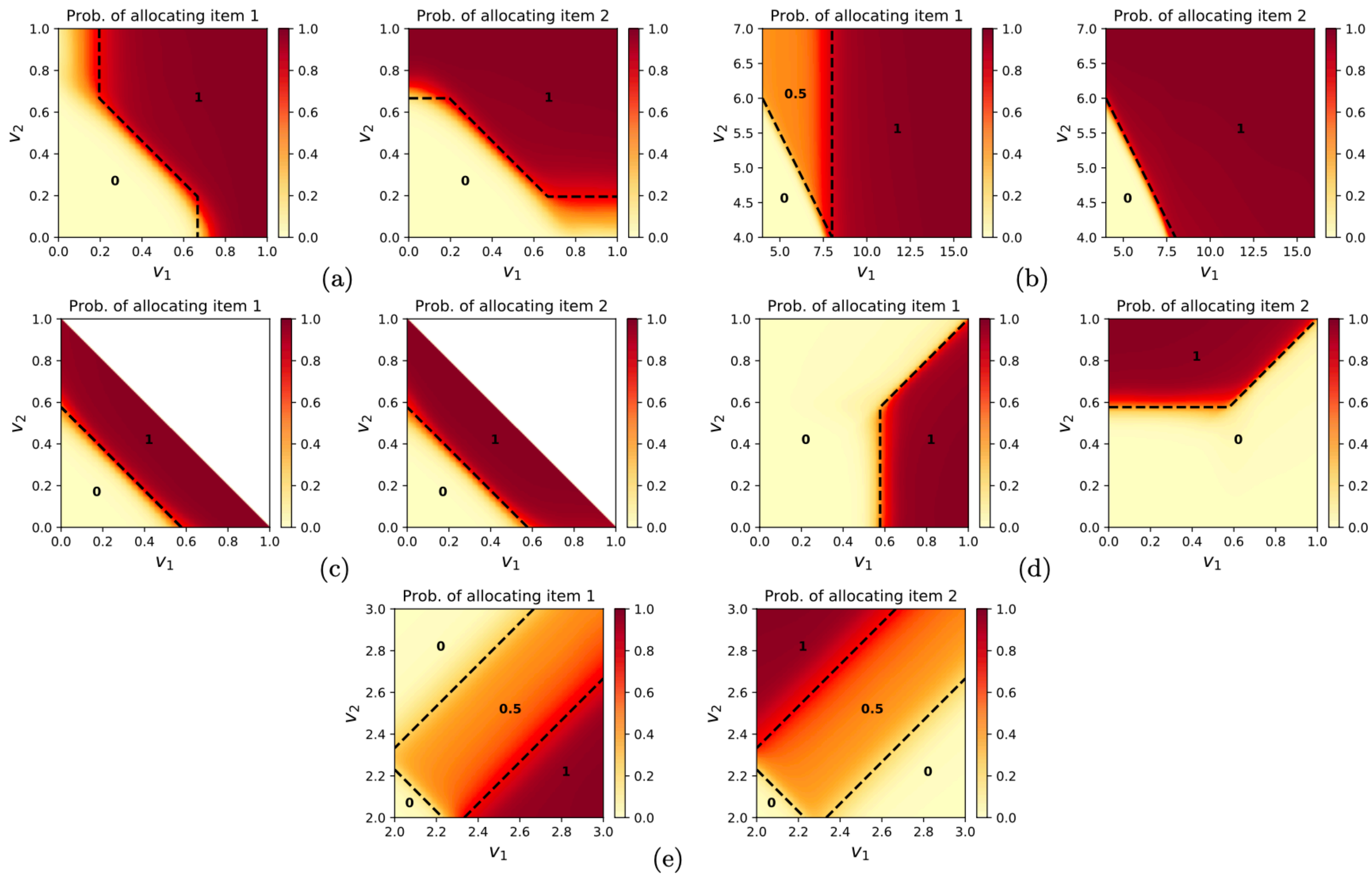
- Dataset is a large number of randomly sampled valuation profiles $\boldsymbol{v}$

- Loss on a single valuation profile:

$$L(\boldsymbol{v}, f(\boldsymbol{v})) = -\sum_i p_i + \sum_i \lambda_i \text{rgt}_i(\boldsymbol{v}) + \frac{\rho}{2}\left(\sum_i \text{rgt}_i(\boldsymbol{v})\right)^2$$

- Minimize L by minibatch SGD, just like any neural network.

Successful results



Dotted lines denote theoretically optimal mechanism; orange/red is what neural networks learned after training

Successful results

Distribution	RegretNet		VVCA	AMA_{bsym}
	rev	*rgt*	*rev*	*rev*
Setting (VI)	**0.878**	< 0.001	0.860	0.862
Setting (VII)	**2.871**	< 0.001	2.741	2.765
Setting (VIII)	**4.270**	< 0.001	4.209	3.748

Distribution	RegretNet		Item-wise Myerson	Bundled Myerson
	rev	*rgt*	*rev*	*rev*
Setting (IX)	**3.461**	< 0.003	2.495	3.457
Setting (X)	**5.541**	< 0.002	5.310	5.009
Setting (XI)	**6.778**	< 0.005	6.716	5.453

They also beat a bunch of strong baselines in more complicated situations where the optimal mechanism is not known

Distribution	Method	*rev*	*rgt*	*IR viol.*	Run-time
2 additive bidders, 3 items with $v_{ij} \sim U[0,1]$	RegretNet	1.291	< 0.001	**0**	**~9 hrs**
	LP (D: 5 bins/value)	**1.53**	0.019	0.027	69 hrs

Agenda

- Mechanism design
    - What is it, why do we care, what are the problems?
- Paper 1: RegretNet
- **Paper 2: Differentiable Optimization**
- Extensions of RegretNet
    - Strategyproof differentiable kidney exchange optimization problems

Differentiable Optimization

- Solvers for optimization problems are quite complicated, so it's somewhat surprising that you can stick an optimization problem in the middle of a neural network as a network layer.

$$z_{i-1} = \sigma(W_{i-1} z_{i-2} + b_{i-1})$$
$$z_i = \arg\min_z f(z, \theta, z_{i-1}) \text{ s.t. } z \in \mathcal{K}$$
$$z_{i+1} = \sigma(W_{i+1} z_i + b_{i+1})$$
$$\cdots$$

- Uses
  - hard constraints on network output
  - some existing layers can be reformulated as optimization (e.g. softmax)
  - meta-learning: learn features from neural network that make e.g. SVM perform well
  - Directly learning unknown parameters of optimization problems

Example Formulation

- "Differentiable optimization as a layer": with network parameters θ, optimization layer z_i outputs:

- $z_i = \arg\min_z \frac{1}{2} z^T Q(\theta) z + q(\theta)^T z$

    - subject to $A(\theta)z = b(\theta), G(\theta)z \leq h(\theta)$

- We would like to compute $\frac{dz_i}{d\theta}$ in order to backpropagate.

- This formulation is from OptNet, with additional explanation from https://arxiv.org/pdf/1804.05098.pdf. You can do all this for general convex programs, not just QPs.

Implicit function theorem (computer scientist version)

- Let $S(\theta) = \{x \,|\, g(\theta, x) = 0\}$ be a "solution map", representing the set of feasible & optimal solutions to some problem.
- If everything is "sufficiently" "nice" this will have a single value (i.e. it is *implicitly* a *function*), at which everything is differentiable, etc. and mathematicians are unable to come up with counterexamples to ruin your day.
- Then we have $D_\theta S(\theta) = -D_x g(\theta, S(\theta))^{-1} D_\theta g(\theta, S(\theta))$, where D_x, D_θ are Jacobians wrt the two inputs.
- In other words, given derivatives of g wrt variables and parameters, we can compute derivatives of the optimal point wrt parameters, evaluated at the optimal point.

KKT conditions and solution map for OptNet

- KKT conditions (primal-dual solution $s = (z^*, \nu^*, \lambda^*)$)
  - $\nabla_z \mathscr{L}(z^*, \nu^*, \lambda^*, \theta) = Qz^* + q + A^T\nu^* + G^T\lambda^* = 0$ (stationarity)
  - $Az^* - b = 0, Gz^* - h \leq 0, \lambda^* \geq 0$ (primal, dual feasibility)
  - $\text{diag}(\lambda^*)(Gz^* - h) = 0$ (complementary slackness)
- Given a feasible point (and some technical assumptions) we have that
  - $$g(s, \theta) = \begin{bmatrix} \nabla_z \mathscr{L}(z, \nu, \lambda, \theta) \\ \text{diag}(\lambda)(Gz - h) \\ Az - b \end{bmatrix} = 0$$ only at the optimal point. This defines our solution map.

Implicit function theorem on KKT conditions

- $$D_x g(s,\theta) = \begin{bmatrix} Q & G^T & A^T \\ \text{diag}(\lambda)G & \text{diag}(Gx - h) & 0 \\ A & 0 & 0 \end{bmatrix}$$

- $$D_\theta g(s,\theta) = \begin{bmatrix} dQz + D_\theta q + dG^T\lambda + dA^T\nu \\ \text{diag}(\lambda)(dGz - D_\theta h) \\ dAz - D_\theta b \end{bmatrix}$$

- Solve the system $-D_x g(s,\theta)^{-1} D_\theta g(s,\theta)$ to get derivatives; various tricks to do this more efficiently

- If Q is 0, the matrix is singular. This means our QP must actually have a quadratic term to be differentiable (can add a small “fudge factor” to differentiate LPs).

Using this for mechanism design?

- Lots of mechanism design problems have hard resource constraints. Often the welfare maximizing solution is a convex optimization problem.
- We can augment the objective to such a problem with input from a neural network, to learn to control the solution.

Example: kidney exchange

- You all know about the kidneys.
- Interesting problem in kidney exchange: hospitals may be incentivized to hide patient-donor pairs from the central mechanism and match them with each other internally. We would like a *strategyproof* mechanism for deriving matchings
- People have come up with theoretical strategyproof mechanisms in some settings. But why not try to learn them?

Differentiable optimization for kidney exchange

- Define an optimization problem
  - $$\max_x \quad w^T x - k\|f(b,\theta) - x\| \quad \text{s.t.}$$
    $$Sx \leq b$$
  - Vector b (and rows of S) is indexed by patient-donor pair type. Each column of S represents a valid matching structure; b is the reported pool of patient-donor pairs from hospitals.
  - Find maximum weight matching, biased from optimum by learned neural network
  - By rights we need integer constraints but we ignore that during training.
- We can learn $f(b,\theta)$ using a RegretNet-style training process. Maximize global welfare (not revenue) s.t. strategyproofness constraints

Questions?