Deep Learning

Day 25: Decision Focused Learning

CSCI 1470

Eric Ewing

Tuesday, 12/2/25

Case Study: Sudoku

8					5			
	7		9				4	
		9		7	8	3	2	5
3		1		9			5	
		6				1		
	9			3		6		2
2	8	3	6	5		7		
	1				2		8	
			1					9

Rules: Each number can only appear once in each row, column, and subgrid.

Fill in the missing numbers

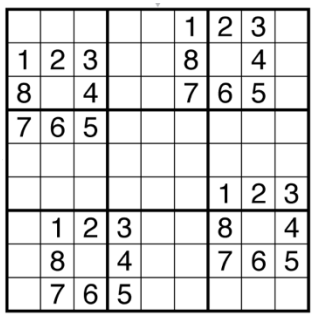
How would you train a neural network to solve this?

Reinforcement Learning

Option 1: Output potential solution, get a reward of 1 if correct, 0 if wrong

What issues might we run into while training?

This is a hard problem…
If we randomly guess solutions we are very unlikely to solve the puzzle and get any reward

f()

6	5	7	9	4	1	2	3	8
1	2	3	6	5	8	9	4	7
8	9	4	2	3	7	6	5	1
7	6	5	1	2	3	4	8	9
2	3	1	8	9	4	5	7	6
9	4	8	7	6	5	1	2	3
5	1	2	3	7	6	8	9	4
3	8	9	4	1	2	7	6	5
4	7	6	5	8	9	3	1	2

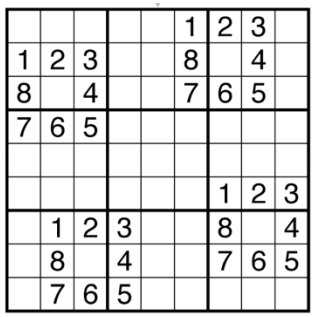
Reinforcement Learning

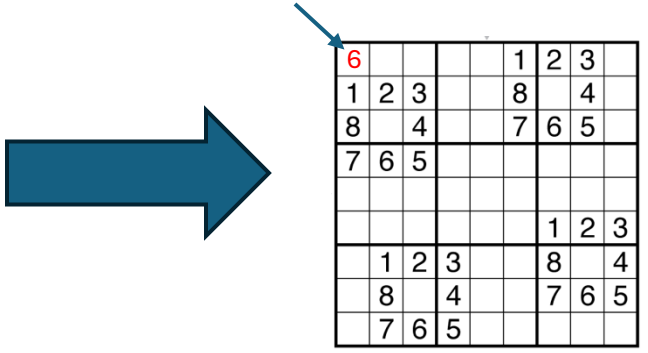
Option 2: Output next value to fill in

What should our reward function be?

+1 if valid (doesn't violate a constraint)?

Model will learn to fill in valid numbers, but maybe not to fill it in perfectly

f()



Plenty of reward shaping is possible (e.g., +100 if puzzle is solved)

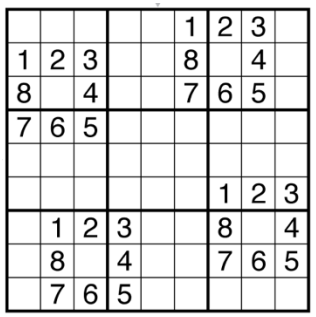
Supervised Learning

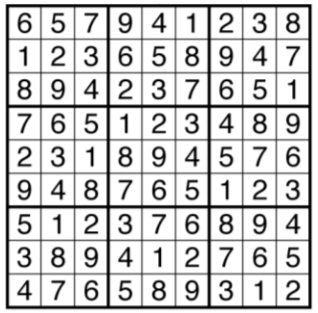
Option 3: Take a dataset of sudoku puzzles, output solutions

What's our loss function?

Option A: BCE (puzzle is correct or not)

Option B: Cross Entropy for every predicted digit

f()

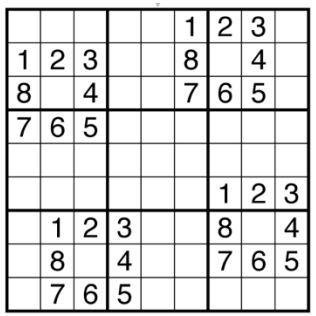


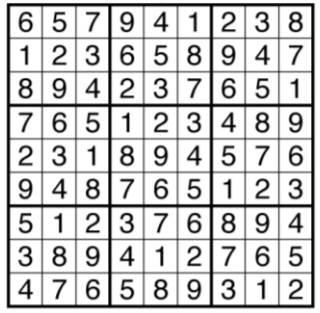
What advantages does Option A have over option B?
What about advantages that B has over A?

Supervised Learning

Most data efficient approach (of our options):

1. Predict missing digits
2. Optimize for accuracy of predicted digits using Cross Entropy

f()



But now our loss function is misaligned with our actual objective…
We are optimizing for accuracy when our primary concern is our decision making quality

Deep Learning predictions are very frequently used in decision making processes, but we aren't optimizing for decision making quality!

Decision-Focused Learning

We should train our models to make predictions that lead to **good decisions**.

This is **not** the same thing as training our models to make accurate predictions.

Decision-Focused Learning

We know Sudoku involves a constraint optimization problem.

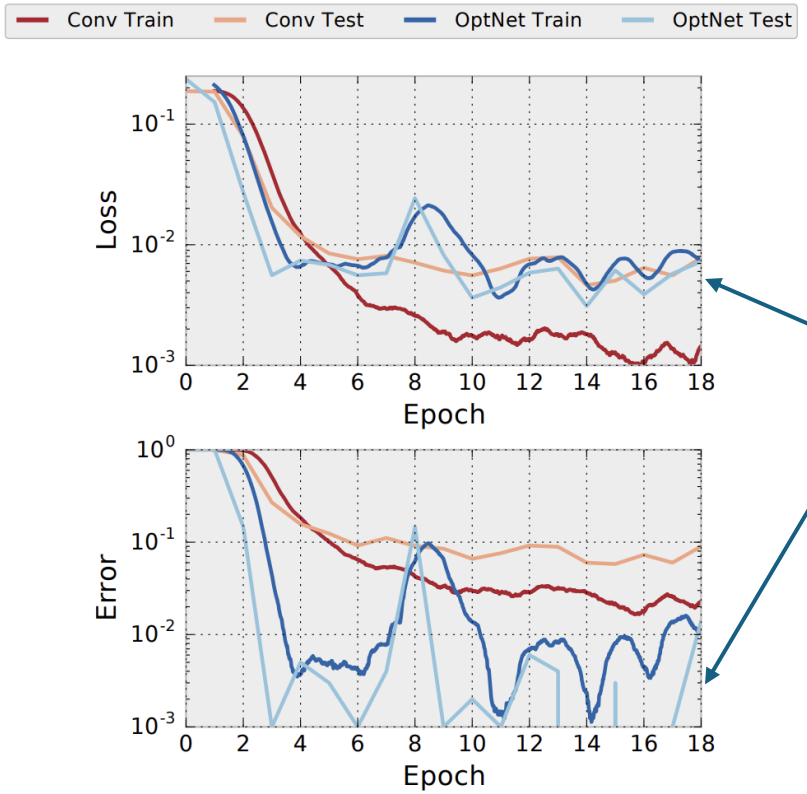
In essence, if we can learn to solve Sudoku, we've learned to solve this constraint optimization problem.

Can we give our network the knowledge that its outputs correspond to a constraint optimization problem?

OptNet: Differentiable Optimization as a Layer

Loss: MSE

Error: percentage of puzzles not solved



Loss is not perfectly correlated with number of puzzles solved!

Figure 5. Sudoku training plots.

Idea: Construct a layer that solves Quadratic Programs

OptNet Layer

Quadratic Program: Optimization problem with quadratic objective, linear constraints, and inequality constraints

Solution to Quadratic Program

Output of layer i+1

$$z_{i+1} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

Input to layer

$Q(z_i), q(z_i), A(z_i), b(z_i), G(z_i), and\ h(z_i)$ are parameters of the optimization problem

General Idea: Learn objective function parameters Q or q or learn constraints A,b,G,h

Quadratic Program

$$z_{i+1} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

Objective level sets shown with dashes

Feasible region (where constraints are satisfied) shaded dark gray

Solution x^* is point in feasible region where objective is minimized

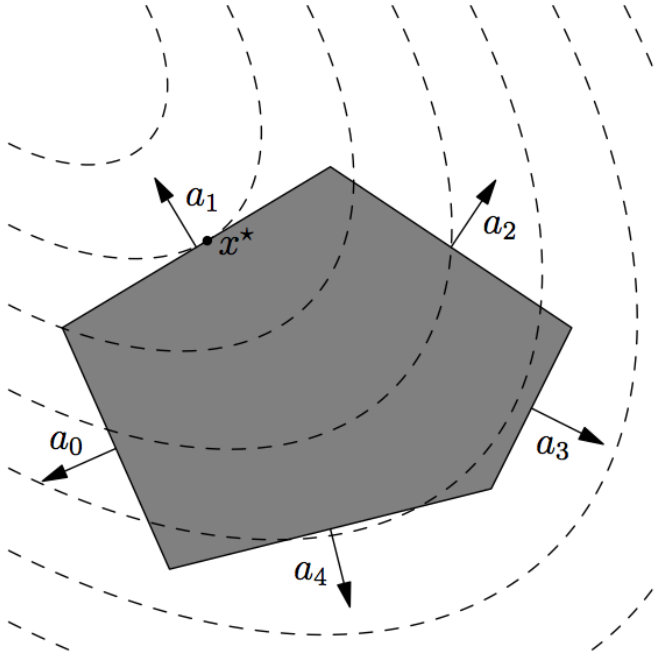


Figure 5.1: Geometric interpretation of quadratic optimization. At the optimal point $x^\star$ the hyperplane $\{x \mid a_1^T x = b\}$ is tangential to an ellipsoidal level curve.

Source: Boyd, Convex Optimization

KKT Conditions

What do we know must be true at an optimal solution?

Unconstrained Optimization:

What must the derivative be at an optimal solution?

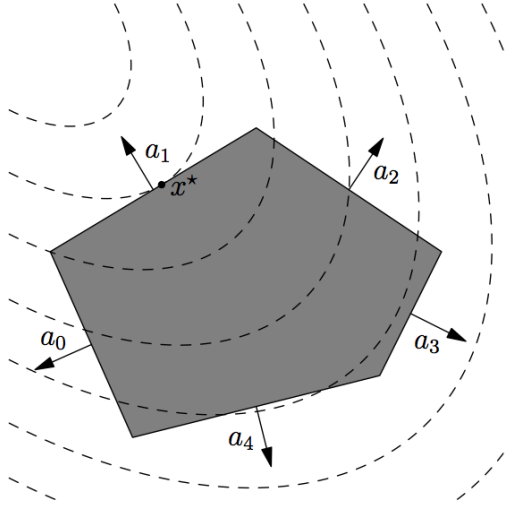
$$\nabla f(x^*) = \vec{0}$$

This is the first order optimality condition for unconstrained optimization!

Constrained Optimization:

What must the derivative be at an optimal solution?

If x^* lies on a constraint, then $\nabla f(x^*) \neq 0$



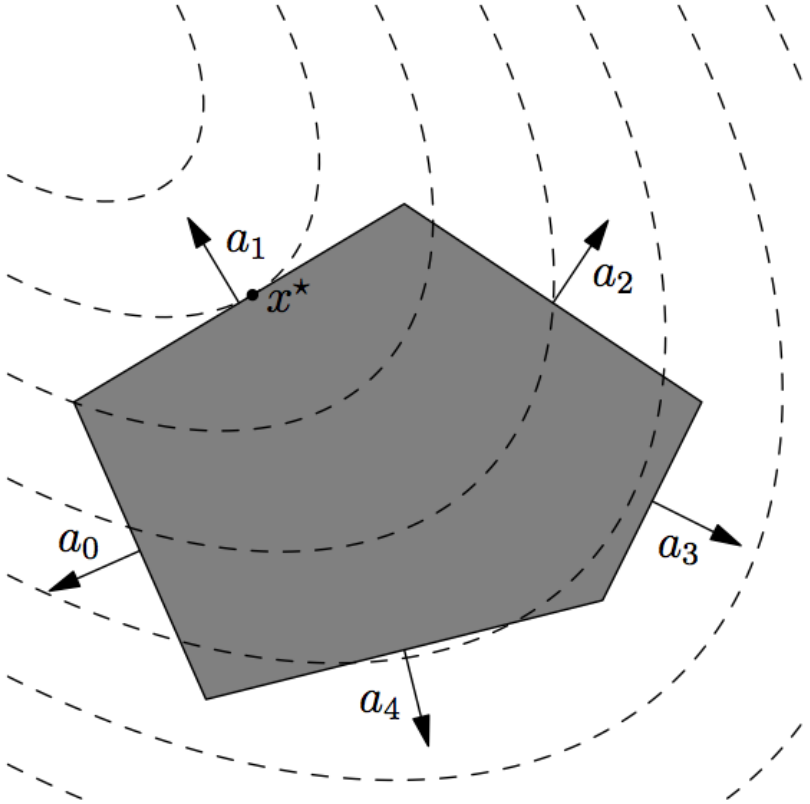
KKT Conditions

x^* is the optimal solution to the constrained problem.

We are unable to move to a better solution because of the first constraint.

Imagine the objective function is gravity pulling on our point (typically how we imagine gradient descent) and the constraint is a wall

If our objective exerts a force on our point (i.e., $\nabla f(x^*)$) how much force must the constraint wall be pushing back with?



KKT Conditions

1. Stationarity:

Amount of force applied by constraint

$$\nabla f(x^*) + \sum_j \lambda_j \nabla a_j(x^*) \quad + \sum_i \mu_i \nabla g_{i(x^*)} = \vec{0}$$

Direction Normal to constraint j

"force+direction of gravity"

Inequality constraints

Equality constraints

KKT Conditions

Necessary conditions for optimality in constrained optimization problems!

1. Stationarity:

$$\nabla f(x^*) + \sum_j \lambda_j \nabla a_j(x^*) \quad + \sum_i \mu_i \nabla g_{i(x^*)} = \vec{0}$$

2. Primal Feasibility (solution satisfies constraints):

$$Ax^* = b, g(x) \leq h$$

3. Dual Feasibility (inequality constraints push with positive force):

$$\mu_i \geq 0$$

4. Complementary Slackness

$$\sum_i \quad \mu_i g_{i(x^*)} = 0$$

If x^* is on a constraint (the constraint is actively pushing), then μ_i is non-zero. Any other time, μ_i is 0.

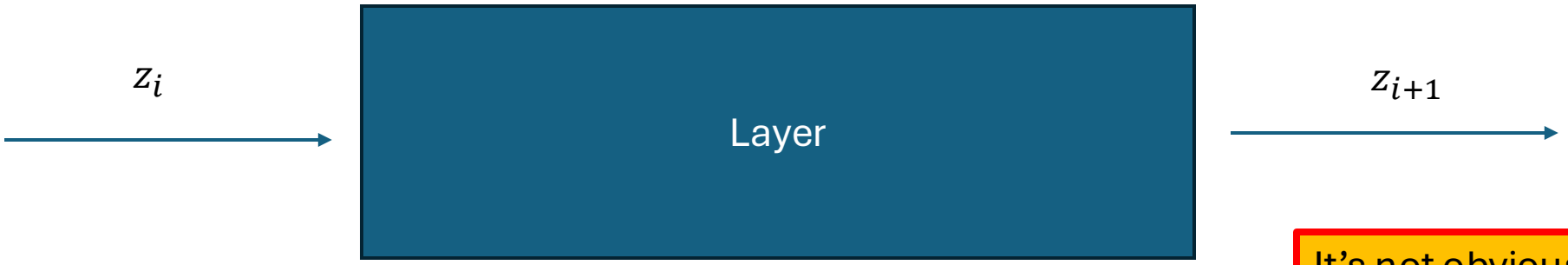
Quadratic Program Layers

$$z_{i+1} = \underset{z}{\text{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

We know how to solve QPs (at least someone else does)

What do we need to turn it into a neural network layer?

Forward Pass: solve QP



Backwards Pass: Find the gradient…

It's not obvious how to take the derivative of an argmin or a problem with constraints…

Implicit Differentiation

What is the derivative of $x^2 + y^2 = 1$, w.r.t x?

Let's treat y as an implicit function of x. How does y change as x changes?

$$\frac{d}{dx}(x^2 + y^2 = 1)$$

$$\frac{d}{dx}x^2 + \frac{d}{dx}y^2 = 0$$

$$2x + 2y\frac{dy}{dx} = 0$$

$$\frac{dy}{dx} = -\frac{x}{y}$$

Implicit differentiation lets us take derivatives of constraints!

The KKT conditions are a set of constraints!

OptNet

Forward Pass

$$z_{i+1} = \underset{z}{\text{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

Note: The backward pass is independent of how the forward pass is solved

Backward Pass

$$\frac{\partial \ell}{\partial q} = d_z \qquad \frac{\partial \ell}{\partial b} = -d_\nu$$
$$\frac{\partial \ell}{\partial h} = -D(\lambda^\star) d_\lambda \qquad \frac{\partial \ell}{\partial Q} = \frac{1}{2}(d_z z^T + z d_z^T)$$
$$\frac{\partial \ell}{\partial A} = d_\nu z^T + \nu d_z^T \qquad \frac{\partial \ell}{\partial G} = D(\lambda^\star)(d_\lambda z^T + \lambda d_z^T)$$

Sudoku Linear Program

Constraints:

$$\sum_{k=1}^{9} x(i, j, k) = 1.$$

Each cell has exactly one digit

$$z_{i+1} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z$$
$$\text{subject to} \ \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

$$\sum_{j=1}^{9} x(i, j, k) = 1.$$

Each row has exactly one of value k

Sudoku formulation only contains equality constraints, no objective function.

$$\sum_{i=1}^{9} x(i, j, k) = 1.$$

Each column has exactly one of value k

$$\sum_{i=1}^{3} \sum_{j=1}^{3} x(i+U, j+V, k) = 1, \text{ where } U, V \, \epsilon \, \{0, 3, 6\}.$$

Each 3x3 subgrid has exactly one of value k

OptNet

Benefit: Works better than pure prediction

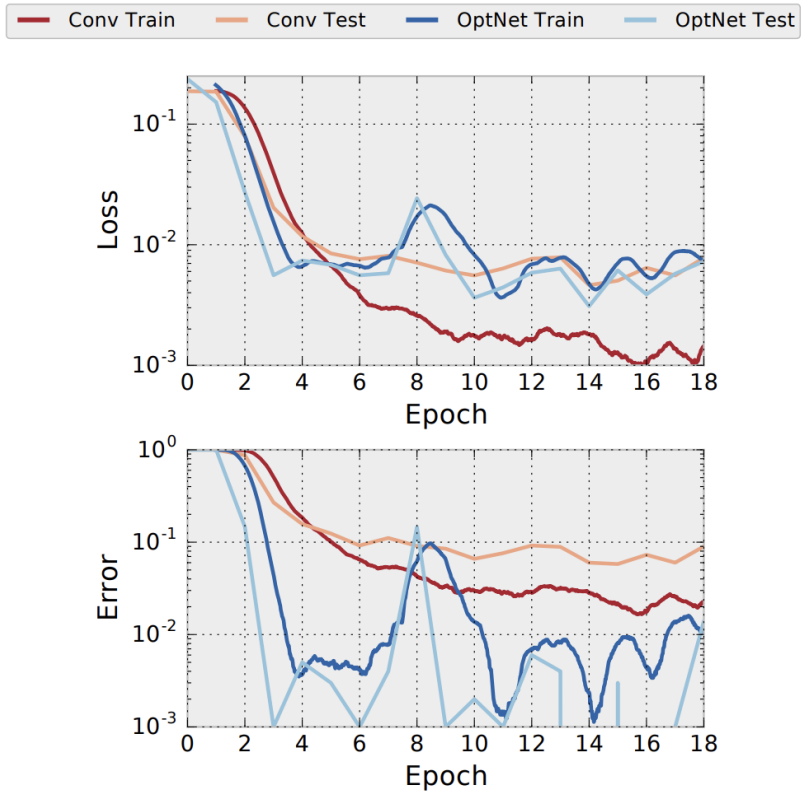
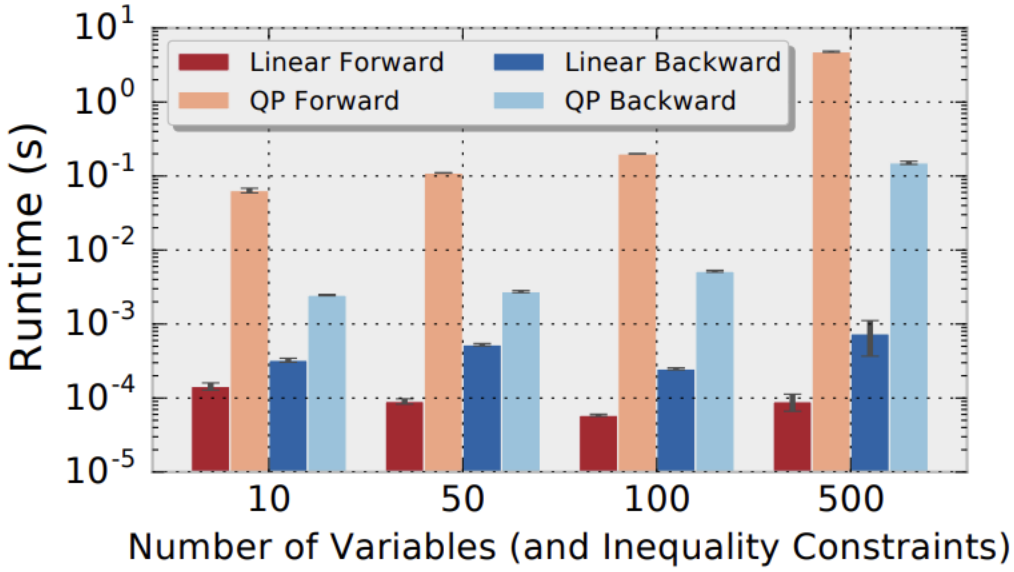


Figure 5. Sudoku training plots.

Downside: it's slower



Model	Train	Test
ConvNet	72.6%	0.04%
ConvNetMask	91.4%	15.1%
SATNet (ours)	99.8%	**98.3%**

(a) Original Sudoku.

Model	Train	Test
ConvNet	0%	0%
ConvNetMask	0.01%	0%
SATNet (ours)	99.7%	**98.3%**

(b) Permuted Sudoku.

Model	Train	Test
ConvNet	0.31%	0%
ConvNetMask	89%	0.1%
SATNet (ours)	93.6%	**63.2%**

(c) Visual Sudoku. (Note: the theoretical "best" test accuracy for our architecture is 74.7%.)

Table 1. Results for 9×9 Sudoku experiments with 9K train/1K test examples. We compare our SATNet model against a vanilla convolutional neural network (ConvNet) as well as one that receives a binary mask indicating which bits need to be learned (ConvNetMask).

SATNet: Differentiable Satisfiability Solver

Similar approach, but with a different problem formulation and optimization procedure.

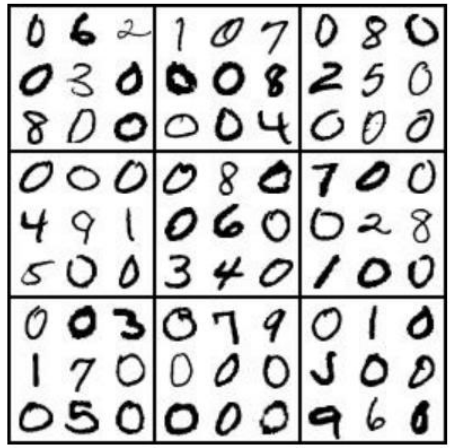


Figure 3. An example visual Sudoku image input, i.e. an image of a Sudoku board constructed with MNIST digits. Cells filled with the numbers 1-9 are fixed, and zeros represent unknowns.

Decision-Focused Learning

There are many ways to add differentiable components to a neural network for specific problems

Solving quadratic programs is just one such component

Deep Learning trend #1:

Bigger, more general networks with more data do better

Deep Learning trend #2:

We can add very specific components to networks to make them more data efficient

End-to-end Learning in Stackelberg Security Games

The Plan

1. Background Info and problem set up 2. Equations 3. Small examples with numbers 4. A very important table 5. Results

Backgroun
d

- Stackleberg Game: One player goes first, second player observes the first players (mixed) action and responds.
- Quantal Response: Player responses are not fully rational, but are still functions of utilities (e.g. Logit QR is a Softmax of utilities for actions)

Maximum Likelihood Estimation

Suppose you have some events X that are sample from a probability distribution P(θ), where θ is an unknown parameter. MLE is the problem of finding the θ that fits the known samples b est.

Example: You flip an unfair (but with unknown probabilities) coin 10 times and it comes up heads 8 times.

Example: You pick 4 balls from a bag with an unknown number of balls in it. Three of the balls are red and white is black. What is

the most likely ratio of red to white balls?

Maximum Likelihood Estimation

Often easiest to minimize Negative Log Likelihood (NLL). Why?

Likelihood often has many probabilities multiplied together, taking the log separates them. Logs also help with numerical stability for small probability events. Sometimes you can find algebraic and exact answers, other times not. We primarily use gradient descent because it had better numerical stability than the exact method.

Problem Set Up

Stackelberg Security Games (SSGs)

Stackelberg Security Game

SSGs are played between a defender and an attacker. The defender allocates k resources to defend T targets. The attacker observes the allocation and attacks a target.

- Motivating example: anti-poaching
- Defender = Park Rangers
- Attacker = Poachers

Problem Setup

- Each target is an area in the park Each target has known utility
- values for the defender The utility values for the attacker are
- unknown to the defender, but known to be based on observable features for each target (e.g. distance from nearest road, animals, proximity to ranger station, etc.)
- The rangers can’t observe all attacks the adversaries make/attackers make infrequent attacks. (There is some amount of noise in observations)

General Sum

- Most poaching activity is along the border of the national park and targets boars for food
- An attack against a boar is much less concerning for Rangers than an attack against a rhino
- Defender and Attacker utilities are not necessarily tied together and is therefore modeled as a general sum (non-zero sum) game.

The Game

- Given a set of targets T, the defender is trying to maximize their expected utility.
- The defender can allocate a resource (or a fraction of a resource) to a target to defend that target.
- The attacker observes the allocation and makes an attack on a target.
- If the attack target is covered by the defender the attacker receives a negative utility, if the attack is not covered, the defender receives the negative utility.
- Defender Expected Utility (DEU): is the probability a target is covered times the probability it is attacked times the utility of that target.

Definition

Variable	Usage The defender's allocated resources to target t
x_t	(proba- bility of covering t) The utility gained for the defender for target t if it is
U_{dt}^{c}	covered and attacked The utility gained* for the defender for target t if is
U_{dt}^{u}	uncovered and attacked. (negative) The attacker's probability of attacking target t
q_t	The features of target t
y_t	The attacker preference function. Maps from features
$\varphi(y_t)$	to attacker utility for target t.

The Defender's Optimization Problem

DEU: Defender's Expected Utility

$$\max_{x} \quad DEU = \sum_{t \in T} x_t q_t^c + (1 - x_t) q_t^u \qquad (1)$$

$$s.t. \quad 0 \le x_t \le 1 \qquad (2)$$

$$\sum_{t \in T} x_t \le k \qquad (3)$$

Subjective Utility Quantal Response: SUQR

Humans are only sort of rational… SUQR is one way of quantifying this.

qt is a softmax of an attractiveness function: wxt + φ(yi),
where w is how strongly the attacker takes into account the

coverage.

$$q_t = \frac{e^{w x_t \varphi(y_i)}}{\sum_{j \in T} e^{w x_j + \varphi(y_j)}}$$

Why SUQR and not QR

Why use wxt + φ(yt) as an attractivness function rather than wxtφ‘(yt) (where φ‘(.) is the attacker’s utility)?

- In general: Better fit to data collected in security games.

  Human’s don’t actually compute explicit utilities for their

  actions and our behavior may more closely match a SUQR model than traditional Logit QR models.

- In this paper: Need to separate decision variable x from φ(.).

  The effects of a change in coverage need to be decomposable from the inherent attractiveness of a target.

Response to w

w is how much the adversary responds to coverage and should be non-p ositive

- w = 0: The attacker does not consider coverage
- w > 0: The attacker is more likely to attack a covered target(???)
- w = –∞ The attacker selects the least covered target

w can be computed with MLE if the attacker makes attacks under two different coverage schemes. w = –4 has been found empirically before for some experiments, so we generally use that value.

The Learning

Learning Attacker Preferences

If we knew $\varphi(\cdot)$ exactly, the problem would be easier. You can use MLE to compute $\tilde{\varphi}(y)$ exactly for the targets you observe attacks on. You could use your favorite non-convex optimizer to solve for

x^*.

The Problem: We may only be able to observe attacks on some targets or we want to transfer our methods to new parks/airports with new targets.

Idea: Two-Stage Approach Fit a Neural Network to predict $\varphi(\cdot)$. Optimize for DEU given your predictions of $\varphi(\cdot)$

given y_t (which is basically equivalent to learning $\varphi($ $_{gt}$

Two-Stage Approach

What's wrong with this predict-then-optimize approach? Our loss function is optimizing for accuracy of prediction, not the end objective value of DEU.

We have bounds for DEU of a target for some given error in prediction $\varepsilon 2$ in Logit QR in general sum and zero sum games.

Key Idea: the defender needs to be pessimistic on predictions for targets that are high value to the defender.

Decision Focused Learning

Better Idea: Decision Focused Learning

We shouldn’t be agnostic to the fact that our learning has an optimization after it. We want to optimize for decision quality, not accuracy in predicting attack probabilities.

The Plan: Learn $\varphi(\cdot)$ that optimizes for solution quality by differentiating through the optimization problem as well and using decision quality as a loss rather than prediction accuracy.

Numbers

Set up

Assume a game with 2 targets, with utilities $U_d^1 = -1, U_d^2 = -2$ and one defender resource k = 1

The defender seeks to optimize DEU:

$$\max \quad -q1(1 - x1) - 2q2(1 - x2)$$
$$\text{s.t.} \quad x1+x2=1$$

If the attacker has uniform preference ($\varphi(y1) = \varphi(y2)$), then the optimal defender coverage is

$$x* = (.244, .755)$$

The resulting attack probabilities would be q = (.885, 115).

But what if not equal preferences?

Say the defender observes a number of attacks with their coverage and observes $\tilde{q}\tilde{} = (.5, .5)$. The attack distribution was different than what the defender assumed (i.e. attacker has non-uniform preferences).

The defender can use MLE to estimate $\tilde{\varphi}(y) = (0, 2)$.

But how can the defender use this attack data to generalize to other targets with different sets of features? Let's fit a neural network to $\tilde{\tilde{\varphi}}(y)$

More Numerical Examples

What happens if we don't fit the true attacker preferences correctly?

Description Observed Preferences	$\tilde{\varphi}(y)$	x^*	q	Cross Entropy of q	$L = DE$
Overestimating Valuable Target	(0, 2) (0,	(.25 , 75)	(. , 5)	.693	0.625
Underestimating Valuable Target	3) (0, 1)	(.18 , 82)	(64 , 36)	.957	.650
Overestimating at Same Cross Entropy	(0, 2.5)	(.31 ; 69)	(38 , 62)	.723	.647
		(.19 , 81)	(62 , 38)	.723	.645

Decision-focused learning

- If we want to include the optimization problem in our gradients, we need to differentiate through a non-convex optimization problem.
- Want to know how the decision quality (L = DEU) changes
- with changes in our model parameters θ. Follow the gradient ∂L $\partial theta$ to train our model.

One big Chain Rule

Our overall gradient is made up of a few different components:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial x^*} \frac{\partial x^*}{\partial q} \frac{\partial q}{\partial \theta}$$

Which one of these terms seems difficult to compute?

Gradients of non-convex optimization

Theorem: Let x be a strict local minimizer of f over X. Then, except on a measure zero set, there exists a convex set I around x such that $x^* = I(\theta) = \arg\min_{x \in I \cap X} f(x, \theta)$ is differentiable. The gradients of x^* with respect to θ are given by the gradients of the solutions to the local quadratic approximation

$\min_{x \in X} 1/2 x^T \nabla^2 f(x, \theta) x + x^T \nabla f(x, \theta)$

(OPTNet Paper contains differentiable QP solver [Amos & Kolter 2017])

Does it matter?

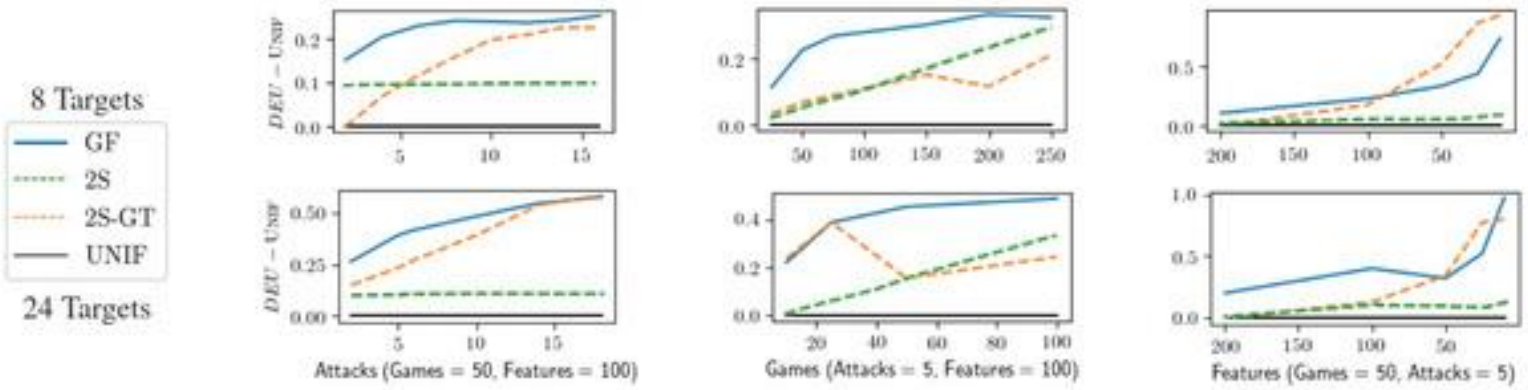


Figure 1: Game focused Learning vs 2 Stage learning on synthetic data

Human Data

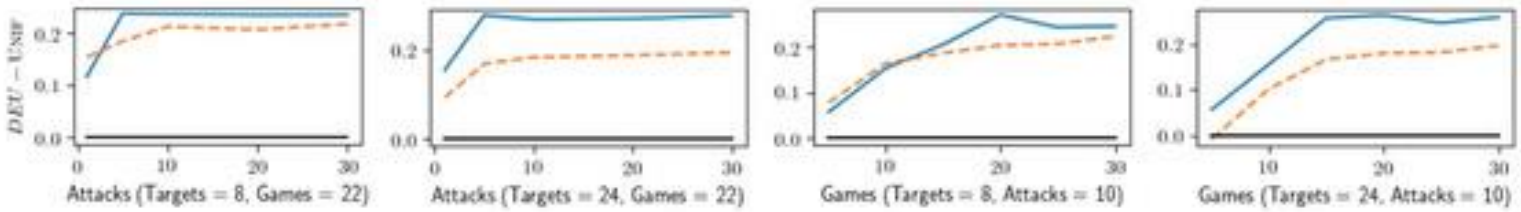


Figure 2: Game focused Learning vs 2 Stage learning on human data

Comparison of Prediction error and performance

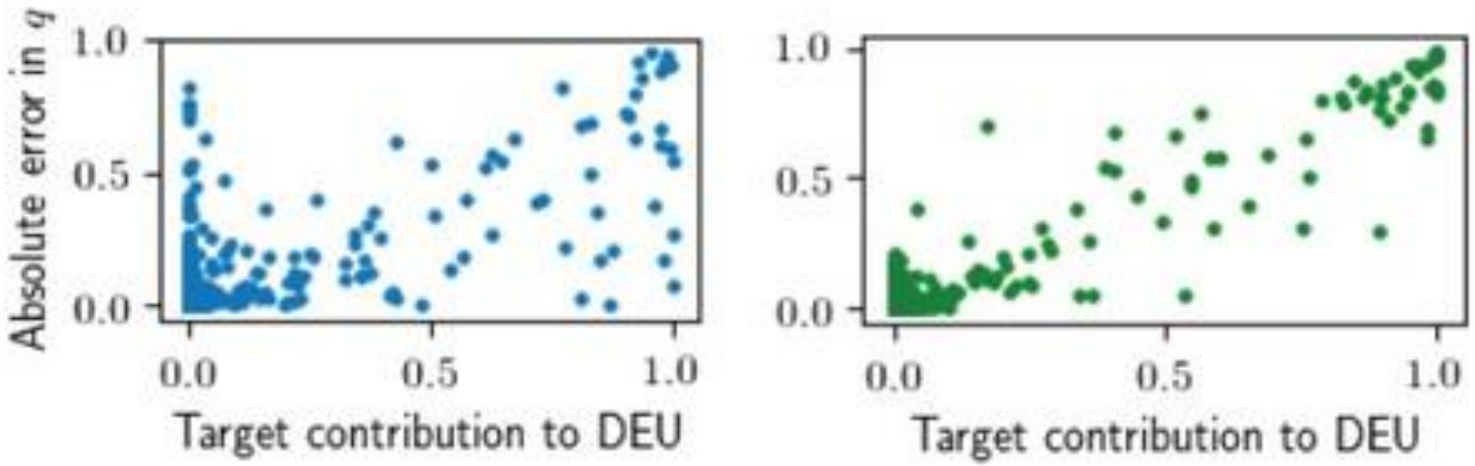


Figure 6: Target contribution to DEU vs. the absolute error in the predicted attacker q. GF (left) has lower estimation errors for targets with high DEU contributions and higher errors for targets with low DEU contributions. 2S (right) estimation errors do not vary with target importance.