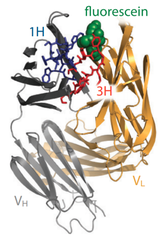
Learning to Make Decisions via Submodular Regularization

Ayya Alieva[3], Aiden Aceves[1], Jialin Song[1], Stephen Mayo[1], Yisong Yue[1], Yuxin Chen[2]

[1]Caltech, [2]University of Chicago, [3]Stanford University

Motivation

- Active learning, experimental design and protein engineering are instances of **combinatorial optimization**.
- These problems are usually NP-hard.
- Greedy could be intractable and/or highly suboptimal.



Our Approach

- ML framework that learns a greedy heuristic based on the **submodular-norm loss**.
- The learned objective is permutation-invariant and approximately submodular with **strong approximation guarantees**.
- The model is easily integrated with ML pipelines.

Submodular Surrogates

Given: ground set $\mathcal{V}$; *value* function $u : 2^{\mathcal{V}} \to \mathbb{R}$; budget k.
Goal: Policy $\pi : 2^{\mathcal{V}} \to \mathcal{V}$ that maximizes $u(S_{\pi,k})$ by iteratively adding to the set:
$S_{\pi,i} = S_{\pi,i-1} \cup \{\pi(S_{\pi,i-1})\}$.
Approach: Design a surragate objective function $f : 2^{\mathcal{V}} \to \mathbb{R}$ which is

- **Monotone**: $\forall\ A \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus A$, $f(A) \leq f(A \cup \{e\})$.
- **Submodular**: $\forall\ A \subseteq B \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus B$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$.
- **Approximately** equal to oracle u.

If we had a "good" **submodular surrogate** f:

- Let $g(A, e) = f(A \cup \{e\}) - f(A)$ for $A \subset \mathcal{V}$ and $e \in \mathcal{V}$.
- $g(A, e) \approx g^{\exp}(A, e) = u(A \cup \{e\}) - u(A)$
- $\pi(S) = \arg\max_{e \in \mathcal{V}} g(S, e)$ is **near-optimal**.

Problem: Hand-engineering f is hard; evaluating $g^{\exp}$ is expensive.

Learning with Submodular Regularization (LeaSuRe)

We train NN $g : 2^{\mathcal{V}} \times \mathcal{V} \to \mathbb{R}$ to approximate $g^{\exp}$:

- DAgger [5] collects **real labelled** dataset $D_{real} = \{(\langle A, x\rangle, g^{\exp}(A, x))\}$.
- For $\langle A, x\rangle \in D_{real}$ we generate random *superset* A' and form an **unlabelled** *synthetic* dataset $D_{synth} = \{(\langle A, x\rangle, \langle A', x\rangle) | A \preceq A', \langle A, x\rangle \in D_{real}\}$
- We update g using **submodular-norm** loss:

$$\begin{aligned}\text{Loss}(g, g^{\exp}) = &\sum_{\langle A,x\rangle \in D_{real}} (g^{\exp}(A, x) - g(A, x))^2 \\ &+ \lambda \sum_{(\langle A,x\rangle, \langle A',x\rangle) \in D_{synth}} \sigma([g(A', x) - g(A, x)]) \\ &+ \gamma \sum_{\langle A',x\rangle \in D_{synth}} \text{ReLu}(-g(A', x)),\end{aligned}$$

LeaSuRe encourages g to model the gain of a **monotone**, **submodular** function approximating the oracle u.

Architecture

Learning element representation $e \in A \cup \{e_{n+1}\} \subset \mathcal{V}$

Element embedding $\bar{e}$

Learning set representation $\{\bar{e}_0, \ldots, \bar{e}_n\}$

Set embedding $\bar{A}$

Learning the score function $(\bar{A}, \bar{e}_{n+1})$

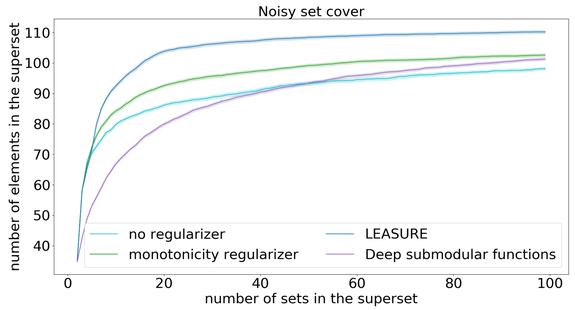
$g(A, e)$

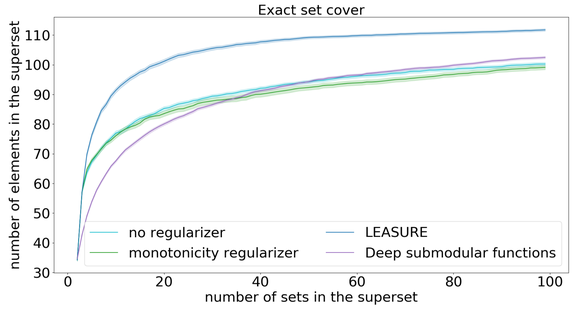
Allows for *modular* and *end-to-end* training:

- Protein Engineering used pretrained element embeddings
- Set Cover and Learning Active Learning trained all networks simultaneously

Protein Engineering used UniRep [2, 1]; Set Cover and Learning Active Learning used a two layer DNN.

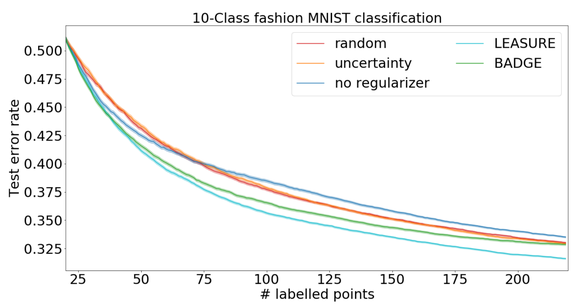
Set Cover





Baselines: "no regularizer" - MSE loss; "monotonicity regularizer" - MSE + ReLu loss; Deep Submodular Functions [4].

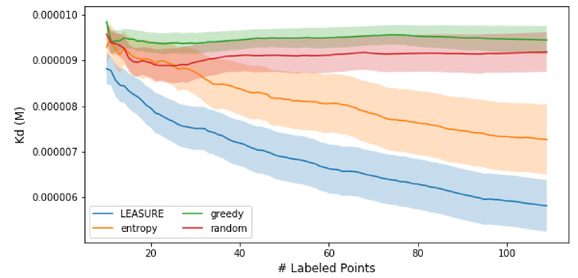
Learning active learning on Fashion MNIST



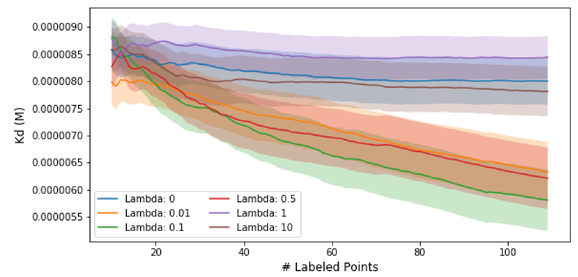
Baselines: "random" - random sampling; "uncertainty" - uncertainty sampling; "no regularizer" - DAgger + MSE loss; BADGE [3].

Protein Engineering

Comparison to baseline methods



Effect of scaling parameter λ for LeaSuRe



Given: A large set of unlabelled proteins.
Goal: Determine a subset of high-potency (low K_D) proteins.
Problem: The protein set is too large to test the potency of all elements.
Baselines: "random" - random sampling; "greedy" - hand-engineered surrogate g; "entropy" - a version of uncertainty sampling.

References

[1] Aceves, A. (2021). Our code. `https://gitlab.com/ajaceves/alilpe`.

[2] Alley, E. C., Khimulya, G., Biswas, S., AlQuraishi, M., and Church, G. M. (2019).

[3] Ash, J. T., Zhang, C., Krishnamurthy, A., Langford, J., and Agarwal, A. (2020).

[4] Dolhansky, B. W. and Bilmes, J. A. (2016).

[5] Ross, S., Gordon, G., and Bagnell, D. (2011).