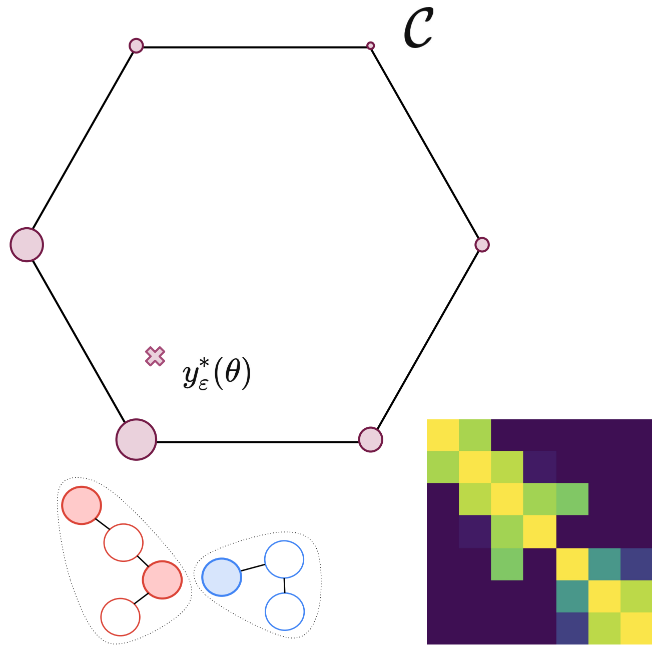
Perturbed optimizers for learning



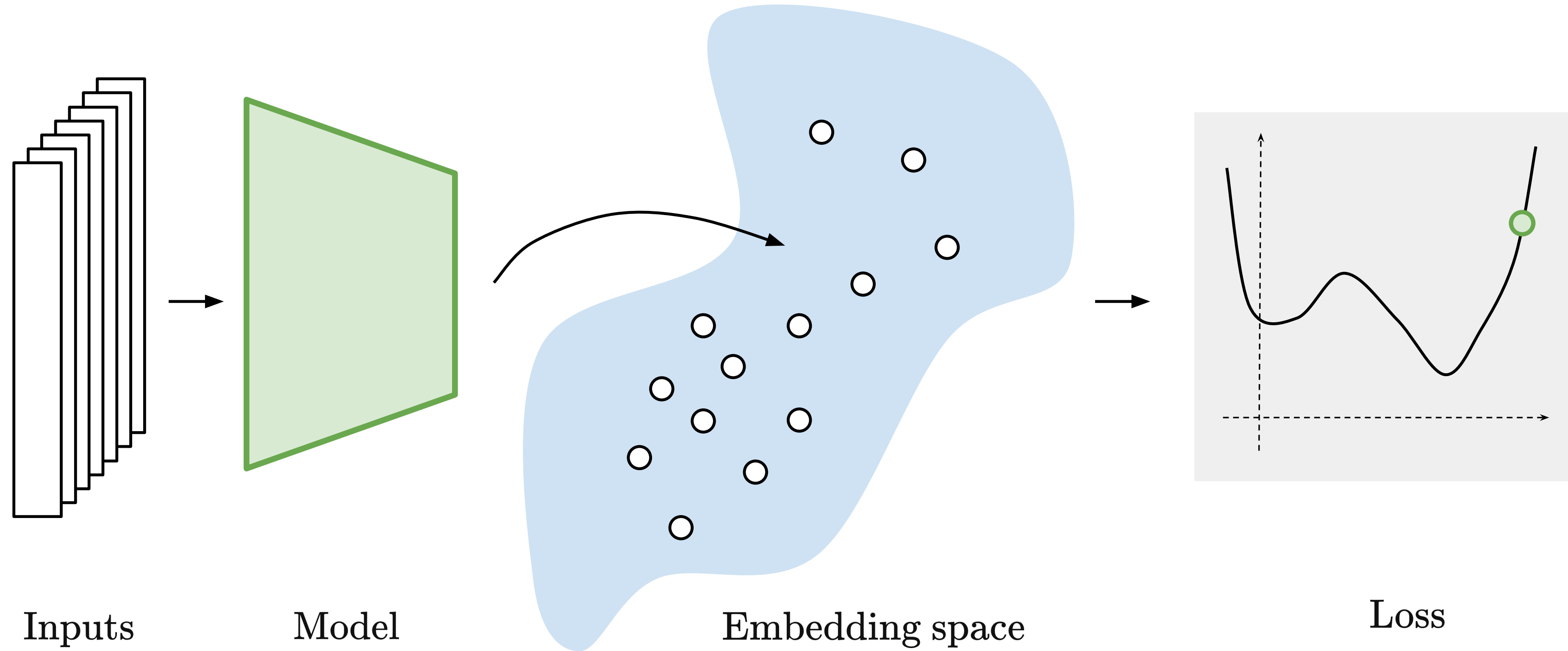
Q. Berthet
(Google DeepMind)

“Differentiable Almost Everything” workshop

ICML 2023, Honolulu, Hawaii 🌺

End-to-end differentiable models

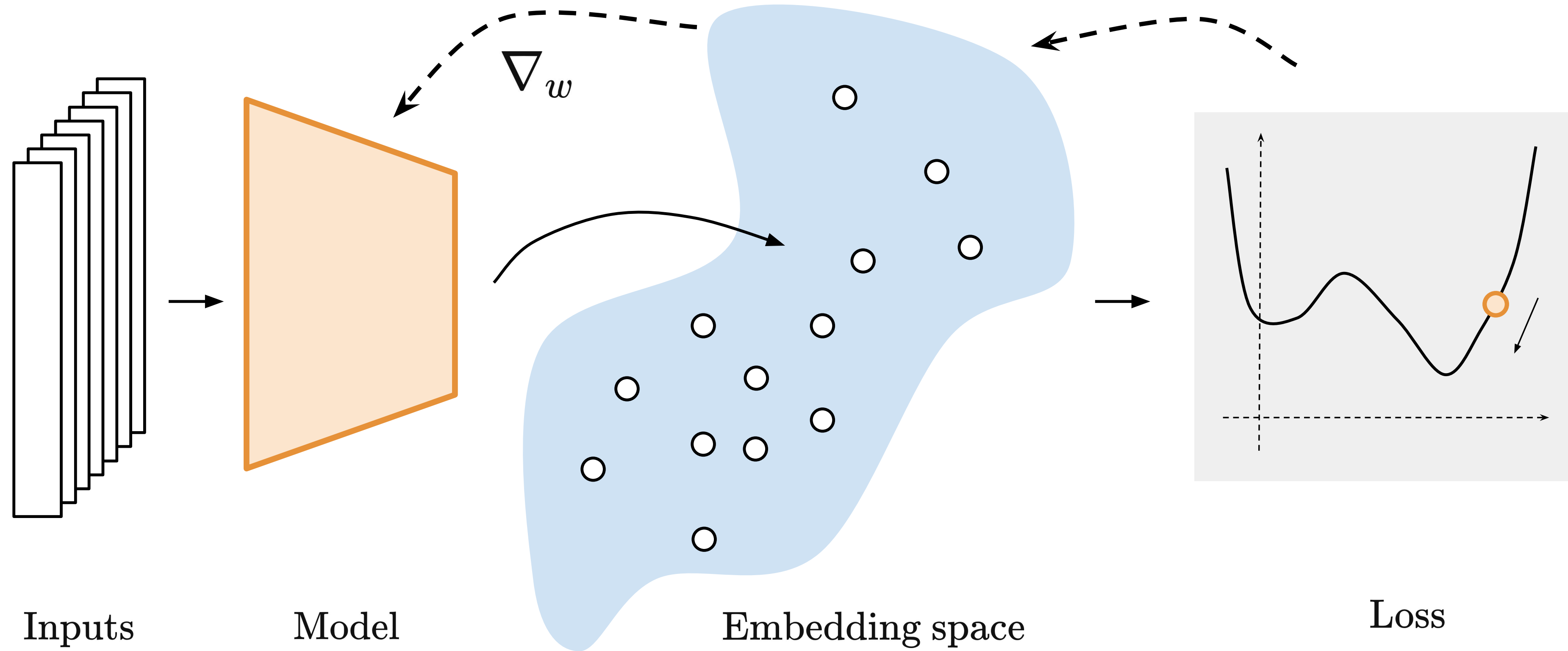
ML training: minimizing a loss w.r.t. weights of a model



- Composition of complex, explicit, analytical operations with **weights**.
- Loss minimization with first-order, **gradient-based** methods.
- Modern, large-scale models and datasets: **automatic differentiation**.

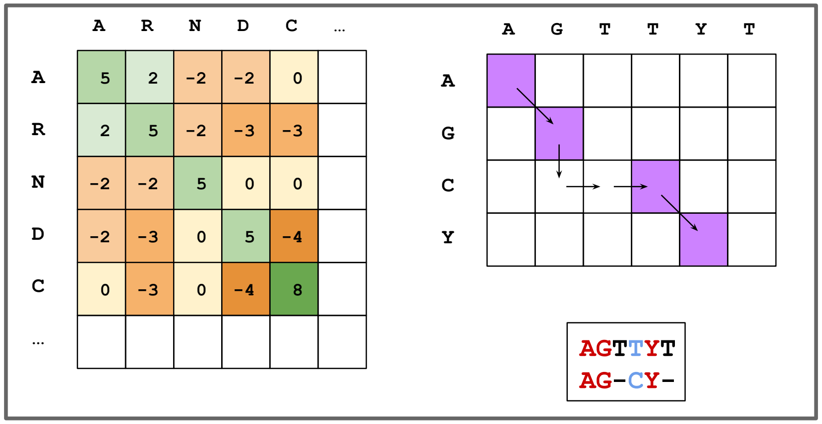
End-to-end differentiable models

ML training: minimizing a loss w.r.t. weights of a model

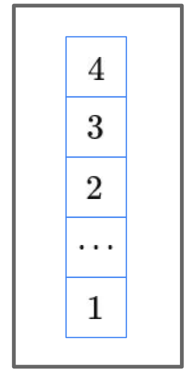


- Composition of complex, explicit, analytical operations with **weights**.
- Loss minimization with first-order, **gradient-based** methods.
- Modern, large-scale models and datasets: **automatic differentiation**.

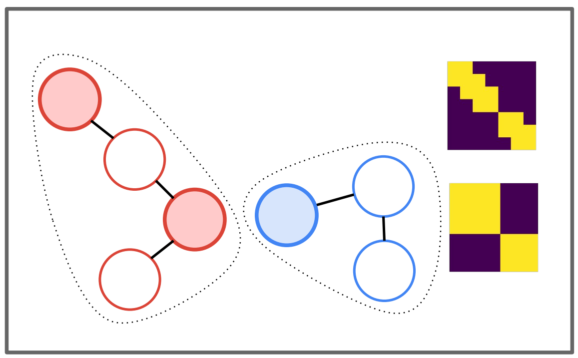
Discrete operations



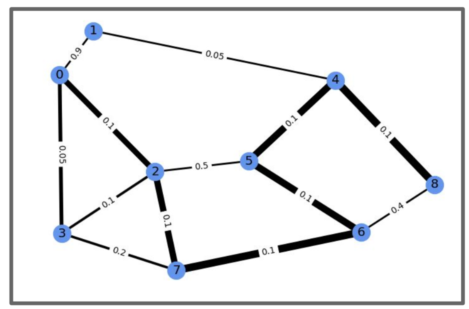
Dynamic programs



Sorting



Clustering

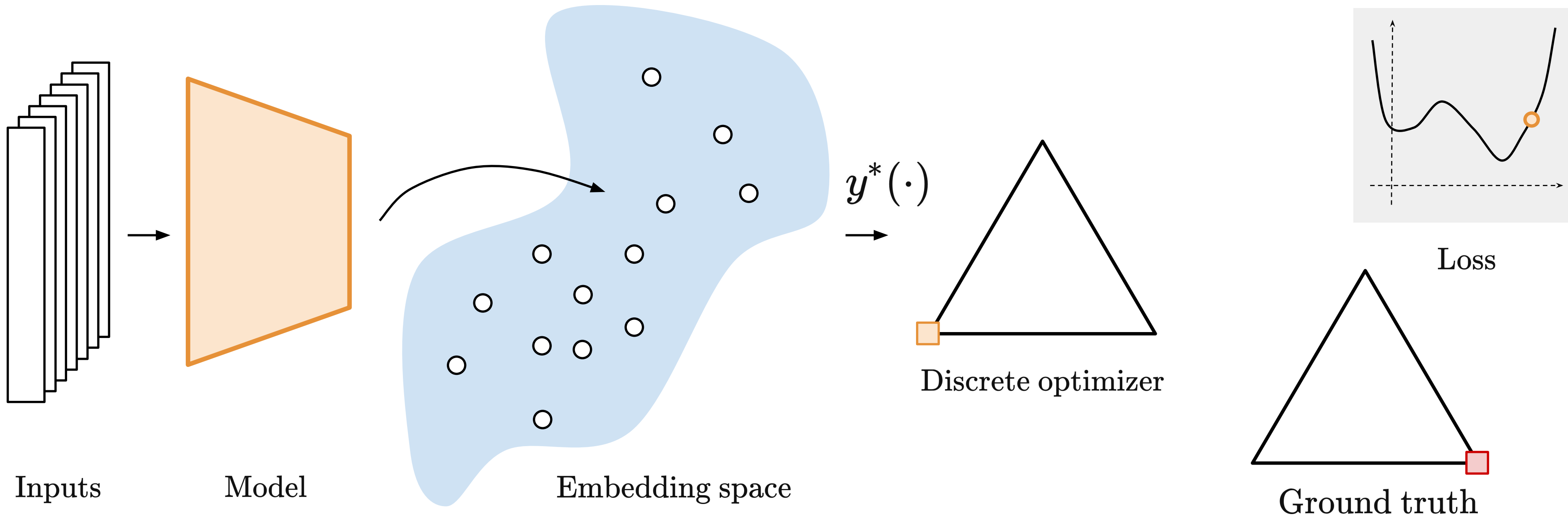


Shortest paths

- **Discrete** operations / algorithms at the heart of computer science.
- Powerful tool to deal with **structured** problems / data.

Structured inference and prediction

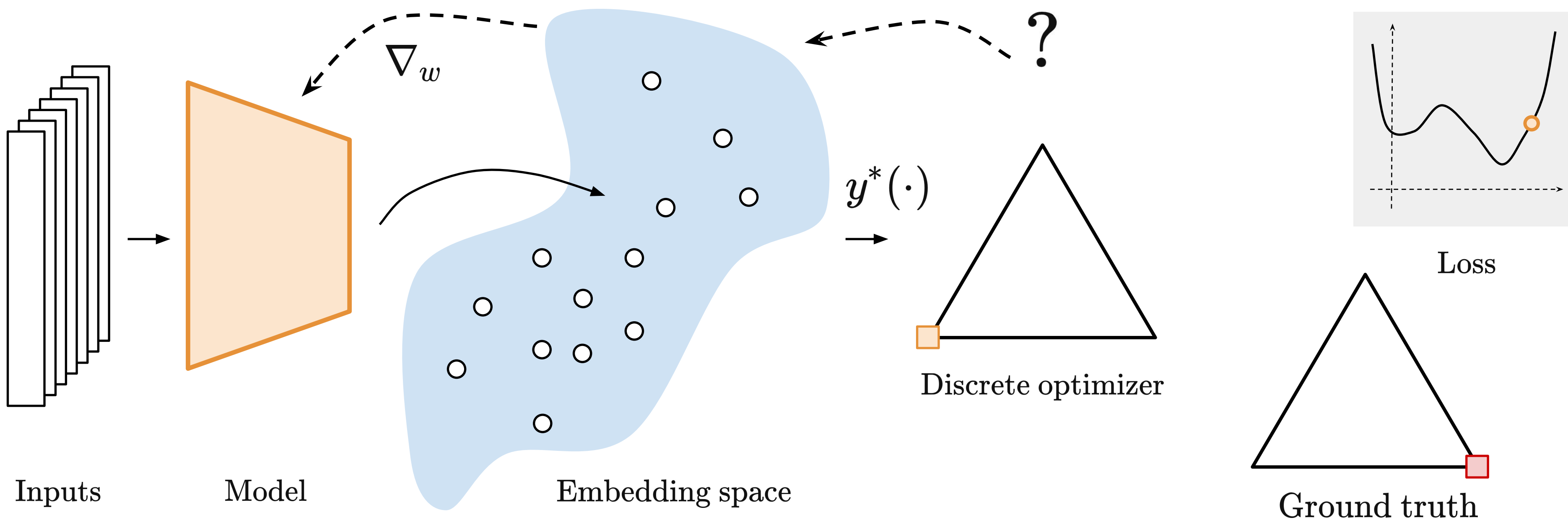
Discrete operators allow us to leverage **structure**.



- Use of an **implicit** and **discrete** function, challenge for **gradients**
- Challenge: moving embeddings towards "correct solution" continuously
- In **classification**, soft "arg" max solution, smooth approximation.

Structured inference and prediction

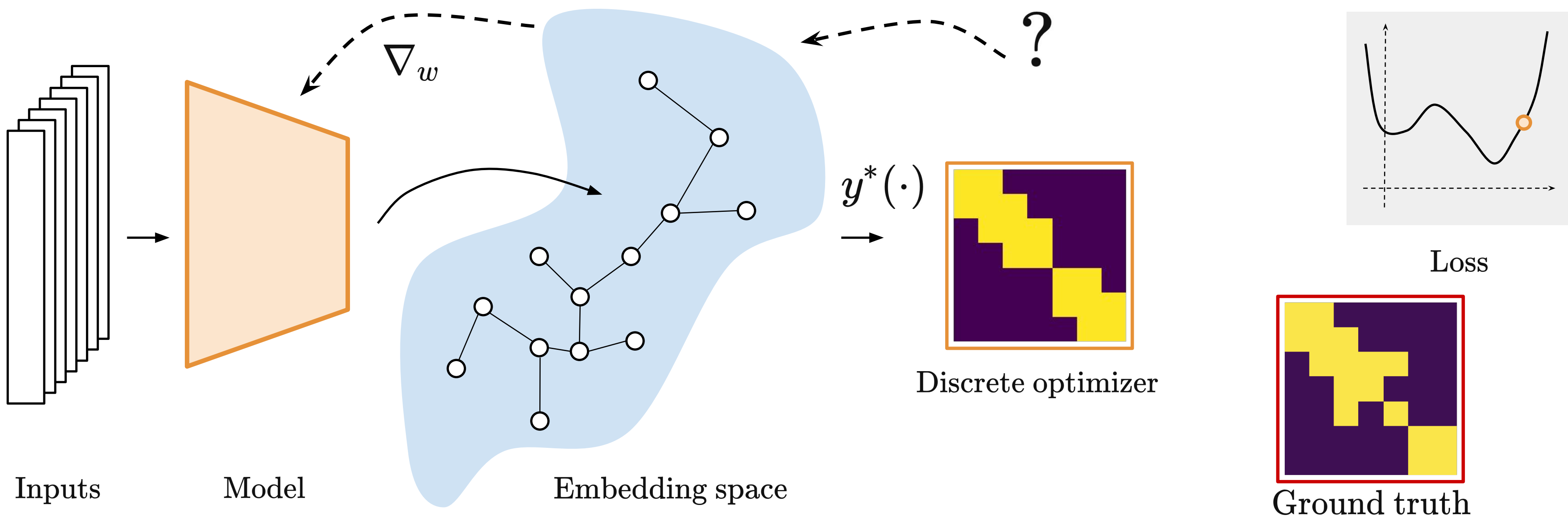
Discrete operators allow us to leverage **structure**.



- Use of an **implicit** and **discrete** function, challenge for **gradients**
- Challenge: moving embeddings towards "correct solution" continuously
- In **classification**, soft "arg" max solution, smooth approximation.

Structured inference and prediction

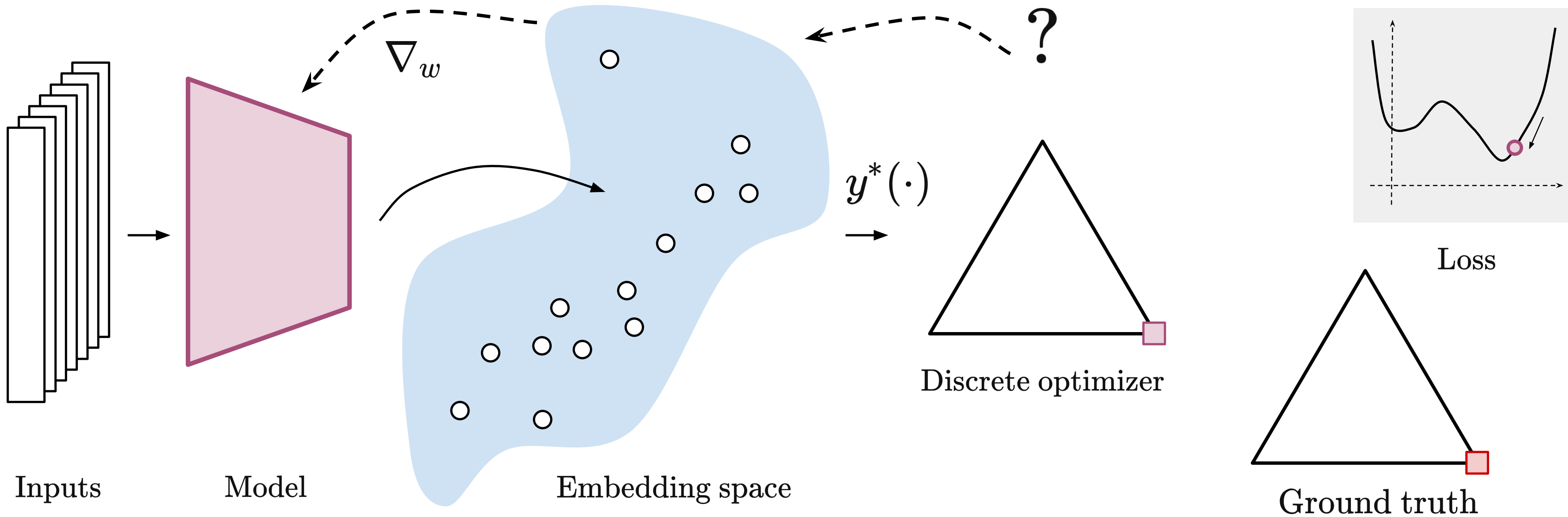
Discrete operators allow us to leverage **structure**.



- Use of an **implicit** and **discrete** function, challenge for **gradients**
- Challenge: moving embeddings towards "correct solution" continuously
- In **classification**, soft "arg" max solution, smooth approximation.

Structured inference and prediction

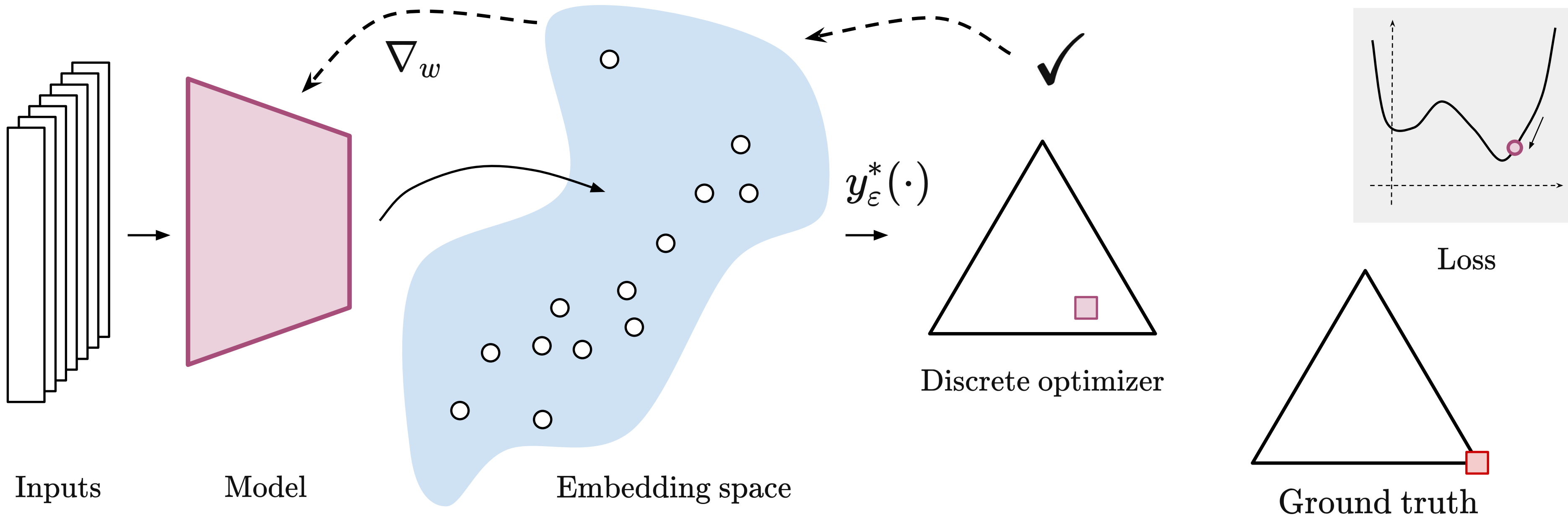
Discrete operators allow us to leverage **structure**.



- Use of an **implicit** and **discrete** function, challenge for **gradients**
- Challenge: moving embeddings towards "correct solution" continuously
- In **classification**, soft "arg" max solution, smooth approximation.

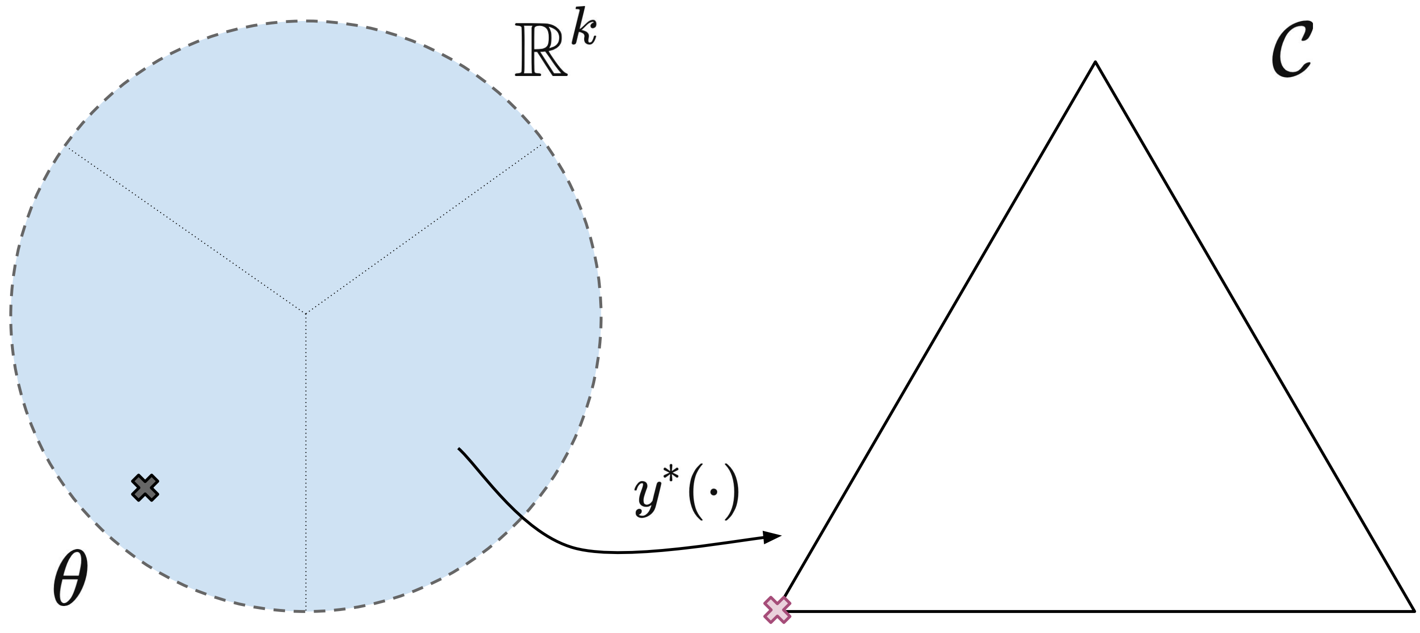
Structured inference and prediction

Discrete operators allow us to leverage **structure**.



- Use of an **implicit** and **discrete** function, challenge for **gradients**
- Challenge: moving embeddings towards "correct solution" continuously
- In **classification**, soft "arg" max solution, smooth approximation.

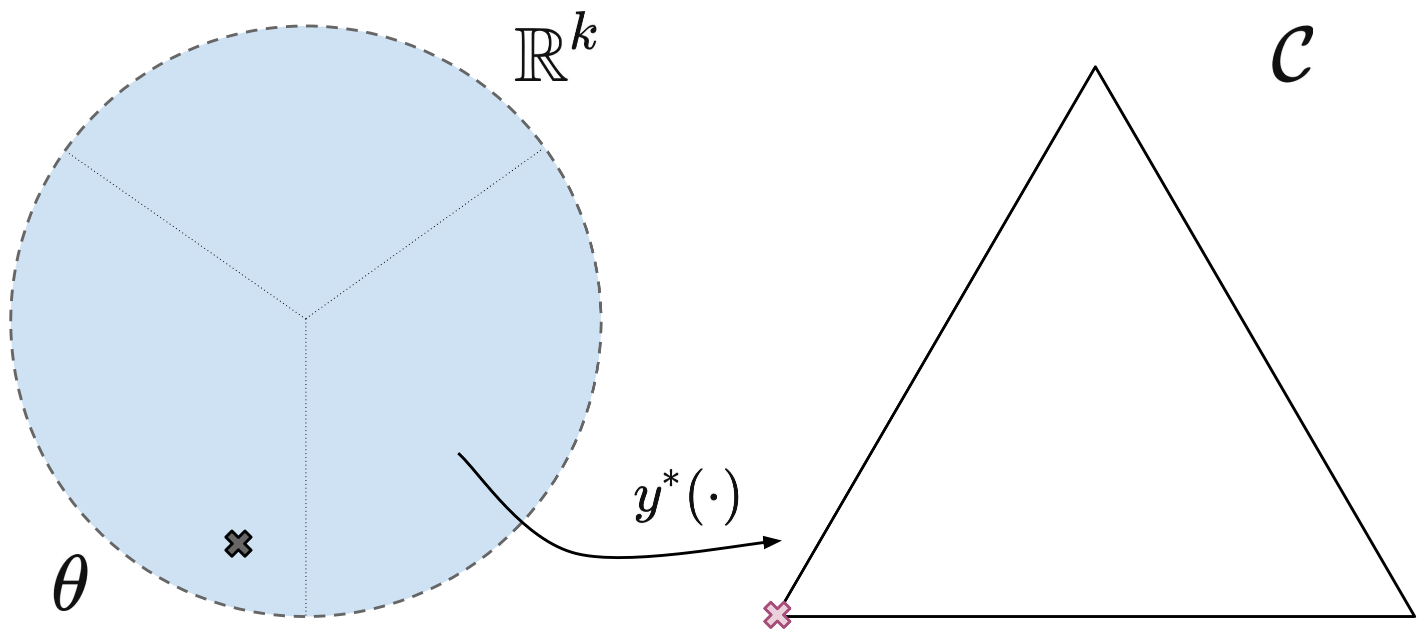
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / perturbed variational definition, with **closed form**

$$y^*(\theta) = \operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta \rangle \,, \quad [y^*(\theta)]_i = \delta_{i^*}(i)\,.$$

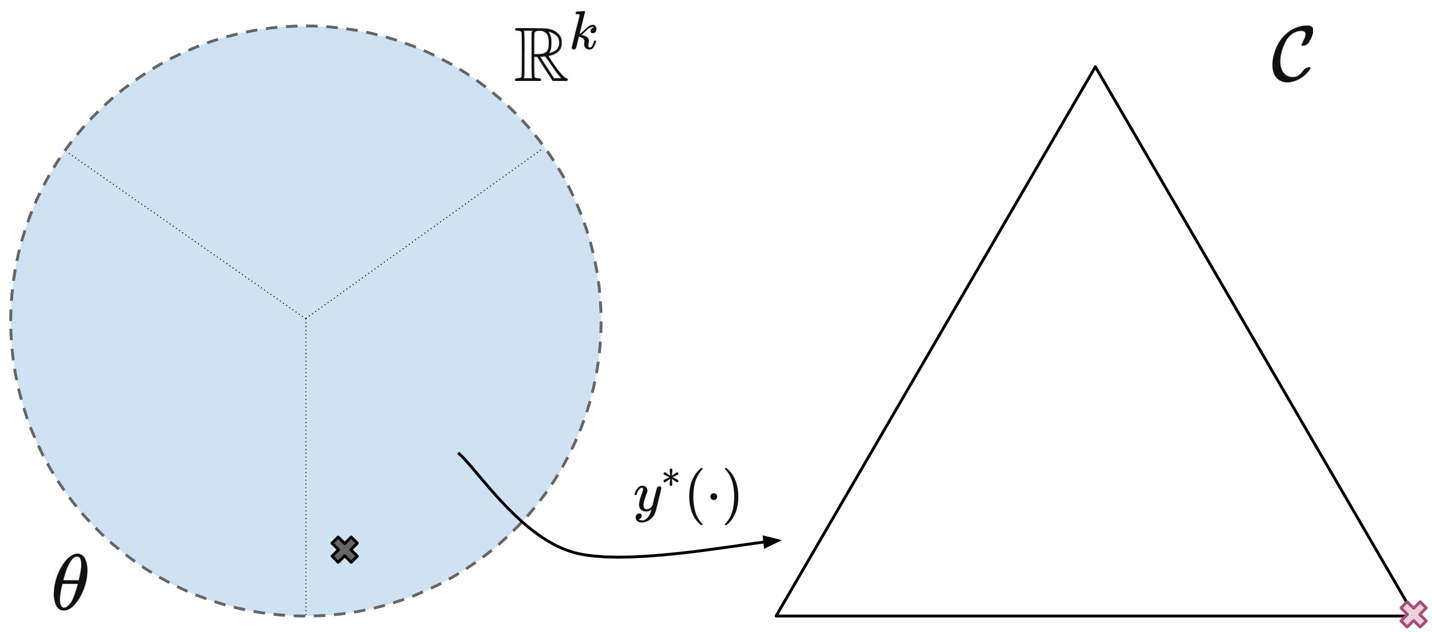
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / perturbed variational definition, with **closed form**

$$y^*(\theta) = \underset{y \in \mathcal{C}}{\operatorname{argmax}} \langle y, \theta \rangle \,, \quad [y^*(\theta)]_i = \delta_{i^*}(i) \,.$$

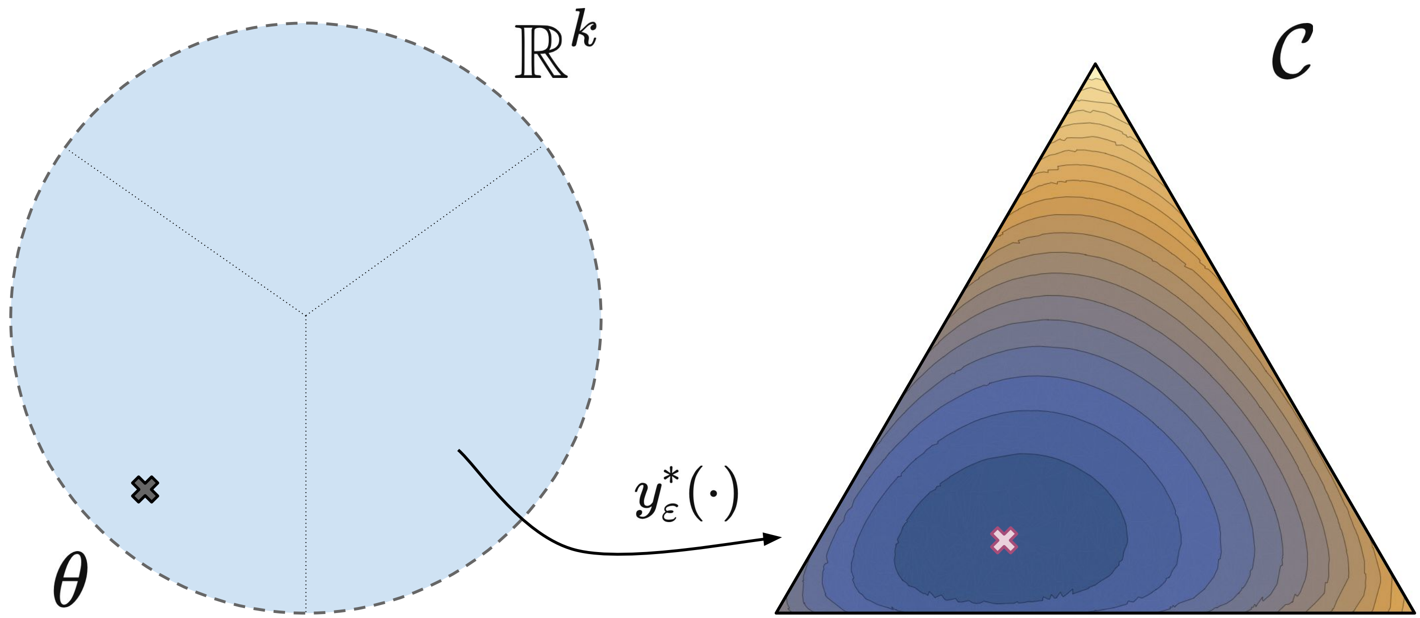
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / perturbed variational definition, with **closed form**

$$y^*(\theta) = \underset{y \in \mathcal{C}}{\mathrm{argmax}} \langle y, \theta \rangle \,, \quad [y^*(\theta)]_i = \delta_{i^*}(i) \,.$$

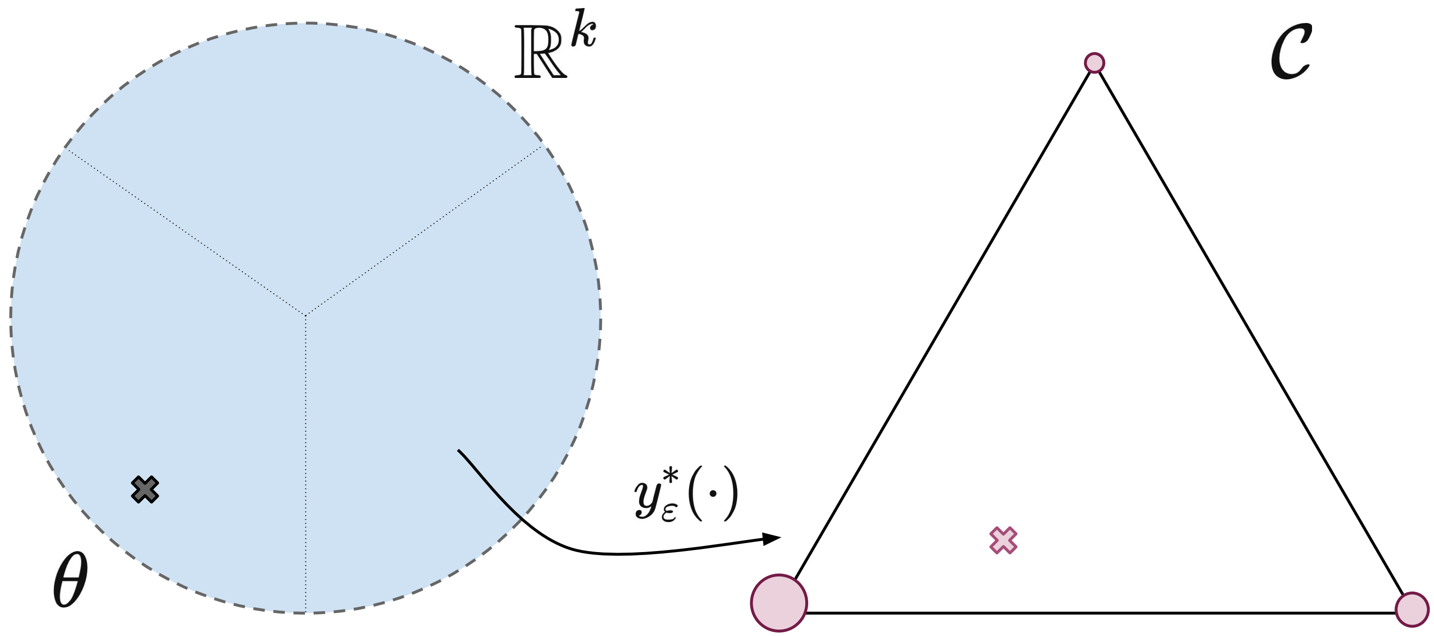
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent **regularized** / perturbed variational definition, with **closed form**

$$y_\varepsilon^*(\theta) = \underset{y \in \mathcal{C}}{\operatorname{argmax}} \langle y, \theta \rangle - \varepsilon \Omega(y) \,, \quad [y_\varepsilon^*(\theta)]_i = \frac{e^{\theta_i/\varepsilon}}{\sum_j e^{\theta_j/\varepsilon}} \,.$$

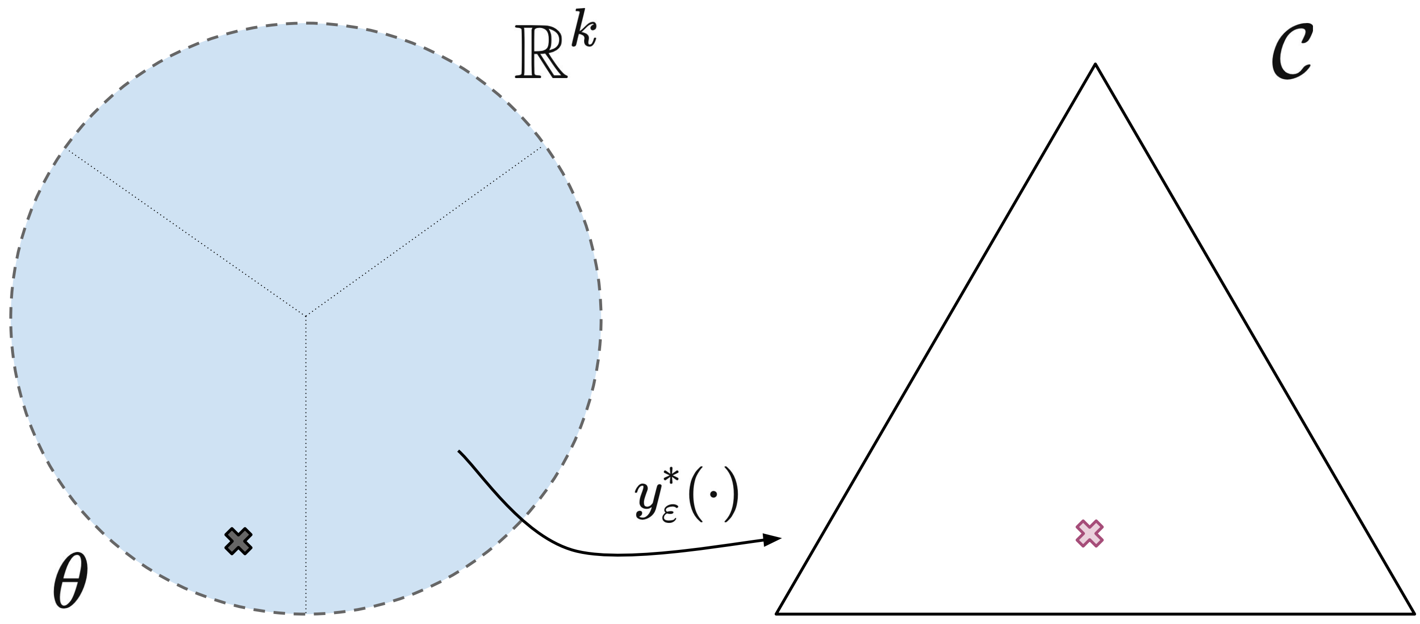
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / **perturbed** variational definition, with **closed form**

$$y^*_\varepsilon(\theta) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,, \quad [y^*_\varepsilon(\theta)]_i = \frac{e^{\theta_i/\varepsilon}}{\sum_j e^{\theta_j/\varepsilon}} \,.$$

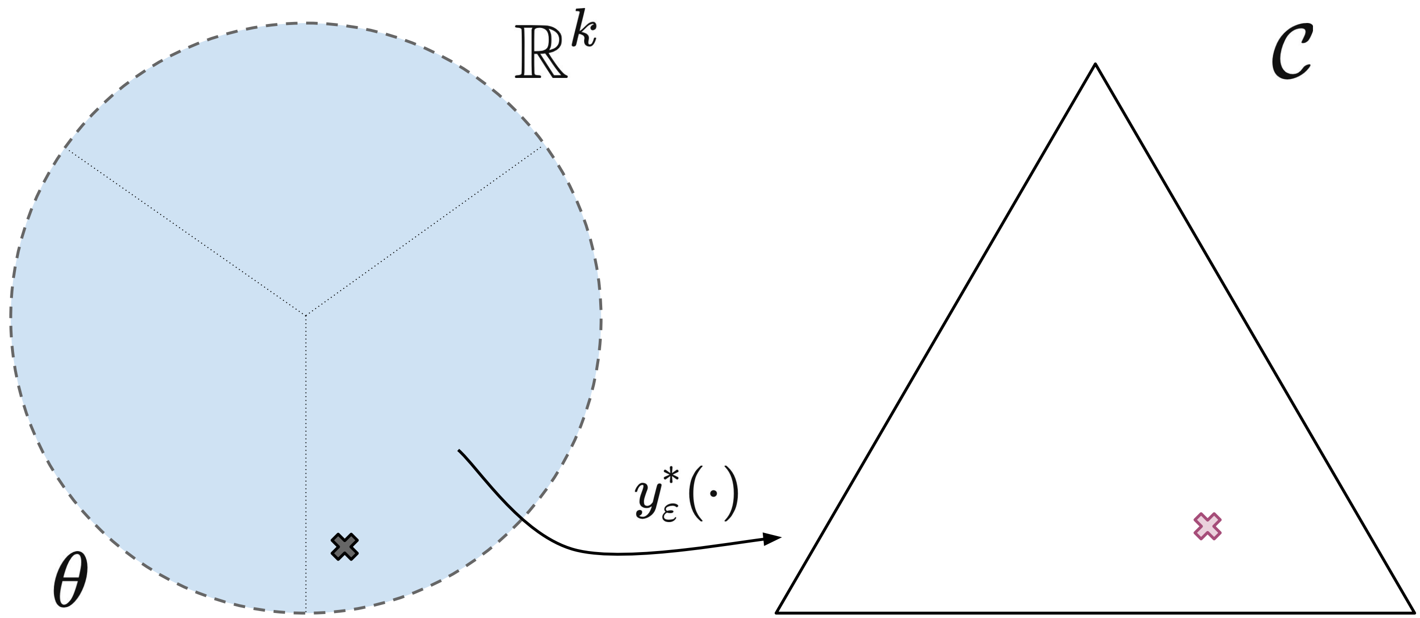
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / **perturbed** variational definition, with **closed form**

$$y^*_\varepsilon(\theta) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,, \quad [y^*_\varepsilon(\theta)]_i = \frac{e^{\theta_i/\varepsilon}}{\sum_j e^{\theta_j/\varepsilon}} \,.$$

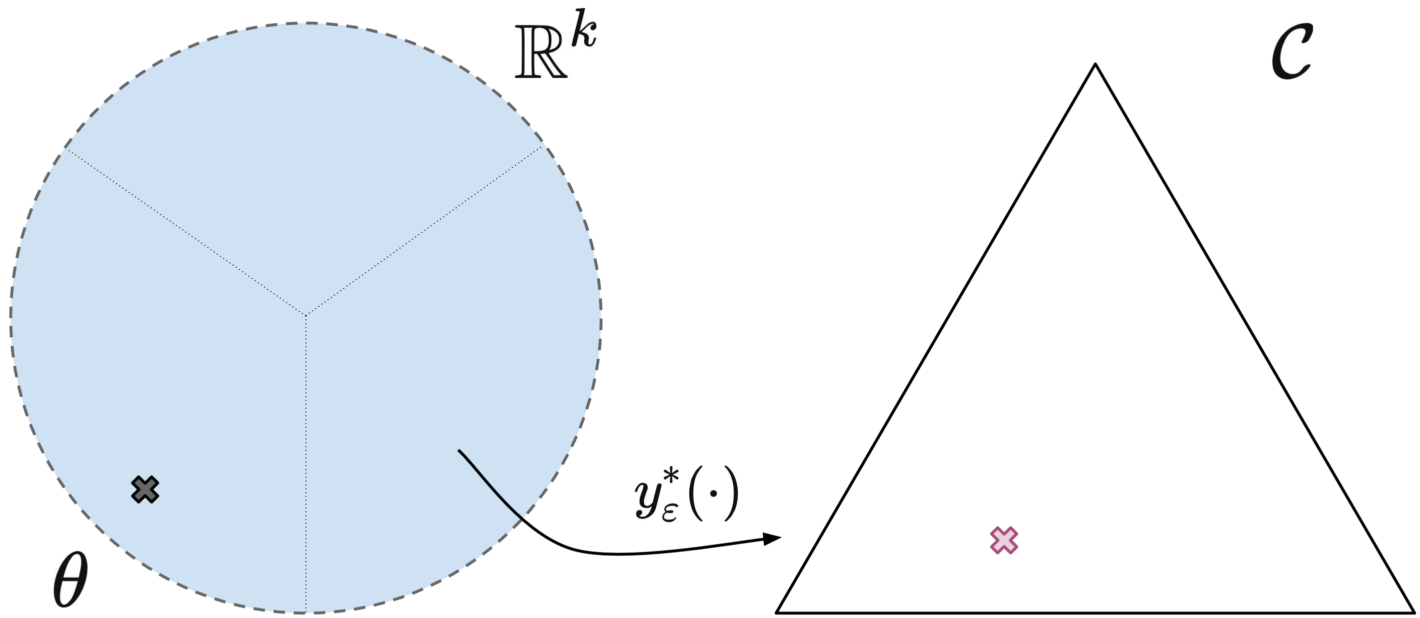
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / **perturbed** variational definition, with **closed form**

$$y^*_\varepsilon(\theta) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,, \quad [y^*_\varepsilon(\theta)]_i = \frac{e^{\theta_i/\varepsilon}}{\sum_j e^{\theta_j/\varepsilon}} \,.$$

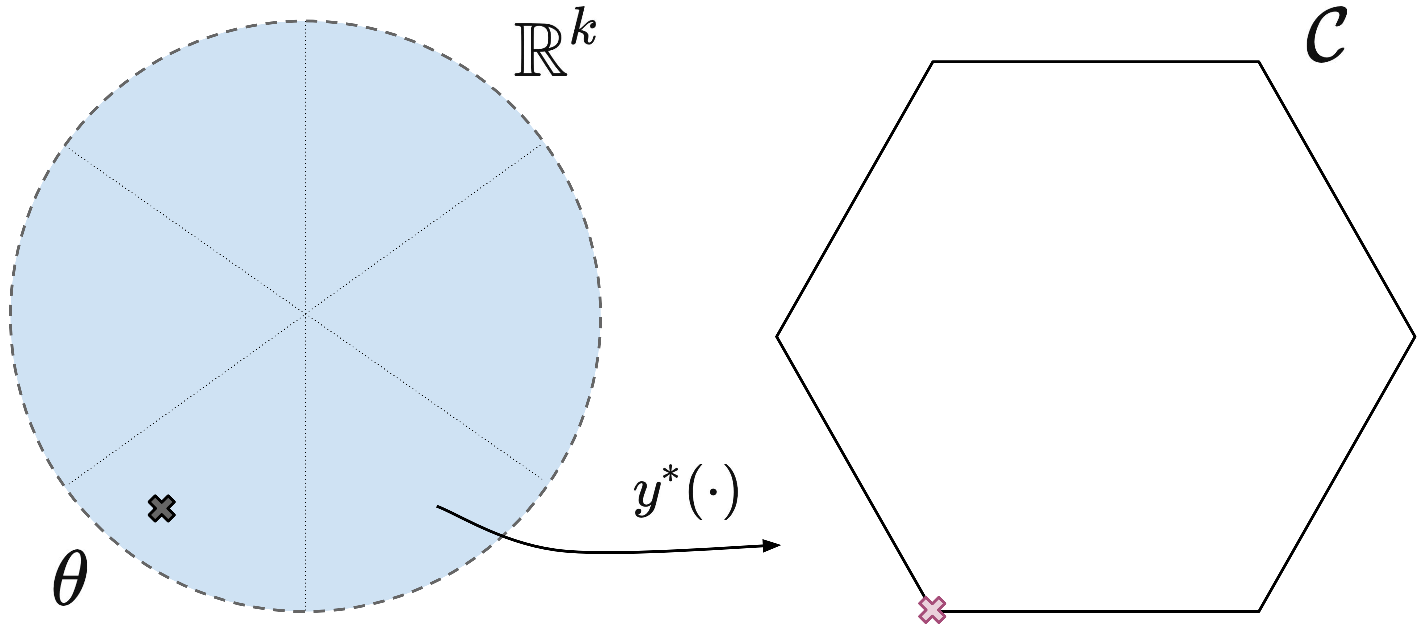
Softmax



- **Softmax** a smooth, explicit approximation of "hard" argmax over **simplex**.
- Equivalent regularized / **perturbed** variational definition, with **closed form**

$$y^*_\varepsilon(\theta) = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle]\,, \quad [y^*_\varepsilon(\theta)]_i = \frac{e^{\theta_i/\varepsilon}}{\sum_j e^{\theta_j/\varepsilon}}\,.$$

Generalized softmax (?)

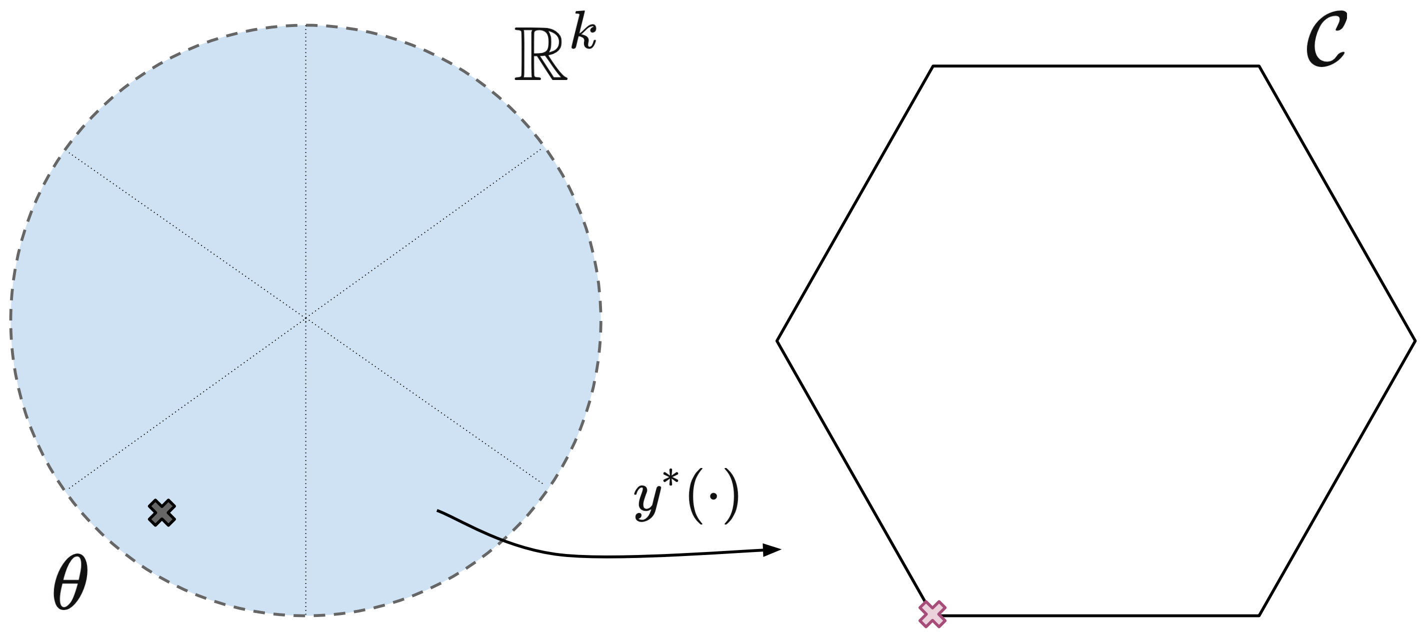


- Most discrete optimizers can be naturally written as

$$y^*(\theta) = \operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta \rangle$$

- How to generalize these methods, to have **differentiable** proxies?

Generalized softmax (?)

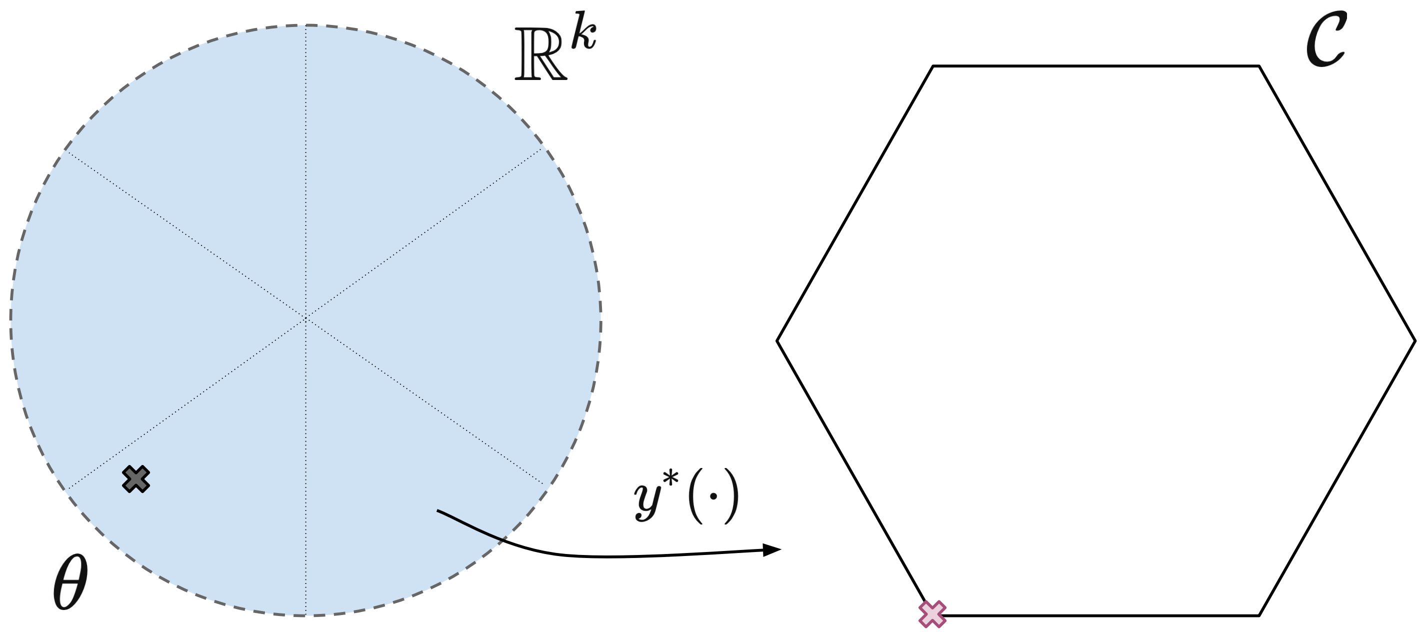


- Most discrete optimizers can be naturally written as

$$y^*(\theta) = \operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta \rangle$$

- How to generalize these methods, to have **differentiable** proxies?

Generalized softmax (?)

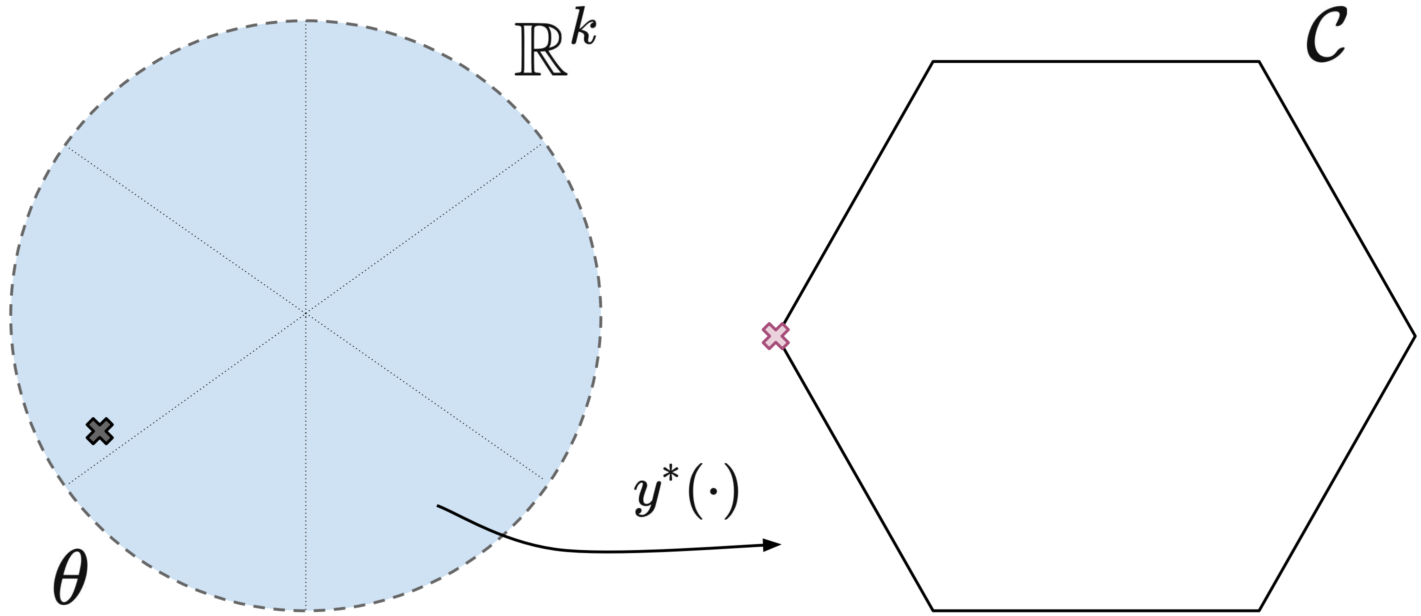


- Most discrete optimizers can be naturally written as

$$y^*(\theta) = \operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta \rangle$$

- How to generalize these methods, to have **differentiable** proxies?

Generalized softmax (?)



- Most discrete optimizers can be naturally written as

$$y^*(\theta) = \underset{y \in \mathcal{C}}{\operatorname{argmax}} \langle y, \theta \rangle$$

- How to generalize these methods, to have **differentiable** proxies?

Q. Berthet

M.Blondel

O.Teboul

M. Cuturi

J-P. Vert

F.Bach

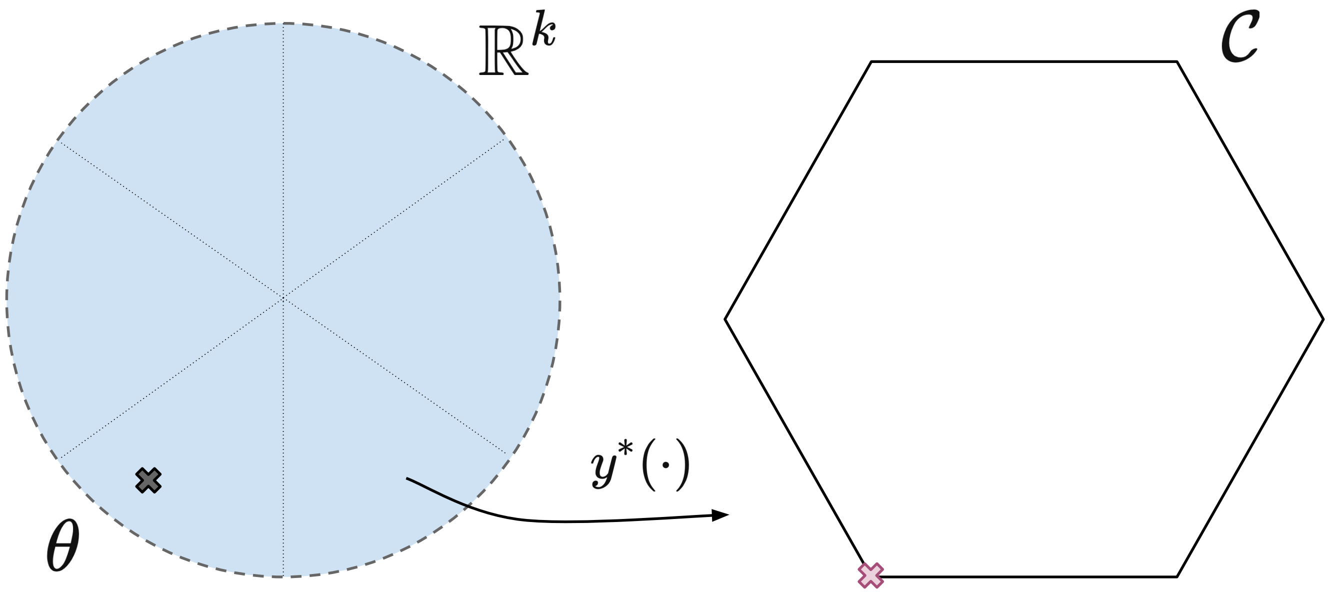
- **Learning with Differentiable Perturbed Optimizers**

  NeurIPS 2020

Perturbed maximizer

Discrete decisions: optimizers of linear program over $\mathcal{C}$, convex hull of $\mathcal{Y} \subseteq \mathbf{R}^k$

$$F(\theta) = \max_{y \in \mathcal{C}} \langle y, \theta \rangle \,, \quad \text{and} \quad y^*(\theta) = \operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta \rangle = \nabla_\theta F(\theta)\,.$$

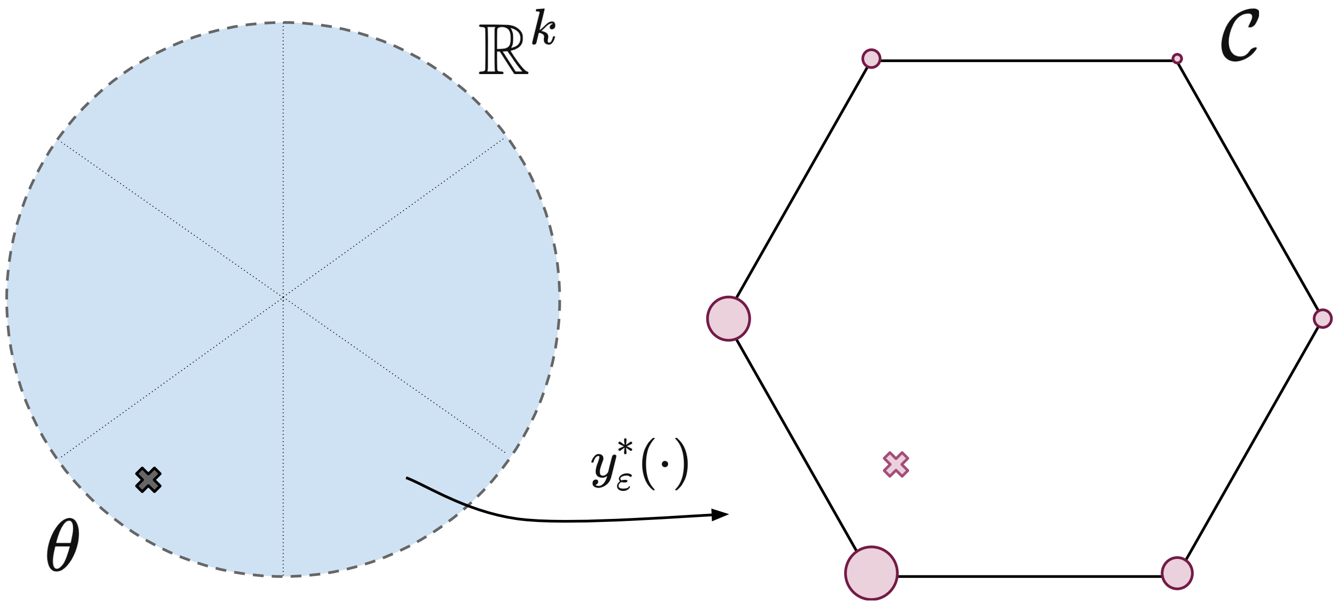


Perturbed maximizer: average of solutions for inputs with noise εZ

$$F_\varepsilon(\theta) = \mathbf{E}[\max_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,,\; y^*_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[\operatorname*{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] = \nabla_\theta F_\varepsilon(\theta)\,.$$

Perturbed maximizer

Discrete decisions: optimizers of linear program over $\mathcal{C}$, convex hull of $\mathcal{Y} \subseteq \mathbf{R}^k$

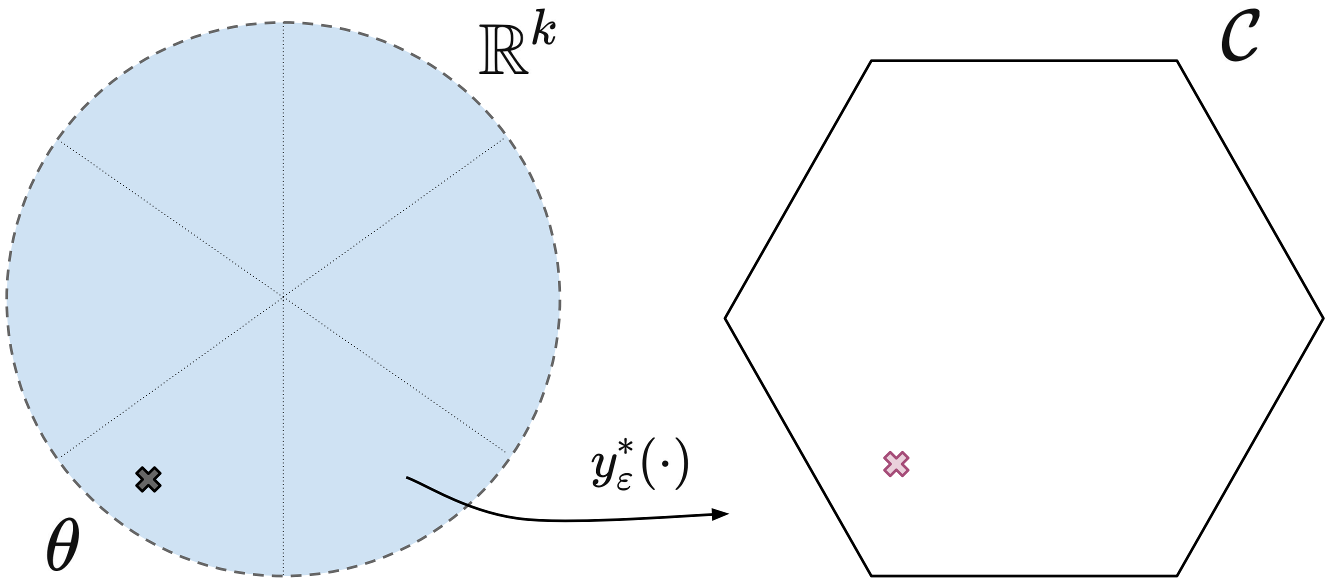


Perturbed maximizer: average of solutions for inputs with noise εZ

$$F_\varepsilon(\theta) = \mathbf{E}[\max_{y\in\mathcal{C}}\langle y, \theta+\varepsilon Z\rangle]\,,\ y^*_\varepsilon(\theta) = \mathbf{E}[y^*(\theta+\varepsilon Z)] = \mathbf{E}[\operatorname*{argmax}_{y\in\mathcal{C}}\langle y, \theta+\varepsilon Z\rangle] = \nabla_\theta F_\varepsilon(\theta)\,.$$

Perturbed maximizer

Discrete decisions: optimizers of linear program over $\mathcal{C}$, convex hull of $\mathcal{Y} \subseteq \mathbf{R}^k$

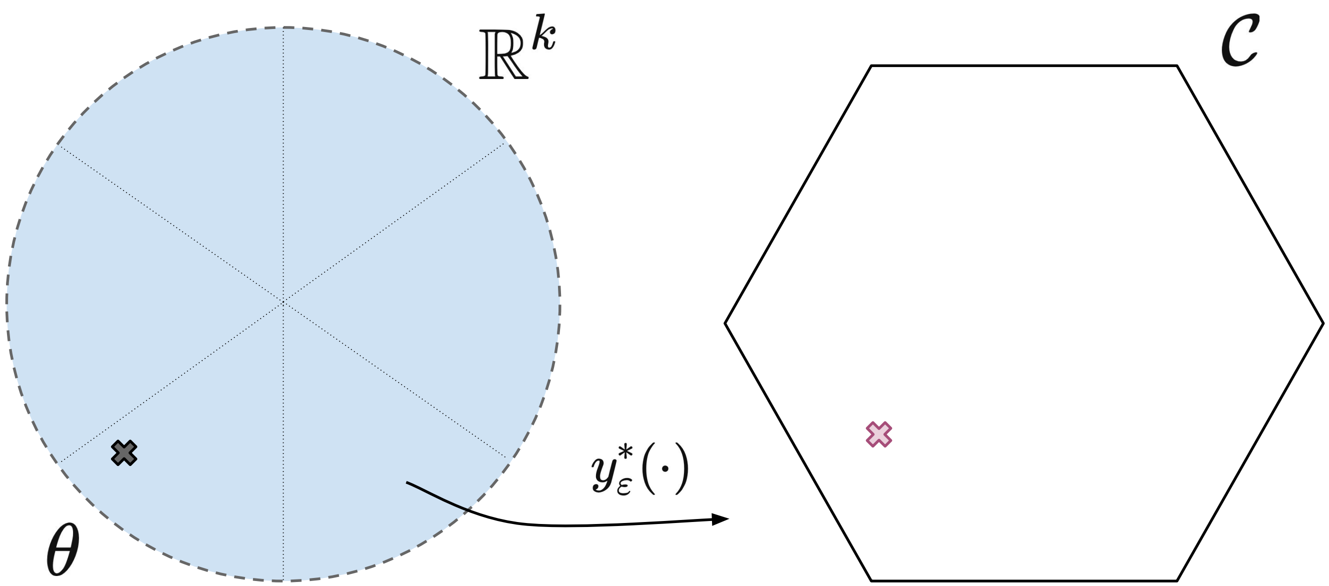


Perturbed maximizer: average of solutions for inputs with noise εZ

$$F_\varepsilon(\theta) = \mathbf{E}[\max_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,, \; y^*_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[\operatorname{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] = \nabla_\theta F_\varepsilon(\theta) \,.$$

Perturbed maximizer

Discrete decisions: optimizers of linear program over $\mathcal{C}$, convex hull of $\mathcal{Y} \subseteq \mathbf{R}^k$

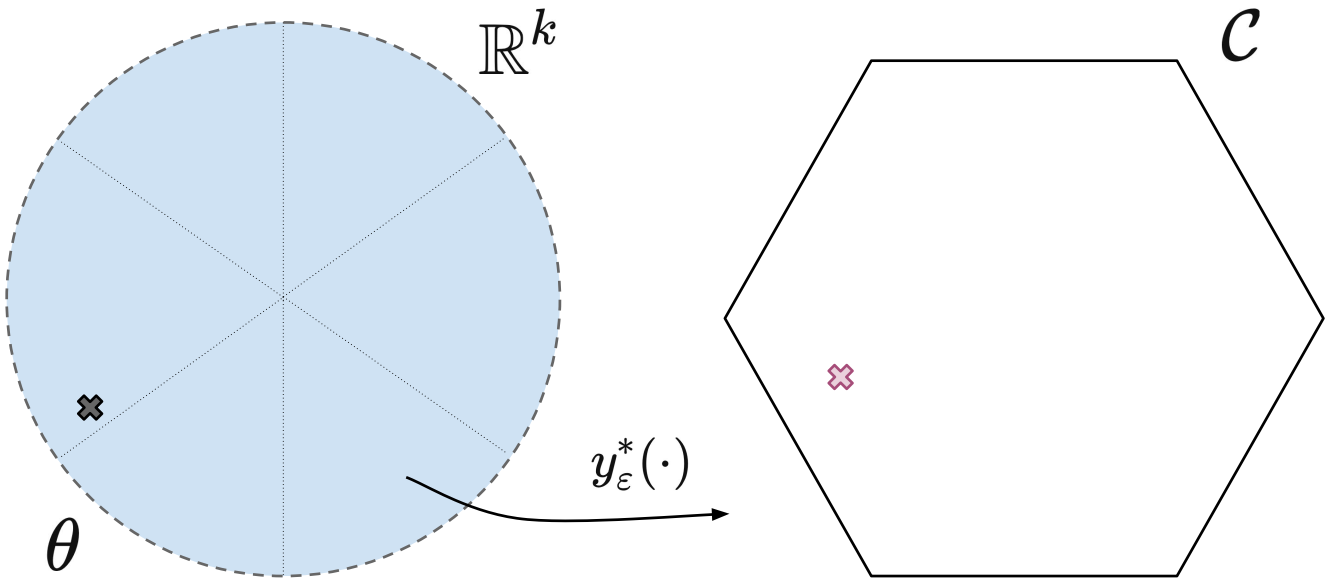


Perturbed maximizer: average of solutions for inputs with noise εZ

$$F_\varepsilon(\theta) = \mathbf{E}[\max_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle]\,,\; y^*_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[\operatorname{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] = \nabla_\theta F_\varepsilon(\theta)\,.$$

Perturbed maximizer

Discrete decisions: optimizers of linear program over $\mathcal{C}$, convex hull of $\mathcal{Y} \subseteq \mathbf{R}^k$



Perturbed maximizer: average of solutions for inputs with noise εZ

$$F_\varepsilon(\theta) = \mathbf{E}[\max_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] \,, \; y^*_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[\operatorname{argmax}_{y \in \mathcal{C}} \langle y, \theta + \varepsilon Z \rangle] = \nabla_\theta F_\varepsilon(\theta) \,.$$

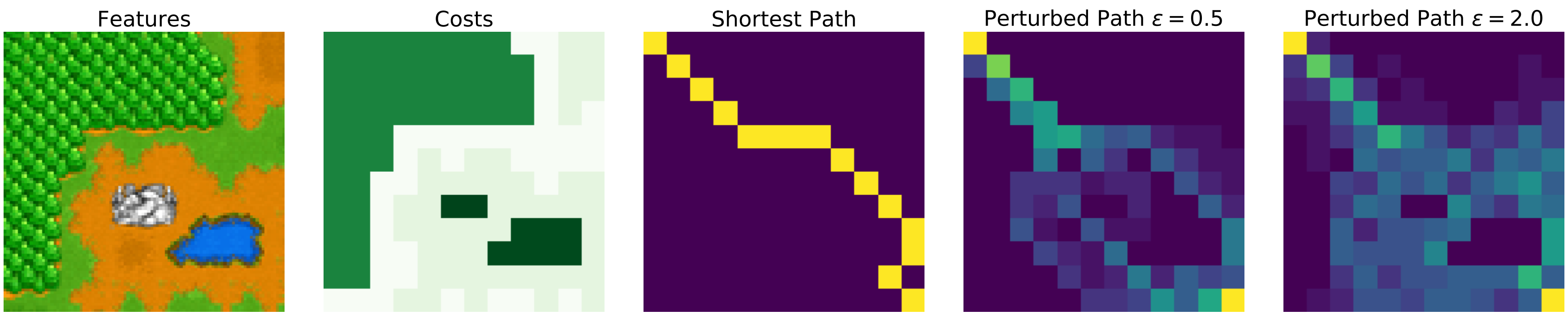
Perturbed model

Model of optimal decision under uncertainty Luce (1959), McFadden et al. (1973)

$$Y = \underset{y \in \mathcal{C}}{\operatorname{argmax}} \langle y, \theta + \varepsilon Z \rangle$$

Follows a **perturbed model** with $Y \sim p_\theta(y)$, expectation $y^*_\varepsilon(\theta) = \mathbf{E}_{p_\theta}[Y]$.

Perturb and map Papandreou & Yuille (2011), FT Perturbed L Kalai & Vempala (2003)



Example. Over the unit simplex $\mathcal{C} = \Delta^d$ with Gumbel noise Z, $F(\theta) = \max_i \theta_i$.

$$F_\varepsilon(\theta) = \varepsilon \log \sum_{i \in [d]} e^{\frac{\theta_i}{\varepsilon}}, \qquad p_\theta(e_i) \propto \exp(\langle \theta, e_i \rangle / \varepsilon), \qquad [y^*_\varepsilon(\theta)]_i = \frac{e^{\frac{\theta_i}{\varepsilon}}}{\sum e^{\frac{\theta_j}{\varepsilon}}}$$

Why? and How?

Learning problems:

Features X_i, model output $\theta_w = g_w(X_i)$, prediction $y_{\text{pred}} = y^*_\varepsilon(\theta_w)$, loss L

$$F(w) = L\big(y^*_\varepsilon(\theta_w)\,, y_i\big)\,, \quad \text{gradients require } \partial_\theta\, y^*_\varepsilon(\theta_w)\,.$$

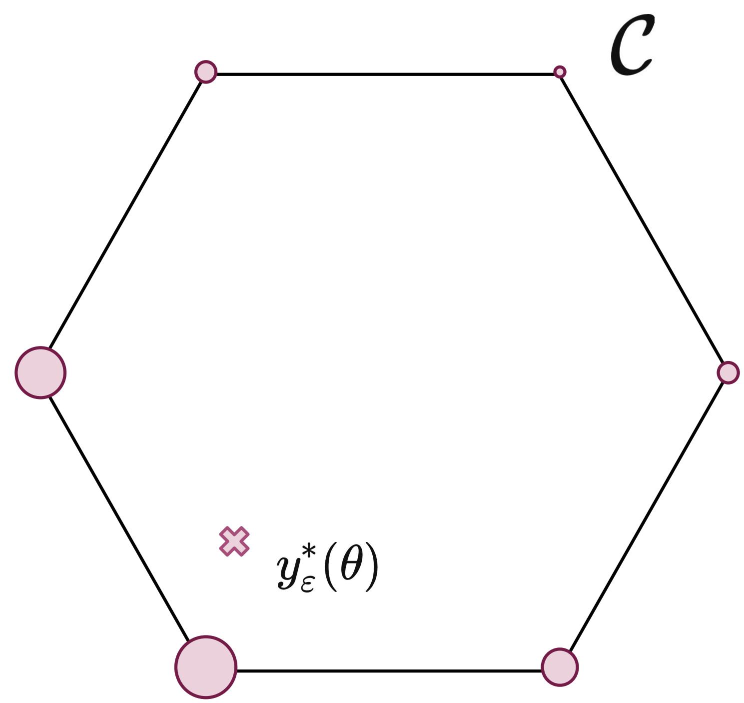
Monte Carlo estimates. Perturbed maximizer and derivatives as expectations.

For $\theta \in \mathbf{R}^d$, $Z^{(1)}, \ldots, Z^{(M)}$ i.i.d. copies

$$y^{(\ell)} = y^*(\theta + \varepsilon Z^{(\ell)})$$

Unbiased estimate of $y^*_\varepsilon(\theta)$ given by

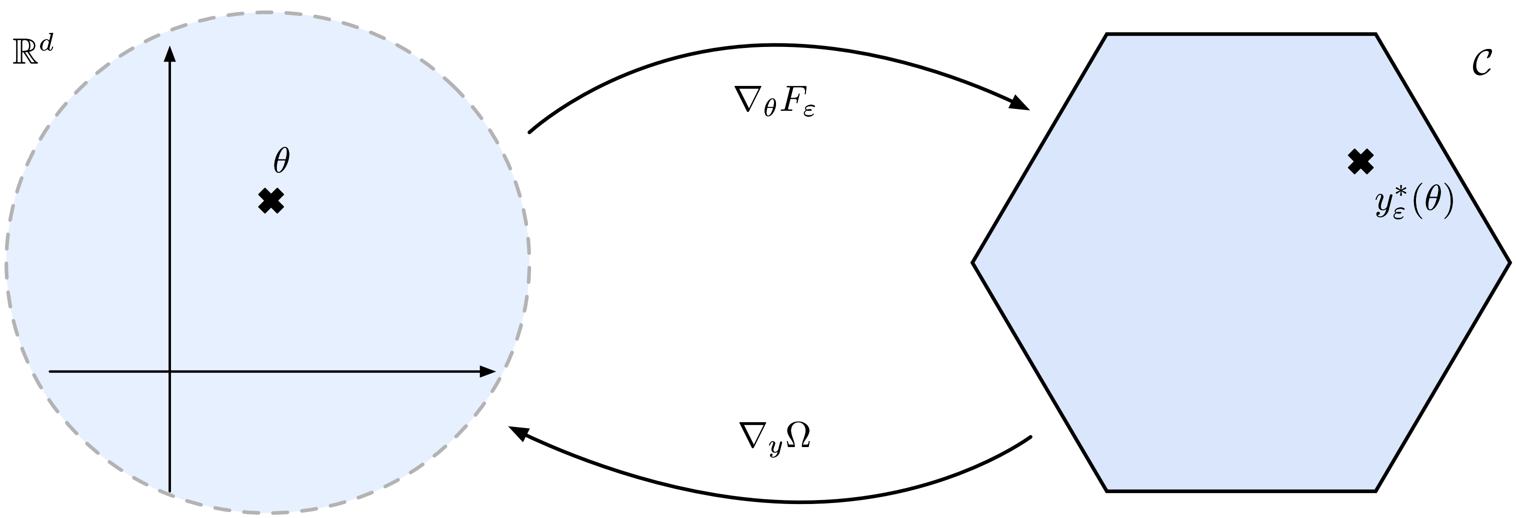
$$\bar{y}_{\varepsilon,M}(\theta) = \frac{1}{M} \sum_{\ell=1}^{M} y^{(\ell)}\,.$$



Properties

Mirror maps: For $\mathcal{C}$ with full interior, Z with smooth density μ, full support

F_ε strictly convex, gradient Lipschitz. Ω strongly convex, Legendre type.



Differentiability. Functions are smooth in the inputs. For $\mu(z) \propto \exp(-\nu(z))$

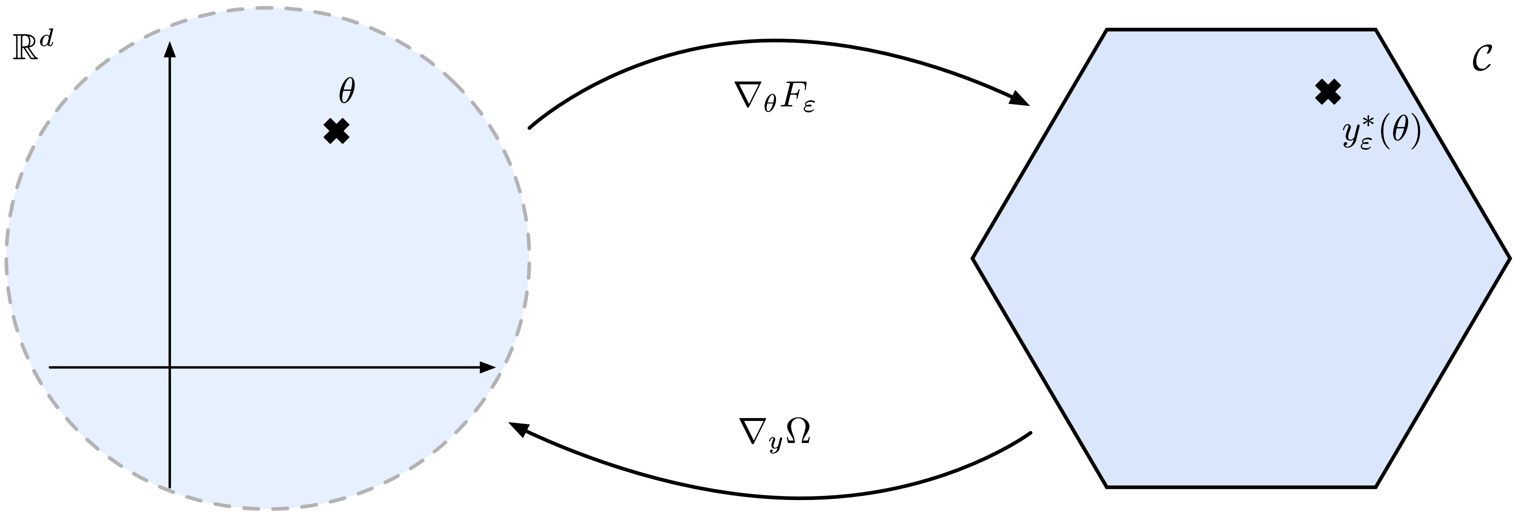
$$y^*_\varepsilon(\theta) = \nabla_\theta F_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[F(\theta + \varepsilon Z)\nabla_z \nu(Z)/\varepsilon]\,,$$
$$\partial_\theta\, y^*_\varepsilon(\theta) = \nabla^2 F_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)\nabla_z \nu(Z)^\top/\varepsilon]\,.$$

Perturbed maximizer y^*_ε never locally constant in θ. Abernethy et al. (2014)

Properties

Mirror maps: For $\mathcal{C}$ with full interior, Z with smooth density μ, full support

F_ε strictly convex, gradient Lipschitz. Ω strongly convex, Legendre type.



Differentiability. Functions are smooth in the inputs. For $\mu(z) \propto \exp(-\nu(z))$

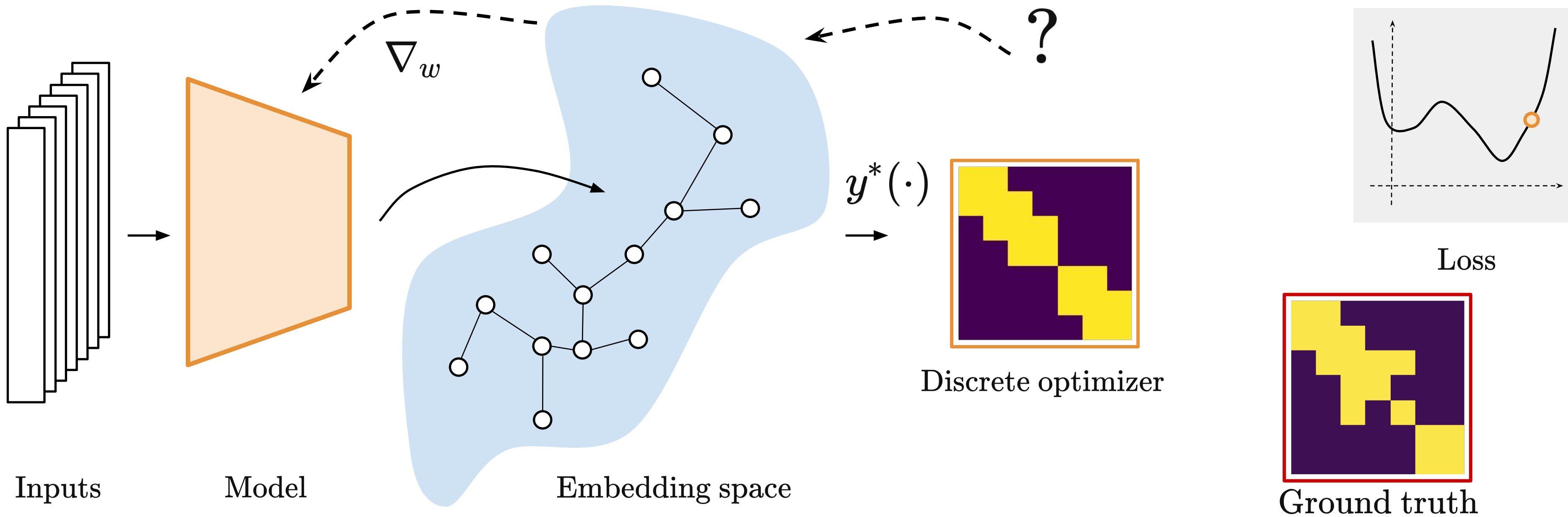
$$y^*_\varepsilon(\theta) = \nabla_\theta F_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)] = \mathbf{E}[F(\theta + \varepsilon Z)\nabla_z \nu(Z)/\varepsilon]\,,$$

$$\partial_\theta\, y^*_\varepsilon(\theta) = \nabla^2 F_\varepsilon(\theta) = \mathbf{E}[y^*(\theta + \varepsilon Z)\nabla_z \nu(Z)^\top/\varepsilon]\,.$$

Perturbed maximizer y^*_ε never locally constant in θ. Abernethy et al. (2014)

Learning with perturbed optimizers

Machine learning pipeline: variable X, discrete label y, model outputs $\theta = g_w(X)$

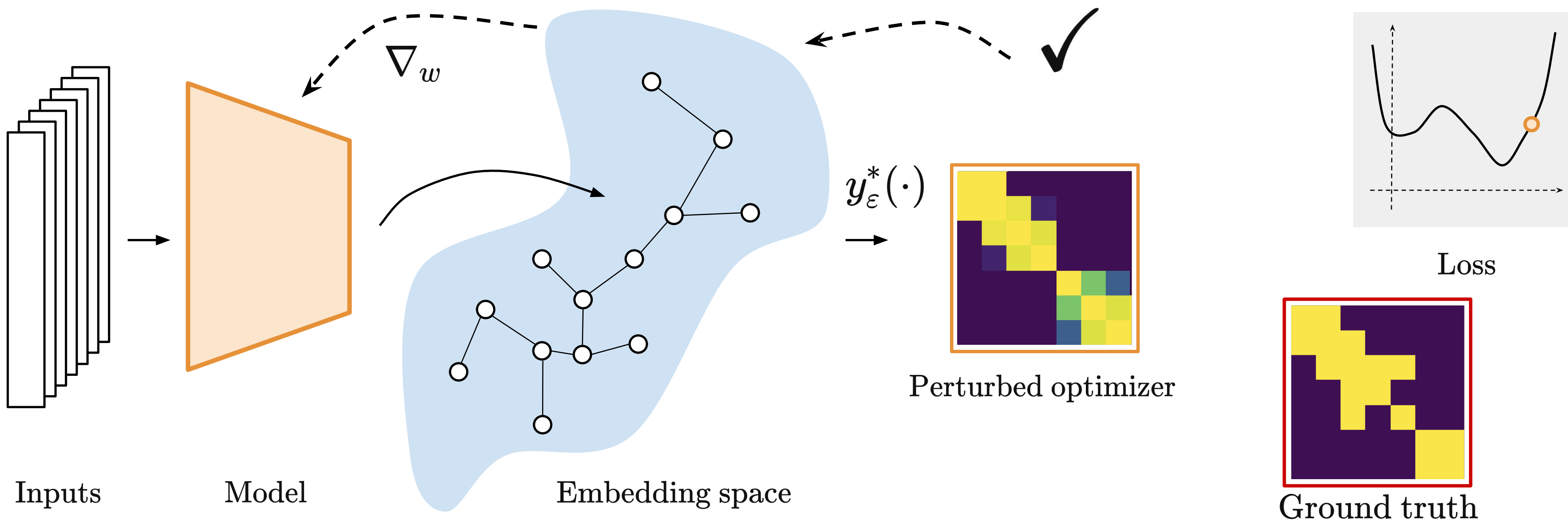


Labels are solutions of optimization problems (one-hots, ranks, shortest paths)

Small modification of the model: end-to-end differentiable

Learning with perturbed optimizers

Machine learning pipeline: variable X, discrete label y, model outputs $\theta = g_w(X)$

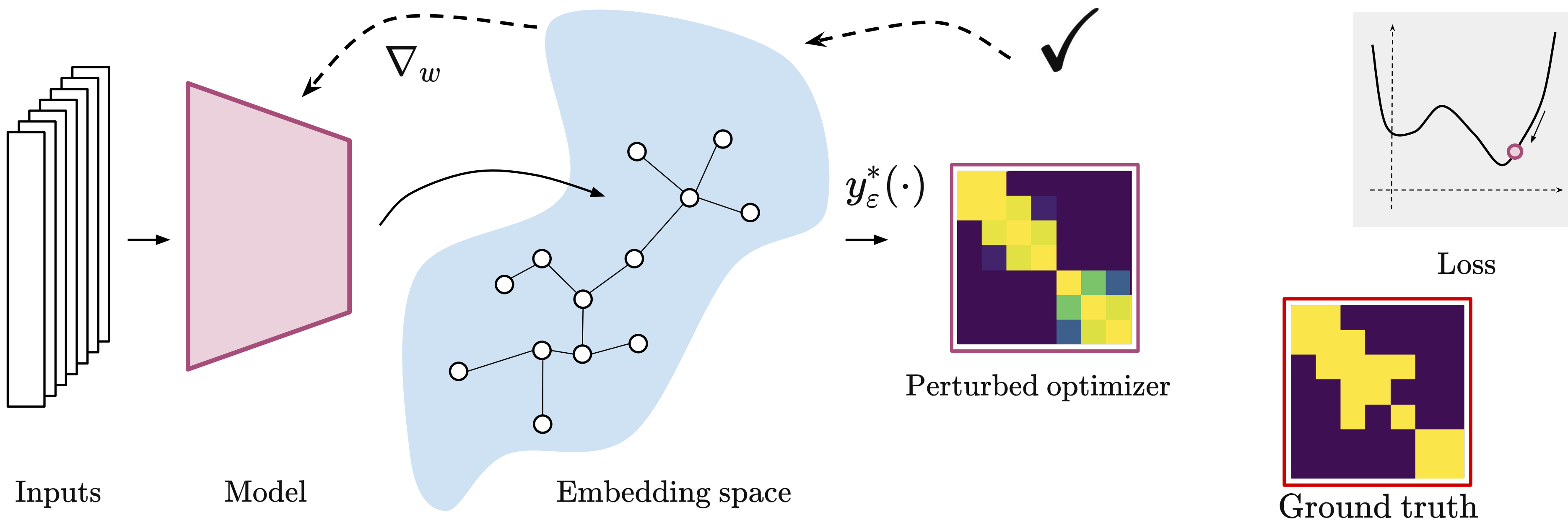


Labels are solutions of optimization problems (one-hots, ranks, shortest paths)

Small modification of the model: end-to-end differentiable

Learning with perturbed optimizers

Machine learning pipeline: variable X, discrete label y, model outputs $\theta = g_w(X)$

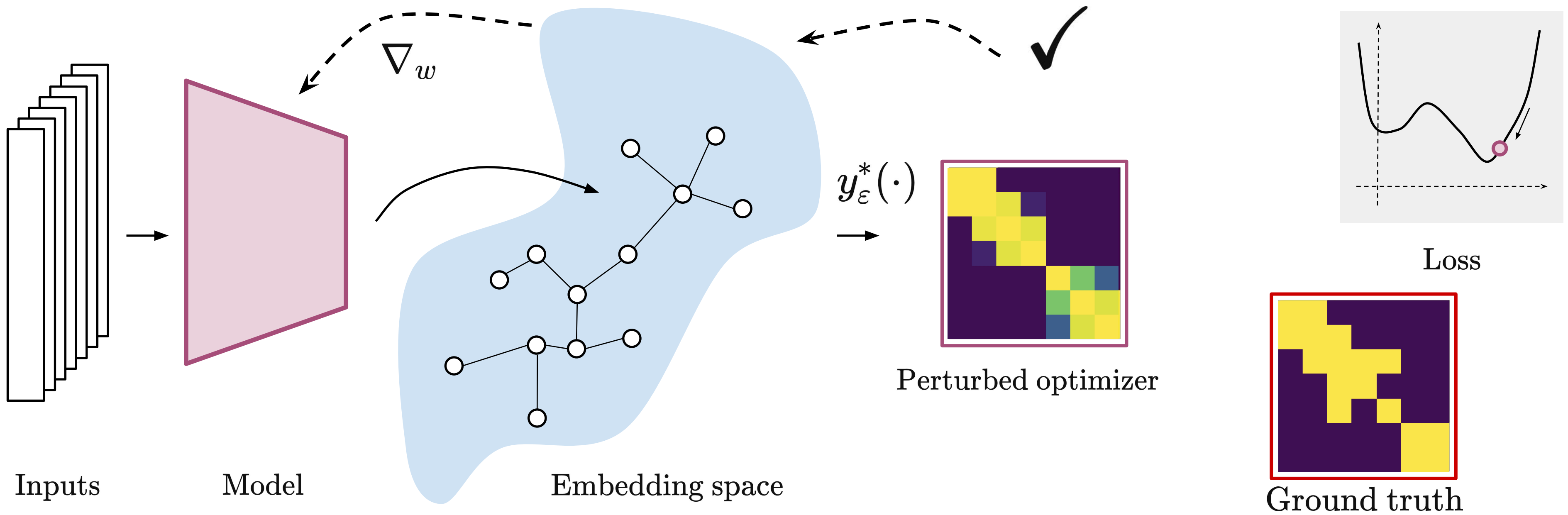


Labels are solutions of optimization problems (one-hots, ranks, shortest paths)

Small modification of the model: end-to-end differentiable

Learning with perturbations and Fenchel-Young losses

Within the same framework, possible to virtually bypass the optimization block

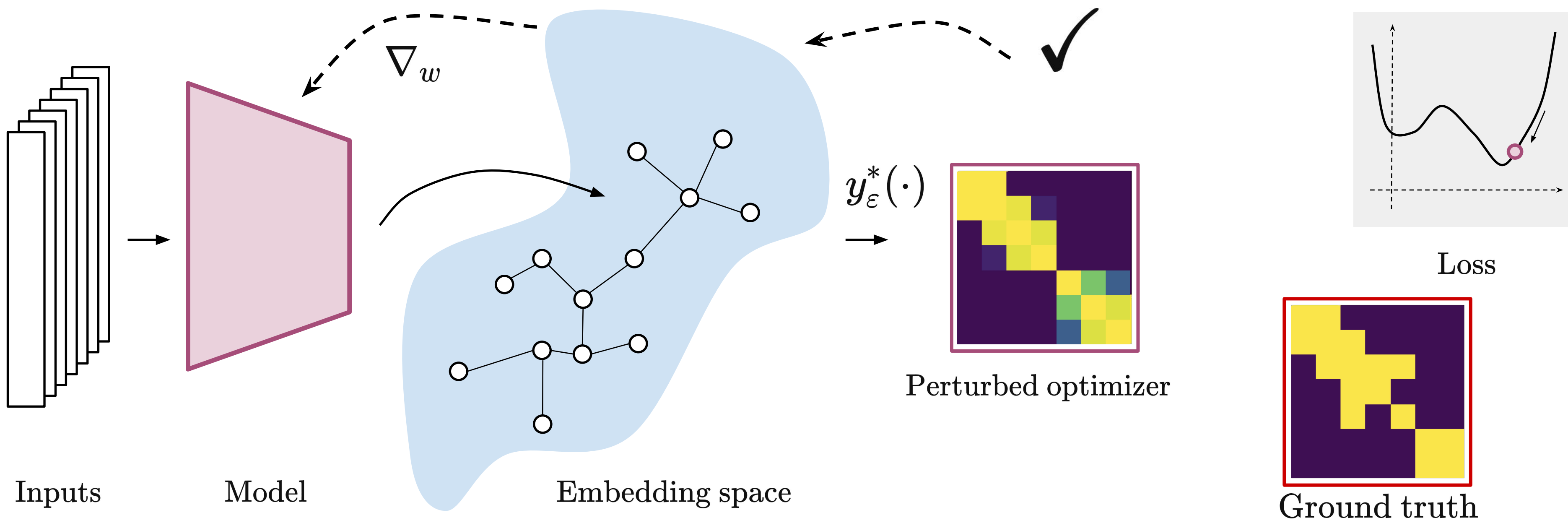


Fenchel-Young losses Easier to implement, no Jacobian of y^*_ε. Blondel et al (20)

Population loss minimized at ground truth for perturbed generative model.

Learning with perturbations and Fenchel-Young losses

Motivated by model where $y_i = \mathrm{argmax}_{y \in \mathcal{C}} \langle g_{w_0}(X_i) + \varepsilon Z_i, y \rangle$



Stochastic gradients for empirical loss only require

$$\nabla_\theta L(\theta = g_w(X_i); y_i) = y^*_\varepsilon(\theta) - y_i = y^*_\varepsilon(g_w(X_i)) - y_i \, .$$

Simulated by a doubly stochastic scheme.

Computations

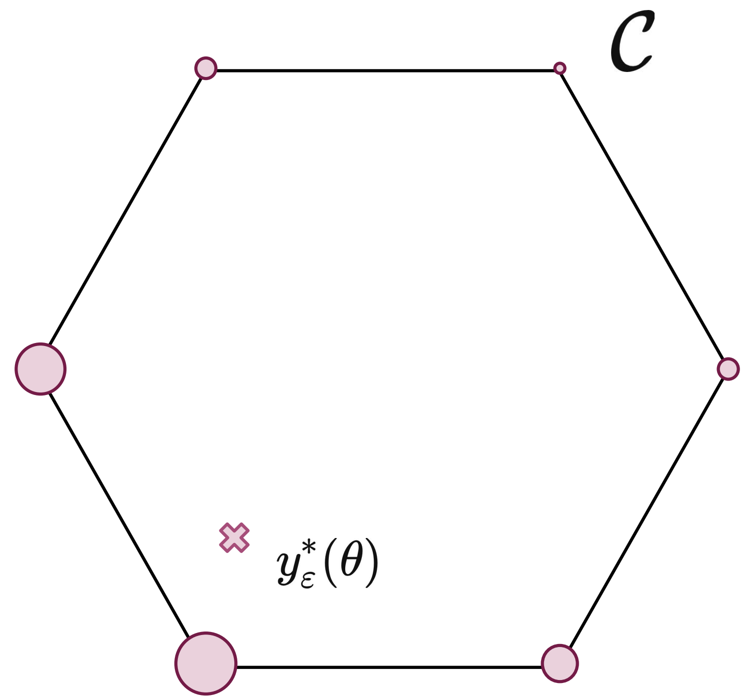
Monte Carlo estimates. Perturbed maximizer and derivatives as expectations.

For $\theta \in \mathbf{R}^d$, $Z^{(1)}, \ldots, Z^{(M)}$ i.i.d. copies

$$y^{(\ell)} = y^*(\theta + \varepsilon Z^{(\ell)})$$

Unbiased estimate of $y^*_\varepsilon(\theta)$ given by

$$\bar{y}_{\varepsilon,M}(\theta) = \frac{1}{M} \sum_{\ell=1}^{M} y^{(\ell)} .$$



Supervised learning:

Features X_i, model output $\theta_w = g_w(X_i)$, prediction $y_{\text{pred}} = y^*_\varepsilon(\theta_w)$.

Stochastic gradient in w:

$$\nabla_w F_i(w) = \partial_w g_w(X_i) \cdot (y^*_\varepsilon(\theta) - Y_i)$$

Computations

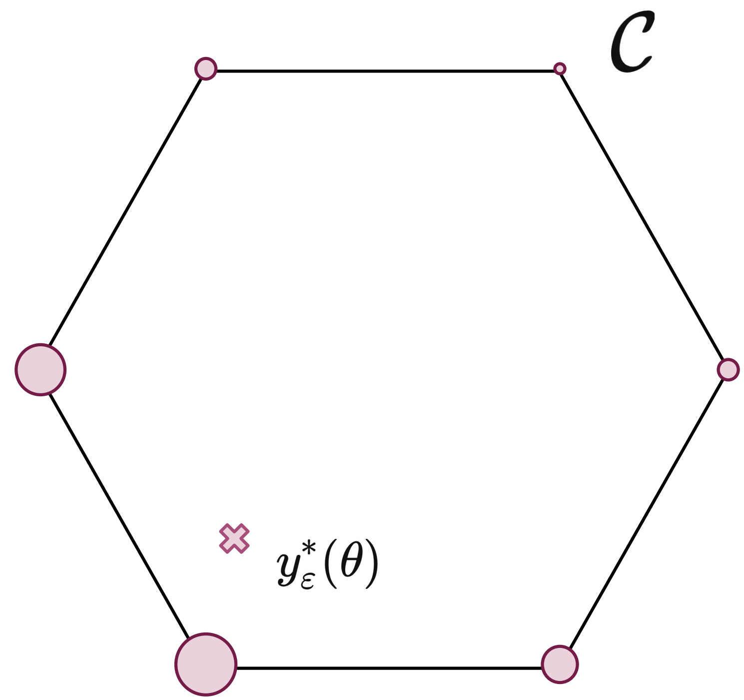
Monte Carlo estimates. Perturbed maximizer and derivatives as expectations.

For $\theta \in \mathbf{R}^d$, $Z^{(1)}, \ldots, Z^{(M)}$ i.i.d. copies

$$y^{(\ell)} = y^*(\theta + \varepsilon Z^{(\ell)})$$

Unbiased estimate of $y^*_\varepsilon(\theta)$ given by

$$\bar{y}_{\varepsilon,M}(\theta) = \frac{1}{M}\sum_{\ell=1}^{M} y^{(\ell)} \,.$$



Supervised learning:

Features X_i, model output $\theta_w = g_w(X_i)$, prediction $y_{\text{pred}} = y^*_\varepsilon(\theta_w)$.

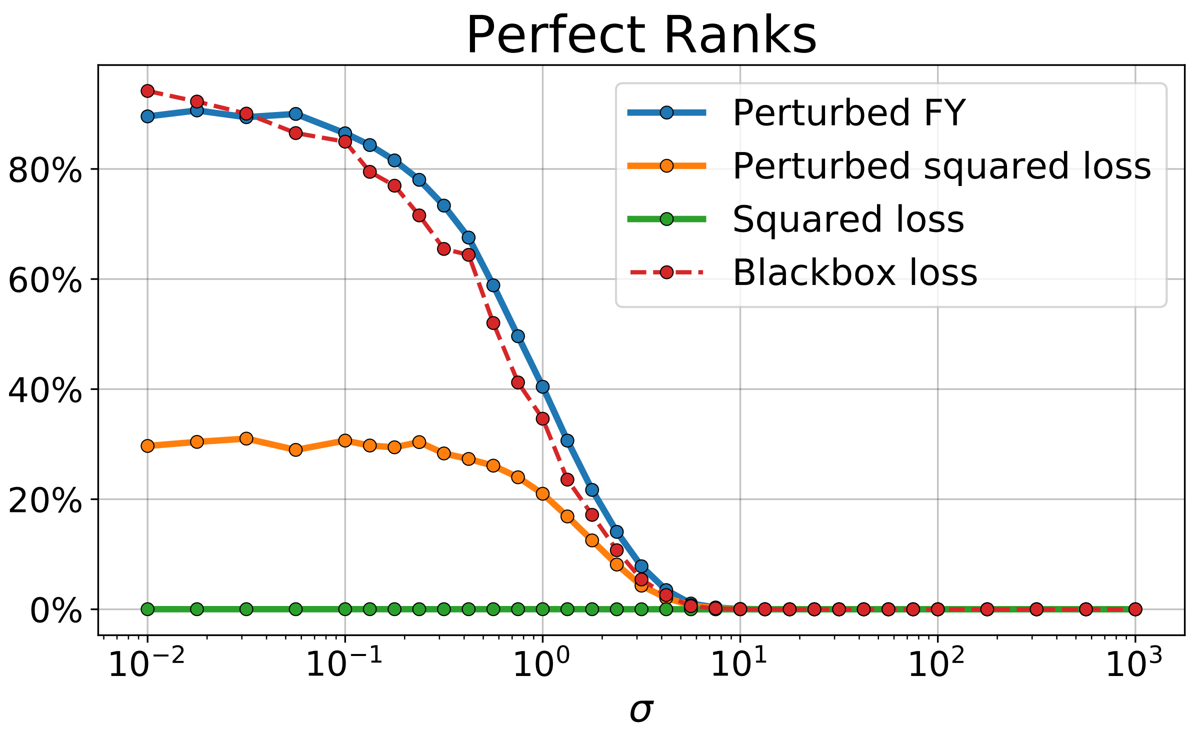
Stochastic gradient in w (doubly stochastic scheme)

$$\nabla_w F_i(w) = \partial_w g_w(X_i) \cdot \Big(\frac{1}{M}\sum_{\ell=1}^{M} y^*(\theta + \varepsilon Z^{(\ell)}) - Y_i\Big) \,.$$

Experiments

Learning to rank: Experiments on 4k instances of 100 vectors to rank.

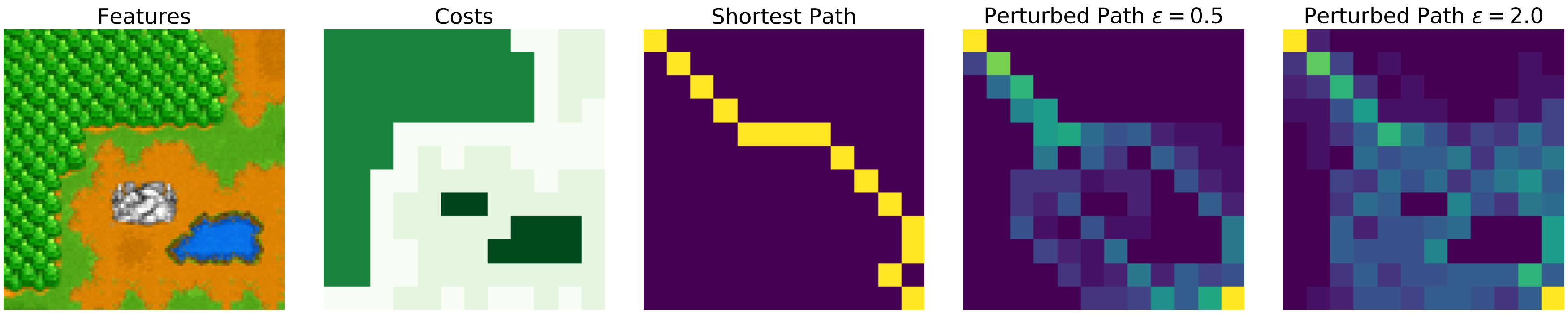
Robustness to noise observed for some tolerated variance



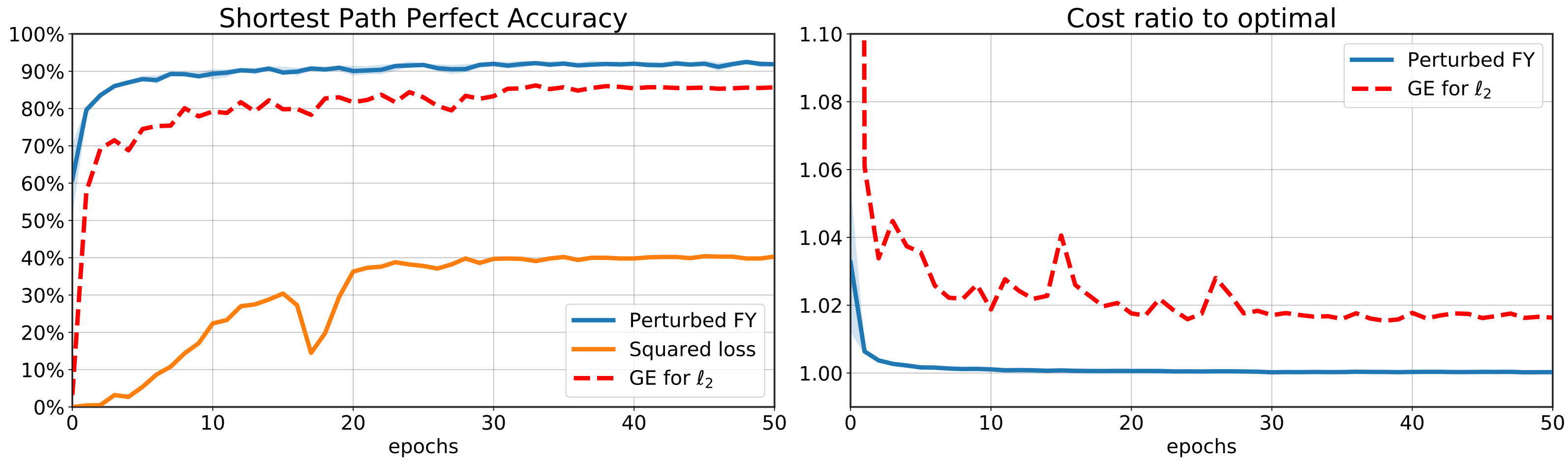
Fenchel-Young loss is convex in w: linear model, possible theoretical analysis.

Experiments

Learning from shortest paths: From 10k examples of Warcraft 96×96 RGB images, representing 12×12 costs, and matrix of shortest paths. (Vlastelica et al. 19)



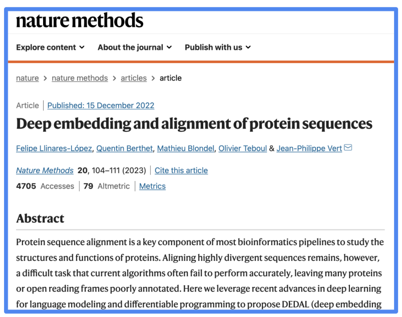
Train a CNN for 50 epochs, to learn costs recovery of optimal paths.



DEDAL

Deep Embedding and Differentiable ALignment

F. Llinares, Q. Berthet, M.Blondel, O. Teboul, J.P. Vert


nature methods

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › nature methods › articles › article

Article | Published: 15 December 2022

Deep embedding and alignment of protein sequences

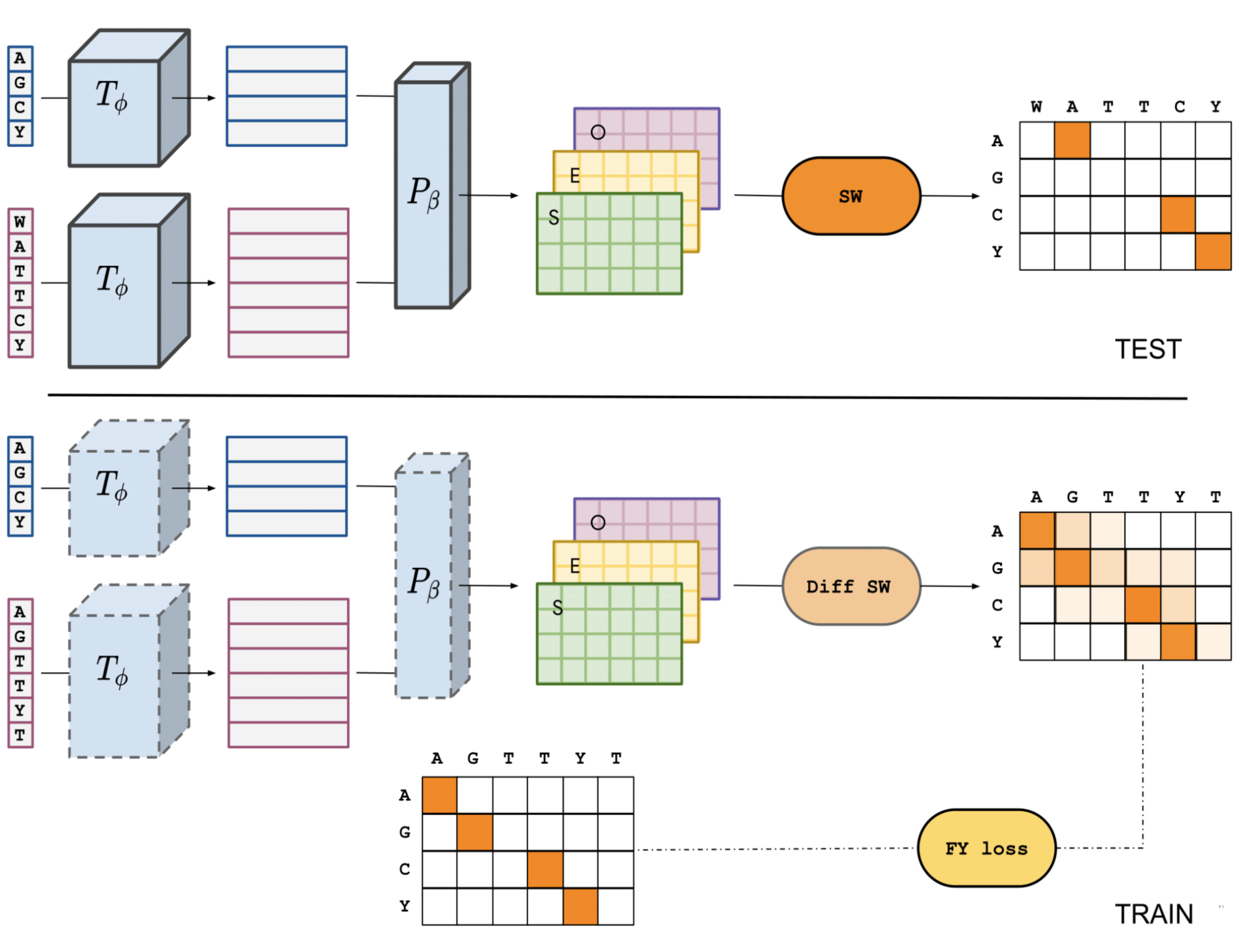
Felipe Llinares-López, Quentin Berthet, Mathieu Blondel, Olivier Teboul & Jean-Philippe Vert

Nature Methods **20**, 104–111 (2023) | Cite this article

4705 Accesses | **79** Altmetric | Metrics

Abstract

Protein sequence alignment is a key component of most bioinformatics pipelines to study the structures and functions of proteins. Aligning highly divergent sequences remains, however, a difficult task that current algorithms often fail to perform accurately, leaving many proteins or open reading frames poorly annotated. Here we leverage recent advances in deep learning for language modeling and differentiable programming to propose DEDAL (deep embedding



- **Deep embedding and alignment of protein sequences**

  Nature methods, 2023

F. Llinares

Q. Berthet

M. Blondel

O. Teboul

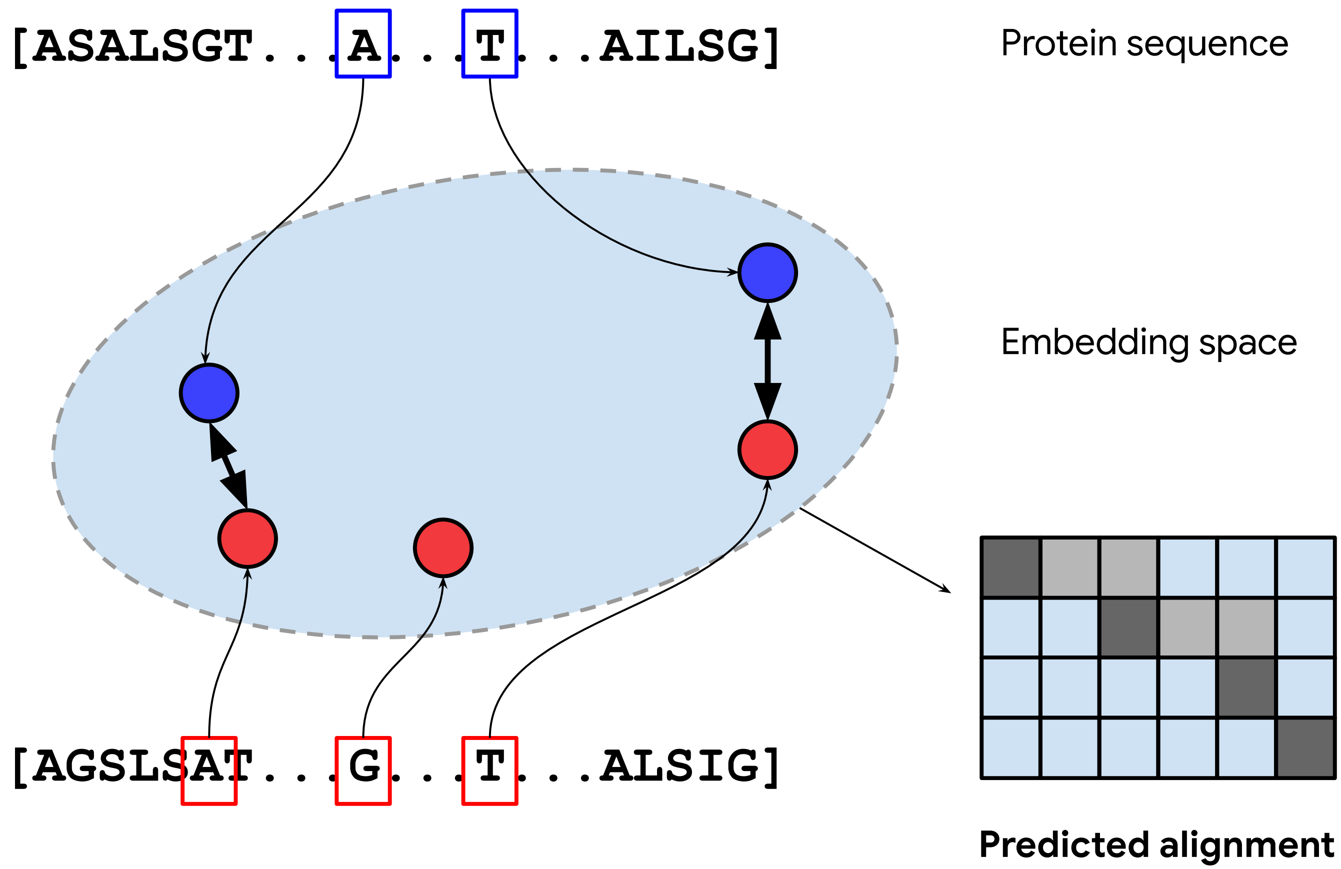
J-P. Vert

- **Deep embedding and alignment of protein sequences**

  Nature methods, 2023

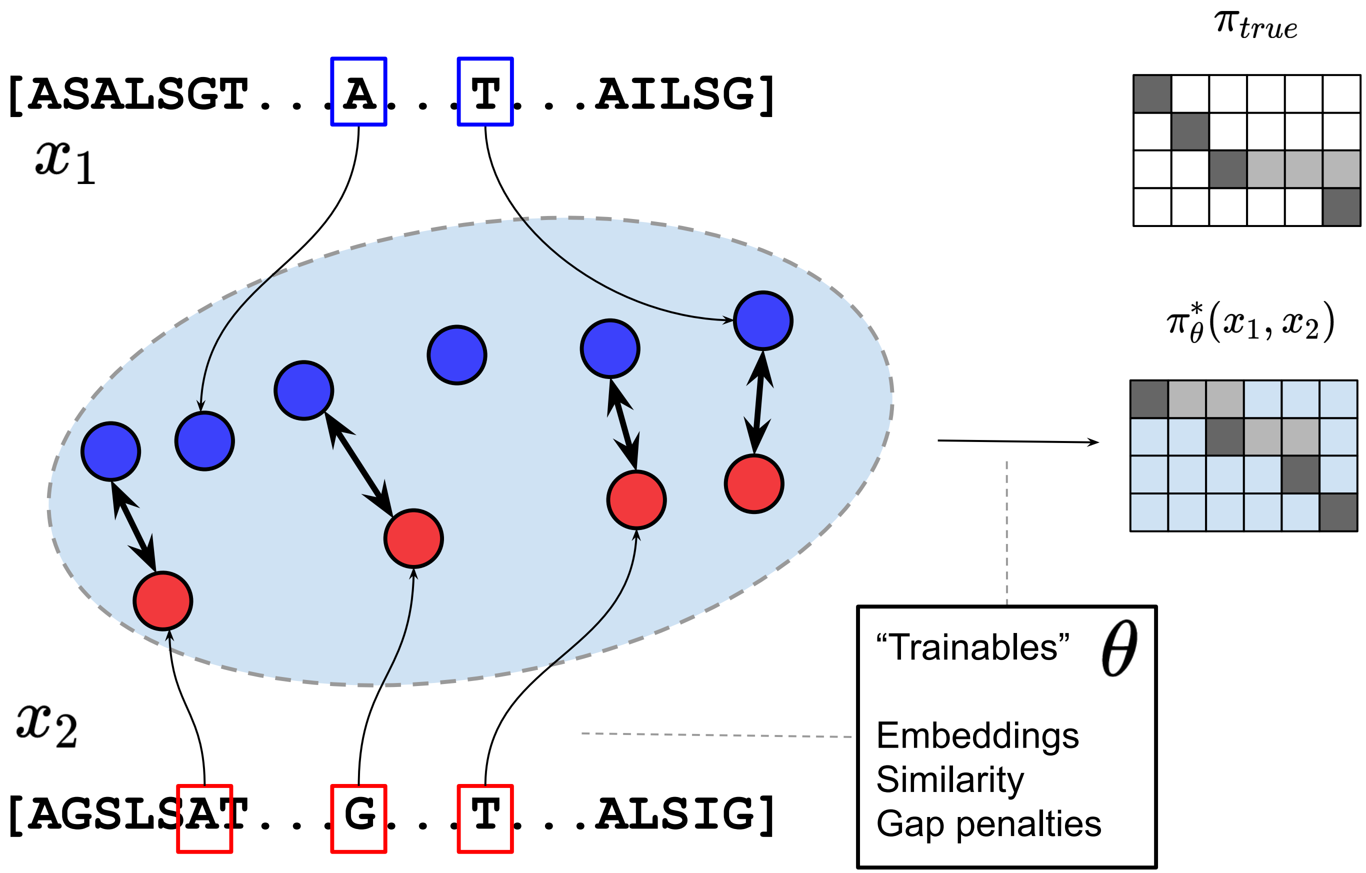
Protein alignment

- Learning character-wise embeddings of protein sequences.
- Using it to compute costs of an alignment problem (dynamic programming).



Protein alignment

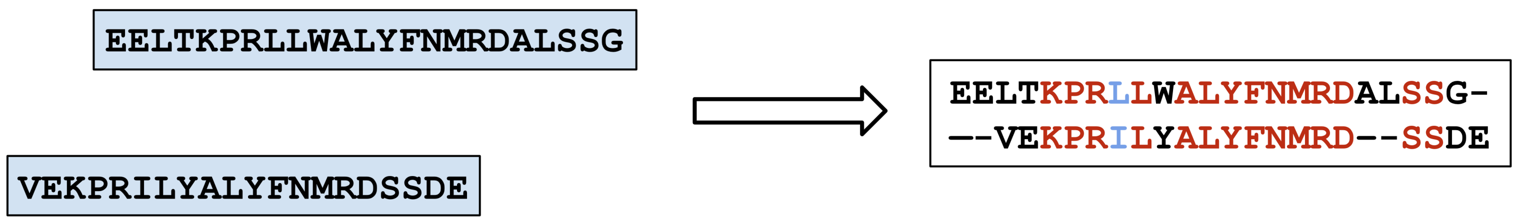
- Learning character-wise embeddings of protein sequences.
- Using it to compute costs of an alignment problem (dynamic programming).



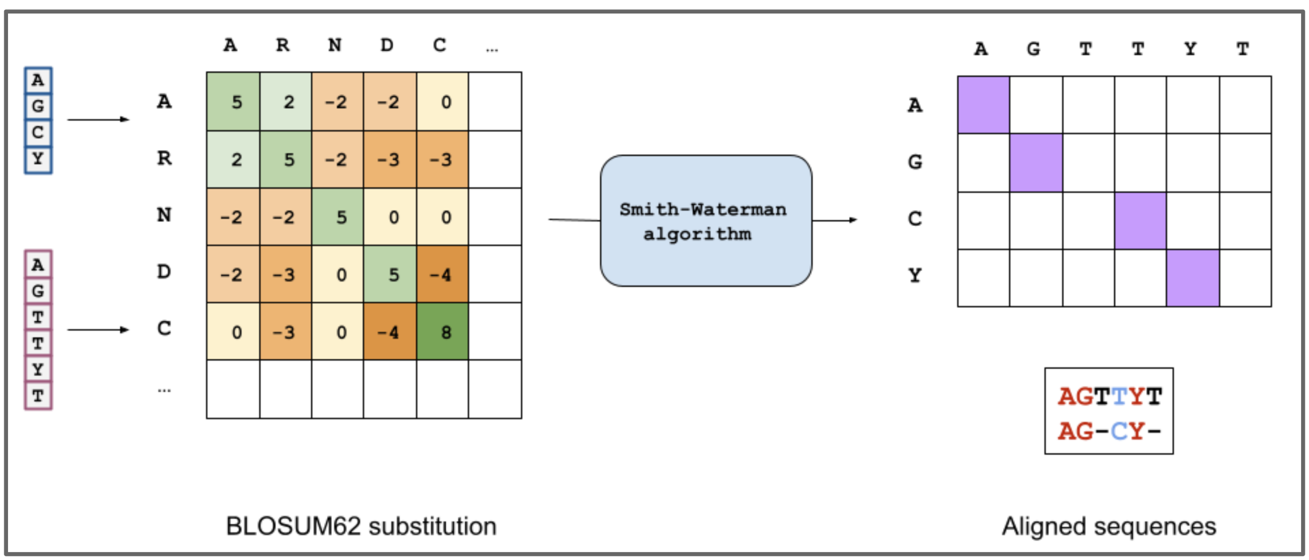
Differentiable alignment

- Given fixed substitution/insertion costs, local alignment problem.
- Smith-Waterman problem solved by DP, to align proteins.
- Non-differentiable solution, introducing perturbations

Alignment as a biologically-plausible sequence similarity measure



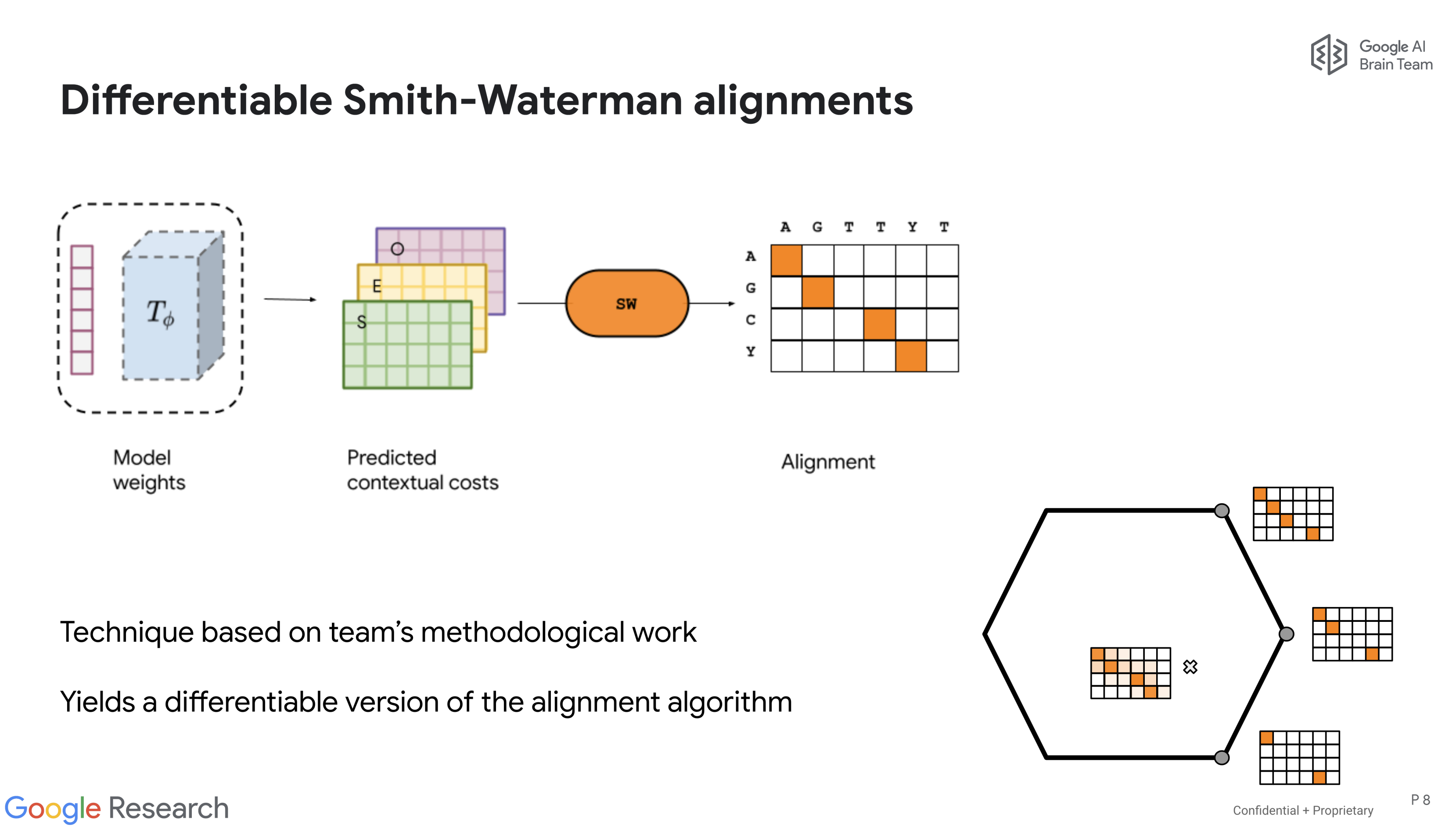
Source: Dr. Vered Caspi



Solved by dynamic program (Smith-Waterman algorithm)

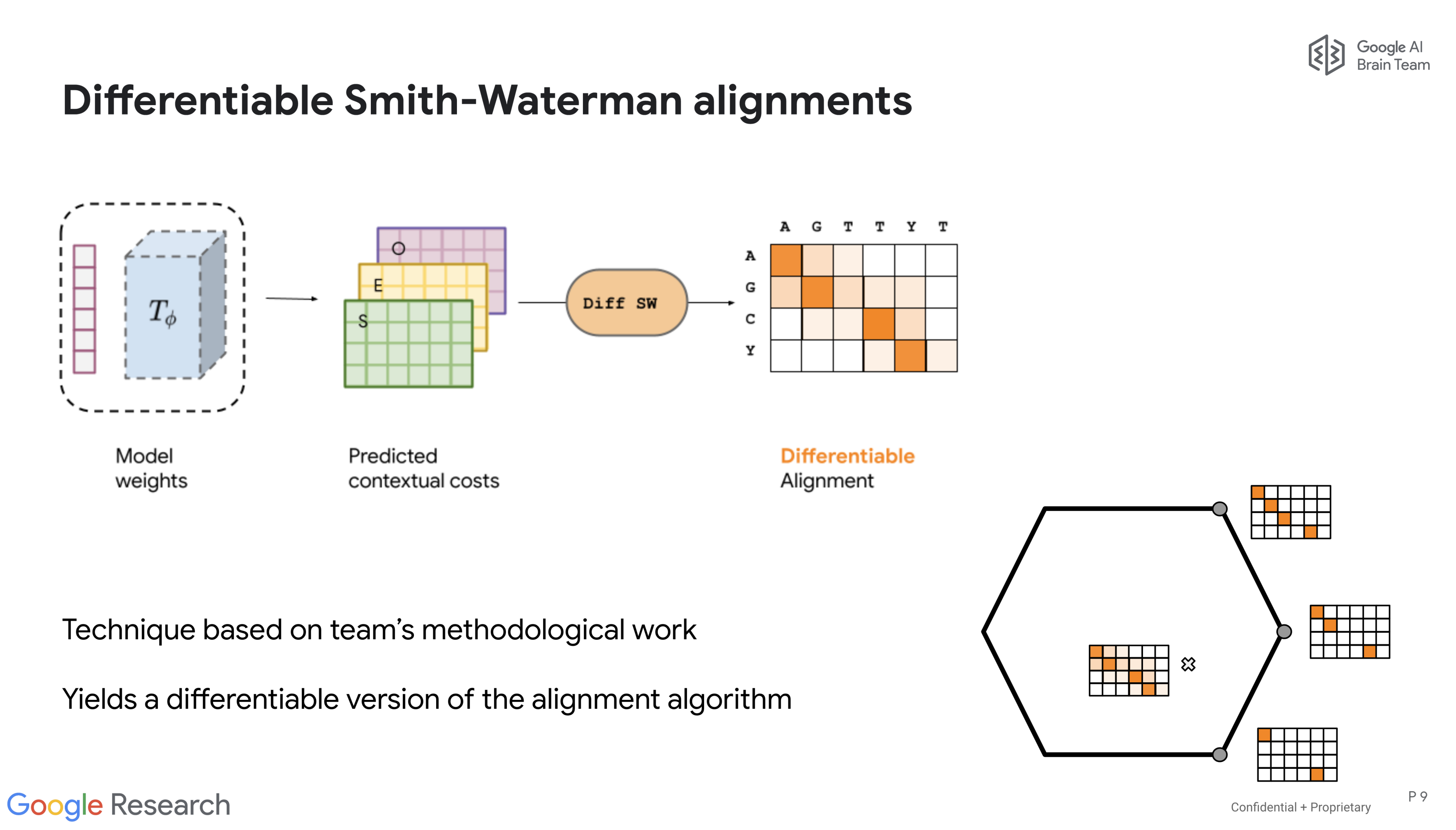
Differentiable alignment

- Given fixed substitution/insertion costs, local alignment problem.
- Smith-Waterman problem solved by DP, to align proteins.
- Non-differentiable solution, introducing perturbations



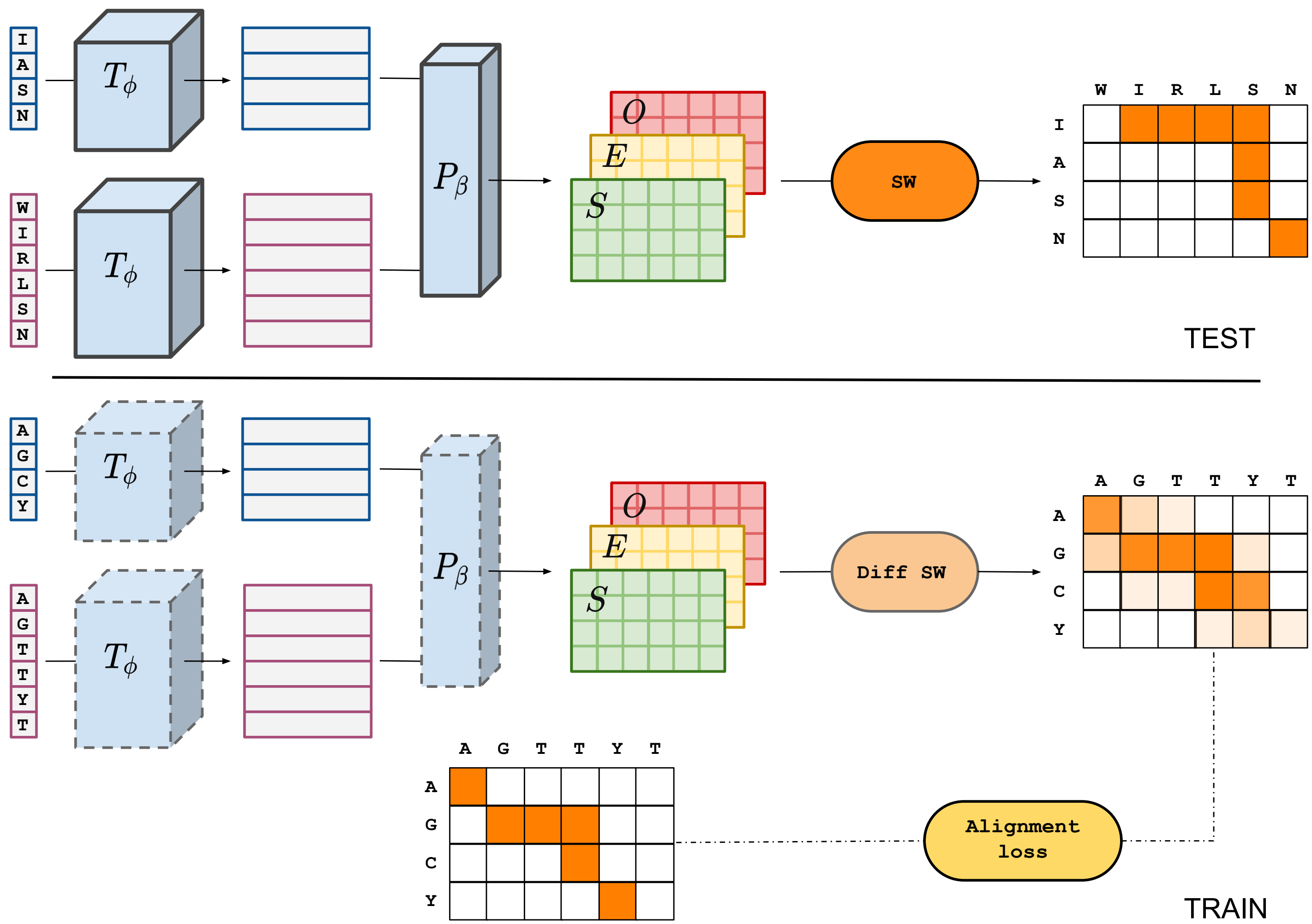
Differentiable alignment

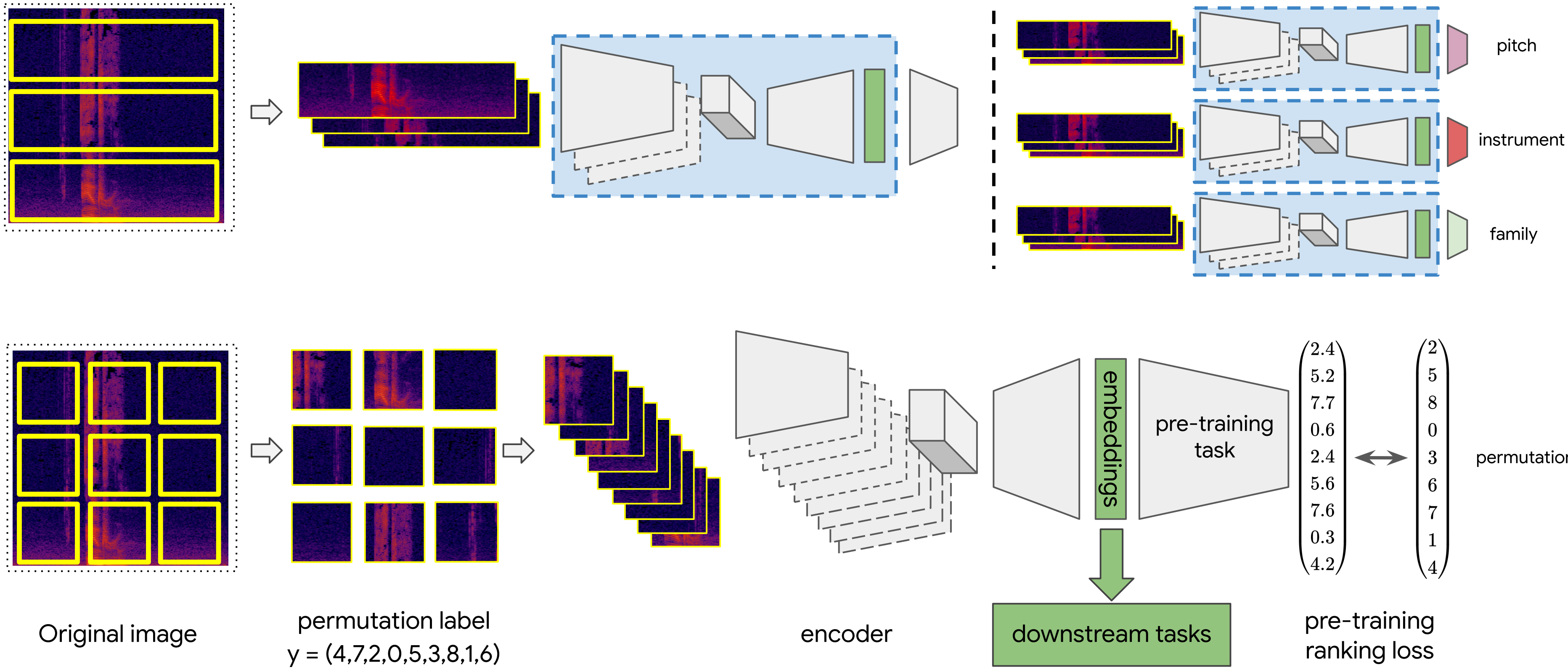
- Given fixed substitution/insertion costs, local alignment problem.
- Smith-Waterman problem solved by DP, to align proteins.
- Non-differentiable solution, introducing perturbations



DEDAL: End-to-end learning to align

- From sequences to embeddings, costs, to perturbed alignments.
- Transformer architecture trained on databases of aligned proteins.

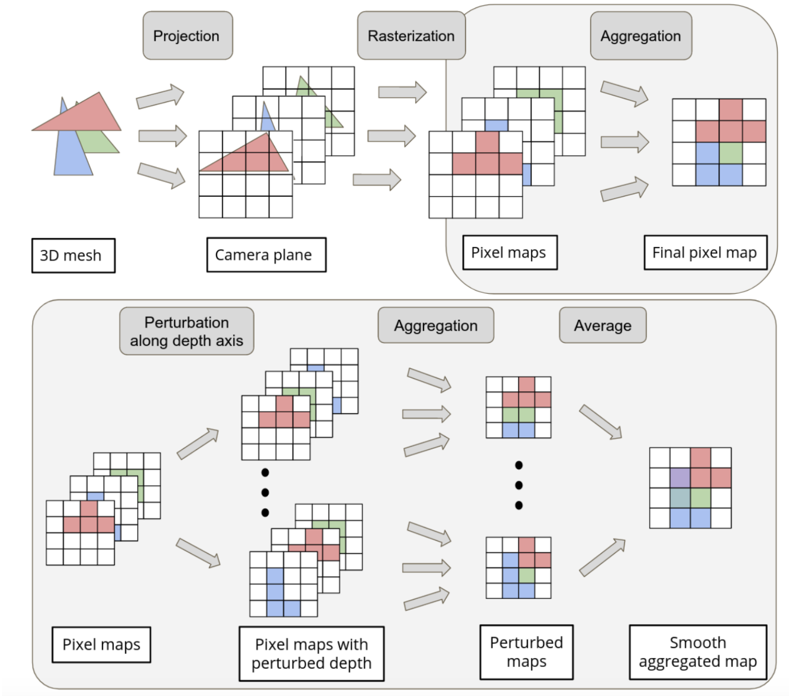




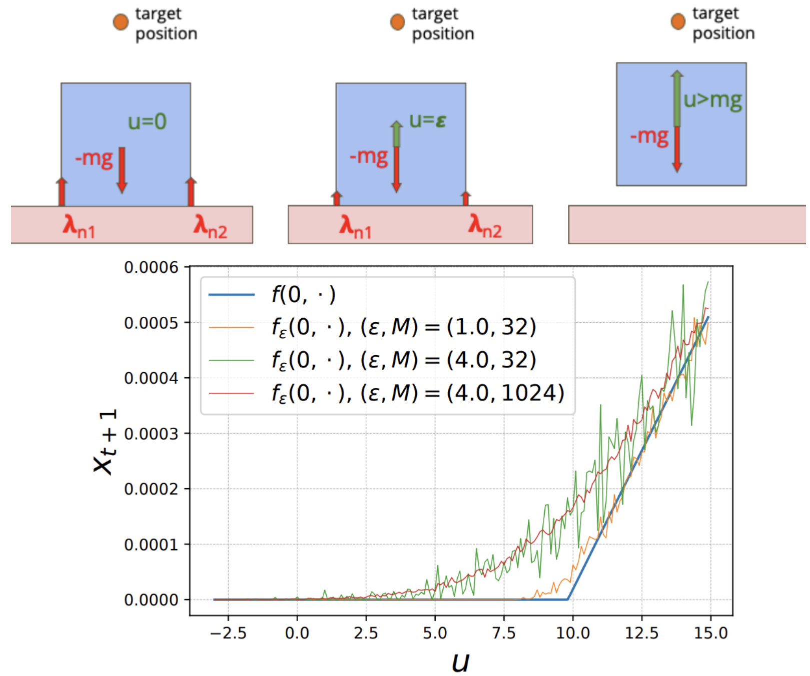
- **Self-supervised learning of audio representations from permutations with differentiable ranking**

A. Carr, Q. Berthet, M. Blondel, O. Teboul, N. Zeghidour

IEEE Signal Processing Letters, 2021



Rendering Le Lidec et al. (21)



Optimal Control Le Lidec et al. (22)

- **Applications**:

  Digital pathology Thandiackal et al. (22), Patch selection Cordonnier et al. (21), Video Token Selection Wang et al. (22), . . .

- **Algorithmic improvements**: parralelized optimization Dubois-Taine et al. (22)

L. Stewart

F. Bach

F. Llinares

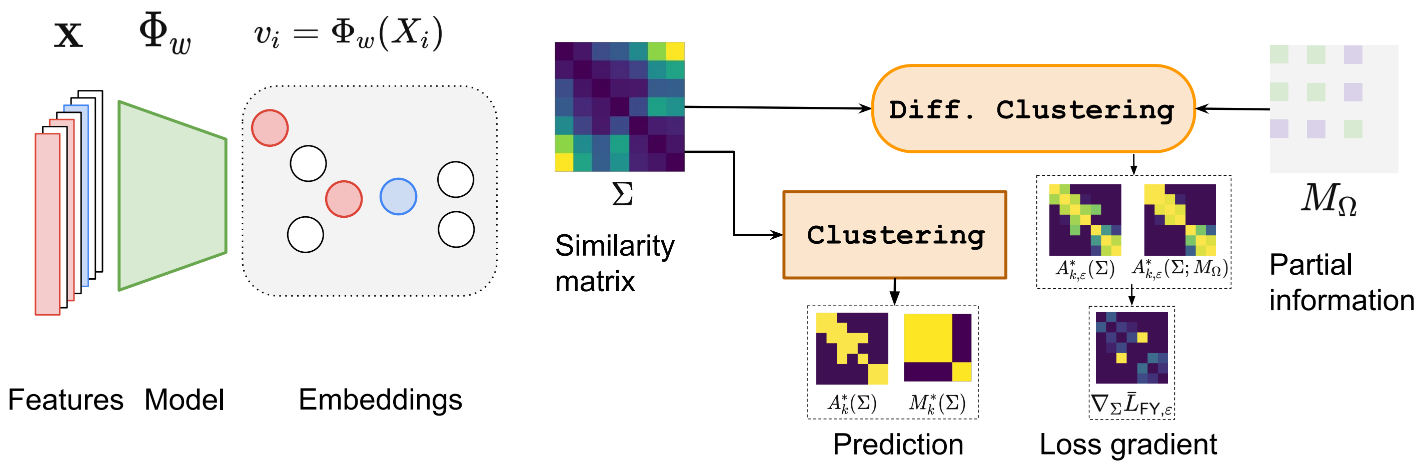
Q. Berthet

- **Differentiable Clustering with Perturbed Spanning Forests**

  Preprint, 2023

Differentiable Clustering

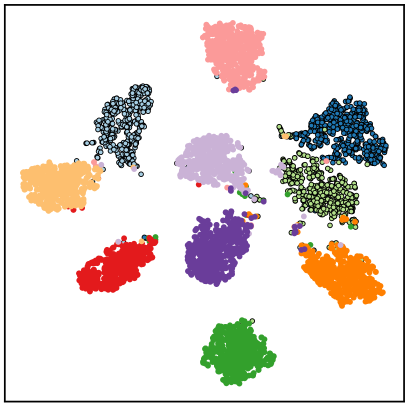
- Transformer architecture trained on databases of aligned proteins.



- Semi-supervised clustering.

- Discovery of held-out classes.

- **Presentation** and **poster** today.



Mahalo!