Adaptive Perturbation-Based Gradient Estimation for Discrete Latent Variable Models

Pasquale Minervini
p.minervini@ed.ac.uk
www.neuralnoise.com
@pminervini

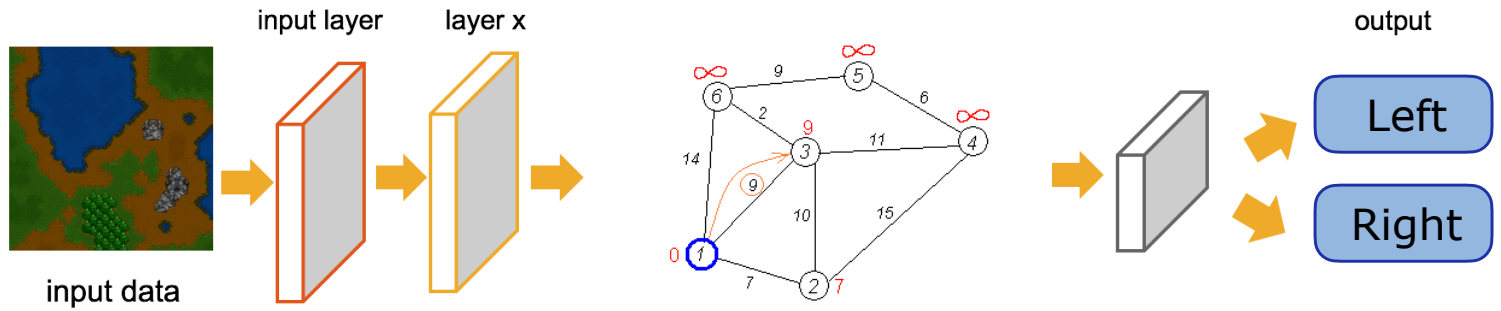
Mathias Niepert

Luca Franceschi

Discrete Representations in Deep Neural Models

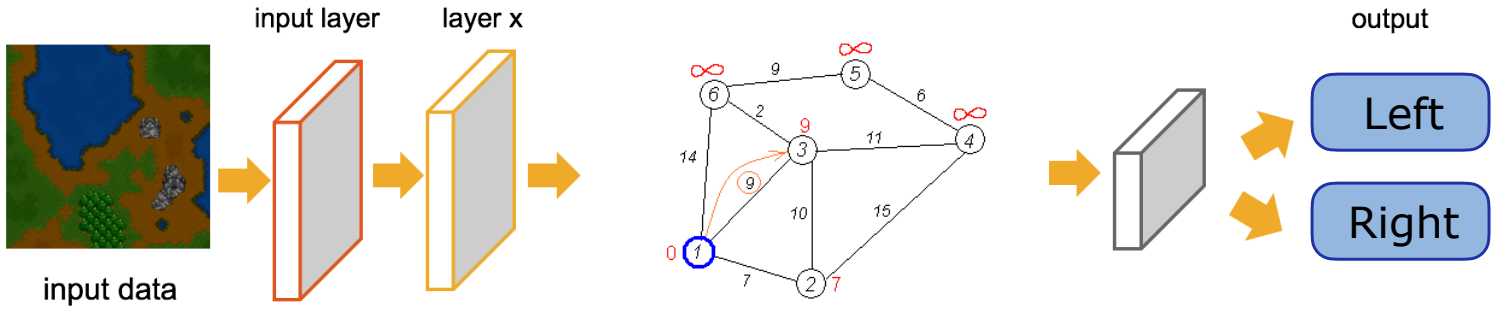
Discrete Representations in Deep Neural Models

- Combination of *discrete optimisation* algorithms (e.g., an ILP solver, Dijkstra's shortest path algorithm) and arbitrary neural modules

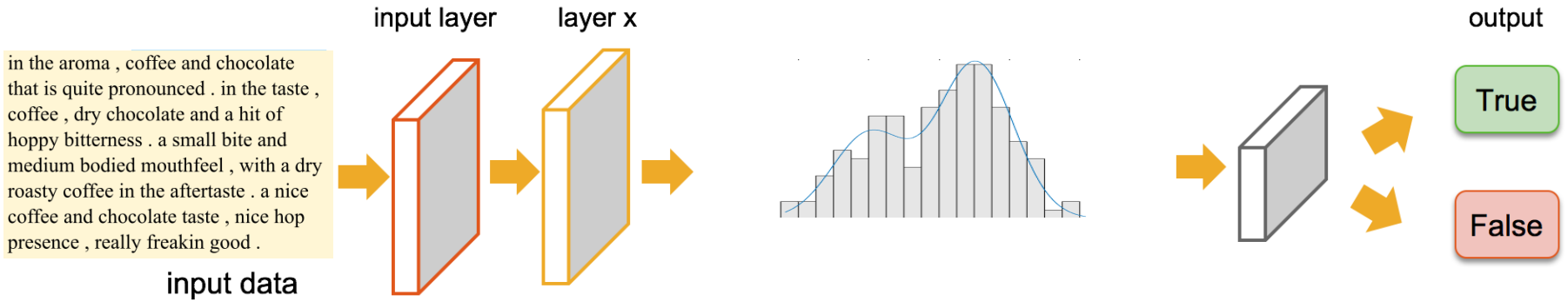


Discrete Representations in Deep Neural Models

- Combination of *discrete optimisation* algorithms (e.g., an ILP solver, Dijkstra's shortest path algorithm) and arbitrary neural modules

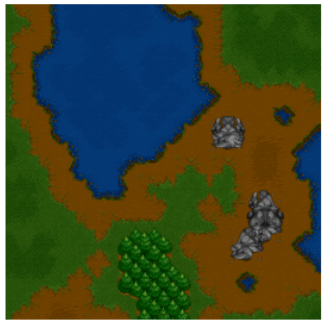


- Combination of *complex discrete probability distributions* (e.g., discrete attention distributions over the input, graph-structured representations) with arbitrary neural modules



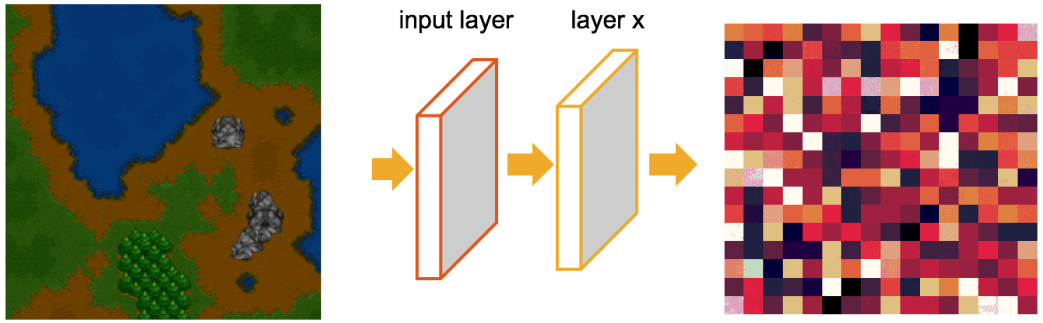
Learning to Plan

1. A neural model assigns weights to cells in a map



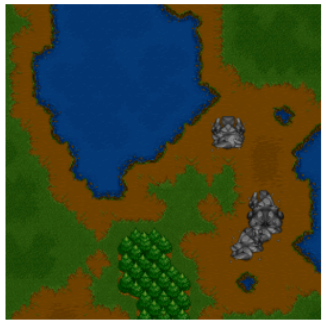
Learning to Plan

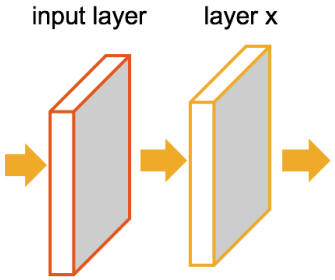
1. A neural model assigns weights to cells in a map

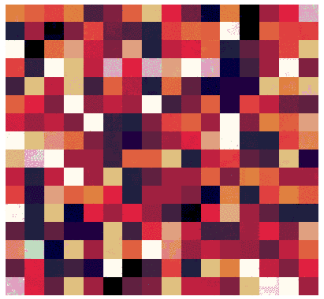


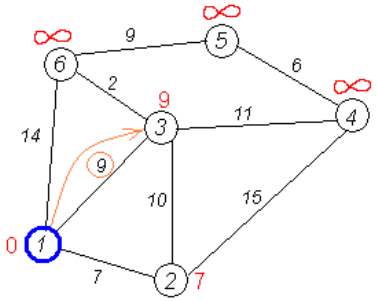
Learning to Plan

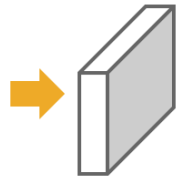
1. A neural model assigns weights to cells in a map
2. Weights are used as input to a shortest path algorithm







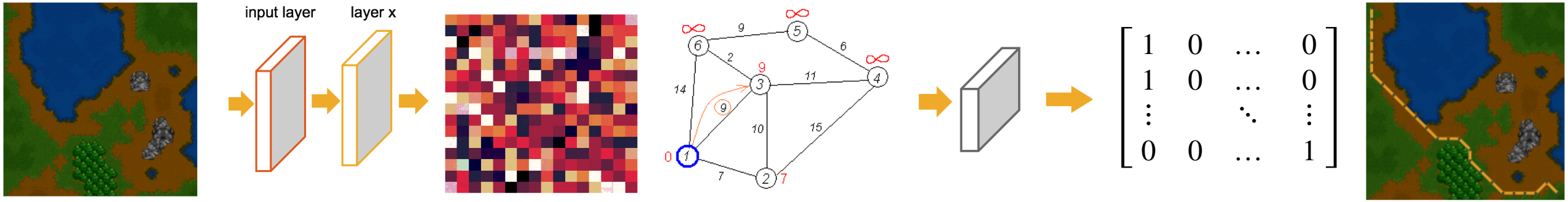




$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Learning to Plan

1. A neural model assigns weights to cells in a map
2. Weights are used as input to a shortest path algorithm
3. The solver returns a shortest path which is then used by the downstream neural model for e.g., deciding the next action, or computing a loss function



Learning to Explain

1. A neural model assigns weights to words in the input

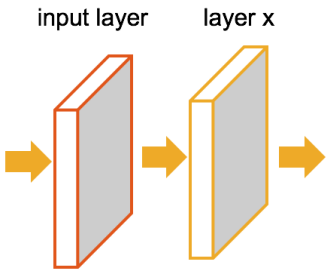
in the aroma , coffee and chocolate that is quite pronounced . in the taste , coffee , dry chocolate and a hit of hoppy bitterness . a small bite and medium bodied mouthfeel , with a dry roasty coffee in the aftertaste . a nice coffee and chocolate taste , nice hop presence , really freakin good .

input text

Learning to Explain

1. A neural model assigns weights to words in the input
2. Weights are used as parameters of a discrete distributions with a k-subset constraint

in the aroma , coffee and chocolate that is quite pronounced . in the taste , coffee , dry chocolate and a hit of hoppy bitterness . a small bite and medium bodied mouthfeel , with a dry roasty coffee in the aftertaste . a nice coffee and chocolate taste , nice hop presence , really freakin good .



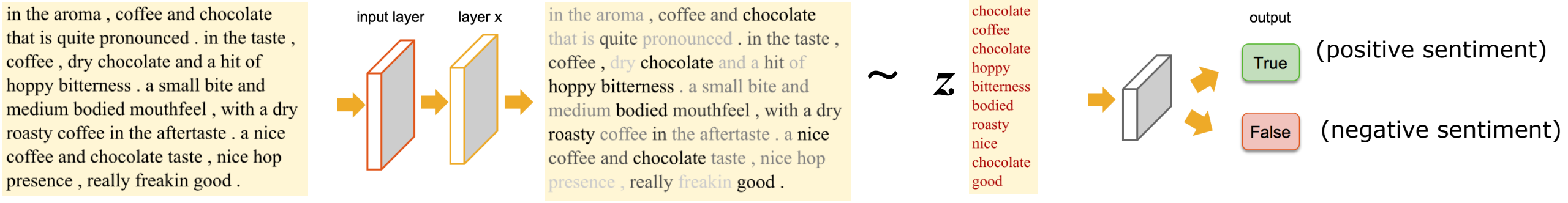
in the aroma , coffee and chocolate that is quite pronounced . in the taste , coffee , dry chocolate and a hit of hoppy bitterness . a small bite and medium bodied mouthfeel , with a dry roasty coffee in the aftertaste . a nice coffee and chocolate taste , nice hop presence , really freakin good .

input text

Weight θ assigned to each token

Learning to Explain

1. A neural model assigns weights to words in the input
2. Weights are used as parameters of a discrete distributions with a k-subset constraint
3. A subset of k input words is sampled from this distribution, and used by a classification model to produce a prediction



input text

Weight θ assigned to each token

Sample discretely *exactly k* tokens

Combinatorial Solvers

We consider the problem of back-propagating through a combinatorial solver, i.e., a (black-box) component that solves the following problem:

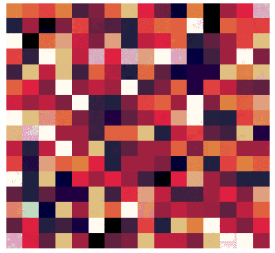
Combinatorial Solvers

We consider the problem of back-propagating through a combinatorial solver, i.e., a (black-box) component that solves the following problem:

E.g., shortest path

$$\theta \mapsto \text{solve}(\theta)$$

Input weights

$\theta =$ 

$$\mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Combinatorial Solvers

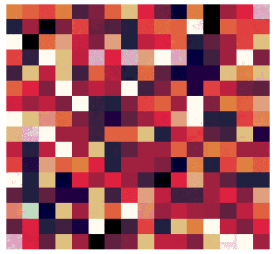
We consider the problem of back-propagating through a combinatorial solver, i.e., a (black-box) component that solves the following problem:

E.g., shortest path

$\theta \mapsto \text{solve}(\theta)$ such that $\text{solve}(\theta) = \arg\max_{z \in \mathcal{C}} \text{score}(\theta, z)$

Input weights

E.g., the set of all paths

$\theta =$ 

$\mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$

Combinatorial Solvers

We consider the problem of back-propagating through a combinatorial solver, i.e., a (black-box) component that solves the following problem:

E.g., shortest path

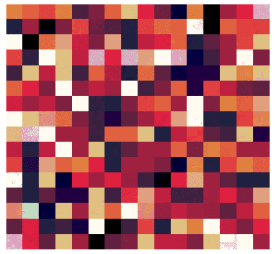
$\theta \mapsto \text{solve}(\theta)$ such that $\text{solve}(\theta) = \arg\max_{z \in \mathcal{C}} \text{score}(\theta, z)$

Input weights

E.g., the set of all paths

with $\text{score}(\theta, z) = \theta^\top z$

Linear score function

$\theta =$ 

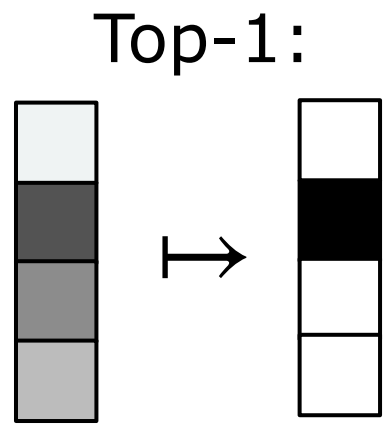
$\mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$

Combinatorial Solvers

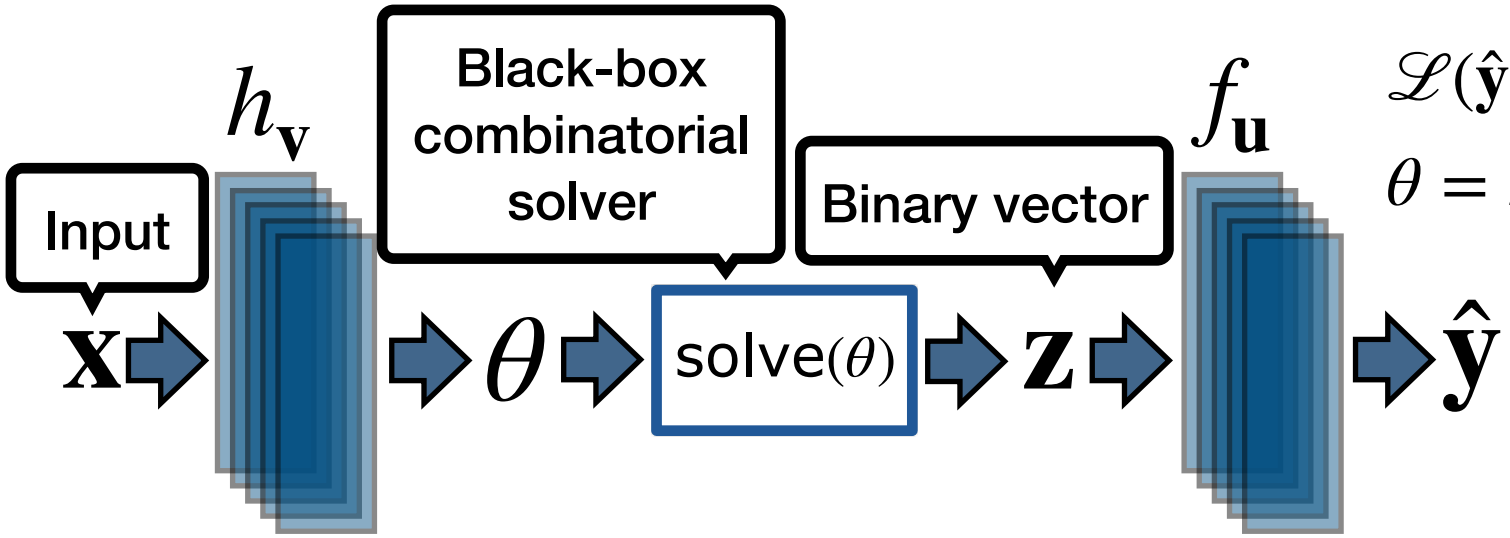
We consider the problem of back-propagating through a combinatorial solver, i.e., a (black-box) component that solves the following problem:

Examples of combinatorial solvers:

- Top-k functions
- Shortest path algorithms
- Maximum spanning three algorithms
- Scheduling algorithms
- Supply chain optimisation algorithms
- .. and many more



Backpropagating through Combinatorial Solvers

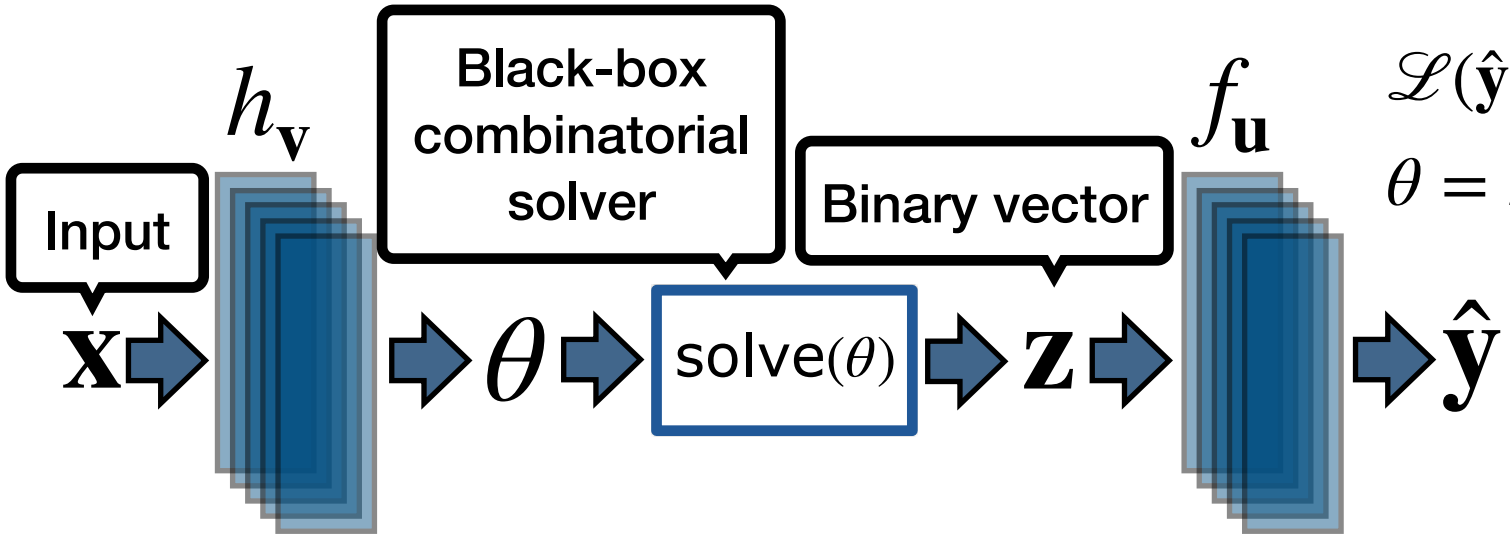


$$\mathscr{L}(\hat{\mathbf{y}}, \mathbf{y}) = \mathscr{L}(f_{\mathbf{u}}(\mathbf{z}), \mathbf{y}) \quad \text{where}$$

$$\theta = h_{\mathbf{v}}(\mathbf{x}) \in \mathbb{R}^n, \quad \mathbf{z} = \text{solve}(\theta) \in \{0,1\}^n$$

$$\theta = \quad \mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Backpropagating through Combinatorial Solvers



$$\mathscr{L}(\hat{\mathbf{y}}, \mathbf{y}) = \mathscr{L}(f_{\mathbf{u}}(\mathbf{z}), \mathbf{y}) \quad \text{where}$$

$$\theta = h_{\mathbf{v}}(\mathbf{x}) \in \mathbb{R}^n, \quad \mathbf{z} = \text{solve}(\theta) \in \{0,1\}^n$$

$$\theta = \quad \mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Problem: How do we estimate the gradient of the loss $\nabla_{\theta}\mathscr{L}(\hat{\mathbf{y}}, \mathbf{y})$?

Combinatorial Solvers and Exponential Families

Implicit MLE is based on observing that calling a **combinatorial solver** $\mathbf{z} \leftarrow \text{solve}(\theta)$ on some **input weights** $\theta \in \mathbb{R}^n$ to obtain a **discrete solution** $\mathbf{z} \in \{0,1\}^n$ is equivalent to computing the MAP state of an **exponential family distribution** $p(\mathbf{z}; \theta)$:

e.g., top-k, shortest path algorithm, maximum spanning tree algorithm..

$$\mathbf{z} \leftarrow \text{solve}(\theta)$$

Combinatorial Solvers and Exponential Families

Implicit MLE is based on observing that calling a **combinatorial solver** $\mathbf{z} \leftarrow \text{solve}(\theta)$ on some **input weights** $\theta \in \mathbb{R}^n$ to obtain a **discrete solution** $\mathbf{z} \in \{0,1\}^n$ is equivalent to computing the MAP state of an **exponential family distribution** $p(\mathbf{z}; \theta)$:

e.g., top-k, shortest path algorithm, maximum spanning tree algorithm..

Maximum a Posteriori (MAP) estimation of $\mathbf{z}$ wrt. $p(\cdot\,; \theta)$

$$\mathbf{z} \leftarrow \text{solve}(\theta) \quad \equiv \quad \mathbf{z} \leftarrow \arg\max_{\overline{\mathbf{z}}} p(\overline{\mathbf{z}}; \theta)$$

Combinatorial Solvers and Exponential Families

Implicit MLE is based on observing that calling a **combinatorial solver** $\mathbf{z} \leftarrow \text{solve}(\theta)$ on some **input weights** $\theta \in \mathbb{R}^n$ to obtain a **discrete solution** $\mathbf{z} \in \{0,1\}^n$ is equivalent to computing the MAP state of an **exponential family distribution** $p(\mathbf{z};\theta)$:

e.g., top-k, shortest path algorithm, maximum spanning tree algorithm..

Maximum a Posteriori (MAP) estimation of $\mathbf{z}$ wrt. $p(\cdot\,;\theta)$

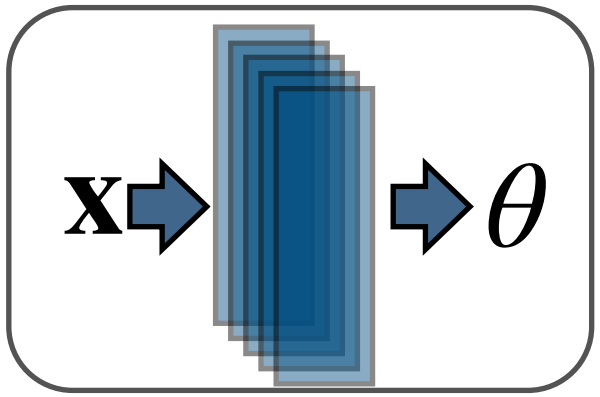
$$\mathbf{z} \leftarrow \text{solve}(\theta) \quad \equiv \quad \mathbf{z} \leftarrow \arg\max_{\overline{\mathbf{z}}} p(\overline{\mathbf{z}};\theta)$$

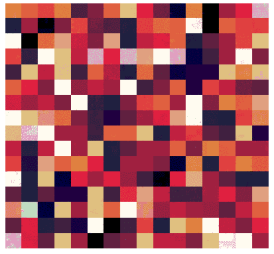
Space of solutions

$$\text{with} \quad p(\mathbf{z};\theta) = \begin{cases} \exp(\langle \mathbf{z}, \theta \rangle) - Z(\theta) & \text{if } \mathbf{z} \in \mathscr{C} \\ 0 & \text{otherwise} \end{cases}$$

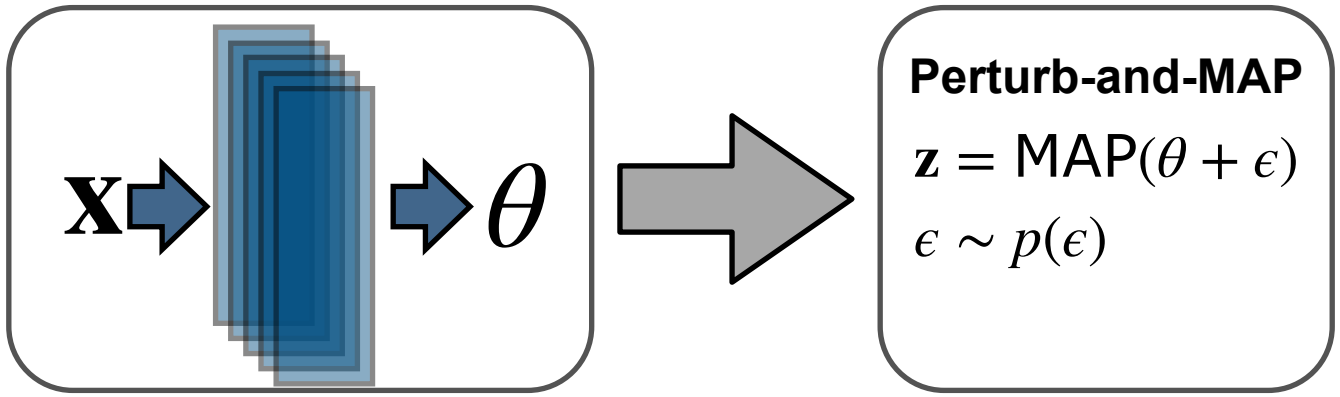
Exponential family distribution parameterised by θ

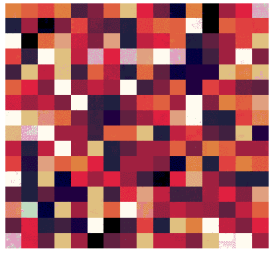
Implicit MLE in a nutshell



$\theta =$ 

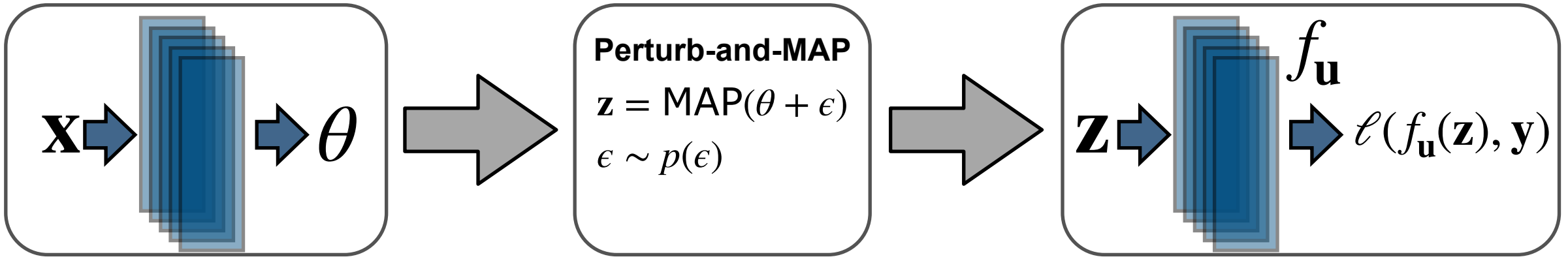
Implicit MLE in a nutshell



$$\theta =$$ 

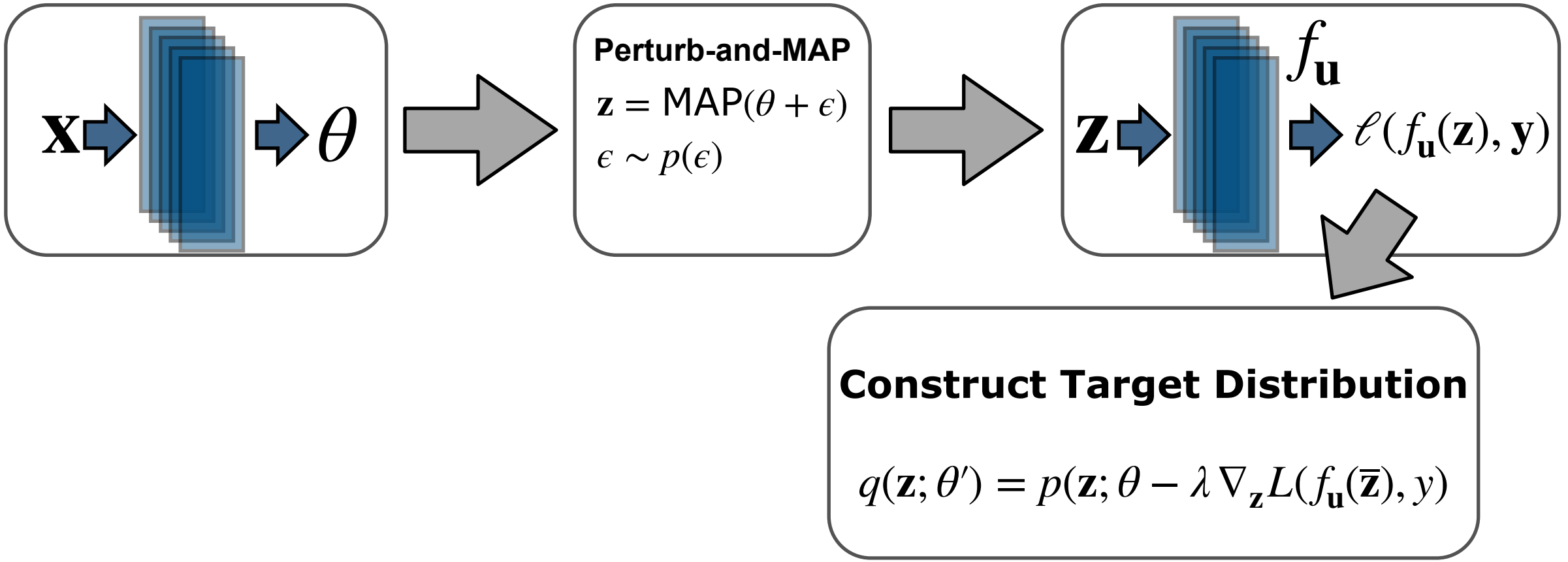
$$\mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Implicit MLE in a nutshell

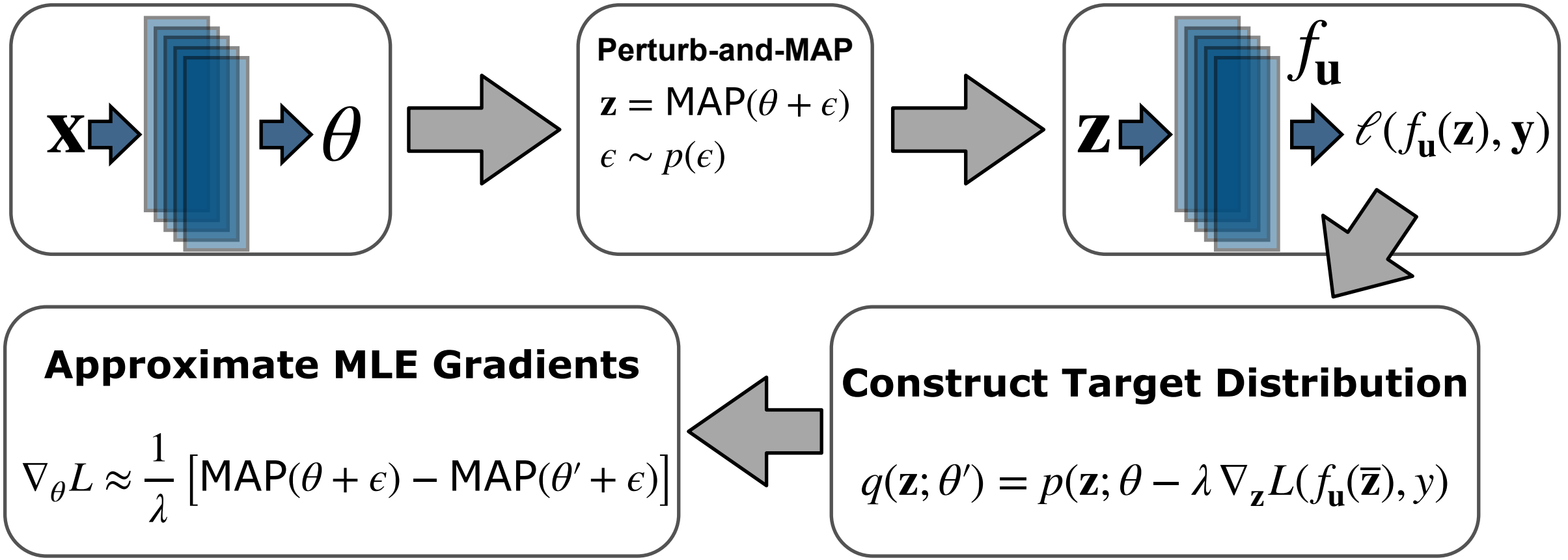


$$\theta = \quad \mathbf{z} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Implicit MLE in a nutshell



Implicit MLE in a nutshell



Constructing the Target Distribution

$$q(\mathbf{z};\theta') = p(\mathbf{z};\theta - \lambda \nabla_{\mathbf{z}} L(f_{\mathbf{u}}(\bar{\mathbf{z}}), y)$$

How do we select the perturbation size λ?

Constructing the Target Distribution

$$q(\mathbf{z};\theta') = p(\mathbf{z};\theta - \lambda \nabla_{\mathbf{z}} L(f_{\mathbf{u}}(\bar{\mathbf{z}}), y)$$

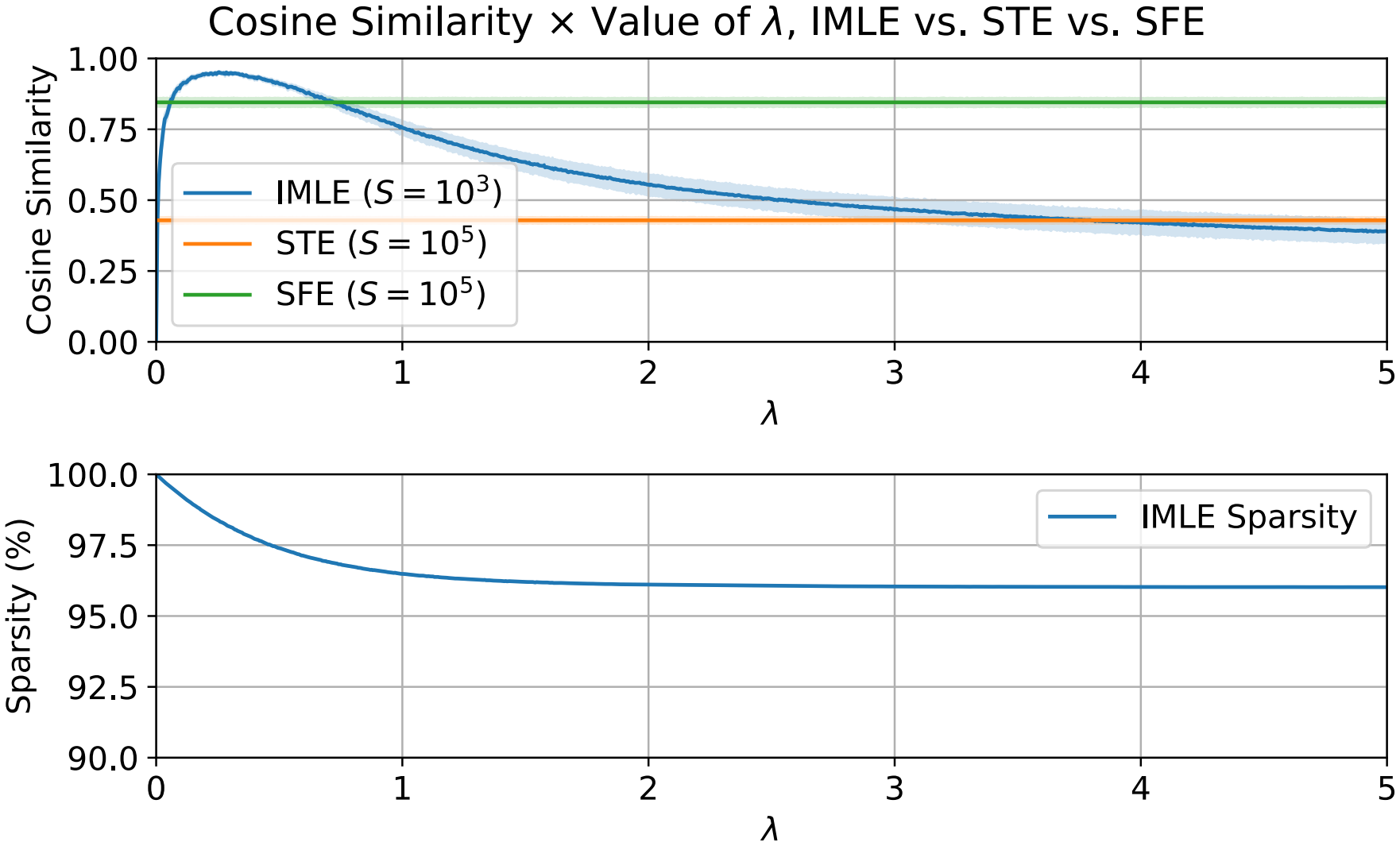
How do we select the perturbation size λ?

If $\lambda \approx 0$, the IMLE estimate of $\nabla_\theta \mathscr{L}(\hat{\mathbf{y}}, \mathbf{y})$ is $\mathbf{0}$

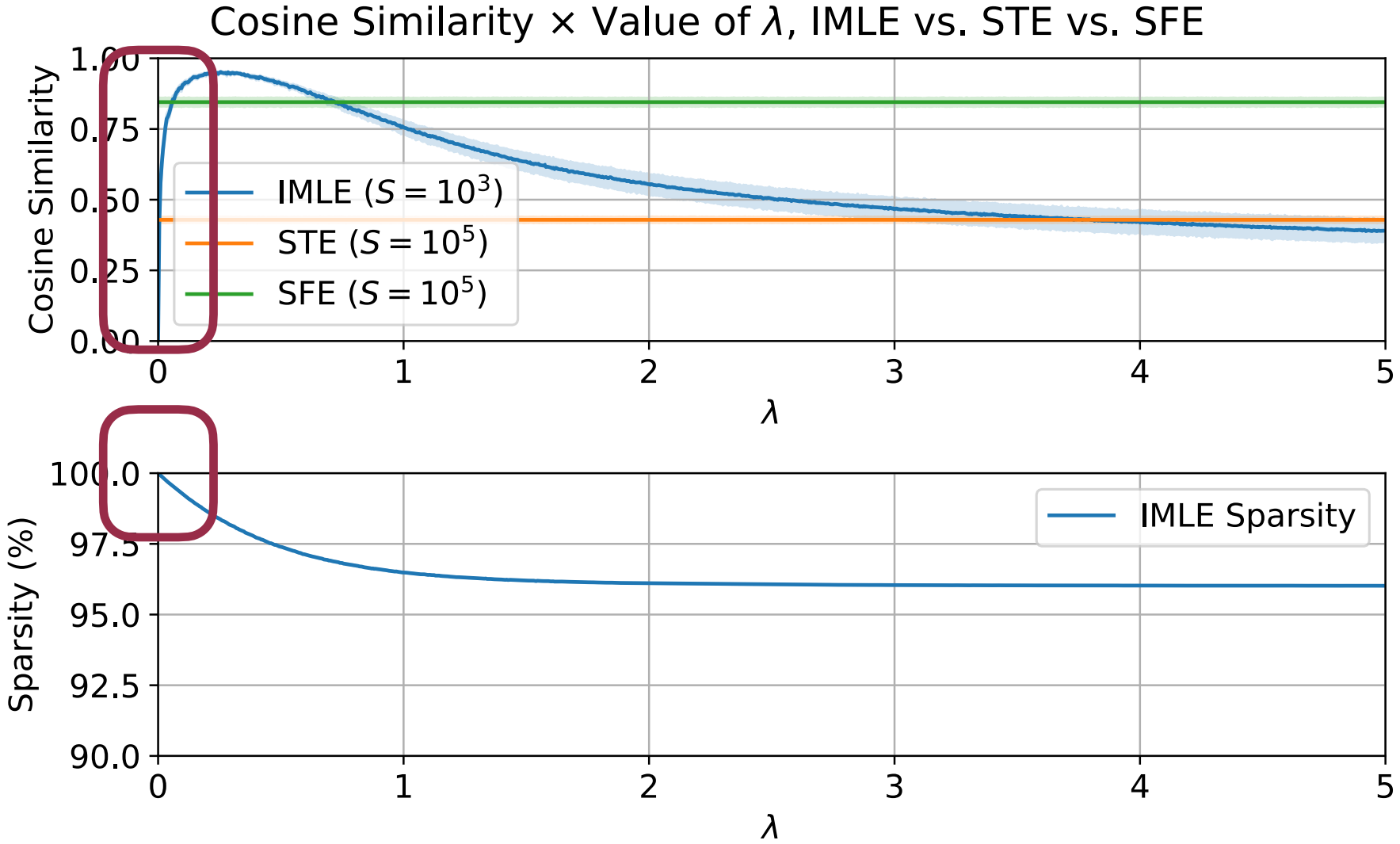
If $\lambda \gg 0$, the IMLE estimate of $\nabla_\theta \mathscr{L}(\hat{\mathbf{y}}, \mathbf{y})$ is **highly biased**

How can we automatically select λ during training?

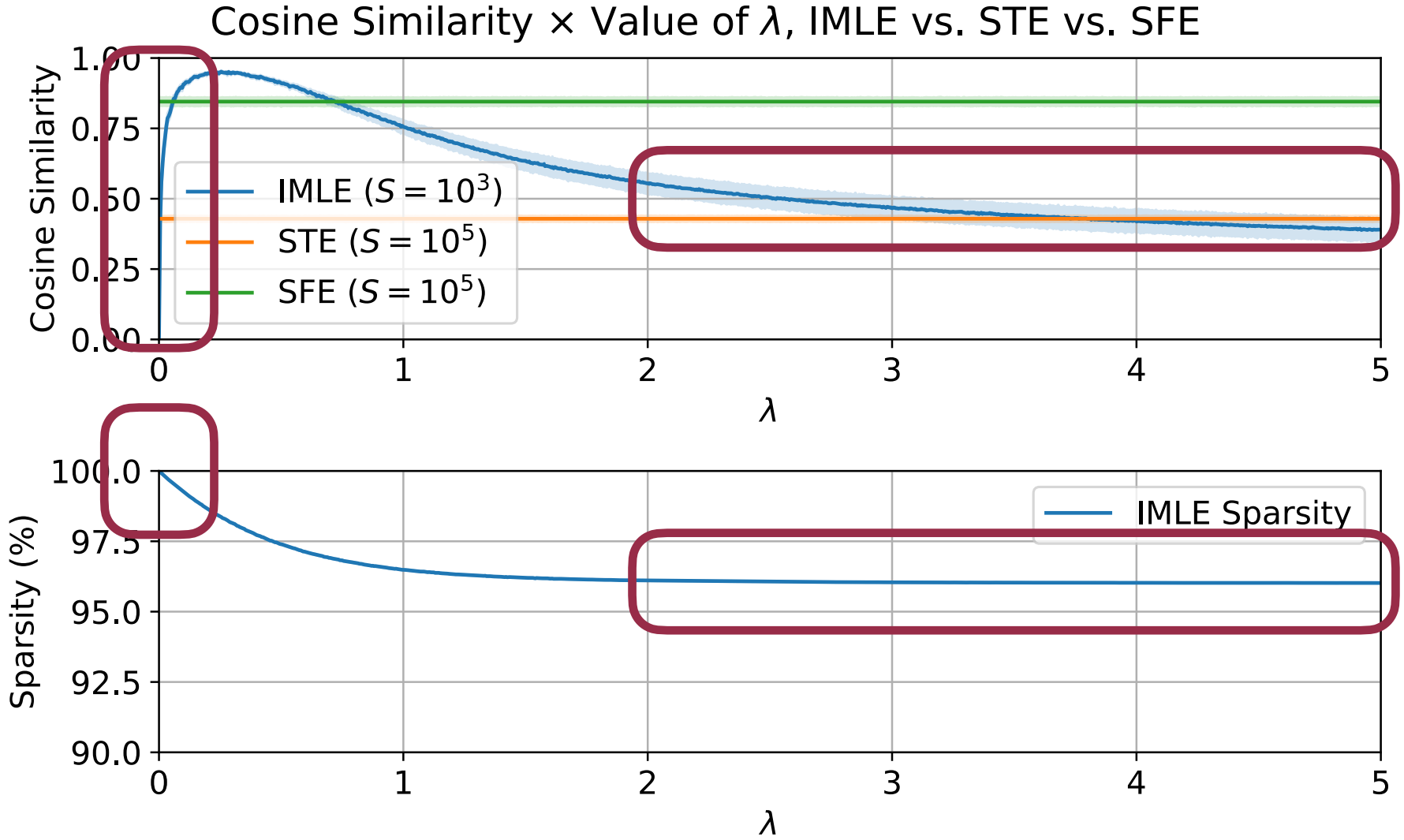
Impact of the Perturbation Size λ



Impact of the Perturbation Size λ



Impact of the Perturbation Size λ



Automatically Selecting λ

Very simple idea:

Automatically Selecting λ

Very simple idea:

- Initialise $\lambda \leftarrow 0$

Automatically Selecting λ

Very simple idea:

- Initialise $\lambda \leftarrow 0$
- During training, **adaptively change** λ until the gradient estimates satisfy some **desired sparsity criteria**

Relationships with Finite Difference Methods

Single-sample IMLE gradient estimate:

$$\nabla_\theta L \approx \frac{1}{\lambda}\left[\text{MAP}(\theta+\epsilon) - \text{MAP}(\theta - \lambda \nabla_{\mathbf{z}} L + \epsilon)\right]$$

Can be seen as a **forward** (one-sided) **finite difference approximation** in the form $f'(x) \approx \left[f(x+h) - f(x)\right]/h$.

Relationships with Finite Difference Methods

Single-sample IMLE gradient estimate:

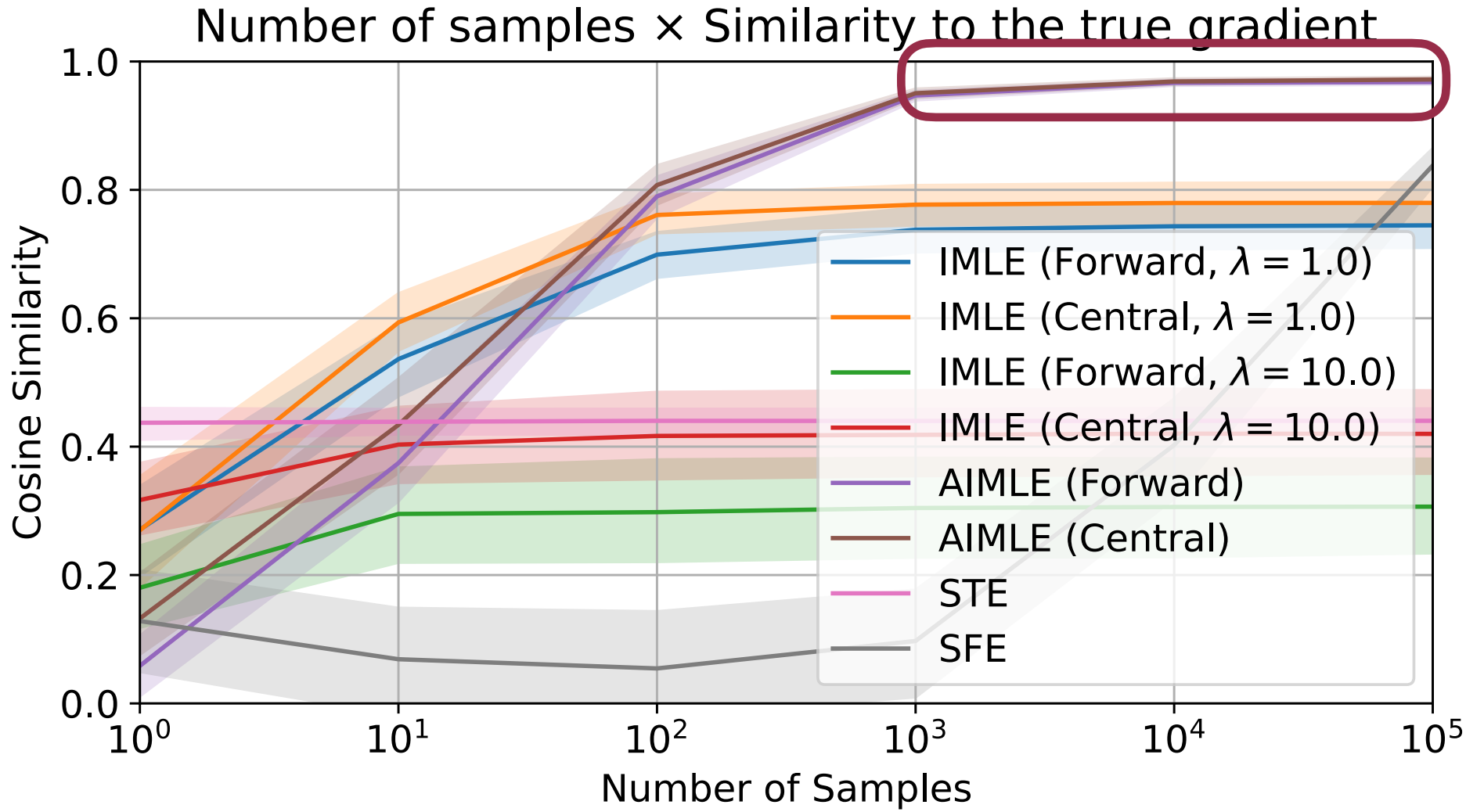
$$\nabla_\theta L \approx \frac{1}{\lambda}\left[\text{MAP}(\theta+\epsilon) - \text{MAP}(\theta - \lambda \nabla_{\mathbf{z}} L + \epsilon)\right]$$

Can be seen as a **forward** (one-sided) **finite difference approximation** in the form $f'(x) \approx \left[f(x+h) - f(x)\right]/h$.

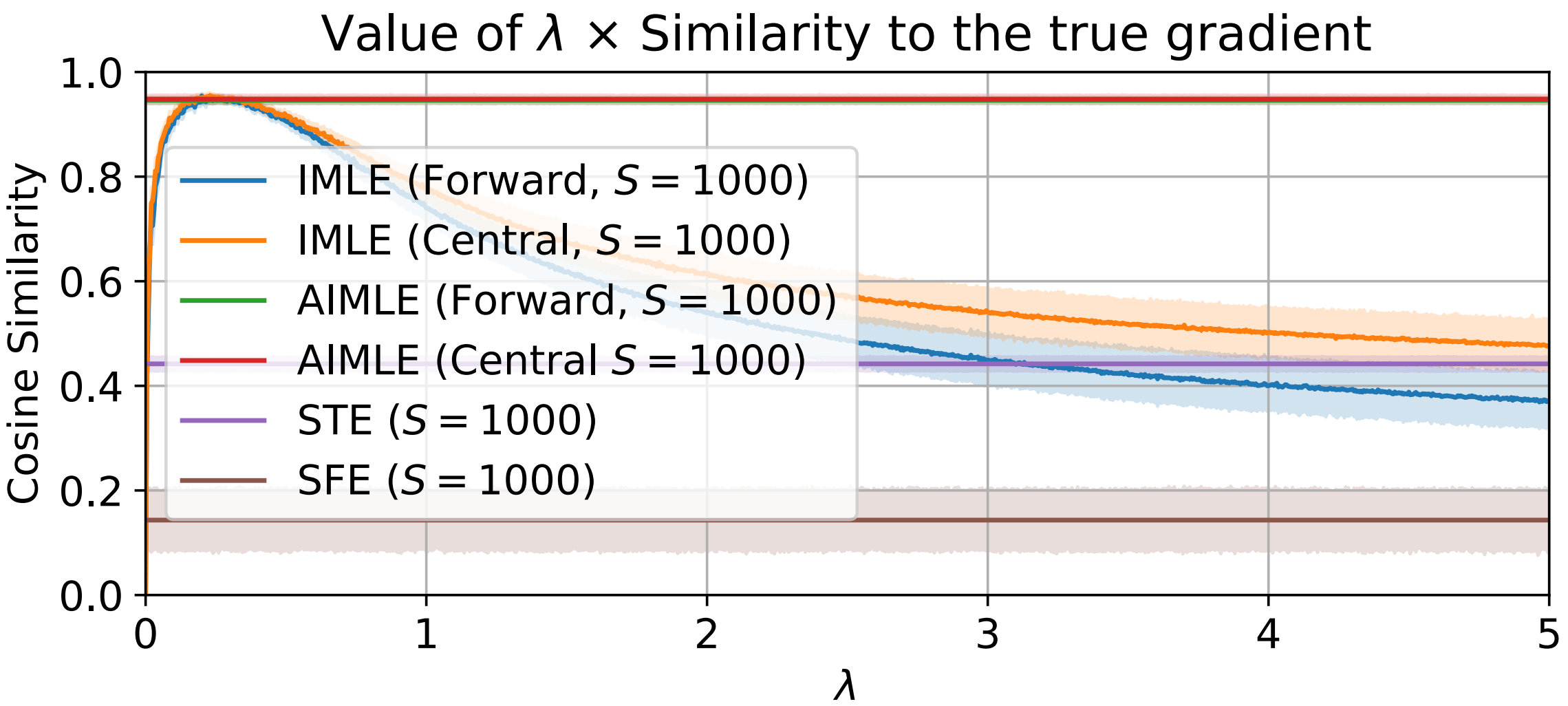
We can obtain a better approximation using a **centred** (two-sided) **difference formula**:

$$\nabla_\theta L \approx \frac{1}{2\lambda}\left[\text{MAP}(\theta+\lambda \nabla_{\mathbf{z}} L + \epsilon) - \text{MAP}(\theta - \lambda \nabla_{\mathbf{z}} L + \epsilon)\right]$$

Results - Estimator Bias and Variance



Results - Manual vs. Automatic λ Selection

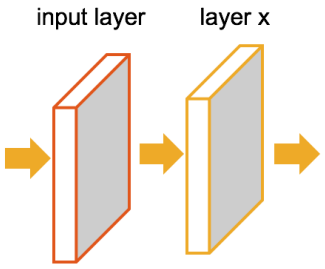


Results - Learning to Explain

Pours a slight tangerine orange and straw yellow. The head is nice and bubbly but fades very quickly with a little lacing. Smells like Wheat and European hops, a little yeast in there too. There is some fruit in there too, but you have to take a good whiff to get it. The taste is of wheat, a bit of malt, and a little fruit flavour in there too. Almost feels like drinking Champagne, medium mouthful otherwise. Easy to drink, but not something I'd be trying every night.

Appearance: 3.5 **Aroma: 4.0** Palate: 4.5 Taste: 4.0 Overall: 4.0

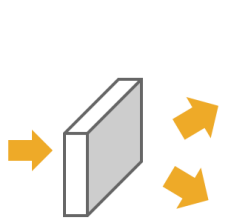
in the aroma , coffee and chocolate that is quite pronounced . in the taste , coffee , dry chocolate and a hit of hoppy bitterness . a small bite and medium bodied mouthfeel , with a dry roasty coffee in the aftertaste . a nice coffee and chocolate taste , nice hop presence , really freakin good .



in the aroma , coffee and chocolate that is quite pronounced . in the taste , coffee , dry chocolate and a hit of hoppy bitterness . a small bite and medium bodied mouthfeel , with a dry roasty coffee in the aftertaste . a nice coffee and chocolate taste , nice hop presence , really freakin good .

$\sim z$

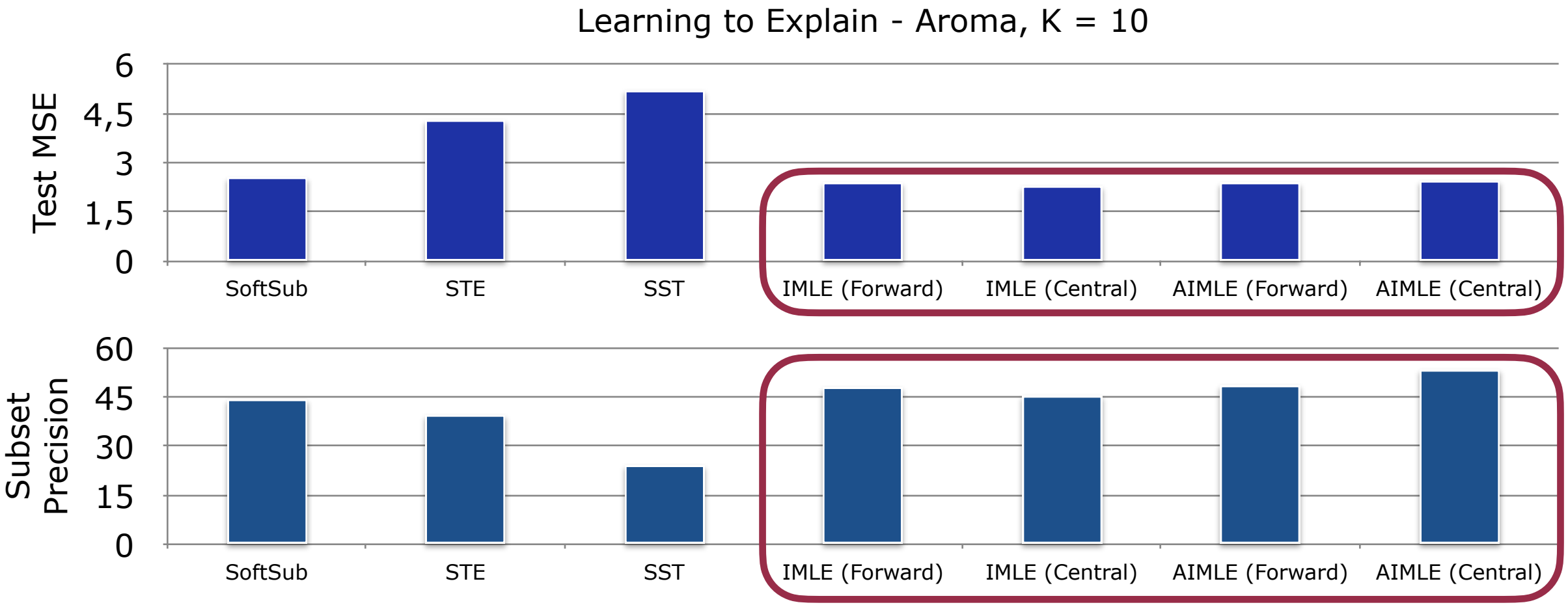
chocolate
coffee
chocolate
hoppy
bitterness
bodied
roasty
nice
chocolate
good



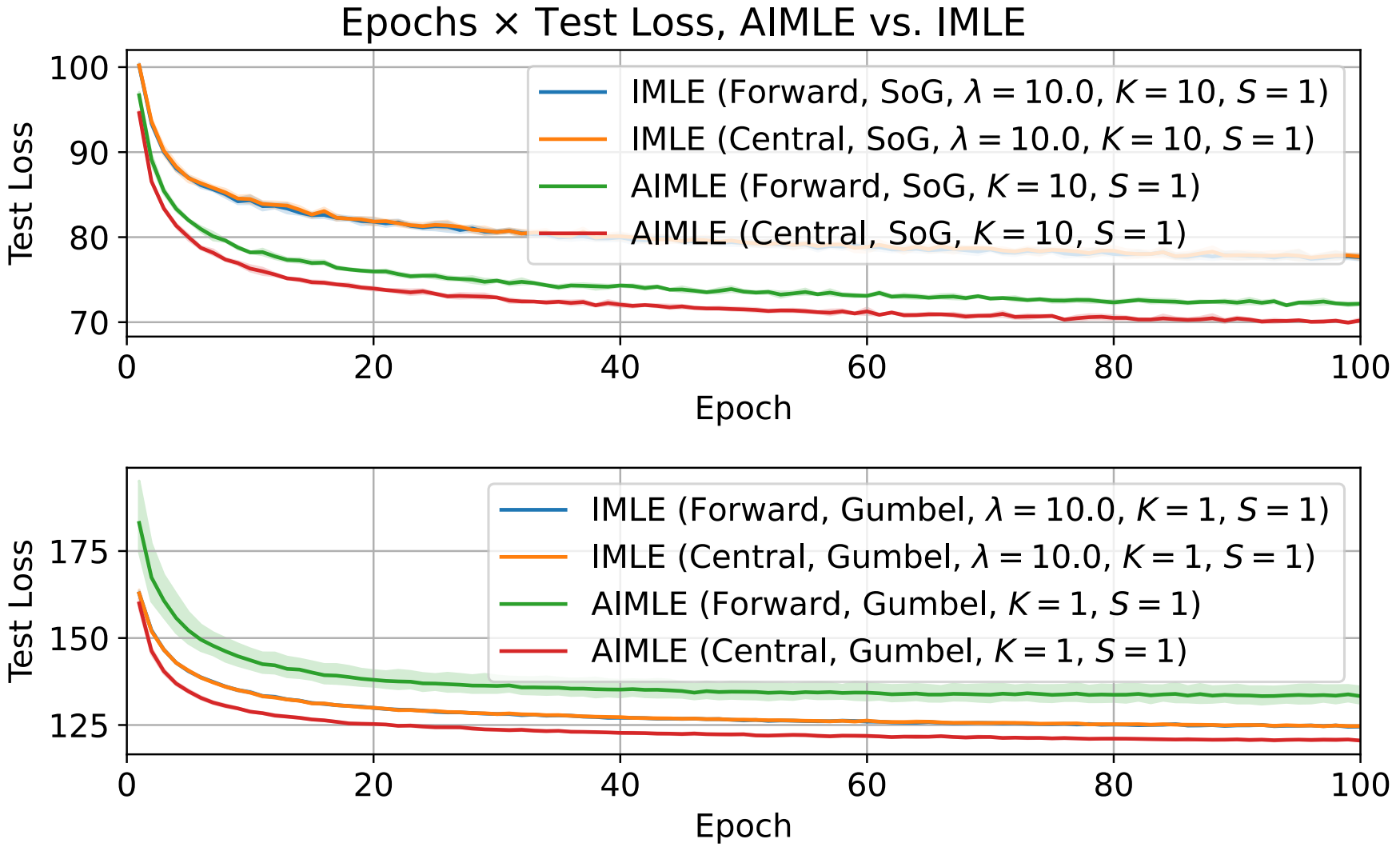
(positive sentiment)

(negative sentiment)

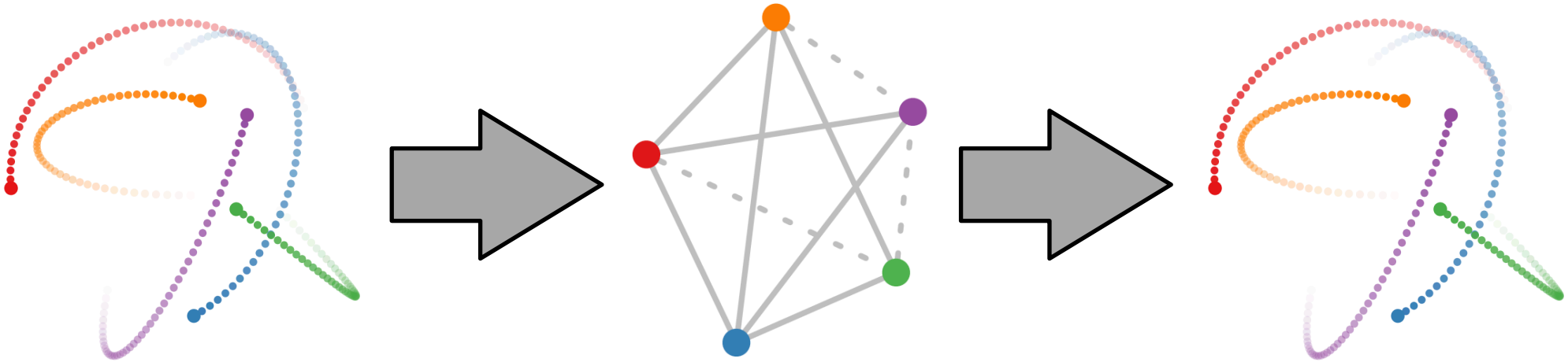
Results - Learning to Explain



Results - Discrete Variational Auto-Encoder



Results - Neural Relational Inference

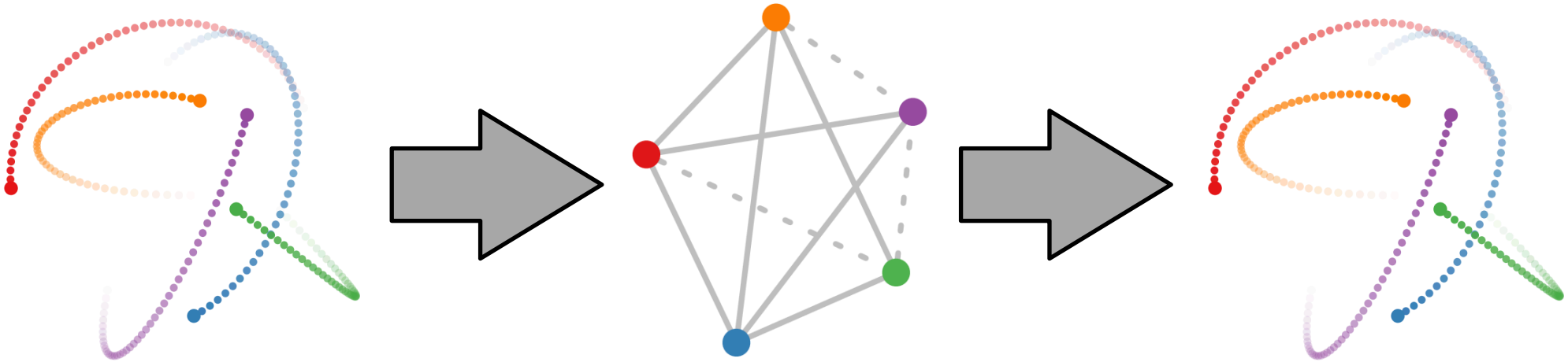


Observed dynamics

Interaction graph

Predicted dynamics

Results - Neural Relational Inference



Observed dynamics | Interaction graph | Predicted dynamics

	T=10			T=20		
Edge Distribution	ELBO	Edge Prec.	Edge Rec.	ELBO	Edge Prec.	Edge Rec.
SST (Hard)	-2301.47 ± 85.86	33.75 ± 9.44	60.40 ± 23.23	-3407.89 ± 221.53	57.40 ± 17.87	70.42 ± 8.22
IMLE (Forward)	-2289.94 ± 4.31	23.94 ± 0.03	95.75 ± 0.14	-3820.68 ± 25.32	20.28 ± 0.12	20.28 ± 0.12
IMLE (Central)	-2341.71 ± 41.68	43.95 ± 7.22	43.95 ± 7.22	-3447.29 ± 550.38	40.25 ± 14.26	40.25 ± 14.26
AIMLE (Forward)	**-1877.90 ± 277.53**	55.23 ± 11.86	55.23 ± 11.86	**-1884.83 ± 124.62**	40.48 ± 4.25	40.48 ± 4.25
AIMLE (Central)	-2018.39 ± 357.16	29.32 ± 6.89	41.83 ± 21.51	-1999.57 ± 856.27	70.89 ± 24.77	83.73 ± 1.31

Summary

Implicit MLE [Niepert et al., NeurIPS 2021] allows you to back-propagate through arbitrary combinatorial solvers, achieving seamless neuro-symbolic integration

- Highly dependent on the perturbation size hyper-parameter λ

Summary

Implicit MLE [Niepert et al., NeurIPS 2021] allows you to back-propagate through arbitrary combinatorial solvers, achieving seamless neuro-symbolic integration

- Highly dependent on the perturbation size hyper-parameter λ

In this work, we propose **Adaptive IMLE**, a method for adaptively selecting λ during training based on analysing the sparsity of the gradient estimates

- Off-the-shelf: just add a decorator to your combinatorial solver to use it as a neural network layer
- Easy-to-use library available at github.com/EdinburghNLP/torch-adaptive-imle