Differentiable Dynamic Programming for Structured Prediction and Attention

Arthur Mensch, Mathieu Blondel

Differentiable Dynamic Programming for Structured Prediction and Attention

Arthur Mensch [1] **Mathieu Blondel** [2]

Abstract

Dynamic programming (DP) solves a variety of structured combinatorial problems by iteratively breaking them down into smaller subproblems. In spite of their versatility, many DP algorithms are non-differentiable, which hampers their use as a layer in neural networks trained by backpropagation. To address this issue, we propose to smooth the max operator in the dynamic programming recursion, using a strongly convex regularizer. This allows to relax both the optimal value and solution of the original combinatorial problem, and turns a broad class of DP algorithms into differentiable operators. Theoretically, we provide a new probabilistic perspective on backpropagating through these DP operators, and relate them to inference in graphical models. We derive two particular instantiations of our framework, a smoothed Viterbi algorithm for sequence prediction and a smoothed DTW algorithm for time-series alignment. We showcase these instantiations on structured prediction (audio-to-score alignment, NER) and on structured and sparse attention for translation.

Modern neural networks are composed of multiple layers of nested functions. Although layers usually consist of of elementary linear algebraic operations and simple nonlinearities, there is a growing need for layers that output the *value* or the *solution* of an optimization problem. This can be used to design loss functions that capture relevant regularities in the input (**??**) or to create layers that impose prior structure on the output (**????**).

Among these works, several involve a convex optimization problem (**???**); others solve certain combinatorial optimization problems by dynamic programming (**???**). However, because dynamic programs (**?**) are usually non-differentiable, virtually all these works resort to the formalism of conditional random fields (CRFs) (**?**), which can be seen as changing the semiring used by the dynamic program — replacing all values by their exponentials and all $(\max, +)$ operations with $(+, \times)$ operations (**?**). While this modification smoothes the dynamic program, it looses the sparsity of solutions, since hard assignments become soft ones. Moreover, a general understanding of how to relax and differentiate dynamic programs is lacking. In this work, we propose to do so by leveraging smoothing (**??**) and backpropagation (**?**). We make the following contributions.

1) We present a unified framework for turning a broad class of dynamic programs (DP) into differentiable operators. Unlike existing works, we propose to change the semiring to use $(\max_\Omega, +)$ operations, where $\max_\Omega$ is a max operator smoothed with a strongly convex regularizer Ω (§1).

2) We show that the resulting DP operators, that we call DP_Ω, are smoothed relaxations of the original DP algorithm and satisfy several key properties, chief among them convexity. In addition, we show that their gradient, ∇DP_Ω, is equal to the *expected trajectory* of a certain random walk and can be used as a sound relaxation to the original dynamic program's solution. Using negative entropy for Ω recovers existing CRF-based works from a different perspective — we provide new arguments as to why this Ω is a good choice. On the other hand, using squared ℓ_2 norm for Ω leads to new algorithms whose expected solution is *sparse*. We derive a clean and efficient method to backpropagate gradients, both through DP_Ω and ∇DP_Ω. This allows us to define differentiable DP layers that can be incorporated in neural networks trained end-to-end (§2).

3) We illustrate how to to derive two particular instantiations of our framework, a smoothed Viterbi algorithm for sequence prediction and a smoothed DTW algorithm for supervised time-series alignment (§3). The latter is illustrated in Figure 1. Finally, we showcase these two instantiations on structured prediction tasks (§4) and on structured attention for neural machine translation (§5).

[1]Inria, CEA, Université Paris-Saclay, Gif-sur-Yvette, France. Work performed at [2]NTT Communication Science Laboratories, Kyoto, Japan. Correspondence to: AM <arthur.mensch@m4x.org>, MB <mathieu@mblondel.org>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

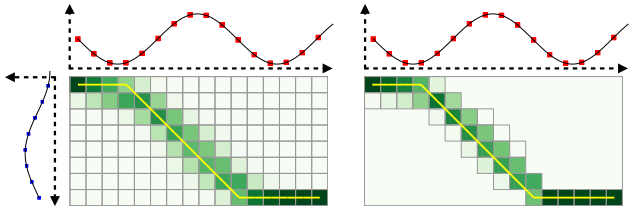

Figure 1. $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is an instantiation of the proposed smoothed dynamic programming operator, $\mathrm{DP}_\Omega(\boldsymbol{\theta})$, to the dynamic time warping (DTW) computational graph. In this picture, $\boldsymbol{\theta}$ is the squared Euclidean distance matrix between the observations of two time-series. The gradient $\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is equal to the expected alignment under a certain random walk characterized in §2.3 and is a sound continuous relaxation to the hard DTW alignment between the two time-series (here depicted with a yellow path). Unlike negentropy regularization (left), ℓ_2^2 regularization leads to exactly sparse alignments (right). Our framework allows to backpropagate through both $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ and $\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta})$, which makes it possible to learn the distance matrix $\boldsymbol{\theta}$ end-to-end.

Notation. We denote scalars, vectors and matrices using lower-case, bold lower-case and bold upper-case letters, *e.g.*, y, $\boldsymbol{y}$ and $\boldsymbol{Y}$. We denote the elements of $\boldsymbol{Y}$ by $y_{i,j}$ and its rows by $\boldsymbol{y}_i$. We denote the Frobenius inner product between $\boldsymbol{A}$ and $\boldsymbol{B}$ by $\langle \boldsymbol{A}, \boldsymbol{B} \rangle \triangleq \sum_{i,j} a_{i,j} b_{i,j}$. We denote the $(D-1)$-probability simplex by $\triangle^D \triangleq \{\boldsymbol{\lambda} \in \mathbb{R}_+^D \colon \|\boldsymbol{\lambda}\|_1 = 1\}$. We write $\mathrm{conv}(\mathcal{Y}) \triangleq \{\sum_{\boldsymbol{Y} \in \mathcal{Y}} \lambda_{\boldsymbol{Y}} \boldsymbol{Y} \colon \boldsymbol{\lambda} \in \triangle^{|\mathcal{Y}|}\}$ the convex hull of $\mathcal{Y}$, $[N]$ the set $\{1, \dots, N\}$ and $\mathrm{supp}(\boldsymbol{x}) \triangleq \{j \in [D] \colon x_j \neq 0\}$ the support of $\boldsymbol{x} \in \mathbb{R}^D$. We denote the Shannon entropy by $H(\boldsymbol{q}) \triangleq \sum_i q_i \log q_i$.

We have released an optimized and modular *PyTorch* implementation for reproduction and reuse.

1. Smoothed max operators

In this section, we introduce smoothed max operators (**???**), that will serve as a powerful and generic abstraction to define differentiable dynamic programs in §2. Let $\Omega : \mathbb{R}^D \to \mathbb{R}$ be a strongly convex function on $\triangle^D$ and let $\boldsymbol{x} \in \mathbb{R}^D$. We define the max operator smoothed by Ω as:

$$\max_\Omega(\boldsymbol{x}) \triangleq \max_{\boldsymbol{q} \in \triangle^D} \langle \boldsymbol{q}, \boldsymbol{x} \rangle - \Omega(\boldsymbol{q}). \tag{1}$$

In other words, $\max_\Omega$ is the convex conjugate of Ω, restricted to the simplex. From the duality between strong convexity and smoothness, $\max_\Omega$ is smooth: differentiable everywhere and with Lipschitz continuous gradient. Since the argument that achieves the maximum in (1) is unique, from Danskin's theorem (**?**), it is equal to the gradient:

$$\nabla\max_\Omega(\boldsymbol{x}) = \operatorname*{argmax}_{\boldsymbol{q} \in \triangle^D} \langle \boldsymbol{q}, \boldsymbol{x} \rangle - \Omega(\boldsymbol{q}).$$

The gradient is differentiable almost everywhere for any strongly-convex Ω (everywhere for negentropy). Next, we state properties that will be useful throughout this paper.

Lemma 1. *Properties of* $\max_\Omega$ *operators*

Let $\boldsymbol{x} = (x_1, \dots, x_D)^\top \in \mathbb{R}^D$.

1. *Boundedness: If Ω is lower-bounded by $L_{\Omega,D}$ and upper-bounded by $U_{\Omega,D}$ on the simplex $\triangle^D$, then* $\max(\boldsymbol{x}) - U_{\Omega,D} \leq \max_\Omega(\boldsymbol{x}) \leq \max(\boldsymbol{x}) - L_{\Omega,D}$.
2. *Distributivity of $+$ over* $\max_\Omega$*:* $\max_\Omega(\boldsymbol{x} + c\boldsymbol{1}) = \max_\Omega(\boldsymbol{x}) + c \quad \forall c \in \mathbb{R}$.
3. *Commutativity: If $\Omega(\boldsymbol{P}\boldsymbol{q}) = \Omega(\boldsymbol{q})$, where $\boldsymbol{P}$ is a permutation matrix, then* $\max_\Omega(\boldsymbol{P}\boldsymbol{x}) = \max_\Omega(\boldsymbol{x})$.
4. *Non-decreasingness in each coordinate:* $\max_\Omega(\boldsymbol{x}) \leq \max_\Omega(\boldsymbol{y}) \quad \forall \boldsymbol{x} \leq \boldsymbol{y}$
5. *Insensitivity to $-\infty$:* $x_j = -\infty \Rightarrow \nabla\max_\Omega(\boldsymbol{x})_j = 0$.

Proofs are given in §A.1. In particular, property 3 holds whenever $\Omega(\boldsymbol{q}) = \sum_{i=1}^D \omega(q_i)$, for some function ω. We focus in this paper on two specific regularizers Ω: the negentropy $-H$ and the squared ℓ_2 norm. For these choices, all properties above are satisfied and we can derive closed-form expressions for $\max_\Omega$, its gradient and its Hessian — see §B.1. When using negentropy, $\max_\Omega$ becomes the *log-sum-exp* and $\nabla\max_\Omega$ the *softmax*. The former satisfies associativity, which as we shall see, makes it natural to use in dynamic programming. With the squared ℓ_2 regularization, as observed by **??**, the gradient $\nabla\max_\Omega$ is *sparse*. This will prove useful to enforce sparsity in the models we study.

2. Differentiable DP layers

Dynamic programming (DP) is a generic way of solving combinatorial optimization problems by recursively solving problems on smaller sets. We first introduce this category of algorithms in a broad setting, then use smoothed max operators to define differentiable DP layers.

2.1. Dynamic programming on a DAG

Every problem solved by dynamic programming reduces to finding the highest-scoring path between a start node and an end node, on a weighted directed acyclic graph (DAG). We therefore introduce our formalism on this generic problem, and give concrete examples in §3.

Formally, let $G = (\mathcal{V}, \mathcal{E})$ be a DAG, with nodes $\mathcal{V}$ and edges $\mathcal{E}$. We write $N = |\mathcal{V}| \geq 2$ the number of nodes. Without loss of generality, we number the nodes in topological order, from 1 (start) to N (end), and thus $\mathcal{V} = [N]$. Node 1 is the only node without parents, and node N the only node without children. Every directed edge (i, j) from a parent node j to a child node i has a weight $\theta_{i,j} \in \mathbb{R}$. We

gather the edge weights in a matrix $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subseteq \mathbb{R}^{N \times N}$, setting $\theta_{i,j} = -\infty$ if $(i,j) \notin \mathcal{E}$ and $\theta_{1,1} = 1$. We consider the set $\mathcal{Y}$ of all paths in G from node 1 to node N. Any path $\boldsymbol{Y} \in \mathcal{Y}$ can be represented as a $N \times N$ binary matrix, with $y_{i,j} = 1$ if the path goes through the edge (i,j) and $y_{i,j} = 0$ otherwise. In the sequel, paths will have a one-to-one correspondence with discrete structures such as sequences or alignments. Using this representation, $\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle$ corresponds to the cumulated sum of edge weights, along the path $\boldsymbol{Y}$. The computation of the *highest score* among all paths amounts to solving the combinatorial problem

$$\mathrm{LP}(\boldsymbol{\theta}) \triangleq \max_{\boldsymbol{Y} \in \mathcal{Y}} \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle \in \mathbb{R}. \tag{2}$$

Although the size of $\mathcal{Y}$ is in general exponential in N, $\mathrm{LP}(\boldsymbol{\theta})$ can be computed in one topologically-ordered pass over G using *dynamic programming*. We let $\mathcal{P}_i$ be the set of parent nodes of node i in graph G and define recursively

$$\begin{aligned} v_1(\boldsymbol{\theta}) &\triangleq 0 \\ \forall\, i \in [2, \dots, N] :\ v_i(\boldsymbol{\theta}) &\triangleq \max_{j \in \mathcal{P}_i} \theta_{i,j} + v_j(\boldsymbol{\theta}). \end{aligned} \tag{3}$$

This algorithm outputs $\mathrm{DP}(\boldsymbol{\theta}) \triangleq v_N(\boldsymbol{\theta})$. We now show that this is precisely the highest score among all paths.

Proposition 1. *Optimality of dynamic programming*

$$\forall \boldsymbol{\theta} \in \boldsymbol{\Theta} : \quad \mathrm{DP}(\boldsymbol{\theta}) = \mathrm{LP}(\boldsymbol{\theta})$$

The optimality of recursion (3) is well-known (**?**). We prove it again with our formalism in §A.2, since it exhibits the two key properties that the max operator must satisfy to guarantee optimality: *distributivity* of $+$ over it and *associativity*. The cost of computing $\mathrm{DP}(\boldsymbol{\theta})$ is $\mathcal{O}(|\mathcal{E}|)$, which is exponentially better than $\mathcal{O}(|\mathcal{Y}|)$.

In many applications, we will often rather be interested in the *argument* that achieves the maximum, *i.e.*, one of the highest-scoring paths

$$\boldsymbol{Y}^\star(\boldsymbol{\theta}) \in \operatorname*{argmax}_{\boldsymbol{Y} \in \mathcal{Y}} \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle. \tag{4}$$

This argument can be computed by *backtracking*, that we now relate to computing subgradients of $\mathrm{LP}(\boldsymbol{\theta})$.

Linear program, lack of differentiability. Unfortunately, $\mathrm{LP}(\boldsymbol{\theta})$ is not differentiable everywhere. To see why this is the case, notice that (2) can be rewritten as a linear program over the convex polytope $\mathrm{conv}(\mathcal{Y})$:

$$\mathrm{LP}(\boldsymbol{\theta}) = \max_{\boldsymbol{Y} \in \mathrm{conv}(\mathcal{Y})} \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle.$$

From the generalized Danskin theorem (**?**),

$$\boldsymbol{Y}^\star(\boldsymbol{\theta}) \in \partial \mathrm{LP}(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{Y} \in \mathrm{conv}(\mathcal{Y})} \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle,$$

where ∂ denotes the subdifferential of $\mathrm{LP}(\boldsymbol{\theta})$, *i.e.*, the set of subgradients. When $\boldsymbol{Y}^\star(\boldsymbol{\theta})$ is unique, $\partial \mathrm{LP}(\boldsymbol{\theta})$ is a singleton and $\boldsymbol{Y}^\star$ is equal to the gradient of $\mathrm{LP}(\boldsymbol{\theta})$, that we write $\nabla \mathrm{LP}(\boldsymbol{\theta})$. Unfortunately, $\boldsymbol{Y}^\star(\boldsymbol{\theta})$ is not always unique, meaning that $\mathrm{LP}(\boldsymbol{\theta})$ is not differentiable everywhere. As we will show in §4.2, this hinders optimization as we can only train models involving $\mathrm{LP}(\boldsymbol{\theta})$ with *subgradient* methods. Worse, $\boldsymbol{Y}^\star(\boldsymbol{\theta})$, a function from $\boldsymbol{\Theta}$ to $\mathcal{Y}$, is discontinuous and has null or undefined derivatives. It is thus impossible to use it in a model trained by gradient descent.

2.2. Smoothed max layers

To address the lack of differentiability of dynamic programming, we introduce the operator $\max_\Omega$, presented in §1, and consider two approaches.

Smoothing the linear program. Let us define the Ω-smoothed maximum of a function $f \colon \mathcal{Y} \to \mathbb{R}$ over a finite set $\mathcal{Y}$ using the following shorthand notation:

$$\max_{\boldsymbol{Y} \in \mathcal{Y}}{}_\Omega\, f(\boldsymbol{Y}) \triangleq \max\nolimits_\Omega((f(\boldsymbol{Y}))_{\boldsymbol{Y} \in \mathcal{Y}}).$$

A natural way to circumvent the lack of differentiability of $\mathrm{LP}(\boldsymbol{\theta})$ is then to replace the *global* $\max$ operator by $\max_\Omega$:

$$\mathrm{LP}_\Omega(\boldsymbol{\theta}) \triangleq \max_{\boldsymbol{Y} \in \mathcal{Y}}{}_\Omega \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle \in \mathbb{R}. \tag{5}$$

From §1, $\mathrm{LP}_\Omega(\boldsymbol{\theta})$ is convex and, as long as Ω is strongly convex, differentiable everywhere. In addition, $\nabla \mathrm{LP}_\Omega(\boldsymbol{\theta})$ is Lipschitz continuous and thus differentiable almost everywhere. Unfortunately, solving (5) for general Ω is likely intractable when $\mathcal{Y}$ has an exponential size.

Smoothing the dynamic program. As a tractable alternative, we propose an *algorithmic* smoothing. Namely, we replace $\max$ by $\max_\Omega$ *locally* within the DP recursion. Omitting the dependence on Ω, this defines a smoothed recursion over the new sequence $(v_i(\boldsymbol{\theta}))_{i=1}^N$:

$$\begin{aligned} v_1(\boldsymbol{\theta}) &\triangleq 0 \\ \forall i \in [2, \dots, N] :\ v_i(\boldsymbol{\theta}) &\triangleq \max_{j \in \mathcal{P}_i}{}_\Omega\, \theta_{i,j} + v_j(\boldsymbol{\theta}). \end{aligned} \tag{6}$$

The new algorithm outputs $\mathrm{DP}_\Omega(\boldsymbol{\theta}) \triangleq v_N(\boldsymbol{\theta})$, the *smoothed highest score*. Smoothing the max operator locally brings the same benefit as before — $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is smooth and $\nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$ is differentiable almost everywhere. However, computing $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is now always tractable, since it simply requires to evaluate $(v_i(\boldsymbol{\theta}))_{i=1}^N$ in topological order, as in the original recursion (3). Although $\mathrm{LP}_\Omega(\boldsymbol{\theta})$ and $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ are generally different (in fact, $\mathrm{LP}_\Omega(\boldsymbol{\theta}) \geq \mathrm{DP}_\Omega(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}$), we now show that $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is a sensible approximation of $\mathrm{LP}(\boldsymbol{\theta})$ in several respects.

Proposition 2. *Properties of DP_Ω*

1. *$DP_\Omega(\boldsymbol{\theta})$ is convex*
2. *$LP(\boldsymbol{\theta}) - DP_\Omega(\boldsymbol{\theta})$ is bounded above and below: it lies in $(N-1)[L_{\Omega,N}, U_{\Omega,N}]$, with Lemma 1 notations.*
3. *When Ω is separable, $DP_\Omega(\boldsymbol{\theta}) = LP_\Omega(\boldsymbol{\theta})$ **if and only if** $\Omega = -\gamma H$, where $\gamma \geq 0$.*

Proofs are given in §A.3. The first claim can be surprising due to the recursive definition of $\mathrm{DP}_\Omega(\boldsymbol{\theta})$. The second claim implies that $\mathrm{DP}_{\gamma\Omega}(\boldsymbol{\theta})$ converges to $\mathrm{LP}(\boldsymbol{\theta})$ when the regularization vanishes: $\mathrm{DP}_{\gamma\Omega}(\boldsymbol{\theta}) \to_{\gamma\to 0} \mathrm{LP}(\boldsymbol{\theta})$; $\mathrm{LP}_{\gamma\Omega}(\boldsymbol{\theta})$ also satisfies this property. The "if" direction of the third claim follows by showing that $\max_{-\gamma H}$ satisfies associativity. This recovers known results in the framework of message passing algorithms for probabilistic graphical models (*e.g.,* **?**, Section 4.1.3), with a more algebraic point of view. The key role that the distributive and associative properties play into breaking down large problems into smaller ones has long been noted (**??**). However, the "and only if" part of the claim is new to our knowledge. Its proof shows that $\max_{-\gamma H}$ is the only $\max_\Omega$ satisfying associativity, exhibiting a functional equation from information theory (**?**). While this provides an argument in favor of entropic regularization, ℓ_2^2 regularization has different benefits in terms of sparsity of the solutions.

2.3. Relaxed argmax layers

It is easy to check that $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta})$ belongs to $\mathrm{conv}(\mathcal{Y})$ and can be interpreted as an expected path under some distribution induced by $\nabla\max_\Omega$, over all possible $\boldsymbol{Y} \in \mathcal{Y}$ — see §A.4 for details. This makes $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta})$ interpretable as a *continuous relaxation* of the highest-scoring path $\boldsymbol{Y}^\star(\boldsymbol{\theta})$ defined in (4). However, like $\mathrm{LP}_\Omega(\boldsymbol{\theta})$, computing $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta})$ is likely intractable in the general case. Fortunately, $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is always easily computable by *backpropagation* and enjoys similar properties, as we now show.

Computing $\nabla\mathbf{DP}_\Omega(\boldsymbol{\theta})$. Computing $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$ can be broken down into two steps. First, we compute and record the local gradients alongside the recursive step (6):

$$\forall\, i \in [N]: \quad \boldsymbol{q}_i(\boldsymbol{\theta}) \triangleq \nabla\max_\Omega(\boldsymbol{\theta}_i + \boldsymbol{v}(\boldsymbol{\theta})) \in \triangle^N,$$

where $\boldsymbol{v}(\boldsymbol{\theta}) \triangleq (v_1(\boldsymbol{\theta}), \dots, v_N(\boldsymbol{\theta}))$. Since we assume that $\theta_{i,j} = -\infty$ if $(i,j) \notin \mathcal{E}$, we have $\mathrm{supp}(\boldsymbol{q}_i(\boldsymbol{\theta})) = \mathcal{P}_i$. This ensures that, similarly to $v_i(\boldsymbol{\theta})$, $\boldsymbol{q}_i(\boldsymbol{\theta})$ exclusively depends on $(v_j(\boldsymbol{\theta}))_{j\in\mathcal{P}_i}$. Let $\mathcal{C}_j$ be the children of node $j \in [N]$. A straighforward application of backpropagation (cf. §A.5) yields a recursion run in *reverse-topological* order, starting from node $j = N-1$ down to $j = 1$:

$$\forall\, i \in \mathcal{C}_j:\ e_{i,j} \leftarrow \bar{e}_i q_{i,j} \text{ then } \bar{e}_j \leftarrow \sum_{i\in\mathcal{C}_j} e_{i,j},$$

where $\bar{e}_N \leftarrow 1$ and $e_{i,j} \leftarrow 0$ for $(i,j) \notin \mathcal{E}$. The final output is $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}) = \boldsymbol{E}$. Assuming $\max_\Omega$ can be computed in linear time, the total cost is $\mathcal{O}(|\mathcal{E}|)$, the same as $\mathrm{DP}(\boldsymbol{\theta})$. Pseudo-code is summarized in §A.5.

Associated path distribution. The backpropagation we derived has a probabilistic interpretation. Indeed, $\boldsymbol{Q}(\boldsymbol{\theta}) \in \mathbb{R}^{N\times N}$ can be interpreted as a transition matrix: it defines a *random walk* on the graph G, *i.e.*, a finite Markov chain with states $\mathcal{V}$ and transition probabilities supported by $\mathcal{E}$. The random walk starts from node N and, when at node i, hops to node $j \in \mathcal{P}_i$ with probability $q_{i,j}$. It always ends at node 1, which is absorbing. The walk follows the path $\boldsymbol{Y} \in \mathcal{Y}$ with a probability $p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y})$, which is simply the product of the $q_{i,j}$ of visited edges. Thus, $\boldsymbol{Q}(\boldsymbol{\theta})$ defines a *path distribution* $p_{\boldsymbol{\theta},\Omega}$. Our next proposition shows that $\nabla\mathrm{DP}_\Omega(\boldsymbol{Y}) \in \mathrm{conv}(\mathcal{Y})$ and is equal to the expected path $\mathbb{E}_{\boldsymbol{\theta},\Omega}[\boldsymbol{Y}]$ under that distribution.

Proposition 3. *$\nabla DP_\Omega(\boldsymbol{\theta})$ as an expected path*

$$\forall\, \boldsymbol{\theta} \in \boldsymbol{\Theta}: \quad \nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta},\Omega}[\boldsymbol{Y}] = \boldsymbol{E} \in \mathrm{conv}(\mathcal{Y}).$$

Proof is provided in §A.5. Moreover, $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is a principled relaxation of the highest-scoring path $\boldsymbol{Y}^\star(\boldsymbol{\theta})$, in the sense that it converges to a subgradient of $\mathrm{LP}(\boldsymbol{\theta})$ as the regularization vanishes: $\nabla\mathrm{DP}_{\gamma\Omega}(\boldsymbol{\theta}) \xrightarrow[\gamma\to 0]{} \boldsymbol{Y}^\star(\boldsymbol{\theta}) \in \partial\mathrm{LP}(\boldsymbol{\theta})$. When $\Omega = -\gamma H$, the distributions underpinning $\mathrm{LP}_\Omega(\boldsymbol{\theta})$ and $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ coincide and reduce to the Gibbs distribution $p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y}) \propto \exp(\langle\boldsymbol{\theta}, \boldsymbol{Y}\rangle/\gamma)$. The value $\mathrm{LP}_\Omega(\boldsymbol{\theta}) = \mathrm{DP}_\Omega(\boldsymbol{\theta})$ is then equal to the log partition. When $\Omega = \gamma\|\cdot\|^2$, some transitions between nodes have zero probability and hence some paths have zero probability under the distribution $p_{\boldsymbol{\theta},\Omega}$. Thus, $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$ is typically *sparse* — this will prove interesting to introspect the various models we consider (typically, the smaller γ, the sparser $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$).

2.4. Multiplication with the Hessian $\nabla^2\mathbf{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$

Using $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$ as a layer involves backpropagating through $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$. This requires to apply the *Jacobian* of $\nabla\mathrm{DP}_\Omega$ operator (a linear map from $\mathbb{R}^{N\times N}$ to $\mathbb{R}^{N\times N}$), or in other words to apply the *Hessian* of DP_Ω, to an input sensibility vector $\boldsymbol{Z}$, computing

$$\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z} = \nabla\langle\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle \in \mathbb{R}^{N\times N},$$

where derivatives are w.r.t. $\boldsymbol{\theta}$. The above vector may be computed in two ways, that differ in the order in which derivatives are computed. Using automatic differentiation frameworks such as *PyTorch* (**?**), we may backpropagate over the computational graph a first time to compute the gradient $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$, while recording operations. We may then compute $\langle\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$, and backpropagate once again. However, due to the structure of the problem,

it proves more efficient, adapting Pearlmutter's approach (**?**), to directly compute $\langle \nabla \mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z} \rangle \in \mathbb{R}$, namely, the *directional derivative* at $\boldsymbol{\theta}$ along $\boldsymbol{Z}$. This is done by applying the chain rule in one topologically-ordered pass over G. Similarly to the gradient computation, we record products with the *local* Hessians $\boldsymbol{H}_i(\boldsymbol{\theta}) \triangleq \nabla^2 \max_\Omega(\boldsymbol{\theta}_i + \boldsymbol{v}(\boldsymbol{\theta}))$ along the way. We then compute the gradient of the directional derivative using backpropagation. This yields a recursion for computing $\nabla^2 \mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ in reverse topological-order over G. The complete derivation and the pseudo-code are given in §A.7. They allow to implement DP_Ω as as a custom twice-differentiable module in existing software. For both approaches, the computational cost is $\mathcal{O}(|\mathcal{E}|)$, the same as for gradient computation. In our experiments in §4.2, our custom Hessian-vector product computation brings a $3\times/12\times$ speed-up during the backward pass on GPU/CPU *vs.* automatic differentiation.

Related works. Smoothing LP formulations was also used for MAP inference (**?**) or optimal transport (**?**) but these works do not address how to differentiate through the smoothed formulation. An alternative approach to create structured prediction layers, fundamentally different both in the forward and backward passes, is SparseMAP (**?**).

Summary. We have proposed $\mathrm{DP}_\Omega(\boldsymbol{\theta})$, a smooth, convex and tractable relaxation to the *value* of $\mathrm{LP}(\boldsymbol{\theta})$. We have also shown that $\nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$ belongs to $\mathrm{conv}(\mathcal{Y})$ and is therefore a sound relaxation to *solutions* of $\mathrm{LP}(\boldsymbol{\theta})$. To conclude this section, we formally define our proposed two layers.

Definition 1. *Differentiable dynamic programming layers*

$$\begin{aligned} \textit{Value layer:} \quad & \mathit{DP}_\Omega(\boldsymbol{\theta}) \in \mathbb{R} \\ \textit{Gradient layer:} \quad & \nabla \mathit{DP}_\Omega(\boldsymbol{\theta}) \in \mathrm{conv}(\mathcal{Y}) \end{aligned}$$

3. Examples of computational graphs

We now illustrate two instantiations of our framework for specific computational graphs.

3.1. Sequence prediction

We demonstrate in this section how to instantiate DP_Ω to the computational graph of the Viterbi algorithm (**??**), one of the most famous instances of DP algorithm. We call the resulting operator Vit_Ω. We wish to tag a sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$ of vectors in $\mathbb{R}^D$ (*e.g.,* word representations) with the most probable output sequence (*e.g.,* entity tags) $\boldsymbol{y} = (y_1, \dots, y_T) \in [S]^T$. This problem can be cast as finding the highest-scoring path on a *treillis* G. While $\boldsymbol{y}$ can always be represented as a sparse $N \times N$ binary matrix, it is convenient to represent it instead as a $T \times S \times S$ binary tensor $\boldsymbol{Y}$, such that $y_{t,i,j} = 1$ if $\boldsymbol{y}$ transitions from node j to node i on time t, and 0 otherwise — we set $y_0 = 1$. The potentials can similarly be organized as a $T \times S \times S$ real tensor, such that $\theta_{t,i,j} = \phi_t(\boldsymbol{x}_t, i, j)$. Traditionally, the potential functions ϕ_t were human-engineered (**?**, §2.5). In recent works and in this paper, they are learned end-to-end (**???**).

Using the above binary tensor representation, the inner product $\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle$ is equal to $\sum_{t=1}^T \phi_t(\boldsymbol{x}_t, y_t, y_{t-1})$, $\boldsymbol{y}$'s cumulated score. This is illustrated in Figure 2 on the task of part-of-speech tagging. The bold arrows indicate one possible output sequence $\boldsymbol{y}$, *i.e.*, one possible path in G.

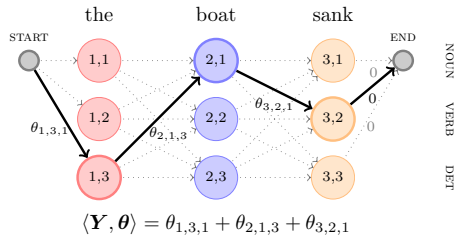

Figure 2. Computational graph of the Viterbi algorithm.

When $\Omega = -H$, we recover linear-chain conditional random fields (CRFs) (**?**) and the probability of $\boldsymbol{y}$ ($\boldsymbol{Y}$ in tensor representation) given $\boldsymbol{X}$ is

$$p_{\boldsymbol{\theta},-H}(\boldsymbol{y}|\boldsymbol{X}) \propto \exp(\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle) = \exp\Big(\sum_{t=1}^T \phi_t(\boldsymbol{x}_t, y_t, y_{t-1})\Big).$$

From Prop. 3, the gradient $\nabla \mathrm{Vit}_{-H}(\boldsymbol{\theta}) = \boldsymbol{E} \in \mathbb{R}^{T \times S \times S}$ is such that $e_{t,i,j} = p_{\boldsymbol{\theta},-H}(y_t = i, y_{t-1} = j|\boldsymbol{X})$. The marginal probability of state i at time t is simply $p_{\boldsymbol{\theta},-H}(y_t = i|\boldsymbol{X}) = \sum_{j=1}^S e_{t,i,j}$. Using a different Ω simply changes the distribution over state transitions. When $\Omega = \|\cdot\|^2$, the marginal probabilities are typically *sparse*. Pseudo-code for $\mathrm{Vit}_\Omega(\boldsymbol{\theta})$, as well as gradient and Hessian-product computations, is provided in §B.2. The case $\Omega = \|\cdot\|^2$ is new to our knowledge.

When $\Omega = -H$, the marginal probabilities are traditionally computed using the forward-backward algorithm (**?**). In contrast, we compute $\nabla \mathrm{Vit}_{-H}(\boldsymbol{\theta})$ using backpropagation while efficiently maintaining the marginalization. An advantage of our approach is that all operations are numerically stable. The relation between forward-backward and backpropagation has been noted before (*e.g.,* **?**). However, the analysis is led using $(+, \times)$ operations, instead of $(\max_\Omega, +)$ as we do. Our Viterbi instantiation can be generalized to graphical models with a tree structure, and to approximate inference in general graphical models, since unrolled loopy belief propagation (**?**) yields a dynamic program. We note that continuous beam search (**?**) can also be cleanly rewritten and extended using Vit_Ω operators.

3.2. Time-series alignment

We now demonstrate how to instantiate DP_Ω to the computational graph of dynamic time warping (DTW) (**?**), whose goal is to seek the *minimal* cost alignment between two time-series. We call the resulting operator DTW_Ω. Formally, let N_A and N_B be the lengths of two time-series, $\boldsymbol{A}$ and $\boldsymbol{B}$. Let $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ be the i^{th} and j^{th} observations of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Since edge weights only depend on child nodes, it is convenient to rearrange $\boldsymbol{Y}$ and $\boldsymbol{\theta}$ as $N_A \times N_B$ matrices. Namely, we represent an alignment $\boldsymbol{Y}$ as a $N_A \times N_B$ binary matrix, such that $y_{i,j} = 1$ if $\boldsymbol{a}_i$ is aligned with $\boldsymbol{b}_j$, and 0 otherwise. Likewise, we represent $\boldsymbol{\theta}$ as a $N_A \times N_B$ matrix. A classical example is $\theta_{i,j} = d(\boldsymbol{a}_i, \boldsymbol{b}_j)$, for some differentiable discrepancy measure d. We write $\mathcal{Y}$ the set of all monotonic alignment matrices, such that the path that connects the upper-left $(1,1)$ matrix entry to the lower-right (N_A, N_B) one uses only $\downarrow, \rightarrow, \searrow$ moves. The DAG associated with $\mathcal{Y}$ is illustrated in Figure 3 with $N_A = 4$ and $N_B = 3$ below.

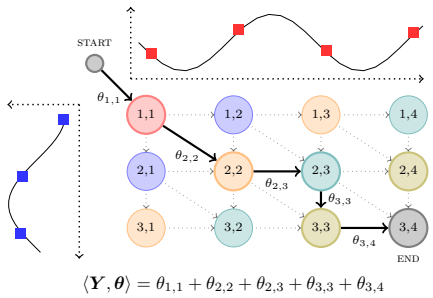

Figure 3. Computational graph of the DTW algorithm.

Again, the bold arrows indicate one possible path $\boldsymbol{Y} \in \mathcal{Y}$ from start to end in the DAG, and correspond to one possible alignment. Using this representation, the cost of an alignment (cumulated cost along the path) is conveniently computed by $\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle$. The value $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ can be used to define a loss between alignments or between time-series. Following Proposition 3, $\nabla \mathrm{DTW}_\Omega(\boldsymbol{\theta}) = \boldsymbol{E} \in \mathbb{R}^{N_A \times N_B}$ can be understood as a soft alignment matrix. This matrix is sparse when $\Omega = \|\cdot\|^2$, as illustrated in Figure 1 (right).

Pseudo-code to compute $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ as well as its gradient and its Hessian products are provided in §B.3. When $\Omega = -H$, $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is a conditional random field known as soft-DTW, and the probability $p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y}|\boldsymbol{A}, \boldsymbol{B})$ is a Gibbs distribution similar to §3.1 (**?**). However, the case $\Omega = \|\cdot\|^2$ and the computation of $\nabla^2 \mathrm{DTW}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ are new and allow new applications.

4. Differentiable structured prediction

We now apply the proposed layers, $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ and $\nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$, to structured prediction (**?**), whose goal is to predict a structured output $\boldsymbol{Y} \in \mathcal{Y}$ associated with a structured input $\boldsymbol{X} \in \mathcal{X}$. We define old and new structured losses, and demonstrate them on two structured prediction tasks: named entity recognition and time-series alignment.

4.1. Structured loss functions

Throughout this section, we assume that the potentials $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ have already been computed using a function from $\mathcal{X}$ to $\boldsymbol{\Theta}$ and let $C \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a cost function between the ground-truth output $\boldsymbol{Y}_{\text{true}}$ and the predicted output $\boldsymbol{Y}$.

Convex losses. Because C is typically non-convex, the cost-augmented structured hinge loss (**?**) is often used instead for linear models

$$\ell_C(\boldsymbol{Y}_{\text{true}}; \boldsymbol{\theta}) \triangleq \max_{\boldsymbol{Y} \in \mathcal{Y}} C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y}) + \langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle - \langle \boldsymbol{Y}_{\text{true}}, \boldsymbol{\theta} \rangle. \quad (7)$$

This is a convex upper-bound on $C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y}^\star(\boldsymbol{\theta}))$, where $\boldsymbol{Y}^\star(\boldsymbol{\theta})$ is defined in (4). To make the cost-augmented decoding tractable, it is usually assumed that $C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y})$ is linear in $\boldsymbol{Y}$, *i. e.*, it can be written as $\langle \boldsymbol{C}_{\boldsymbol{Y}_{\text{true}}}, \boldsymbol{Y} \rangle$ for some matrix $\boldsymbol{C}_{\boldsymbol{Y}_{\text{true}}}$. We can then rewrite (7) using our notation as

$$\ell_C(\boldsymbol{Y}_{\text{true}}; \boldsymbol{\theta}) = \mathrm{LP}(\boldsymbol{\theta} + \boldsymbol{C}_{\boldsymbol{Y}_{\text{true}}}) - \langle \boldsymbol{Y}_{\text{true}}, \boldsymbol{\theta} \rangle.$$

However, this loss function is non-differentiable. We therefore propose to relax LP by substituting it with DP_Ω:

$$\ell_{C,\Omega}(\boldsymbol{Y}_{\text{true}}; \boldsymbol{\theta}) \triangleq \mathrm{DP}_\Omega(\boldsymbol{\theta} + \boldsymbol{C}_{\boldsymbol{Y}_{\text{true}}}) - \langle \boldsymbol{Y}_{\text{true}}, \boldsymbol{\theta} \rangle.$$

Losses in this class are convex, smooth, tractable for any Ω, and by Proposition 2 property 2 a sensible approximation of ℓ_C. In addition, they only require to backpropagate through $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ at training time. It is easy to check that we recover the structured perceptron loss with $\ell_{0,0}$ (**?**), the structured hinge loss with $\ell_{C,0}$ (**?**) and the CRF loss with $\ell_{0,-H}$ (**?**). The last one has been used on top of LSTMs in several recent works (**??**). Minimizing $\ell_{0,-H}(\boldsymbol{\theta})$ is equivalent to maximizing the likelihood $p_{\boldsymbol{\theta},-H}(\boldsymbol{Y}_{\text{true}})$. However, minimizing $\ell_{0,\|\cdot\|^2}$ is *not* equivalent to maximizing $p_{\boldsymbol{\theta},\|\cdot\|^2}(\boldsymbol{Y}_{\text{true}})$. In fact, the former is convex while the latter is not.

Non-convex losses. A direct approach that uses the output distribution $\boldsymbol{p}_{\theta,\Omega}$ consists in minimizing the risk $\sum_{\boldsymbol{y} \in \mathcal{Y}} p_{\boldsymbol{\theta},-H}(\boldsymbol{Y}) C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y})$. As shown by **?**, this can be achieved by backpropagating through the minimum risk decoder. However, the risk is usually non-differentiable, piecewise constant (**?**) and several smoothing heuristics are necessary to make the method work (**?**).

Another principled approach is to consider a differentiable approximation $\Delta \colon \mathcal{Y} \times \mathrm{conv}(\mathcal{Y}) \to \mathbb{R}_+$ of the cost C. We can then relax $C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y}^\star(\boldsymbol{\theta}))$ by $\Delta(\boldsymbol{Y}_{\text{true}}, \nabla \mathrm{DP}_\Omega(\boldsymbol{\theta}))$.

Unlike minimum risk training, this approach is differentiable everywhere when $\Omega = -H$. Both approaches require to backpropagate through $\nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$, which is twice as costly as backpropagating through $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ (see §2.4).

4.2. Named entity recognition

Let $\boldsymbol{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T)$ be an input sentence, where each word $\boldsymbol{x}_t$ is represented by a vector in $\mathbb{R}^D$, computed using a neural recurrent architecture trained end-to-end. We wish to tag each word with named entities, *i.e.*, identify blocks of words that correspond to names, locations, dates, etc. We use the specialized operator Vit_Ω described in §3.1. We construct the potential tensor $\boldsymbol{\theta}(\boldsymbol{X}) \in \mathbb{R}^{T \times S \times S}$ as

$$\forall\, t > 1, \quad \theta(\boldsymbol{X})_{t,i,j} \triangleq \boldsymbol{w}_i^\top \boldsymbol{x}_t + b_i + t_{i,j},$$

and $\theta(\boldsymbol{X})_{1,i,j} \triangleq \boldsymbol{w}_i^\top \boldsymbol{x}_t + b_i$, where $(\boldsymbol{w}_i, b_i) \in \mathbb{R}^D \times \mathbb{R}$ is the linear classifier associated with tag i and $\boldsymbol{T} \in \mathbb{R}^{S \times S}$ is a transition matrix. We learn $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{T}$ along with the network producing $\boldsymbol{X}$, and compare two losses:

$$\begin{aligned} &\text{Surrogate convex loss:} && \ell_{0,\Omega}(\boldsymbol{Y}_{\text{true}}; \boldsymbol{\theta}), \\ &\text{Relaxed loss:} && \Delta(\boldsymbol{Y}_{\text{true}}, \nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})), \end{aligned}$$

where $\Delta(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y})$ is the squared ℓ_2 distance when $\Omega = \|\cdot\|_2^2$ and the Kullback-Leibler divergence when $\Omega = -H$, applied row-wise to the marginalization of $\boldsymbol{Y}_{\text{true}}$ and $\boldsymbol{Y}$.

Experiments. We measure the performance of the different losses and regularizations on the four languages of the CoNLL 2003 dataset. Following **?**, who use the $\ell_{0,-H}$ loss, we use a character LSTM and FastText (**?**) pretrained embeddings computed using *o*n Wikipedia. Those are fed to a word bidirectional LSTM to obtain $\boldsymbol{X}$. Architecture details are provided in §C.1. Results are reported in Table 1, along with reference results with different pretrained embeddings. We first note that the non-regularized structured perceptron loss $\ell_{0,0}$, that involves working with subgradients of $\mathrm{DP}(\boldsymbol{\theta})$, perform significantly worse than regularized losses. With proper parameter selections, all regularized losses perform within 1% F_1-score of each other, although entropy-regularized losses perform slightly better on ¾ languages. However, the ℓ_2^2-regularized losses yield sparse predictions, whereas entropy regularization always yields dense probability vectors. Qualitatively, this allows to identify ambiguous predictions more easily, as illustrated in §C.1. Sparse predictions also allows to enumerate all non-zero probability entities, and to trade precision for recall at test time.

4.3. Supervised audio-to-score transcription

We use our framework to perform supervised audio-to-score alignment on the Bach 10 dataset (**?**). The dataset consists of 10 music pieces with audio tracks, MIDI transcriptions, and annotated alignments between them. We transform the audio tracks into a sequence of audio frames using a feature extractor (see §C.2) to obtain a sequence $\boldsymbol{A} \in \mathbb{R}^{N_A \times D}$, while the associated score sequence is represented by $\boldsymbol{B} \in \mathbb{R}^{N_B \times K}$ (each row $\boldsymbol{b}_j$ is a one-hot vector corresponding to one key b_j). Each pair $(\boldsymbol{A}, \boldsymbol{B})$ is associated to an alignment $\boldsymbol{Y}_{\text{true}} \in \mathbb{R}^{N_A \times N_B}$. As described in §3.2, we define a discrepancy matrix $\boldsymbol{\theta} \in \mathbb{R}^{N_A \times N_B}$ between the elements of the two sequences. We set the cost between an audio frame and a key to be the log-likelihood of this key given a multinomial linear classifier:

$$\begin{aligned} \forall\, i \in [N_A], \boldsymbol{l}_i \triangleq -\log(\mathrm{softmax}(\boldsymbol{W}^\top \boldsymbol{a}_i + \boldsymbol{c})) \in \mathbb{R}^K \\ \text{and } \forall\, j \in [N_B],\ \theta_{i,j} \triangleq l_{i,b_j}, \end{aligned}$$

where $(\boldsymbol{W}, \boldsymbol{c}) \in \mathbb{R}^{D \times K} \times \mathbb{R}^K$ are learned classifier parameters. We predict a soft alignment by $\boldsymbol{Y} = \nabla \mathrm{DTW}_{-H}(\boldsymbol{\theta})$. Following (**?**), we define the relaxed loss

$$\Delta(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y}) \triangleq \|\boldsymbol{L}(\boldsymbol{Y} - \boldsymbol{Y}_{\text{true}})^\top\|_F^2,$$

where $\boldsymbol{L}$ a the lower triangular matrix filled with 1. When $\boldsymbol{Y} \in \mathcal{Y}$ is a true alignement matrix, $\Delta(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y})$ is the area between the path of $\boldsymbol{Y}_{\text{true}}$ and $\boldsymbol{Y}$, which corresponds to the *mean absolute deviation* in the audio literature. When $\boldsymbol{Y} \in \mathrm{conv}(\mathcal{Y})$, it is a convex relaxation of the area. At test time, once $\boldsymbol{\theta}$ is learned, we use the non-regularized DTW algorithm to output a hard alignment $\boldsymbol{Y}^\star(\boldsymbol{\theta}) \in \mathcal{Y}$.

Table 1. F_1 score comparison on CoNLL03 NER datasets.

Ω	Loss	English	Spanish	German	Dutch
Negent.	Surrogate	90.80	**86.68**	77.35	**87.56**
	Relaxed	90.47	86.20	**77.56**	87.37
ℓ_2^2	Surrogate	**90.86**	85.51	76.01	86.58
	Relaxed	89.49	84.07	76.91	85.90
0	Struct. perceptron	86.52	81.48	68.81	80.49
	(**?**)	*90.96*	*85.75*	*78.76*	*81.74*

Table 2. Mean absolute deviation of alignment using an end-to-end trained multinomial classifier and a pre-trained one.

Linear model	Train	Test
End-to-end trained	$\mathbf{0.17 \pm 0.01}$	$\mathbf{1.07 \pm 0.61}$
Pretrained	1.80 ± 0.14	3.69 ± 2.85
Random $\boldsymbol{\theta}$	14.64 ± 2.63	14.64 ± 0.29

Results. We perform a leave-one-out cross-validation of our model performance, learning the multinomial classifier on 9 pieces and assessing the quality of the alignment on the remaining piece. We report the mean absolute deviation on both train and test sets. A solid baseline consists in learning the multinomial classifier $(\boldsymbol{W}, \boldsymbol{c})$ beforehand, *i.e.*, without end-to-end training. We then use this model to compute $\boldsymbol{\theta}$ as in (4.3) and obtain $\boldsymbol{Y}^\star(\boldsymbol{\theta})$. As shown in

Table 2, our end-to-end technique outperforms this baseline by a large margin. We also demonstrate in §C.2 that the alignments obtained by end-to-end training are visibly closer to the ground truth. End-to-end training thus allows to *fine-tune* the distance matrix $\boldsymbol{\theta}$ for the task at hand.

5. Structured and sparse attention

We show in this section how to apply our framework to neural sequence-to-sequence models augmented with an attention mechanism (**?**). An encoder first produces a list of vectors $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$ representing the input sequence. A decoder is then used to greedily produce the corresponding output sequence. To simplify the notation, we focus on one time step of the decoding procedure. Given the decoder's current hidden state $\boldsymbol{z}$ and $\boldsymbol{X}$ as inputs, the role of the attention mechanism is to produce a distribution $\boldsymbol{w} \in \triangle^T$ over $\boldsymbol{X}$, for the current time step. This distribution is then typically used to produce a context vector $\boldsymbol{c} \triangleq \boldsymbol{X}^\top \boldsymbol{w}$, that is in turn invoved in the computation of the output sequence's next element.

Structured attention layers. **?** proposed a segmentation attention layer, which is capable of taking into account the transitions between elements of $\boldsymbol{X}$. They use a linear-chain CRF to model the probability $p_{\boldsymbol{\theta},-H}(\boldsymbol{y}|\boldsymbol{X})$ of a sequence $\boldsymbol{y} = (y_1, \dots, y_T)$, where each y_t is either 1 ("pay attention") or 0. They then propose to use normalized marginal probabilities as attention weights: $w_t \propto p_{\boldsymbol{\theta},-H}(y_t = 1|\boldsymbol{X})$. They show how to backpropagate gradients through the forward-backward algorithm, which they use to compute the marginal probabilities.

Generalizing structured attention. Using the notation from §3.1, any $\boldsymbol{y}$ can be represented as a tensor $\boldsymbol{Y} \in \{0,1\}^{T\times 2\times 2}$ and the potentials as a tensor $\boldsymbol{\theta} \in \mathbb{R}^{T\times 2\times 2}$. Similarly to **?**, we define

$$\theta_{t,1,j} \triangleq \boldsymbol{x}_t \boldsymbol{M} \boldsymbol{z} + t_{1,j} \quad \text{and} \quad \theta_{t,0,j} \triangleq t_{0,j},$$

where $\boldsymbol{x}\boldsymbol{M}\boldsymbol{z}$ is a learned bilinear form and $\boldsymbol{T} \in \mathbb{R}^{2\times 2}$ is a learned transition matrix. Following §3.1, the gradient $\nabla \mathrm{Vit}_\Omega(\boldsymbol{\theta})$ is equal to the expected matrix $\boldsymbol{E} \in \mathbb{R}^{T\times 2\times 2}$ and the marginals are obtained by marginalizing that matrix. Hence, we can set $w_t \propto p_{\boldsymbol{\theta},\Omega}(y_t = 1|\boldsymbol{X}) = e_{t,1,0} + e_{t,1,1}$. Backpropagating through $\nabla \mathrm{Vit}_\Omega(\boldsymbol{\theta})$ can be carried out using our approach outlined in §2.4. This approach is not only more general, but also simpler and more robust to underflow problems than backpropagating through the forward-backward algorithm as done by **?**.

Experiments. We demonstrate structured attention layers with an LSTM encoder and decoder to perform French to English translation using data from a 1 million sentence subset of the WMT14 FR-EN challenge. We illustrate an example of attenion map obtained with negentropy and ℓ_2^2 regularizations in Figure 4. Non-zero elements are underlined with borders: ℓ_2^2-regularized attention maps are sparse and more interpretable — this provides a structured alternative to sparsemax attention (**?**). Results were all within 0.8 point of BLEU score on the newstest2014 dataset. For French to English, standard softmax attention obtained **27.96**, while entropy and ℓ_2^2 regularized structured attention obtained **27.96** and **27.19** — introducing structure and sparsity therefore provides enhanced interpretability with comparable peformance. We provide model details, full results and further visualizations in §C.3.

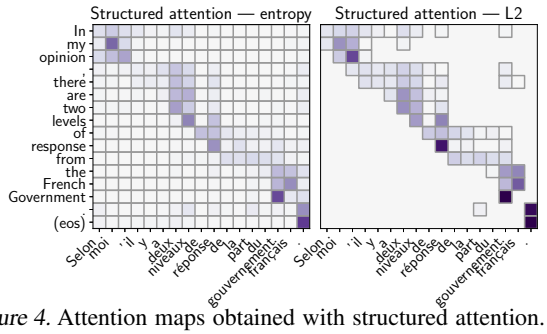

Figure 4. Attention maps obtained with structured attention. Although both regularizations led to the same translation (y-axis) in this example, attention is sparse and more interpretable with ℓ_2^2.

6. Conclusion

We proposed a theoretical framework for turning a broad class of dynamic programs into convex, differentiable and tractable operators, using the novel point of view of smoothed max operators. Our work sheds a new light on how to transform dynamic programs that predict hard assignments (*e.g.,* the maximum a-posteriori estimator in a probabilistic graphical model or an alignment matrix between two time-series) into continuous and probabilistic ones. We provided a new argument in favor of negentropy regularization by showing that it is the only one to preserve *associativity* of the smoothed max operator. We showed that different regularizations induce different distributions over outputs and that ℓ_2^2 regularization has other benefits, in terms of sparsity of the expected outputs. Generally speaking, performing inference in a graphical model and backpropagating through it reduces to computing the first and second-order derivatives of a relaxed maximum-likelihood estimation — leveraging this observation yields elegant and efficient algorithms that are readily usable in deep learning frameworks, with various promising applications.

Acknowledgements

MB thanks Vlad Niculae and Marco Cuturi for many fruitful discussions. AM thanks Julien Mairal, Inria Thoth and Inria Parietal for lending him the computational resources necessary to run the experiments. He thanks University Paris-Saclay for allowing him to do an internship at NTT, and Olivier Grisel for his insightful comments on natural language processing models.

Appendix

A. Proofs and detailed derivations

This section contains the proofs of the propositions and lemmas presented in the main text. It also contains derivations of gradient, directional derivative and Hessian-product computations.

A.1. Proof of Lemma 1 (properties of $\max_\Omega$)

Property 1 (boundedness). Let $\boldsymbol{q}^\star$ and $\boldsymbol{q}^\star_\Omega$ be the solutions of $\max_{\boldsymbol{q}\in\triangle^D} \boldsymbol{q}^\top \boldsymbol{x}$ and $\max_{\boldsymbol{q}\in\triangle^D} \boldsymbol{q}^\top \boldsymbol{x} - \Omega(\boldsymbol{q})$, respectively. Then, we have

$$\max_\Omega(\boldsymbol{x}) = \langle \boldsymbol{q}^\star_\Omega, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}^\star_\Omega) \geq \langle \boldsymbol{q}^\star, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}^\star) = \max(\boldsymbol{x}) - \Omega(\boldsymbol{q}^\star)$$

and

$$\max(\boldsymbol{x}) - \Omega(\boldsymbol{q}^\star_\Omega) \geq \langle \boldsymbol{q}^\star_\Omega, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}^\star_\Omega) = \max_\Omega(\boldsymbol{x}).$$

Combining the two and using $L_{\Omega,D} \leq \Omega(\boldsymbol{q}) \leq U_{\Omega,D}\ \forall \boldsymbol{q} \in \triangle^D$, we obtain

$$\max(\boldsymbol{x}) - U_{\Omega,D} \leq \max(\boldsymbol{x}) - \Omega(\boldsymbol{q}^\star) \leq \max_\Omega(\boldsymbol{x}) \leq \max(\boldsymbol{x}) - \Omega(\boldsymbol{q}^\star_\Omega) \leq \max(\boldsymbol{x}) - L_{\Omega,D}.$$

When $\Omega(\boldsymbol{q}) = \sum_i q_i \log q_i$, we have the tight inequality $-\log D \leq \Omega(\boldsymbol{q}) \leq 0\ \forall \boldsymbol{q} \in \triangle^D$ and hence

$$\max(\boldsymbol{x}) \leq \max_\Omega(\boldsymbol{x}) \leq \max(\boldsymbol{x}) + \log D.$$

When $\Omega(\boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{q}\|^2$, we have the tight inequality $\frac{1}{2D} \leq \Omega(\boldsymbol{q}) \leq \frac{1}{2}\ \forall \boldsymbol{q} \in \triangle^D$ and hence

$$\max(\boldsymbol{x}) - \frac{1}{2} \leq \max_\Omega(\boldsymbol{x}) \leq \max(\boldsymbol{x}) - \tfrac{1}{2D}.$$

Note that the difference $U_{\Omega,D} - L_{\Omega,D}$ is equal to $\log D$ when Ω is the negative entropy and to $\frac{D-1}{2D} \leq \frac{1}{2}$ when Ω is the squared ℓ_2 norm. Since $\log D > \frac{1}{2}$ for all integers $D \geq 2$, we get a better approximation of the $\max$ operator using squared ℓ_2 norm than using negative entropy, whenever $D \geq 2$.

Property 2 (distributivity of $+$ over $\max_\Omega$). This follows immediately from

$$\max_\Omega(\boldsymbol{x} + c\mathbf{1}) = \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{x} + c\mathbf{1}\rangle - \Omega(\boldsymbol{q}) = \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}) + c = \max_\Omega(\boldsymbol{x}) + c.$$

Using our shorthand notation, this simply becomes $\max_{\boldsymbol{Y}\in\mathcal{Y}}{}_\Omega\ (f(\boldsymbol{Y}) + c) = \left(\max_{\boldsymbol{Y}\in\mathcal{Y}}{}_\Omega\ f(\boldsymbol{Y})\right) + c.$

Property 3 (commutativity). Assume $\Omega(\boldsymbol{P}\boldsymbol{q}) = \Omega(\boldsymbol{q})$ for all permutation matrices $\boldsymbol{P}$. Let $\boldsymbol{P}^{-1}$ be the inverse permutation matrix associated with $\boldsymbol{P}$. Then we have

$$\begin{aligned}\max_\Omega(\boldsymbol{P}\boldsymbol{x}) &= \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{P}\boldsymbol{x}\rangle - \Omega(\boldsymbol{q}) = \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{P}^{-1}\boldsymbol{q}, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}) \\ &= \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{x}\rangle - \Omega(\boldsymbol{P}\boldsymbol{q}) = \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q}).\end{aligned}$$

Property 4 (non-decreasingness in each coordinate). If $\boldsymbol{x} \leq \boldsymbol{y}$, then for all $\boldsymbol{q} \in \triangle^D$, $\langle \boldsymbol{x}, \boldsymbol{q}\rangle - \Omega(\boldsymbol{q}) \leq \langle \boldsymbol{y}, \boldsymbol{q}\rangle - \Omega(\boldsymbol{q})$, as all $\boldsymbol{q}$ coordinates are non-negative. Thus $\max_\Omega(\boldsymbol{x}) \leq \max_\Omega(\boldsymbol{y})$.

Property 5 (insensitivity to $-\infty$). Since $\max_\Omega(\boldsymbol{x}) = \max_{\boldsymbol{q}\in\triangle^D} \langle \boldsymbol{q}, \boldsymbol{x}\rangle - \Omega(\boldsymbol{q})$, if $x_j = -\infty$, then $q_j = \nabla \max_\Omega(\boldsymbol{x})_j = 0$ is the only feasible solution for the j^{th} coordinate.

A.2. Proof of Proposition 1 (optimality of DP recursion)

Let $v_i(\boldsymbol{\theta})$ be the highest-score path up to node $i \in [N]$. Let $\mathcal{Y}_i$ be the set of paths $\boldsymbol{y} = (y_1, \dots, y_L)$ starting from node 1 and reaching node i, that is $y_1 = 1$ and $y_L = i$. Note that L may depend on $\boldsymbol{y}$ but we do not make this dependency explicit. Because nodes are sorted in topological order, we can compute $v_i(\boldsymbol{\theta})$ by

$$v_i(\boldsymbol{\theta}) = \max_{\boldsymbol{y}\in\mathcal{Y}_i} \sum_{t=2}^{L} \theta_{y_t,y_{t-1}} = \max_{\boldsymbol{y}\in\mathcal{Y}_i} \sum_{t=2}^{L-1} \theta_{y_t,y_{t-1}} + \theta_{y_L,y_{L-1}} = \max_{\boldsymbol{y}\in\mathcal{Y}_i} \sum_{t=2}^{L-1} \theta_{y_t,y_{t-1}} + \theta_{i,y_{L-1}}.$$

Recall that $\mathcal{P}_i$ is the set of parent nodes of node i. From the *associativity* of the max operator,

$$v_i(\boldsymbol{\theta}) = \max_{j\in\mathcal{P}_i} \max_{\substack{\boldsymbol{y}\in\mathcal{Y}_i \\ y_{L-1}=j}} \left(\sum_{t=2}^{L-1} \theta_{y_t,y_{t-1}} + \theta_{i,y_{L-1}} \right) = \max_{j\in\mathcal{P}_i} \max_{\substack{\boldsymbol{y}\in\mathcal{Y}_i \\ y_{L-1}=j}} \left(\sum_{t=2}^{L-1} \theta_{y_t,y_{t-1}} + \theta_{i,j} \right).$$

From the *distributivity* of $+$ over $\max$, we obtain

$$v_i(\boldsymbol{\theta}) = \max_{j\in\mathcal{P}_i} \left(\max_{\substack{\boldsymbol{y}\in\mathcal{Y}_i \\ y_{L-1}=j}} \sum_{t=2}^{L-1} \theta_{y_t,y_{t-1}} \right) + \theta_{i,j} = \max_{j\in\mathcal{P}_i} v_j(\boldsymbol{\theta}) + \theta_{i,j},$$

where we used the fact that the inner max operations are independent of $y_L = i$. This concludes the proof of the optimality of (3).

A.3. Proof of Proposition 2 (properties of $\text{DP}_\Omega(\boldsymbol{\theta})$)

We prove in this section the three main claims of Proposition 2. For the first two claims, we rewrite (3) and (6) using the following notations:

$$\begin{aligned}
&v_i^0(\boldsymbol{\theta}) \triangleq \max(\boldsymbol{u}_i^0(\boldsymbol{\theta})) \quad \text{and} \quad v_i^\Omega(\boldsymbol{\theta}) \triangleq \max(\boldsymbol{u}_i^\Omega(\boldsymbol{\theta})), \quad \text{where} \\
&\boldsymbol{u}_i^0(\boldsymbol{\theta}) \triangleq (\theta_{i,1} + v_1^0(\boldsymbol{\theta}), \dots, \theta_{i,i-1} + v_{i-1}^0(\boldsymbol{\theta}), -\infty, -\infty, \dots, -\infty) \in \mathbb{R}^N \quad \text{and} \\
&\boldsymbol{u}_i^\Omega(\boldsymbol{\theta}) \triangleq (\theta_{i,1} + v_1^\Omega(\boldsymbol{\theta}), \dots, \theta_{i,i-1} + v_{i-1}^\Omega(\boldsymbol{\theta}), \underbrace{-\infty}_{i}, -\infty, \dots, -\infty) \in \mathbb{R}^N.
\end{aligned}$$

These definitions are indeed valid as per Lemma 1, property 5.

Proof of $\text{DP}_\Omega(\boldsymbol{\theta})$ convexity. Since $v_1^\Omega(\boldsymbol{\theta}) = 0$, it is trivially convex. Assume that $v_2^\Omega(\boldsymbol{\theta}), \dots, v_{i-1}^\Omega(\boldsymbol{\theta})$ are convex. Then, $v_i^\Omega(\boldsymbol{\theta})$ is the composition of $\max_\Omega$ and $\boldsymbol{u}_i^\Omega$, a convex function and a function which outputs a vector whose each coordinate is convex in $\boldsymbol{\theta}$. By induction, since $\max_\Omega$ is non-decreasing per coordinate (cf. Lemma 1 property 4), $v_i^\Omega(\boldsymbol{\theta})$ is convex (*e.g.,* **?**, §3.2.4). Therefore $v_i^\Omega(\boldsymbol{\theta})$ is convex for all $i \in [N]$ and $\text{DP}_\Omega(\boldsymbol{\theta}) = v_N^\Omega(\boldsymbol{\theta})$ is convex.

Proof of $\text{DP}_\Omega(\boldsymbol{\theta})$ bound. We clearly have $v_1^\Omega(\boldsymbol{\theta}) \geq v_1^0(\boldsymbol{\theta})$. Assume that $v_j^\Omega(\boldsymbol{\theta}) \geq v_j^0(\boldsymbol{\theta}) - (j-1)U_{\Omega,N}$ for all $j \in \{2, \dots, i-1\}$. That is, $\boldsymbol{u}_i^\Omega(\boldsymbol{\theta}) \geq \boldsymbol{u}_i^0(\boldsymbol{\theta}) - (i-2)U_{\Omega,N}\mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^N$ is the unit vector. Then, by induction, we have

$$\max_\Omega(\boldsymbol{u}_i^\Omega(\boldsymbol{\theta})) \geq \max_\Omega(\boldsymbol{u}_i^0(\boldsymbol{\theta})) - (i-2)U_{\Omega,N} \geq \max(\boldsymbol{u}_i^0(\boldsymbol{\theta})) - (i-1)U_{\Omega,N},$$

where we used Lemma 1, properties 1, 2 and 4. Therefore $v_i^\Omega(\boldsymbol{\theta}) \geq v_i^0(\boldsymbol{\theta}) - (i-1)U_{\Omega,N}$ for all $i \in [N]$ and hence, $\text{DP}_\Omega(\boldsymbol{\theta}) \geq \text{LP}(\boldsymbol{\theta}) - (N-1)U_{\Omega,N}$. Using a similar reasoning we

obtain $v_i^0(\boldsymbol{\theta}) - (i-1)L_{\Omega,N} \geq v_i^\Omega(\boldsymbol{\theta})$ and therefore $\mathrm{LP}(\boldsymbol{\theta}) - (N-1)L_{\Omega,N} \geq \mathrm{DP}_\Omega(\boldsymbol{\theta})$. To summarize, we obtain

$$\mathrm{LP}(\boldsymbol{\theta}) - (N-1)L_{\Omega,N} \geq \mathrm{DP}_\Omega(\boldsymbol{\theta}) \geq \mathrm{LP}(\boldsymbol{\theta}) - (N-1)U_{\Omega,N},$$

which concludes the proof. Note that using property 1 of Lemma 1, this immediately implies a bound involving $\mathrm{LP}_\Omega(\boldsymbol{\theta})$ instead of $\mathrm{LP}(\boldsymbol{\theta})$.

Proof that $\Omega = -\gamma H \Rightarrow \mathbf{DP}_\Omega(\boldsymbol{\theta}) = \mathbf{LP}_\Omega(\boldsymbol{\theta})$. We first show that $\max_\Omega$ is associative.

Lemma 2. *Associativity of $\max_\Omega$ when $\Omega = -\gamma H$*

We have $\max_\Omega(\max_\Omega(\boldsymbol{x}), c) = \max_\Omega(\boldsymbol{x}, c) \quad \forall \boldsymbol{x} \in \mathbb{R}^D, c \in \mathbb{R}$.

Proof. We simply use the closed form of $\max_\Omega$ when $\Omega = -\gamma H$ (cf. §B.1):

$$\begin{aligned}
\max_\Omega(\max_\Omega(\boldsymbol{x}), c) &= \gamma \log(\exp(\max_\Omega(\boldsymbol{x})/\gamma) + \exp(c/\gamma)) \\
&= \gamma \log\left(\exp\left(\log \sum_{i=1}^D \exp(x_i/\gamma)\right) + \exp(c/\gamma)\right) \\
&= \gamma \log\left(\sum_{i=1}^D \exp(x_i/\gamma) + \exp(c/\gamma)\right) \\
&= \max_\Omega(\boldsymbol{x}, c),
\end{aligned}$$

and the lemma follows. □

Using our shorthand notation, Lemma 2 can be used to write

$$\max_{\Omega \atop (y_1,\ldots,y_i,\ldots,y_L)} f(\boldsymbol{y}) = \max_{\Omega \atop v} \max_{\Omega \atop (y_1,\ldots,v,\ldots,y_L)} f(\boldsymbol{y}).$$

This is precisely the associative property that we used in the proof of Proposition 1. The second property that we used, the distributivity of $+$ over $\max$, holds for any $\max_\Omega$, as per Lemma 1 property 2. Thus, the same proof as Proposition 1 is also valid when we substitute $\max$ with $\max_\Omega$, when $\Omega = -\gamma H$, which yields $\mathrm{LP}_\Omega(\boldsymbol{\theta}) = \mathrm{DP}_\Omega(\boldsymbol{\theta})$.

Proof that $\Omega = -\gamma H \Leftarrow \mathbf{DP}_\Omega(\boldsymbol{\theta}) = \mathbf{LP}_\Omega(\boldsymbol{\theta})$. Mirroring the previous proof, we first characterize the regularizations Ω for which $\max_\Omega$ is associative.

Lemma 3. *Let $\Omega \colon \triangle^D \to \mathbb{R}$ be a regularization function, i.e., $\operatorname{dom}\Omega = \triangle^D$. Assume that there exist ω convex lower-semi-continuous defined on $[0,1]$ such that $\Omega(\boldsymbol{q}) = \sum_{i=1}^d \omega(q_i)$. If*

$$\max_\Omega(\max_\Omega(\boldsymbol{x}), c) = \max_\Omega(\boldsymbol{x}, c) \quad \forall \boldsymbol{x} \in \mathbb{R}^D, c \in \mathbb{R},$$

then $\Omega(\boldsymbol{q}) = -\gamma \sum_{i=1}^d q_i \log(q_i)$ for some $\gamma \geq 0$.

Proof. We start by writing the associativity property for three elements. For all $x_1, x_2, x_3 \in \mathbb{R}$,

$$\begin{aligned}
\max_\Omega\big((x_1, x_2, x_3)\big) &= \max_\Omega\big(\max_\Omega(x_1, x_2), x_3\big) \\
&= \max_{\substack{q+q_3=1 \\ q,q_3\geq 0}} q \max_{\substack{\tilde{q}_1+\tilde{q}_2=1 \\ \tilde{q}_i\geq 0}} \big(\tilde{q}_1 x_1 + \tilde{q}_2 x_2 - \omega(\tilde{q}_1) - \omega(\tilde{q}_2)\big) + q_3 x_3 - \omega(q_3) - \omega(q) \\
&= \max_{\substack{q_1+q_2+q_3=1 \\ q_i\geq 0}} q_1 x_1 + q_2 x_2 + q_3 x_3 - \Phi(q_1, q_2, q_3), \quad \text{where} \\
\Phi(q_1, q_2, q_3) &\triangleq (q_1+q_2)\Big(\omega\big(\frac{q_1}{q_1+q_2}\big) + \omega\big(\frac{q_2}{q_1+q_2}\big)\Big) + \omega(q_1+q_2) + \omega(q_3).
\end{aligned}$$

We have performed a variable change $q_{1,2} = q\,\tilde{q}_{1,2}$ at the second line, and noticed $q = q_1 + q_2$. Therefore

$$\max_\Omega\big((x_1, x_2, x_3)\big) = \Phi^\star(x_1, x_2, x_3),$$

where $\Phi^\star$ is the convex conjugate of Φ restricted to $]0, 1]^3$. By definition, we also have $\max_\Omega\big((x_1, x_2, x_3)\big) = \Omega^\star(x_1, x_2, x_3)$, so that $\Omega^\star = \Phi^\star$ on $\mathbb{R}^3$. As Ω is convex and lower semi-continous, we can apply Moreau-Yoshida theorem and obtain $\Omega^{\star\star} = \Omega = \Phi^{\star\star} \leq \Phi$.

Suppose that there exists $\boldsymbol{q} = (q_1, q_2, q_3) \in \triangle^3$ such that $\Phi(q_1, q_2, q_3) < \Omega(q_1, q_2, q_3)$. Given the forms of Φ and Ω, $\Phi(q_1, q_2, 0) < \Omega(q_1, q_2, 0)$. We let $\boldsymbol{x} = (x_1, x_2, -\infty) \in \mathbb{R}^3$ such that

$$\begin{aligned}\max_\Omega(x_1, x_2, -\infty) = \max_\Omega(x_1, x_2) = x_1 q_1 + x_2 q_2 - \omega(q_1) - \omega(q_2) = \langle \boldsymbol{x}, \boldsymbol{q}\rangle - \Omega(\boldsymbol{q}) \\ < \langle \boldsymbol{x}, \boldsymbol{q}\rangle - \Phi(\boldsymbol{q}) \leq \max_{\boldsymbol{q}\in\triangle^3} \langle \boldsymbol{x}, \boldsymbol{q}\rangle - \Phi(\boldsymbol{q}) = \max_\Omega\big(\max_\Omega(x_1, x_2), -\infty\big),\end{aligned}$$

leading to a contradiction. Therefore $\Omega \geq \Phi$ over $\triangle^3$, and finally $\Omega = \Phi$. We have used the fact that the operator $\nabla\max_\Omega : \mathbb{R}^2 \to \triangle^2$ is surjective, as $\triangle^2$ is a one-dimensional segment, $\nabla\max_\Omega$ is continuous and reaches the extreme values $\nabla\max_\Omega(0, -\infty) = (1, 0)$ and $\nabla\max_\Omega(-\infty, 0) = (0, 1)$ — which allows to use the intermediate value theorem.

To conclude, for all $q_1, q_2 \in]0, 1]$ such that $q_1 + q_2 \leq 1$, we have

$$\begin{aligned}\omega(q_1) + \omega(q_2) &= (q_1 + q_2)\Big(\omega\big(\frac{q_1}{q_1 + q_2}\big) + \omega\big(\frac{q_2}{q_1 + q_2}\big)\Big) + \omega(q_1 + q_2) \\ \omega(xy) + \omega((1 - x)y) - \omega(y) &= y(\omega(x) + \omega(1 - x)) \quad \forall\, 0 < y \leq 1,\ 0 < x < 1, \qquad (8)\end{aligned}$$

where we have set $y = q_1 + q_2$ and $x = \frac{q_1}{q_1 + q_2}$. The functional equation (8) was first studied in the field of information theory. As first shown by **?**, Theorem 0, and further extended (**?**), all measurable solutions have the form

$$\omega(x) = -\gamma x \log(x),$$

where $\gamma \geq 0$ is a constant. The lemma follows. □

Assuming that Ω is not equal to $-\gamma H$ for any $\gamma \geq 0$, the previous lemma tells us that the associativity property is not met for a triplet $(x_1, x_2, x_3) \in \mathbb{R}^3$. In Figure 5, we construct a graph G such that

$$\mathrm{DP}_\Omega(\boldsymbol{\theta}) = \max_\Omega(\max_\Omega(x_1, x_2), x_3) \neq \mathrm{LP}_\Omega(\boldsymbol{\theta}) = \max_\Omega(x_1, x_2, x_3)$$

The proposition follows.

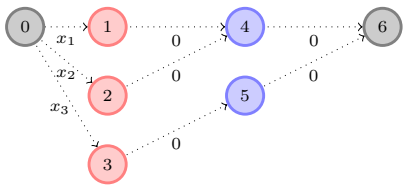

Figure 5. In general, $v_6(\boldsymbol{\theta}) = \mathrm{DP}_\Omega(\boldsymbol{\theta}) \neq \mathrm{LP}_\Omega(\boldsymbol{\theta})$.

A.4. Computation of $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta})$ and interpretation as an expectation

We show that $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta}) \in \mathrm{conv}(\mathcal{Y})$, and characterize a path distribution of which $\nabla\mathrm{LP}_\Omega(\boldsymbol{\theta})$ is the expectation.

Convex hull of $\mathcal{Y}$. We rewrite $\mathrm{LP}_\Omega(\boldsymbol{\theta}) = \max_\Omega(\boldsymbol{u}(\boldsymbol{\theta}))$, where $\boldsymbol{u}(\boldsymbol{\theta}) \triangleq (\langle \boldsymbol{Y}, \boldsymbol{\theta} \rangle)_{\boldsymbol{Y} \in \mathcal{Y}}$. Using the chain rule, we have

$$\nabla \mathrm{LP}_\Omega(\boldsymbol{\theta}) = \boldsymbol{J}_{\boldsymbol{u}}(\boldsymbol{\theta})^\top \nabla \max_\Omega(\boldsymbol{u}(\boldsymbol{\theta})), \tag{9}$$

where $\boldsymbol{J}_{\boldsymbol{u}}$ is the Jacobian of $\boldsymbol{u}$ w.r.t. $\boldsymbol{\theta}$, a matrix of size $|\mathcal{Y}| \times (N \times N)$. The horizontal slices of $\boldsymbol{J}_{\boldsymbol{u}}$ are exactly all the paths $\boldsymbol{Y}$ of $\mathcal{Y}$. Using $\nabla \max_\Omega(\boldsymbol{u}(\boldsymbol{\theta})) \in \triangle^{|\mathcal{Y}|}$, we conclude that $\nabla \mathrm{LP}_\Omega(\boldsymbol{\theta}) \in \mathrm{conv}(\mathcal{Y})$.

Induced distribution. From (9), we see that $\nabla \mathrm{LP}_\Omega(\boldsymbol{\theta}) = \sum_{\boldsymbol{Y} \in \mathcal{Y}} p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y})\ \boldsymbol{Y}$, where we defined the distribution

$$p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y}) \triangleq \Big(\nabla \max_\Omega(\boldsymbol{u}(\boldsymbol{\theta}))\Big)_{\boldsymbol{Y}}.$$

Unfortunately, since $\boldsymbol{u}(\boldsymbol{\theta}) \in \mathbb{R}^{|\mathcal{Y}|}$, computing $p_{\boldsymbol{\theta},\Omega}(\boldsymbol{Y})$, let alone the expectation $\mathbb{E}_{\boldsymbol{\theta},\Omega}[\boldsymbol{Y}]$ under that distribution, is intractable for general Ω.

A.5. Proof of Proposition 3 (computation of $\nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$)

Gradient computation. We first derive the recursion over $\boldsymbol{E} \triangleq \nabla \mathrm{DP}_\Omega(\boldsymbol{\theta})$ using sensitivity analysis, a.k.a backpropagation calculus. For any $(i,j) \in \mathcal{E}$, since $\theta_{i,j}$ influences only v_i, a straighforward application of the chain rule gives

$$e_{i,j} = \frac{\partial v_N}{\partial \theta_{i,j}} = \frac{\partial v_N}{\partial v_i} \frac{\partial v_i}{\partial \theta_{i,j}}. \tag{10}$$

Recall that $\boldsymbol{v} = (v_1, \dots, v_N)$ and $\boldsymbol{q}_i \triangleq \nabla \max_\Omega(\boldsymbol{\theta}_i + \boldsymbol{v})$. With this vector defined, we can now easily derive the two terms on the r.h.s of (10). Differentiating (6) w.r.t. $\theta_{i,j}$ straighforwardly gives the second term $\frac{\partial v_i}{\partial \theta_{i,j}} = q_{i,j}$.

The first term must be computed recursively. Recall that $\mathcal{C}_j$ denotes the children of node j. Since a node j influences only its children $i \in \mathcal{C}_j$, using the chain rule, we get

$$\frac{\partial v_N}{\partial v_j} = \sum_{i \in \mathcal{C}_j} \frac{\partial v_N}{\partial v_i} \frac{\partial v_i}{\partial v_j} \triangleq \bar{e}_j. \tag{11}$$

Differentiating (6) w.r.t. v_j again gives $\frac{\partial v_i}{\partial v_j} = q_{i,j}$. By definition, we also have $\frac{\partial v_N}{\partial v_i} = \bar{e}_i$ and $e_{i,j} = \bar{e}_i q_{i,j}$. Hence,

$$\bar{e}_j = \sum_{i \in \mathcal{C}_j} \bar{e}_i q_{i,j} = \sum_{i \in \mathcal{C}_j} e_{i,j}.$$

Combining the above, for any $j \in [N-1]$, we obtain the following two-step recursion

$$\forall\, i \in \mathcal{C}_j,\ e_{i,j} = \bar{e}_i q_{i,j} \quad \text{and} \quad \bar{e}_j = \sum_{i \in \mathcal{C}_j} e_{i,j}.$$

The values $(e_{i,j})_{(i,j) \in \mathcal{E}}$ can thus be computed in reverse topological order over the nodes of G, initializing $\bar{e}_N = \frac{\partial v_N}{\partial v_N} = 1$. The pseudo-code is summarized in Algorithm 1.

Associated random walk. It remains to show that $\boldsymbol{E}$ is also the expectation of $\boldsymbol{Y} \in \mathcal{Y}$ support of the following random walk, defined informally in the main text. Formally, we define the random sequence $(w_t)_t$ as

$$w_0 = N, \quad \forall\, t > 0, \forall\, i \in [N], \forall\, j \in \mathcal{P}_i, \quad \mathbb{P}[w_t = j | w_{t-1} = i] = q_{i,j}.$$

We set $y_{i,j} \triangleq \mathbf{1}\{\exists\, t > 0 \text{ s.t. } w_{t-1} = i, w_t = j\}$ where $\mathbf{1}$ is the characteristic function of an event, thereby defining a random variable $\boldsymbol{Y} \in \mathcal{Y}$, with distribution $\mathcal{D}$. We leave implicit the dependency of $\mathbb{P}$ in $\boldsymbol{\theta}$ and Ω. As the depth of w_t (number of edges to connect to the root

Algorithm 1 Compute $\mathrm{DP}_\Omega(\boldsymbol{\theta})$ and $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$

Input: Edge weights $\boldsymbol{\theta} \in \mathbb{R}^{N\times N}$
$v_1 \leftarrow 0, \quad \bar{e}_N \leftarrow 1, \qquad \boldsymbol{Q}, \boldsymbol{E} \leftarrow \mathbf{0} \in \mathbb{R}^{N\times N}$
for $i \in [2, \dots, N]$ **do** ▷ Topological order
$v_i \leftarrow \max_\Omega \theta_{i,j} + v_j$
$(\boldsymbol{q}_{i,j})_{j\in\mathcal{P}_i} \leftarrow \nabla\max_\Omega \theta_{i,j} + v_j$
for $j \in [N-1, \dots, 1]$ **do** ▷ Reverse topological order
$\forall\, i \in \mathcal{C}_j,\ e_{i,j} \leftarrow q_{i,j}\bar{e}_i, \qquad \bar{e}_j \leftarrow \sum_{i\in\mathcal{C}_j} e_{i,j}$
Return: $\mathrm{DP}_\Omega(\boldsymbol{\theta}) = v_N$, $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}) = \boldsymbol{E} \in \mathbb{R}^{N\times N}$
Intermediate computation for Algorithm 2
$\bar{\boldsymbol{e}} \triangleq [\bar{e}]_{i=1}^N \in \mathbb{R}^N$, $\boldsymbol{Q} \in \mathbb{R}^{N\times N}$

Algorithm 2 Compute $\langle\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$ and $\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$

Input: Edge weights and perturbation $\boldsymbol{\theta}, \boldsymbol{Z} \in \mathbb{R}^{N\times N}$
Call Algorithm 1 with input $\boldsymbol{\theta}$ to get $\bar{\boldsymbol{e}}$ and $\boldsymbol{Q}$
$\dot{v}_1 \leftarrow 0; \qquad \dot{e_N} \leftarrow 0, \qquad \dot{\boldsymbol{Q}}, \dot{\boldsymbol{E}} \leftarrow \mathbf{0} \in \mathbb{R}^{N\times N}$
for $i \in [2, \dots, N]$ **do** ▷ Topological order
$\dot{v}_i \leftarrow \sum_{j\in\mathcal{P}_i} q_{i,j}(z_{i,j} + \dot{v}_j)$ (A1)
$(\dot{\boldsymbol{q}}_{i,j})_{j\in\mathcal{P}_i} \leftarrow \boldsymbol{J}_\Omega\big((\boldsymbol{q}_{i,j})_{j\in\mathcal{P}_i}\big)(z_{i,j} + \dot{v}_j)_{j\in\mathcal{P}_i}$ (A2)
for $j \in [N-1, \dots, 1]$ **do** ▷ Reverse topological order
$\forall\, i \in \mathcal{C}_j,\ \dot{e}_{i,j} \leftarrow \dot{q}_{i,j}\bar{e}_i + q_{i,j}\dot{\bar{e}}_i$ (A3)
$\dot{\bar{e}}_j \leftarrow \sum_{i\in\mathcal{C}_j} \dot{e}_{i,j}$
Return: $\langle\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle = \dot{v}_N$
$\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z} = \dot{\boldsymbol{E}} \in \mathbb{R}^{N\times N}$

node) is stricly decreasing with t, $(w_t)_t$ reaches node 1 in finite time with probability one and is constant after this event. We introduce the random variables $(\bar{y}_j)_j$, defined for all $j \in [N]$ as

$$\bar{y}_j \triangleq \mathbf{1}\{\exists\, t \geq 0, w_t = j\} = \sum_{i\in\mathcal{C}_j} y_{i,j} \text{ if } j \neq N, 0 \text{ otherwise.}$$

By definition, using the fact that $\mathbb{P}[w_t = j | w_{t-1} = i]$ is independent of t (Markov property), for all $i \in \mathcal{C}_j$ and for all $j \in [N-1]$, we have

$$\mathbb{P}[y_{i,j} = 1] = \mathbb{E}[y_{i,j}] = \mathbb{P}[\exists\, t > 0, w_{t-1} = i]\mathbb{P}[w_t = j | w_{t-1} = i] = \mathbb{E}[\bar{y}_i]q_{i,j}.$$

Linearity of the expectation then provides

$$\mathbb{E}[\bar{y}_j] = \sum_{i\in\mathcal{C}_j} \mathbb{E}[y_{i,j}],$$

with initialization $\mathbb{E}[\bar{y}_N] = 1$. We recover the same two-step recursion as the one defining $\boldsymbol{E}$ and $\bar{\boldsymbol{e}}$, with the same initialization. Hence the probabilistic interpretation of the gradient, where the expectation is taken with respect to the distribution $\mathcal{D}$ of $\boldsymbol{Y}$:

$$\boldsymbol{E} = \mathbb{E}_{\boldsymbol{\theta},\Omega}[\boldsymbol{Y}] \quad \text{and} \quad \bar{\boldsymbol{e}} = \mathbb{E}_{\boldsymbol{\theta},\Omega}[\bar{\boldsymbol{y}}].$$

A.6. Computation of the directional derivative $\langle\nabla\mathbf{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$

The derivations of the following two sections allows to write Algorithm 2. Let $\dot{v}_i \triangleq \langle\nabla v_i(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$, where $v_i(\boldsymbol{\theta})$ is defined in (6). Since v_i only directly depends on $v_j + \theta_{i,j}$ for $j \in \mathcal{P}_i$, a straighforward differentiation of $\langle\nabla v_i(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$ gives

$$\dot{v}_i = \sum_{j\in\mathcal{P}_i} \frac{\partial v_i}{\partial v_j}\ (\dot{v}_j + z_{i,j})\,.$$

Recall that $\frac{\partial v_i}{\partial v_j} = q_{i,j}$ and has already been obtained when computing $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$. Hence equation (A1), reproduced here:

$$\forall\, i \in [2, \dots, N]: \qquad \dot{v}_i = \sum_{j\in\mathcal{P}_i} q_{i,j}(\dot{v}_j + z_{i,j}). \tag{12}$$

This recursion can be computed in topological order, starting from $\dot{v}_1 = 0$ to finish at $\dot{v}_N = \langle\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$.

A.7. Computation of the Hessian-vector product $\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$

For convenience, let us define $\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z} \triangleq \dot{\boldsymbol{E}}$. For $(i,j) \notin \mathcal{E}$, we evidently have $\dot{e}_{i,j} = 0$. For $(i,j) \in \mathcal{E}$, since $\theta_{i,j}$ influences only v_i and $\dot{v}_i$, we obtain

$$\dot{e}_{i,j} = \frac{\partial \dot{v}_N}{\partial \theta_{i,j}} = \frac{\partial \dot{v}_N}{\partial v_i}\frac{\partial v_i}{\partial \theta_{i,j}} + \frac{\partial \dot{v}_N}{\partial \dot{v}_i}\frac{\partial \dot{v}_i}{\partial \theta_{i,j}}.$$

We will now show how to derive each of the right-hand side terms in turn. We already know that $\frac{\partial v_i}{\partial \theta_{i,j}} = q_{i,j}$. We also have $\frac{\partial \dot{v}_N}{\partial \dot{v}_i} = u_i$. Indeed, observe that $\dot{v}_j$ only directly influences $\dot{v}_i$ for $i \in \mathcal{C}_i$. Therefore, we have

$$\frac{\partial \dot{v}_N}{\partial \dot{v}_j} = \sum_{i \in \mathcal{C}_j} \frac{\partial \dot{v}_N}{\partial \dot{v}_i} q_{i,j} \quad \forall j \in [N-1] \tag{13}$$

and $\frac{\partial \dot{v}_N}{\partial \dot{v}_1} = 1$. Comparing (11) and (13), we see that $(\frac{\partial \dot{v}_N}{\partial \dot{v}_i})_i$ follows the same recursion as $(\frac{\partial v_N}{\partial v_i})_i$. Since $\frac{\partial \dot{v}_N}{\partial \dot{v}_n} = \frac{\partial v_N}{\partial v_n}$, both sequences are equal:

$$\frac{\partial \dot{v}_N}{\partial \dot{v}_i} = \frac{\partial v_N}{\partial v_i} = e_i.$$

Next, we derive $\frac{\partial \dot{v}_i}{\partial \theta_{i,j}}$. Since, for $j \in \mathcal{P}_i$, $\dot{v}_j + z_{i,j}$ does not depend on $\theta_{i,j}$, differentiating (12) w.r.t. $\theta_{i,j}$, we obtain

$$\begin{aligned}\frac{\partial \dot{v}_i}{\partial \theta_{i,j}} &= \sum_{k \in \mathcal{P}_i} \frac{\partial q_{i,j}}{\partial \theta_{i,j}}(\dot{v}_k + z_{i,k}) \\ &= \sum_{k \in \mathcal{P}_i} \frac{\partial^2 v_i}{\partial \theta_{i,j}\partial \theta_{i,k}}(\dot{v}_k + z_{i,k}) \triangleq \dot{q}_{i,j}.\end{aligned}$$

This can be conveniently rewritten in a vectorial form as

$$\dot{\boldsymbol{q}}_i = \nabla^2 \max_\Omega(\boldsymbol{\theta}_i + \boldsymbol{v})\,(\boldsymbol{z}_i + \dot{\boldsymbol{v}}) = \boldsymbol{J}_\Omega(\boldsymbol{q}_i)\,(\boldsymbol{z}_i + \dot{\boldsymbol{v}}),$$

where we have defined $\dot{\boldsymbol{v}} \triangleq (\dot{v}_1, \ldots, \dot{v}_N)$ and where we have used the function $\boldsymbol{J}_\Omega$ defined in §B.1, that conveniently computes the Hessian of $\max_\Omega$ from its gradient. The Hessian has this form for both negentropy and ℓ_2^2 regularizations. In a practical implementation, we only need to compute the coordinates (i,j) of $\dot{\boldsymbol{Q}}$, for $j \in \mathcal{P}_i$. Namely, as specified in (A2),

$$(\dot{\boldsymbol{q}}_{i,j})_{j \in \mathcal{P}_i} \leftarrow \boldsymbol{J}_\Omega\big((\boldsymbol{q}_{i,j})_{j \in \mathcal{P}_i}\big)(z_{i,j} + \dot{v}_j)_{j \in \mathcal{P}_i}.$$

Finally, we derive $\frac{\partial \dot{v}_N}{\partial v_i}$. Since v_j influences only v_i and $\dot{v}_i$ for $i \in \mathcal{C}_j$, the chain rule gives

$$\frac{\partial \dot{v}_N}{\partial v_i} = \sum_{j \in \mathcal{C}_i} \frac{\partial \dot{v}_N}{\partial v_j}\frac{\partial v_j}{\partial v_i} + \frac{\partial \dot{v}_N}{\partial \dot{v}_j}\frac{\partial \dot{v}_j}{\partial v_i} = \sum_{j \in \mathcal{C}_j} \dot{e}_{i,j} \triangleq \dot{\dot{e}}_i.$$

Combining the above, for any $j \in [N-1]$, we obtain the following two-step recursion (A3), reproduced here:

$$\forall\, i \in \mathcal{C}_j, \quad \dot{e}_{i,j} = \dot{q}_{i,j} e_i + q_{i,j}\dot{\dot{e}}_i \quad \text{and} \quad \dot{\dot{e}}_j = \sum_{i \in \mathcal{C}_j} \dot{e}_{i,j}.$$

Similarly to the computation of $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$, our algorithm computes this recursion in reverse topological order over the graph G, yielding $\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z} = \dot{\boldsymbol{E}}$.

B. Examples of algorithm instantiations

We provide the explicit forms of $\max_\Omega$ and its derivative for the negentropy and ℓ_2^2 regularizations. Then, we provide details and pseudo-code for the two instances of differentiable dynamic programming presented in §3.

B.1. Examples of $\max_\Omega$

Negative entropy. When $\Omega(\boldsymbol{q}) = \gamma \sum_{i=1}^D q_i \log q_i$, where $\gamma > 0$ (smaller is less regularized), we obtain

$$\begin{aligned}
\max\nolimits_\Omega(\boldsymbol{x}) &= \gamma \log \left(\sum_{i=1}^D \exp(x_i/\gamma) \right) \\
\nabla \max\nolimits_\Omega(\boldsymbol{x}) &= \exp(\boldsymbol{x}/\gamma) \Big/ \sum_{i=1}^D \exp(x_i/\gamma) \\
\nabla^2 \max\nolimits_\Omega(\boldsymbol{x}) &= \boldsymbol{J}_\Omega(\nabla \max\nolimits_\Omega(\boldsymbol{x})),
\end{aligned}$$

where $\boldsymbol{J}_\Omega(\boldsymbol{q}) \triangleq (\mathrm{Diag}(\boldsymbol{q}) - \boldsymbol{q}\boldsymbol{q}^\top)/\gamma$. Note that $\nabla \max_\Omega(\boldsymbol{x})$ recovers the usual "softmax" with temperature $\gamma = 1$. For a proof of the expression of $\max_\Omega$, see, *e.g.,* (**?**, Example 3.25).

Squared ℓ_2 norm. When $\Omega(\boldsymbol{x}) = \frac{\gamma}{2}\|\boldsymbol{x}\|_2^2$ with $\gamma > 0$, we obtain the following expressions

$$\begin{aligned}
\max\nolimits_\Omega(\boldsymbol{x}) &= \langle \boldsymbol{q}^\star, \boldsymbol{x} \rangle - \frac{\gamma}{2}\|\boldsymbol{q}^\star\|_2^2 \\
\nabla \max\nolimits_\Omega(\boldsymbol{x}) &= \operatorname*{argmin}_{\boldsymbol{q} \in \triangle^D} \|\boldsymbol{q} - \boldsymbol{x}/\gamma\|_2^2 = \boldsymbol{q}^\star \\
\nabla^2 \max\nolimits_\Omega(\boldsymbol{x}) &= \boldsymbol{J}_\Omega(\nabla \max\nolimits_\Omega(\boldsymbol{x})),
\end{aligned}$$

where $\boldsymbol{J}_\Omega(\boldsymbol{q}) \triangleq (\mathrm{Diag}(\boldsymbol{s}) - \boldsymbol{s}\boldsymbol{s}^\top/\|\boldsymbol{s}\|_1)/\gamma$ and $\boldsymbol{s} \in \{0,1\}^D$ is a vector that indicates the support of $\boldsymbol{q}$. Note that $\nabla \max_\Omega(\boldsymbol{x})$ is precisely the Euclidean projection onto the simplex of $\boldsymbol{x}/\gamma$ and can be computed exactly in worst-case $\mathcal{O}(D \log D)$ time using the algorithm of (**?**) or in expected $\mathcal{O}(D)$ time using the randomized pivot algorithm of (**?**). It can be efficiently performed on Nvidia GPUs since recently. An important benefit of the squared ℓ_2 norm, compared to the negative entropy, is that $\nabla \max_\Omega(\boldsymbol{x})$ tends to be sparse. This is useful, among other things, to define sparse attention mechanisms (**??**).

B.2. Sequence prediction with the smoothed Viterbi algorithm

Computational graph. As illustrated in §3, the DAG contains a start node, S nodes for each time step and end node. Therefore $|\mathcal{V}| = N = TS + 2$. Only nodes from consecutive time steps are connected to each other. Taking into account the start and end nodes, the total number of edges is therefore $|\mathcal{E}| = (T-1)S^2 + 2S$.

Representation. We follow the notation of §3, *i.e.* we represent $\boldsymbol{Y}$ and $\boldsymbol{\theta}$ as $T \times S \times S$ tensors (we can safely ignore the edges connected to the end node since their value is 0). We represent $\boldsymbol{Y}$ as a binary tensor such that $y_{t,i,j} = 1$ if $\boldsymbol{Y}$ is in states i and j in time steps t and $t-1$, and $y_{t,i,j} = 0$ otherwise. Likewise, we represent the potentials $\boldsymbol{\theta}$ as a real tensor such that $\theta_{t,i,j}$ contains the potential of transitioning from state j to state i on time t.

Algorithms. Applying recursion (6) to this specific DAG, we obtain a smoothed version of the Viterbi algorithm. Let $v_{t,i}$ be the score of being in state i up to time t. We can rewrite the smoothed Bellman recursion as

$$v_{t,i}(\boldsymbol{\theta}) \triangleq \max_{j \in [S]}\nolimits_\Omega \, v_{t-1,j}(\boldsymbol{\theta}) + \theta_{t,i,j} = \max\nolimits_\Omega(\boldsymbol{v}_{t-1}(\boldsymbol{\theta}) + \boldsymbol{\theta}_{t,i}).$$

Algorithm 3 Compute $\text{Vit}_\Omega(\boldsymbol{\theta})$ and $\nabla\text{Vit}_\Omega(\boldsymbol{\theta})$

Input: Potential scores $\boldsymbol{\theta} \in \mathbb{R}^{T\times S\times S}$
$\triangleright$ Forward pass
$\boldsymbol{v}_0 = \mathbf{0}_S$
for $t \in [1,\dots,T], i \in [S]$ **do**
  $v_{t,i} = \max_\Omega(\boldsymbol{\theta}_{t,i} + \boldsymbol{v}_{t-1})$
  $\boldsymbol{q}_{t,i} = \nabla\max_\Omega(\boldsymbol{\theta}_{t,i} + \boldsymbol{v}_{t-1})$
$v_{T+1,1} = \max_\Omega(\boldsymbol{v}_T); \quad \boldsymbol{q}_{T+1,1} = \nabla\max_\Omega(\boldsymbol{v}_T)$
$\triangleright$ Backward pass
$\boldsymbol{u}_{T+1} = (1,0,\dots,0) \in \mathbb{R}^S$
for $t \in [T,\dots,0], j \in [S]$ **do**
  $\boldsymbol{e}_{t,\cdot,j} = \boldsymbol{q}_{t+1,\cdot,j} \circ \boldsymbol{u}_{t+1}; \quad u_{t,j} = \langle \boldsymbol{e}_{t,\cdot,j}, \mathbf{1}_S\rangle$
Return: $\text{Vit}_\Omega(\boldsymbol{\theta}) = v_{T+1,1}$
$\nabla\text{Vit}_\Omega(\boldsymbol{\theta}) = (e_{t-1,i,j})_{t=1,i,j=1}^{T,S,S}$
Intermediary computations for Alg. 4:
$\boldsymbol{Q} \triangleq (q)_{t=1,i,j=1}^{T+1,S,S}, \boldsymbol{U} \triangleq (u)_{t=1,j=1}^{T+1,S}$

Algorithm 4 Compute $\langle\nabla\text{Vit}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$ and $\nabla^2\text{Vit}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$

Input: $\boldsymbol{Z} \in \mathbb{R}^{T\times S\times S}, \boldsymbol{\theta} \in \mathbb{R}^{T\times S\times S}$
Call Alg. 3 with input $\boldsymbol{\theta}$ to get $\boldsymbol{U}, \boldsymbol{Q}$
$\triangleright$ Forward pass
$\dot{v}_0 = \mathbf{0}_S$
for $t \in [1,\dots,T], i \in [S]$ **do**
  $\dot{v}_{t,i} = \langle \boldsymbol{q}_{t,i}, \boldsymbol{z}_{t,i} + \dot{\boldsymbol{v}}_{t-1}\rangle$
  $\dot{\boldsymbol{q}}_{t,i} = \boldsymbol{J}_\Omega(\boldsymbol{q}_{t,i})\,(\boldsymbol{z}_t + \dot{\boldsymbol{v}}_{t-1})$
$\dot{v}_{T+1,1} = \langle \boldsymbol{q}_{T+1,1}, \dot{\boldsymbol{v}}_T\rangle; \quad \dot{\boldsymbol{q}}_{T+1,1} = \boldsymbol{J}_\Omega(\dot{\boldsymbol{q}}_{T+1,1})\,\dot{\boldsymbol{v}}_T$
$\triangleright$ Backward pass
$\dot{\boldsymbol{u}}_{T+1} = \mathbf{0}_S; \quad \dot{\boldsymbol{Q}}_{T+1} = \mathbf{0}_{S\times S}$
for $t \in [T,\dots,0], j \in [S]$ **do**
  $\dot{\boldsymbol{e}}_{t,\cdot,j} = \boldsymbol{q}_{t+1,\cdot,j} \circ \dot{\boldsymbol{u}}_{t+1} + \dot{\boldsymbol{q}}_{t+1,\cdot,j} \circ \boldsymbol{u}_{t+1}$
  $\dot{u}_{t,j} = \langle \dot{\boldsymbol{e}}_{t,\cdot,j}, \mathbf{1}_S\rangle$
Return: $\langle\text{Vit}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle = \dot{v}_{T+1}$
$\nabla^2\text{Vit}_\Omega(\boldsymbol{\theta})\boldsymbol{Z} = (\dot{e}_{t-1,i,j})_{t=1,i,j=1}^{T,S,S}$

The value $\text{Vit}_\Omega(\boldsymbol{\theta}) \triangleq \max_\Omega(\boldsymbol{v}_T(\boldsymbol{\theta}))$ can be computed in topological order, starting from $\boldsymbol{v}_0(\boldsymbol{\theta})$. The total computational cost is $\mathcal{O}(TS^2)$. Using the computations of §2.3 and §2.4 to this specific DAG, we can compute $\nabla\text{Vit}_\Omega(\boldsymbol{\theta})$, $\langle\nabla\text{Vit}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$ and $\nabla^2\text{Vit}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ with the same complexity. The procedures are summarized in Algorithm 3 and Algorithm 4, respectively. From Proposition 2 property 1, $\text{Vit}_\Omega(\boldsymbol{\theta})$ is a convex function for any Ω.

B.3. Monotonic aligment prediction with the smoothed DTW

Computational graph. As illustrated in §3, the DAG contains a start node and $N_A N_B$ nodes. Therefore, $|\mathcal{V}| = N = N_A N_B + 1$. Due to the monotonic constraint, each node may only be connected with at most 3 other nodes. The cardinality of $\mathcal{Y}$ is the delannoy(N_A-1, N_B-1) number (**??**). That number grows exponentially with N_A and N_B.

Representation. We follow the notation of §3, *i.e.* we represent $\boldsymbol{Y}$ and $\boldsymbol{\theta}$ as $N_A \times N_B$ matrices. We represent $\boldsymbol{Y}$ as a binary matrix such that $y_{i,j} = 1$ if $\boldsymbol{a}_i$ is aligned with $\boldsymbol{b}_j$, and $y_{i,j} = 0$ otherwise. Likewise, we represent $\boldsymbol{\theta}$ as a real matrix such that $\theta_{i,j}$ is a measure of "discrepancy" between $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$.

Algorithms. Following the DTW literature (**?**), we seek an alignment with *minimal* cost. For that reason, we introduce the smoothed min operator, its gradient and its Hessian as follows

$$\begin{aligned}
\min_\Omega(\boldsymbol{x}) &\triangleq -\max_\Omega(-\boldsymbol{x}) \\
\nabla\min_\Omega(\boldsymbol{x}) &= \nabla\max_\Omega(-\boldsymbol{x}) \\
\nabla^2\min_\Omega(\boldsymbol{x}) &= -\nabla^2\max_\Omega(-\boldsymbol{x}) \\
&= -\boldsymbol{J}_\Omega(\nabla\max_\Omega(-\boldsymbol{x})) \\
&= -\boldsymbol{J}_\Omega(\nabla\min_\Omega(\boldsymbol{x})).
\end{aligned}$$

Applying (6) to the DTW DAG gives rise to a smoothed version of the algorithm. Let $v_{i,j}(\boldsymbol{\theta})$ be the alignment cost up to cell (i,j). Then the smoothed DTW recursion is

$$v_{i,j}(\boldsymbol{\theta}) = \theta_{i,j} + \min_\Omega(v_{i,j-1}(\boldsymbol{\theta}), v_{i-1,j-1}(\boldsymbol{\theta}), v_{i-1,j}(\boldsymbol{\theta}))$$

The value $\text{DTW}_\Omega(\boldsymbol{\theta}) \triangleq \boldsymbol{v}_{N_A,N_B}(\boldsymbol{\theta})$ can be computed in $\mathcal{O}(N_A N_B)$ time. Applying the derivations of §2.3 and §2.4 to this specific DAG, we can compute $\nabla\text{DTW}_\Omega(\boldsymbol{\theta})$, $\langle\nabla\text{DTW}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$ and $\nabla^2\text{DTW}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ with the same complexity. The procedures, with appropriate handling of the edge cases, are summarized in Algorithm 5 and 6, respectively.

Algorithm 5 Compute $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ and $\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta})$

Input: Distance matrix $\boldsymbol{\theta} \in \mathbb{R}^{N_A \times N_B}$
▷ Forward pass
$v_{0,0} = 0$; $v_{i,0} = v_{0,j} = \infty$, $i \in [N_A], j \in [N_B]$
for $i \in [1, \dots, N_A], j \in [1, \dots, N_B]$ **do**
  $v_{i,j} = d_{i,j} + \min_\Omega(v_{i,j-1}, v_{i-1,j-1}, v_{i-1,j})$
  $\boldsymbol{q}_{i,j} = \nabla\min_\Omega(v_{i,j-1}, v_{i-1,j-1}, v_{i-1,j}) \in \mathbb{R}^3$
▷ Backward pass
$\boldsymbol{q}_{i,N_B+1} = \boldsymbol{q}_{N_A+1,j} = \mathbf{0}_3$, $i \in [N_A], j \in [N_B]$
$e_{i,N_B+1} = e_{N_A+1,j} = 0$, $i \in [N_A], j \in [N_B]$
$\boldsymbol{q}_{N_A+1,N_B+1} = (0, 1, 0)$; $e_{N_A+1,N_B+1} = 1$
for $j \in [N_B, \dots, 1], i \in [N_A, \dots, 1]$ **do**
  $e_{i,j} = q_{i,j+1,1}\, e_{i,j+1} + q_{i+1,j+1,2}\, e_{i+1,j+1} + q_{i+1,j,3}\, e_{i+1,j}$

Return: $\mathrm{DTW}_\Omega(\boldsymbol{\theta}) = v_{N_A,N_B}$
$\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta}) = (e)_{i,j=1}^{N_A,N_B}$
Intermediate computations for Algo. 6:
$\boldsymbol{Q} \triangleq (q)_{i,j,k=1}^{N_A+1,N_B+1,3}$; $\boldsymbol{E} \triangleq (e)_{i,j=1}^{N_A+1,N_B+1}$

Algorithm 6 Compute $\langle\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle$, $\nabla^2\mathrm{DTW}_\Omega(\boldsymbol{\theta})\, \boldsymbol{Z}$

Input: $\boldsymbol{\theta} \in \mathbb{R}^{N_A \times N_B}$, $\boldsymbol{Z} \in \mathbb{R}^{N_A \times N_B}$
Call Algo. 5 with input $\boldsymbol{\theta}$ to retrieve $\boldsymbol{Q}$ and $\boldsymbol{E}$
▷ Forward pass
$\dot{v}_{i,0} = \dot{v}_{0,j} = 0$, $i \in [0, \dots, N_A], j \in [N_B]$
for $i \in [1, \dots, N_B], j \in [1, \dots, N_A]$ **do**
  $\dot{v}_{i,j} = z_{i,j} + q_{i,j,1}\, \dot{v}_{i,j-1} + q_{i,j,2}\, \dot{v}_{i-1,j-1} + q_{i,j,3}\, \dot{v}_{i-1,j}$
  $\dot{\boldsymbol{q}}_{i,j} = -\boldsymbol{J}_\Omega(\boldsymbol{q}_{i,j})\, (\dot{v}_{i,j-1}, \dot{v}_{i-1,j-1}, \dot{v}_{i-1,j}) \in \mathbb{R}^3$
▷ Backward pass
$\dot{\boldsymbol{q}}_{i,N_B+1} = \dot{\boldsymbol{q}}_{N_A+1,j} = \mathbf{0}_3$, $i \in [0, \dots, N_A], j \in [N_B]$
$\dot{e}_{i,N_B+1} = \dot{e}_{N_A+1,j} = 0$, $i \in [0, \dots, N_A], j \in [N_B]$
for $j \in [N_B, \dots, 1], i \in [N_A, \dots, 1]$ **do**
  $\dot{e}_{i,j} = \dot{q}_{i,j+1,1}\, e_{i,j+1} + q_{i,j+1,1}\, \dot{e}_{i,j+1} + \dot{q}_{i+1,j+1,2}\, e_{i+1,j+1} + q_{i+1,j+1,2}\, \dot{e}_{i+1,j+1} + \dot{q}_{i+1,j,3}\, e_{i+1,j} + q_{i+1,j,3}\, \dot{e}_{i+1,j}$

Return: $\langle\nabla\mathrm{DTW}_\Omega(\boldsymbol{\theta}), \boldsymbol{Z}\rangle = \dot{v}_{N_A,N_B}$
$\nabla^2\mathrm{DTW}_\Omega(\boldsymbol{\theta})\, \boldsymbol{Z} = (\dot{e})_{i,j=1}^{N_A,N_B}$

Note that when Ω is the negative entropy, $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is known as soft-DTW (**?**). While the DP computation of $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ and of its gradient were already known, the generalization to any strongly convex Ω and the computation of $\nabla^2\mathrm{DTW}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ are new. From Proposition 2 property 1, $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is a *concave* function of the discrepancy matrix $\boldsymbol{\theta}$ for any Ω. With respect to time-series, DTW_Ω is neither convex nor concave.

C. Experimental details and further results

We finally provide details on the architecture used in experiments, with additionnals figures.

C.1. Named entity recognition (section §4.2)

Our model extracts word embedding from a 300-dimensional lookup table concatenated with a 50-dimensional character embedding. This character embedding corresponds to the concatenation of the last hidden unit of a bi-directional character LSTM, as in **?**. Character embedding size is set to 50. A word LSTM then produces sentence-aware features for each word. This LSTM is bi-directional with 100-dimensional hidden units per direction. The final features $\boldsymbol{X}$ used to build the potential tensor $\boldsymbol{\theta}$ are thus 200-dimensional. Note that, in contrast with **?**:

- The look-up table is initialized with 300-dimensional embeddings from *FastText* (**?**), trained on Wikipedia corpus.
- We do not pad letters prior to feeding the character LSTM as it is not principled.
- We do not train the unknown word embedding as we found it had no effect.

We convert tags to the IOBES (Inside-Outside-Begin-End-Stop) scheme to build a richer Vit_Ω model than if we used the simpler IOB (Inside-Outside-Begin) scheme, that has a lower number of tags. We performed a small grid-search to select the step-size and batch-size used for optimization: $s \in \{0.005, 0.01, 0.02\}$, $b \in \{8, 32, 128\}$. For each language and each loss, we select the highest-scoring model on the validation set, and report the test score.

The model is strongly subject to overfitting using the convex surrogate loss and the log likelihood. We have to use a small batch size ($b = 8$) and vanilla SGD with large step size ($s = 0.01$) to avoid this overfitting issue. For all losses, accelerated stochastic optimizers have all lower generalization performance than SGD, as also noticed in (**?**) when using the

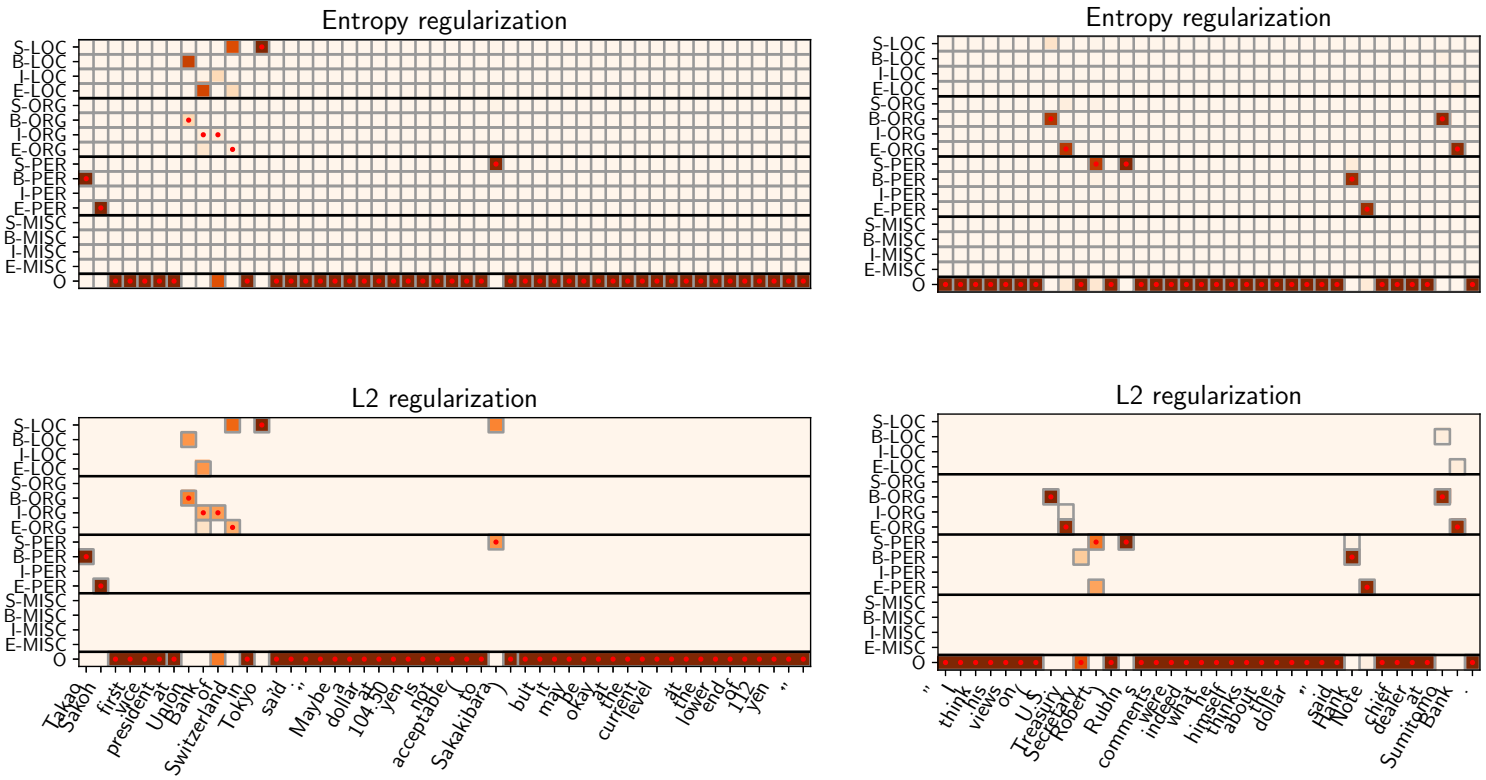

Figure 6. Test predictions from the entropy and ℓ_2^2 regularized named entity recognition (NER) models. Red dots indicate ground truth. When using ℓ_2^2 regularization, model predictions are sparse (grey borders indicates non-zero cells). They are thus easier to introspect for ambiguities, as we can list a finite number of possible outputs.

classical negative log-likelihood as a loss.

Visualization. The models using ℓ_2^2 regularization perform nearly on par with the ones using negentropy, as demonstrated in Table 1. On the other hand, ℓ_2^2 regularization leads to tag probability vectors that are sparse and hence easier to parse. They allow to detect ambiguities more easily. We display a few tagged English sequences in Figure 6. The model using ℓ_2^2 regularization correctly identifies an ambiguous entity (*Union Bank of Switzerland*) and can be used to propose two tag sequences: *(B-ORG, I-ORG, I-ORG, E-ORG)* or *(B-ORG, E-ORG, O, S-LOC)*. Probabilities of every tag sequence can be computed using the matrix $\boldsymbol{Q}$, as described in §2.3 — this remains tractable as long as the matrix $\boldsymbol{Q}$ is *sparse enough*, so that the number of non-zero probabilities sequence remains low. On the other hand, the model using negentropy regularization never assign a zero probability to any tag sequence — it is therefore not tractable to provide the user with a small set of interesting sequences.

C.2. Supervised audio-to-score transcription (section §4.3)

Audio sequences, sampled at 22.05 kHz, are split into frames of 512 samples. We extract the following features from these sequences: energy, spectral centroid, spectral bandwidth, and the 5 first MFCC features. All features are centered around the median and normalized.

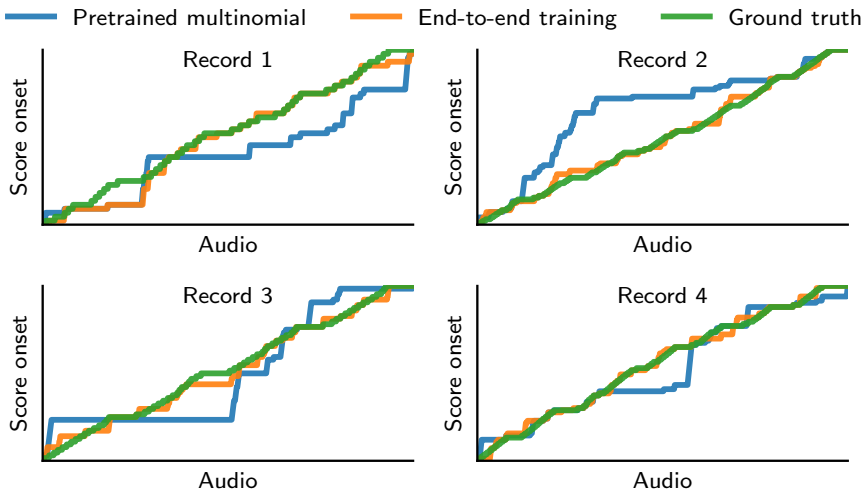

Figure 7. Alignment maps between score onsets and audio frames on test data from the Bach10 dataset. Our end-to-end trained model qualitatively performs better than the baseline model.

The ∇DTW_Ω layer is written in *Cython*[1], and hence run on CPU. This technical choice was suggested by the fact that we have to write explicit loops to specify the topological and reverse topological pass over the DTW computation graph (see Algorithm 5). However, it is possible to use only contiguous vector operations and thus take advantage of GPU computations — this is left for future work. We use *SciPy*'s[2] LBFGS-B solver to perform end-to-end training and multinomial regression. We use a ℓ_2^2 regularization on the weight $\boldsymbol{W}_{,:}$ we selected it using a grid search over $\{10^{-5}, 10^{-4}, \dots, 1\}$ and selected 10^{-3}.

Further vizualisation. In Figure 7, we display the alignment maps we obtained using our algorithm and using the baseline multinomial model followed by a hard-DTW alignment computation. These alignment maps correspond to the predicted onsets of keys. Our model (in orange) performs visibly better in predicting onsets.

C.3. Structured and sparse attention (section §5)

We use *OpenNMT-py* library[3] to fit our structured attention model. Model architecture and optimization details are as follow:

- We use a bidirectional LSTM encoder and decoder, with 500 units in each direction and a depth of 2 layers .
- The decoder is fed with the input representation as in **?**.
- SGD training with $s = 1$ learning rate, decaying from epoch 8 to epoch 15 with rate 0.65, batch size of size 256.
- Training sentence of lengths superior to 50 are ignored, and translated sentence are forced to a length inferior to 100.
- The temperature parameter is set to $\gamma = 2$ for entropy, and $\gamma = 10$ for ℓ_2^2. Performance is not affected much by this parameter, provided that it is not set too low in the ℓ_2^2 case — with a too small γ, Vit_Ω reduces to unregularized MAP estimation and ∇Vit_Ω has zero derivatives.

We use a 1-million sentence subject of WMT14 English-to-French corpus, available at

[1] http://cython.org/
[2] http://scipy.org/
[3] http://opennmt.net/

Table 3. Detokenized BLEU score on newstest2014 data using regularized and unregularized attention.

Attention model	WMT14 1M fr→en	WMT14 en→fr
Softmax	**27.96**	**28.08**
Entropy regularization	**27.96**	27.98
ℓ_2^2 reg.	27.21	27.28

http://nmt-benchmark.net/. We use Moses tokenizer and do not perform any post-processing, before computing BLEU score on detokenized sentences (*multi_bleu.perl* script).

Implementation. We implemeted a batch version of the $\nabla \mathrm{Vit}_\Omega$ layer on GPU, using the *PyTorch* tensor API. Model with negentropy-regularized attention mechanism runs $1/2$ as fast as the softmax attention mechanism (approximately 7500 tokens/s vs 15000 tokens/s on a single Nvidia Titan X Pascal). With ℓ_2^2 regularization, it is only $1/3$ as fast: approximately 5000 tokens/s. Although this remains reasonable, it could certainly be optimized by rewriting kernels using lower-level languages (*e.g.,* using *ATen* API from *PyTorch*.)

Further results. Table 3 provides BLEU scores for both translation directions on the 1 million sentence subset of WMT14 we used. We observe that the introduction of structure and sparsity does not hinder the general performance of the model. We provide several examples of attention maps in Figure 8, that illustrate the sparsity patterns ℓ_2^2 regularization uncovers.

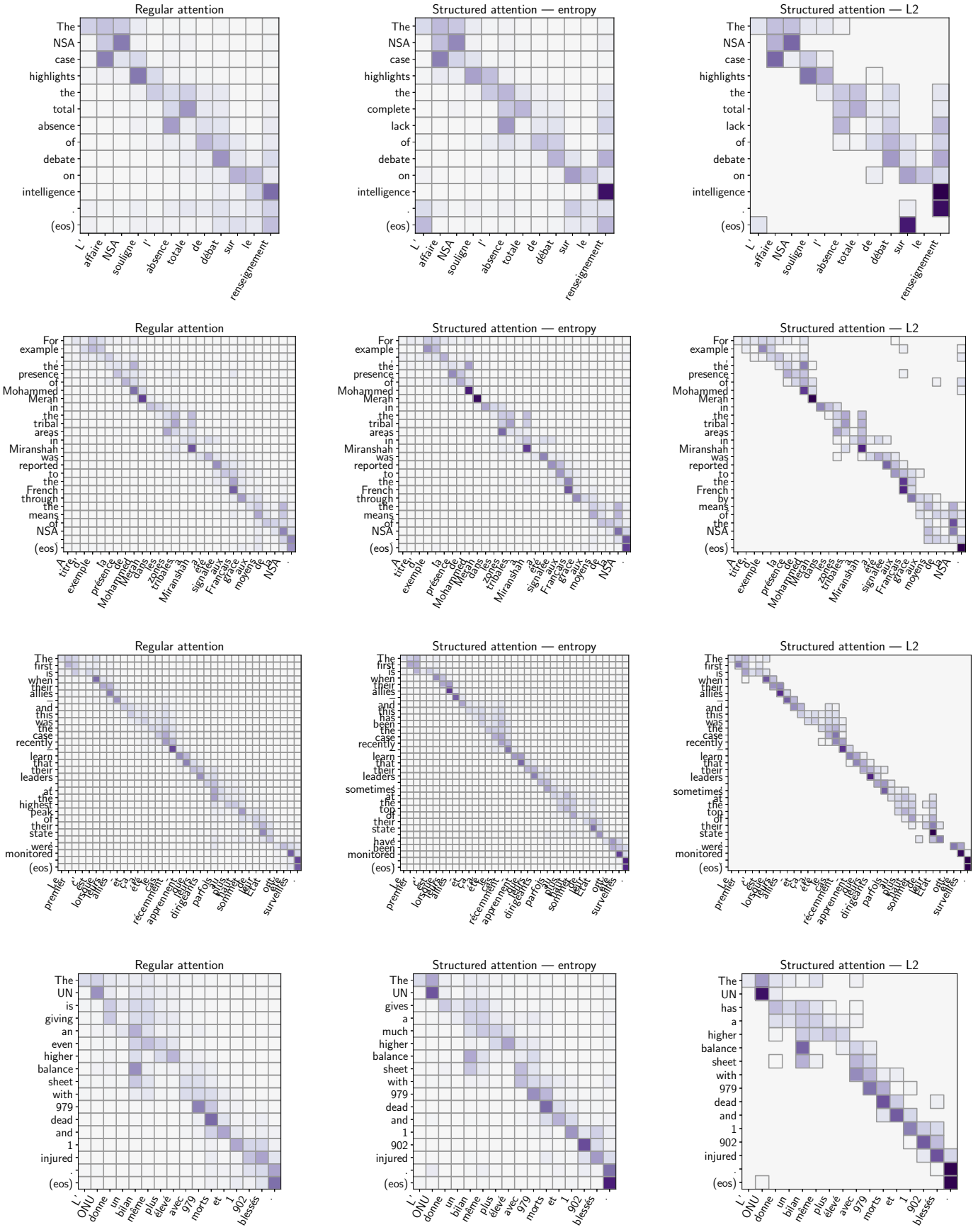

Figure 8. Attention on test samples from Newstest2014. Borders indicate non-zero cells. Translations (y-axis) are often qualitatively equivalent, while attentions maps are sparse in the ℓ_2^2 case.