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 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 (Lample et al., 2016; Cuturi & Blondel, 2017) or to create layers that impose prior structure on the output (Kim et al., 2017; Amos & Kolter, 2017; Niculae & Blondel, 2017; Djolonga & Krause, 2017).

Among these works, several involve a convex optimization problem (Amos & Kolter, 2017; Niculae & Blondel, 2017; Djolonga & Krause, 2017); others solve certain combinatorial optimization problems by dynamic programming (Kim et al., 2017; Cuturi & Blondel, 2017; Nowak et al., 2018). However, because dynamic programs (Bellman, 1952) are usually non-differentiable, virtually all these works resort to the formalism of conditional random fields (CRFs) (Lafferty et al., 2001), 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 (Verdu & Poor, 1987). 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 (Moreau, 1965; Nesterov, 2005) and backpropagation (Linnainmaa, 1970). 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 instanti-

[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.

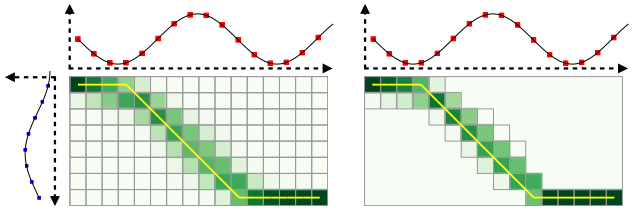

Figure 1. $\mathrm{DTW}_\Omega(\boldsymbol{\theta})$ is an instantiation of the proposed smoothed dynamic programming operator, $\mathrm{DP}_\Omega(\boldsymbol{\theta})$, to the dynamic time warping (DTW) computational graph. In this picture, $\boldsymbol{\theta}$ is the squared Euclidean distance matrix 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.

ations on structured prediction tasks (§4) and on structured attention for neural machine translation (§5).

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 (Nesterov, 2005; Beck & Teboulle, 2012; Niculae & Blondel, 2017), 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 (1966), 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 Martins & Astudillo (2016); Niculae & Blondel (2017), 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 (Bellman, 1952). 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 (Bertsekas, 1971),

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

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.,* Wainwright & Jordan, 2008, 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 (Verdu & Poor, 1987; Aji & McEliece, 2000). 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 (Horibe, 1988). 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 $\nabla\mathrm{DP}_\Omega(\boldsymbol{\theta})$. 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 $\nabla^2\mathrm{DP}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$

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* (Paszke et al., 2017), 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 (1994), 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 (Meshi et al., 2015) or optimal transport (Blondel et al., 2018) 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 (Niculae et al., 2018).

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*

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

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 (Viterbi, 1967; Rabiner, 1990), one of the most famous instances of DP algorithm. We call the resulting operator Vit_Ω. We wish to tag a sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ of vectors in $\mathbb{R}^D$ (*e.g.,* word representations) with the most probable output sequence (*e.g.,* entity tags) $\boldsymbol{y} = (y_1, \ldots, y_T) \in [S]^T$. 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 (Sutton et al., 2012, §2.5). In recent works and in this paper, they are learned end-to-end (Bottou et al., 1997; Collobert et al., 2011; Lample et al., 2016).

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.

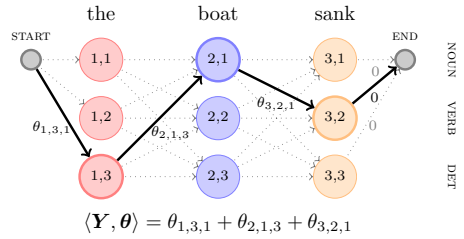

Figure 2. Computational graph of the Viterbi algorithm.

When $\Omega = -H$, we recover linear-chain conditional random fields (CRFs) (Lafferty et al., 2001) 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 (Baum & Petrie, 1966). 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.,* Eisner (2016)). 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 (Pearl, 1988) yields a dynamic program. We note that continuous beam search (Goyal et al., 2017) 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) (Sakoe & Chiba, 1978), 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.

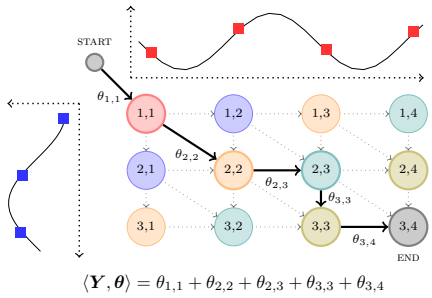

Figure 3. Computational graph of the DTW algorithm.

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 $\text{DTW}_\Omega(\boldsymbol{\theta})$ can be used to define a loss between alignments or between time-series. Following Proposition 3, $\nabla\text{DTW}_\Omega(\boldsymbol{\theta}) = \boldsymbol{E} \in \mathbb{R}^{N_A \times N_B}$ 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 $\text{DTW}_\Omega(\boldsymbol{\theta})$ as well as its gradient and its Hessian products are provided in §B.3. When $\Omega = -H$, $\text{DTW}_\Omega(\boldsymbol{\theta})$ 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 (Cuturi & Blondel, 2017). However, the case $\Omega = \|\cdot\|^2$ and the computation of $\nabla^2\text{DTW}_\Omega(\boldsymbol{\theta})\boldsymbol{Z}$ are new and allow new applications.

4. Differentiable structured prediction

We now apply the proposed layers, $\text{DP}_\Omega(\boldsymbol{\theta})$ and $\nabla\text{DP}_\Omega(\boldsymbol{\theta})$, to structured prediction (Bakır et al., 2007), 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 (Tsochantaridis et al., 2005) 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

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

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

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

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 $\text{DP}_\Omega(\boldsymbol{\theta})$ at training time. It is easy to check that we recover the structured perceptron loss with $\ell_{0,0}$ (Collins, 2002), the structured hinge loss with $\ell_{C,0}$ (Tsochantaridis et al., 2005) and the CRF loss with $\ell_{0,-H}$ (Lafferty et al., 2001). The last one has been used on top of LSTMs in several recent works (Lample et al., 2016; Ma & Hovy, 2016). 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 Stoyanov &

Eisner (2012), this can be achieved by backpropagating through the minimum risk decoder. However, the risk is usually non-differentiable, piecewise constant (Smith & Eisner, 2006) and several smoothing heuristics are necessary to make the method work (Stoyanov & Eisner, 2012).

Another principled approach is to consider a differentiable approximation $\Delta\colon \mathcal{Y} \times \operatorname{conv}(\mathcal{Y}) \to \mathbb{R}_+$ of the cost C. We can then relax $C(\boldsymbol{Y}_{\text{true}}, \boldsymbol{Y}^\star(\boldsymbol{\theta}))$ by $\Delta(\boldsymbol{Y}_{\text{true}}, \nabla \text{DP}_\Omega(\boldsymbol{\theta}))$. Unlike minimum risk training, this approach is differentiable everywhere when $\Omega = -H$. Both approaches require to backpropagate through $\nabla \text{DP}_\Omega(\boldsymbol{\theta})$, which is twice as costly as backpropagating through $\text{DP}_\Omega(\boldsymbol{\theta})$ (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:

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

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 Lample et al. (2016), who use the $\ell_{0,-H}$ loss, we use a character LSTM and FastText (Joulin et al., 2016) 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 $\text{DP}(\boldsymbol{\theta})$, 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.

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
(Lample et al., 2016)		*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

4.3. Supervised audio-to-score transcription

We use our framework to perform supervised audio-to-score alignment on the Bach 10 dataset (Duan & Pardo, 2011). 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:

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

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 $\boldsymbol{Y} = \nabla \text{DTW}_{-H}(\boldsymbol{\theta})$. Following (Garreau et al., 2014), 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 $\boldsymbol{Y} \in \operatorname{conv}(\mathcal{Y})$, 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 $\boldsymbol{Y}^\star(\boldsymbol{\theta}) \in \mathcal{Y}$.

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 (Bahdanau et al., 2015). 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. Kim et al. (2017) 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 Kim et al. (2017), 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 Kim et al. (2017).

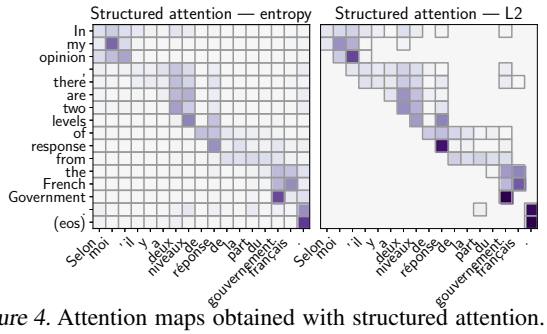

Figure 4. Attention maps obtained with structured attention. Although both regularizations led to the same translation (y-axis) in this example, attention is sparse and more interpretable with ℓ_2^2.

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 (Martins & Astudillo, 2016). 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.

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.

References

Aji, S. M. and McEliece, R. J. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2): 325–343, 2000.

Amos, B. and Kolter, J. Z. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*, pp. 136–145, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proc. of ICLR*, 2015.

Bakır, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. V. N. *Predicting Structured Data*. The MIT Press, 2007.

Banderier, C. and Schwer, S. Why Delannoy numbers? *Journal of Statistical Planning and Inference*, 135(1): 40–54, 2005.

Baum, L. E. and Petrie, T. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

Beck, A. and Teboulle, M. Smoothing and first order methods: A unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

Bellman, R. On the theory of dynamic programming. *Proc. of the National Academy of Sciences*, 38(8):716–719, 1952.

Bertsekas, D. P. *Control of uncertain systems with a set-membership description of the uncertainty.* PhD thesis, Massachusetts Institute of Technology, 1971.

Blondel, M., Seguy, V., and Rolet, A. Smooth and sparse optimal transport. In *Proc. of AISTATS*, 2018.

Bottou, L., Bengio, Y., and Cun, Y. L. Global training of document processing systems using graph transformer networks. In *Proc. of CVPR*, pp. 489–494, 1997.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Collins, M. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of ACL*, pp. 1–8, 2002.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

Cuturi, M. and Blondel, M. Soft-DTW: a differentiable loss function for time-series. In *Proc. of ICML*, pp. 894–903, 2017.

Danskin, J. M. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

Djolonga, J. and Krause, A. Differentiable learning of submodular functions. In *Proc. of NIPS*, pp. 1014–1024, 2017.

Duan, Z. and Pardo, B. Bach 10 dataset. http://music.cs.northwestern.edu/data/Bach10.html.

Duan, Z. and Pardo, B. Soundprism: An online system for score-informed source separation of music audio. *IEEE Journal of Selected Topics in Signal Processing*, 5(6): 1205–1215, 2011.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proc. of ICML*, pp. 272–279, 2008.

Eisner, J. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proc. of the Workshop on Structured Prediction for NLP*, pp. 1–17, 2016.

Garreau, D., Lajugie, R., Arlot, S., and Bach, F. Metric learning for temporal sequence alignment. In *Proc. of NIPS*, pp. 1817–1825, 2014.

Goyal, K., Neubig, G., Dyer, C., and Berg-Kirkpatrick, T. A continuous relaxation of beam search for end-to-end training of neural sequence models. *arXiv:1708.00111 [cs]*, 2017.

Gselmann, E. Entropy functions and functional equations. *Mathematical Communications*, (16):347–357, 2011.

Horibe, Y. Entropy of terminal distributions and the Fibonnacci trees. *The Fibonacci Quarterly*, (26):135–140, 1988.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

Kim, Y., Denton, C., Hoang, L., and Rush, A. M. Structured attention networks. In *Proc. of ICLR*, 2017.

Lafferty, J., McCallum, A., and Pereira, F. C. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pp. 282–289, 2001.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. Neural architectures for named entity recognition. In *Proc. of NAACL*, pp. 260–270, 2016.

Linnainmaa, S. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Univ. Helsinki, 1970.

Luong, T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*, pp. 1412–1421, 2015.

Ma, X. and Hovy, E. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proc. of ACL*, pp. 1064–1074, 2016.

Martins, A. F. and Astudillo, R. F. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*, pp. 1614–1623, 2016.

Meshi, O., Mahdavi, M., and Schwing, A. G. Smooth and strong: MAP inference with linear convergence. In *Proc. of NIPS*, 2015.

Michelot, C. A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 50(1):195–200, 1986.

Moreau, J.-J. Proximité et dualité dans un espace hilbertien. *Bullet de la Société Mathémathique de France*, 93 (2):273–299, 1965.

Nesterov, Y. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

Niculae, V. and Blondel, M. A regularized framework for sparse and structured neural attention. In *Proc. of NIPS*, pp. 3340–3350, 2017.

Niculae, V., Martins, A. F., Blondel, M., and Cardie, C. Sparsemap: Differentiable sparse structured inference. In *Proc. of ICML*, 2018.

Nowak, A., Folqué, D., and Bruna, J. Divide and conquer networks. *Proc. of ICLR*, 2018.

Paszke, A., Gross, S., Chintala, S., and Chanan, G. Pytorch: Tensors and dynamic neural networks in Python with strong GPU acceleration, 2017.

Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier, 1988.

Pearlmutter, B. A. Fast exact multiplication by the Hessian. *Neural computation*, 6(1):147–160, 1994.

Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, pp. 267–296. 1990.

Sakoe, H. and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26: 43–49, 1978.

Smith, D. A. and Eisner, J. Minimum risk annealing for training log-linear models. In *Proc. of COLING/ACL*, pp. 787–794, 2006.

Stoyanov, V. and Eisner, J. Minimum-risk training of approximate CRF-based NLP systems. In *Proc. of NAACL*, pp. 120–130, 2012.

Sulanke, R. A. Objects counted by the central Delannoy numbers. *Journal of Integer Sequences*, 6(1):3, 2003.

Sutton, C., McCallum, A., et al. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Verdu, S. and Poor, H. V. Abstract dynamic programming models under commutativity conditions. *SIAM Journal on Control and Optimization*, 25(4):990–1006, 1987.

Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008.