A Regularized Framework for Sparse and Structured Neural Attention

Vlad Niculae[*]
Cornell University
Ithaca, NY
vlad@cs.cornell.edu

Mathieu Blondel
NTT Communication Science Laboratories
Kyoto, Japan
mathieu@mblondel.org

Abstract

Modern neural networks are often augmented with an attention mechanism, which tells the network where to focus within the input. We propose in this paper a new framework for sparse and structured attention, building upon a smoothed max operator. We show that the gradient of this operator defines a mapping from real values to probabilities, suitable as an attention mechanism. Our framework includes softmax and a slight generalization of the recently-proposed sparsemax as special cases. However, we also show how our framework can incorporate modern structured penalties, resulting in more interpretable attention mechanisms, that focus on entire segments or groups of an input. We derive efficient algorithms to compute the forward and backward passes of our attention mechanisms, enabling their use in a neural network trained with backpropagation. To showcase their potential as a drop-in replacement for existing ones, we evaluate our attention mechanisms on three large-scale tasks: textual entailment, machine translation, and sentence summarization. Our attention mechanisms improve interpretability without sacrificing performance; notably, on textual entailment and summarization, we outperform the standard attention mechanisms based on softmax and sparsemax.

1 Introduction

Modern neural network architectures are commonly augmented with an attention mechanism, which tells the network where to look within the input in order to make the next prediction. Attention-augmented architectures have been successfully applied to machine translation [2, 29], speech recognition [10], image caption generation [44], textual entailment [38, 31], and sentence summarization [39], to name but a few examples. At the heart of attention mechanisms is a mapping function that converts real values to probabilities, encoding the relative importance of elements in the input. For the case of sequence-to-sequence prediction, at each time step of generating the output sequence, attention probabilities are produced, conditioned on the current state of a decoder network. They are then used to aggregate an input representation (a variable-length list of vectors) into a single vector, which is relevant for the current time step. That vector is finally fed into the decoder network to produce the next element in the output sequence. This process is repeated until the end-of-sequence symbol is generated. Importantly, such architectures can be trained end-to-end using backpropagation.

Alongside empirical successes, neural attention—while not necessarily correlated with human attention—is increasingly crucial in bringing more **interpretability** to neural networks by helping explain how individual input elements contribute to the model's decisions. However, the most commonly used attention mechanism, *softmax*, yields dense attention weights: all elements in the input always make at least a small contribution to the decision. To overcome this limitation, *sparsemax* was recently proposed [31], using the Euclidean projection onto the simplex as a sparse alternative to

[*]Work performed during an internship at NTT Commmunication Science Laboratories, Kyoto, Japan.

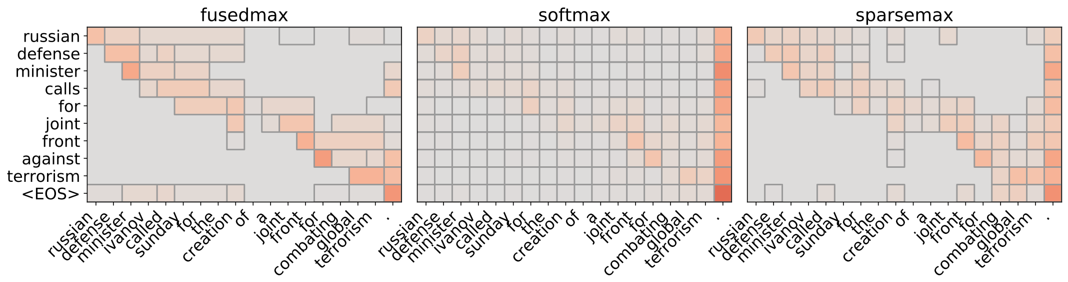

Figure 1: Attention weights produced by the proposed *fusedmax*, compared to *softmax* and *sparsemax*, on sentence summarization. The input sentence to be summarized (taken from [39]) is along the x-axis. From top to bottom, each row shows where the attention is distributed when producing each word in the summary. All rows sum to 1, the grey background corresponds to exactly 0 (never achieved by softmax), and adjacent positions with exactly equal weight are not separated by borders. Fusedmax pays attention to contiguous segments of text with equal weight; such segments never occur with softmax and sparsemax. In addition to enhancing interpretability, we show in §4.3 that fusedmax outperforms both softmax and sparsemax on this task in terms of ROUGE scores.

softmax. Compared to softmax, sparsemax outputs more interpretable attention weights, as illustrated in [31] on the task of textual entailment. The principle of parsimony, which states that simple explanations should be preferred over complex ones, is not, however, limited to sparsity: it remains open whether new attention mechanisms can be designed to benefit from more structural prior knowledge.

Our contributions. The success of sparsemax motivates us to explore new attention mechanisms that can both output sparse weights and take advantage of structural properties of the input through the use of modern sparsity-inducing penalties. To do so, we make the following contributions:

1) We propose a new **general framework** that builds upon a $\max$ operator, regularized with a strongly convex function. We show that this operator is differentiable, and that its gradient defines a mapping from real values to probabilities, suitable as an attention mechanism. Our framework **includes as special cases** both **softmax** and a slight generalization of **sparsemax**. (§2)

2) We show how to incorporate the fused lasso [42] in this framework, to derive a new attention mechanism, named *fusedmax*, which encourages the network to pay attention to **contiguous segments of text** when making a decision. This idea is illustrated in Figure 1 on sentence summarization. For cases when the contiguity assumption is too strict, we show how to incorporate an OSCAR penalty [7] to derive a new attention mechanism, named *oscarmax*, that encourages the network to pay equal attention to **possibly non-contiguous groups of words**. (§3)

3) In order to use attention mechanisms defined under our framework in an autodiff toolkit, two problems must be addressed: evaluating the attention itself and computing its Jacobian. However, our attention mechanisms require solving a convex optimization problem and do not generally enjoy a simple analytical expression, unlike softmax. Computing the Jacobian of the solution of an optimization problem is called argmin/argmax differentiation and is currently an area of active research (cf. [1] and references therein). One of **our key algorithmic contributions is to show how to compute this Jacobian** under our general framework, as well as for fused lasso and OSCAR. (§3)

4) To showcase the potential of our new attention mechanisms as a **drop-in replacement** for existing ones, we show empirically that our new attention mechanisms enhance interpretability while achieving comparable or better accuracy on three diverse and challenging tasks: **textual entailment, machine translation, and sentence summarization**. (§4)

Notation. We denote the set $\{1, \ldots, d\}$ by $[d]$. We denote the $(d-1)$-dimensional probability simplex by $\Delta^d := \{\boldsymbol{x} \in \mathbb{R}^d \colon \|\boldsymbol{x}\|_1 = 1, \boldsymbol{x} \geq 0\}$ and the Euclidean projection onto it by $P_{\Delta^d}(\boldsymbol{x}) := \arg\min_{\boldsymbol{y} \in \Delta^d} \|\boldsymbol{y} - \boldsymbol{x}\|^2$. Given a function $f \colon \mathbb{R}^d \to \mathbb{R} \cup \{\infty\}$, its convex conjugate is defined by $f^*(\boldsymbol{x}) := \sup_{\boldsymbol{y} \in \operatorname{dom} f} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} - f(\boldsymbol{y})$. Given a norm $\|\cdot\|$, its dual is defined by $\|\boldsymbol{x}\|_* := \sup_{\|\boldsymbol{y}\| \leq 1} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}$. We denote the subdifferential of a function f at $\boldsymbol{y}$ by $\partial f(\boldsymbol{y})$. Elements of the subdifferential are called subgradients and when f is differentiable, $\partial f(\boldsymbol{y})$ contains a single element, the gradient of f at $\boldsymbol{y}$, denoted by $\nabla f(\boldsymbol{y})$. We denote the Jacobian of a function $g \colon \mathbb{R}^d \to \mathbb{R}^d$ at $\boldsymbol{y}$ by $J_g(\boldsymbol{y}) \in \mathbb{R}^{d \times d}$ and the Hessian of a function $f \colon \mathbb{R}^d \to \mathbb{R}$ at $\boldsymbol{y}$ by $H_f(\boldsymbol{y}) \in \mathbb{R}^{d \times d}$.

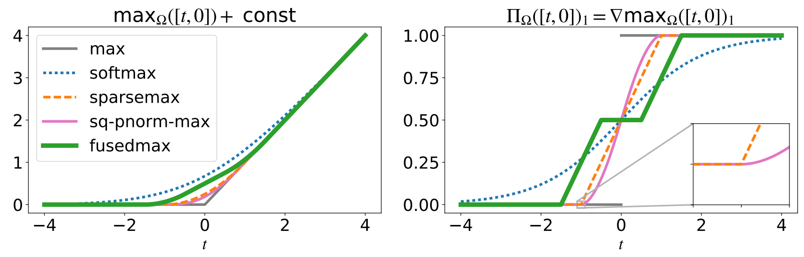

Figure 2: The proposed $\max_\Omega(\boldsymbol{x})$ operator up to a constant (left) and the proposed $\Pi_\Omega(\boldsymbol{x})$ mapping (right), illustrated with $\boldsymbol{x} = [t, 0]$ and $\gamma = 1$. In this case, $\max_\Omega(\boldsymbol{x})$ is a ReLu-like function and $\Pi_\Omega(\boldsymbol{x})$ is a sigmoid-like function. Our framework recovers *softmax* (negative entropy) and *sparsemax* (squared 2-norm) as special cases. We also introduce three new attention mechanisms: *sq-pnorm-max* (squared p-norm, here illustrated with $p = 1.5$), *fusedmax* (squared 2-norm + fused lasso), and *oscarmax* (squared 2-norm + OSCAR; not pictured since it is equivalent to fusedmax in 2-d). Except for softmax, which never exactly reaches 0, all mappings shown on the right encourage sparse outputs.

2 Proposed regularized attention framework

2.1 The max operator and its subgradient mapping

To motivate our proposal, we first show in this section that the subgradients of the maximum operator define a mapping from $\mathbb{R}^d$ to Δ^d, but that this mapping is highly unsuitable as an attention mechanism. The maximum operator is a function from $\mathbb{R}^d$ to $\mathbb{R}$ and can be defined by

$$\max(\boldsymbol{x}) := \max_{i\in[d]} x_i = \sup_{\boldsymbol{y}\in\Delta^d} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}.$$

The equality on the r.h.s comes from the fact that the supremum of a linear form over the simplex is always achieved at one of the vertices, i.e., one of the standard basis vectors $\{\boldsymbol{e}_i\}_{i=1}^d$. Moreover, it is not hard to check that any solution $\boldsymbol{y}^\star$ of that supremum is precisely a subgradient of $\max(\boldsymbol{x})$: $\partial \max(\boldsymbol{x}) = \{\boldsymbol{e}_{i^\star} : i^\star \in \arg\max_{i\in[d]} x_i\}$. We can see these subgradients as a mapping $\Pi\colon \mathbb{R}^d \to \Delta^d$ that **puts all the probability mass onto a single element**: $\Pi(\boldsymbol{x}) = \boldsymbol{e}_i$ for any $\boldsymbol{e}_i \in \partial \max(\boldsymbol{x})$. However, this behavior is undesirable, as the resulting mapping is a discontinuous function (a Heaviside step function when $\boldsymbol{x} = [t, 0]$), which is not amenable to optimization by gradient descent.

2.2 A regularized max operator and its gradient mapping

These shortcomings encourage us to consider a regularization of the maximum operator. Inspired by the seminal work of Nesterov [35], we apply a smoothing technique. The conjugate of $\max(\boldsymbol{x})$ is

$$\max{}^*(\boldsymbol{y}) = \begin{cases} 0, & \text{if } \boldsymbol{y}\in\Delta^d \\ \infty, & \text{o.w.} \end{cases}.$$

For a proof, see for instance [33, Appendix B]. We now add regularization to the conjugate

$$\max{}^*_\Omega(\boldsymbol{y}) := \begin{cases} \gamma\Omega(\boldsymbol{y}), & \text{if } \boldsymbol{y}\in\Delta^d \\ \infty, & \text{o.w.} \end{cases},$$

where we assume that $\Omega\colon \mathbb{R}^d \to \mathbb{R}$ is β**-strongly convex** w.r.t. some norm $\|\cdot\|$ and $\gamma > 0$ controls the regularization strength. To define a smoothed max operator, we take the conjugate once again

$$\max{}_\Omega(\boldsymbol{x}) = \max{}^{**}_\Omega(\boldsymbol{x}) = \sup_{\boldsymbol{y}\in\mathbb{R}^d} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} - \max{}^*_\Omega(\boldsymbol{y}) = \sup_{\boldsymbol{y}\in\Delta^d} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} - \gamma\Omega(\boldsymbol{y}). \tag{1}$$

Our main proposal is a mapping $\Pi_\Omega\colon \mathbb{R}^d \to \Delta^d$, defined as the *argument* that achieves this supremum.

$$\boxed{\Pi_\Omega(\boldsymbol{x}) := \arg\max_{\boldsymbol{y}\in\Delta^d} \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} - \gamma\Omega(\boldsymbol{y}) = \nabla \max{}_\Omega(\boldsymbol{x})}$$

The r.h.s. holds by combining that i) $\max_\Omega(\boldsymbol{x}) = (\boldsymbol{y}^\star)^{\mathrm{T}}\boldsymbol{x} - \max^*_\Omega(\boldsymbol{y}^\star) \Leftrightarrow \boldsymbol{y}^\star \in \partial\max_\Omega(\boldsymbol{x})$ and ii) $\partial\max_\Omega(\boldsymbol{x}) = \{\nabla\max_\Omega(\boldsymbol{x})\}$, since (1) has a unique solution. Therefore, Π_Ω is a **gradient mapping**. We illustrate $\max_\Omega$ and Π_Ω for various choices of Ω in Figure 2 (2-d) and in Appendix C.1 (3-d).

Importance of strong convexity. Our β-strong convexity assumption on Ω plays a crucial role and should not be underestimated. Recall that a function $f\colon \mathbb{R}^d \to \mathbb{R}$ is β-strongly convex w.r.t. a norm $\|\cdot\|$ if and only if its conjugate f^* is $\frac{1}{\beta}$-smooth w.r.t. the dual norm $\|\cdot\|_*$ [46, Corollary 3.5.11] [22, Theorem 3]. This is sufficient to ensure that $\max_\Omega$ is $\frac{1}{\gamma\beta}$**-smooth**, or, in other words, that it is differentiable everywhere and its gradient, Π_Ω, is $\frac{1}{\gamma\beta}$-Lipschitz continuous w.r.t. $\|\cdot\|_*$.

Training by backpropagation. In order to use Π_Ω in a neural network trained by backpropagation, two problems must be addressed for any regularizer Ω. The first is the **forward** computation: how to evaluate $\Pi_\Omega(\boldsymbol{x})$, i.e., how to solve the optimization problem in (1). The second is the **backward** computation: how to evaluate the Jacobian of $\Pi_\Omega(\boldsymbol{x})$, or, equivalently, the Hessian of $\max_\Omega(\boldsymbol{x})$. One of our key contributions, presented in §3, is to show how to solve these two problems for general differentiable Ω, as well as for two structured regularizers: fused lasso and OSCAR.

2.3 Recovering softmax and sparsemax as special cases

Before deriving new attention mechanisms using our framework, we now show how we can recover softmax and sparsemax, using a specific regularizer Ω.

Softmax. We choose $\Omega(\boldsymbol{y}) = \sum_{i=1}^d y_i \log y_i$, the negative entropy. The conjugate of the negative entropy restricted to the simplex is the $\log\operatorname{sum}\exp$ [9, Example 3.25]. Moreover, if $f(\boldsymbol{x}) = \gamma g(\boldsymbol{x})$ for $\gamma > 0$, then $f^*(\boldsymbol{y}) = \gamma g^*(\boldsymbol{y}/\gamma)$. We therefore get a closed-form expression: $\max_\Omega(\boldsymbol{x}) = \gamma\ \log\operatorname{sum}\exp(\boldsymbol{x}/\gamma) := \gamma \log \sum_{i=1}^d e^{x_i/\gamma}$. Since the negative entropy is 1-strongly convex w.r.t. $\|\cdot\|_1$ over Δ^d, we get that $\max_\Omega$ is $\frac{1}{\gamma}$-smooth w.r.t. $\|\cdot\|_\infty$. We obtain the classical softmax, with temperature parameter γ, by taking the gradient of $\max_\Omega(\boldsymbol{x})$,

$$\Pi_\Omega(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}/\gamma}}{\sum_{i=1}^d e^{x_i/\gamma}}, \tag{softmax}$$

where $e^{\boldsymbol{x}/\gamma}$ is evaluated element-wise. Note that some authors also call $\max_\Omega$ a "soft max." Although Π_Ω is really a soft *arg max*, we opt to follow the more popular terminology. When $\boldsymbol{x} = [t, 0]$, it can be checked that $\max_\Omega(\boldsymbol{x})$ reduces to the softplus [16] and $\Pi_\Omega(\boldsymbol{x})_1$ to a sigmoid.

Sparsemax. We choose $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2$, also known as Moreau-Yosida regularization in proximal operator theory [35, 36]. Since $\frac{1}{2}\|\boldsymbol{y}\|_2^2$ is 1-strongly convex w.r.t. $\|\cdot\|_2$, we get that $\max_\Omega$ is $\frac{1}{\gamma}$-smooth w.r.t. $\|\cdot\|_2$. In addition, it is easy to verify that

$$\Pi_\Omega(\boldsymbol{x}) = P_{\Delta^d}(\boldsymbol{x}/\gamma) = \operatorname*{arg\,min}_{\boldsymbol{y}\in\Delta^d} \|\boldsymbol{y} - \boldsymbol{x}/\gamma\|^2. \tag{sparsemax}$$

This mapping was introduced *as is* in [31] with $\gamma = 1$ and was named sparsemax, due to the fact that it is a sparse alternative to softmax. Our derivation thus gives us a slight generalization, where γ controls the sparsity (the smaller, the sparser) and could be tuned; in our experiments, however, we follow the literature and set $\gamma = 1$. The Euclidean projection onto the simplex, P_{Δ^d}, can be computed exactly [34, 15] (we discuss the complexity in Appendix B). Following [31], the Jacobian of Π_Ω is

$$J_{\Pi_\Omega}(\boldsymbol{x}) = \frac{1}{\gamma} J_{P_{\Delta^d}}(\boldsymbol{x}/\gamma) = \frac{1}{\gamma}\left(\operatorname{diag}(\boldsymbol{s}) - \boldsymbol{s}\boldsymbol{s}^{\mathrm{T}}/\|\boldsymbol{s}\|_1\right),$$

where $\boldsymbol{s} \in \{0,1\}^d$ indicates the nonzero elements of $\Pi_\Omega(\boldsymbol{x})$. Since Π_Ω is Lipschitz continuous, Rademacher's theorem implies that Π_Ω is differentiable almost everywhere. For points where Π_Ω is not differentiable (where $\max_\Omega$ is not twice differentiable), we can take an arbitrary matrix in the set of Clarke's generalized Jacobians [11], the convex hull of Jacobians of the form $\lim_{\boldsymbol{x}_t \to \boldsymbol{x}} J_{\Pi_\Omega}(\boldsymbol{x}_t)$ [31].

3 Deriving new sparse and structured attention mechanisms

3.1 Differentiable regularizer Ω

Before tackling more structured regularizers, we address in this section the case of general differentiable regularizer Ω. Because $\Pi_\Omega(\boldsymbol{x})$ involves maximizing (1), a concave function over the simplex, it can be computed globally using any off-the-shelf projected gradient solver. Therefore, the main challenge is how to compute the Jacobian of Π_Ω. This is what we address in the next proposition.

Proposition 1 *Jacobian of Π_Ω for any differentiable Ω (backward computation)*

Assume that Ω is differentiable over Δ^d and that $\Pi_\Omega(\boldsymbol{x}) = \arg\max_{\boldsymbol{y}\in\Delta^d} \boldsymbol{y}^\mathrm{T}\boldsymbol{x} - \gamma\Omega(\boldsymbol{y}) = \boldsymbol{y}^\star$ has been computed. Then the Jacobian of Π_Ω at $\boldsymbol{x}$, denoted J_{Π_Ω}, can be obtained by solving the system

$$(I + A(B - I))\, J_{\Pi_\Omega} = A,$$

where we defined the shorthands $A := J_{P_{\Delta^d}}(\boldsymbol{y}^\star - \gamma\nabla\Omega(\boldsymbol{y}^\star) + \boldsymbol{x})$ and $B := \gamma H_\Omega(\boldsymbol{y}^\star)$.

The proof is given in Appendix A.1. Unlike recent work tackling argmin differentiation through matrix differential calculus on the Karush–Kuhn–Tucker (KKT) conditions [1], our proof technique relies on differentiating the fixed point iteration $\boldsymbol{y}^* = P_{\Delta^d}(\boldsymbol{y}^\star - \nabla f(\boldsymbol{y}^\star))$. To compute $J_{\Pi_\Omega}\boldsymbol{v}$ for an arbitrary $\boldsymbol{v} \in \mathbb{R}^d$, as required by backpropagation, we may directly solve $(I + A(B - I))\,(J_{\Pi_\Omega}\boldsymbol{v}) = A\boldsymbol{v}$. We show in Appendix B how this system can be solved efficiently thanks to the structure of A.

Squared p-norms. As a useful example of a differentiable function over the simplex, we consider squared p-norms: $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_p^2 = \left(\sum_{i=1}^d y_i^p\right)^{2/p}$, where $\boldsymbol{y} \in \Delta^d$ and $p \in (1, 2]$. For this choice of p, it is known that the squared p-norm is strongly convex w.r.t. $\|\cdot\|_p$ [3]. This implies that $\max_\Omega$ is $\frac{1}{\gamma(p-1)}$ smooth w.r.t. $\|.\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$. We call the induced mapping function *sq-pnorm-max*:

$$\Pi_\Omega(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\Delta^d} \frac{\gamma}{2}\|\boldsymbol{y}\|_p^2 - \boldsymbol{y}^\mathrm{T}\boldsymbol{x}. \qquad \text{(sq-pnorm-max)}$$

The gradient and Hessian needed for Proposition 1 can be computed by $\nabla\Omega(\boldsymbol{y}) = \frac{\boldsymbol{y}^{p-1}}{\|\boldsymbol{y}\|_p^{p-2}}$ and

$$H_\Omega(\boldsymbol{y}) = \mathrm{diag}(\boldsymbol{d}) + \boldsymbol{u}\boldsymbol{u}^\mathrm{T}, \quad \text{where} \quad \boldsymbol{d} = \frac{(p-1)}{\|\boldsymbol{y}\|_p^{p-2}}\,\boldsymbol{y}^{p-2} \quad \text{and} \quad \boldsymbol{u} = \sqrt{\frac{(2-p)}{\|\boldsymbol{y}\|_p^{2p-2}}}\,\boldsymbol{y}^{p-1},$$

with the exponentiation performed element-wise. sq-pnorm-max recovers sparsemax with $p = 2$ and, like sparsemax, encourages sparse outputs. However, as can be seen in the zoomed box in Figure 2 (right), the transition between $\boldsymbol{y}^\star = [0, 1]$ and $\boldsymbol{y}^\star = [1, 0]$ can be smoother when $1 < p < 2$. Throughout our experiments, we use $p = 1.5$.

3.2 Structured regularizers: fused lasso and OSCAR

Fusedmax. For cases when the input is sequential and the order is meaningful, as is the case for many natural languages, we propose *fusedmax*, an attention mechanism based on *fused lasso* [42], also known as 1-d total variation (TV). Fusedmax encourages paying attention to **contiguous segments**, with equal weights within each one. It is expressed under our framework by choosing $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2 + \lambda\sum_{i=1}^{d-1}|y_{i+1} - y_i|$, i.e., the sum of a strongly convex term and of a 1-d TV penalty. It is easy to verify that this choice yields the mapping

$$\Pi_\Omega(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\Delta^d} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}/\gamma\|^2 + \lambda\sum_{i=1}^{d-1}|y_{i+1} - y_i|. \qquad \text{(fusedmax)}$$

Oscarmax. For situations where the contiguity assumption may be too strict, we propose *oscarmax*, based on the OSCAR penalty [7], to encourage attention weights to **merge into clusters with the same value**, regardless of position in the sequence. This is accomplished by replacing the 1-d TV penalty in fusedmax with an ∞-norm penalty on each pair of attention weights, i.e., $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2 + \lambda\sum_{i<j}\max(|y_i|, |y_j|)$. This results in the mapping

$$\Pi_\Omega(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\Delta^d} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}/\gamma\|^2 + \lambda\sum_{i<j}\max(|y_i|, |y_j|). \qquad \text{(oscarmax)}$$

Forward computation. Due to the $\boldsymbol{y} \in \Delta^d$ constraint, computing fusedmax/oscarmax does not seem trivial on first sight. The next proposition shows how to do so, without any iterative method.

Proposition 2 *Computing fusedmax and oscarmax (forward computation)*

$$\textit{fusedmax: } \Pi_\Omega(\boldsymbol{x}) = P_{\Delta^d}\left(P_{TV}(\boldsymbol{x}/\gamma)\right), \quad P_{TV}(\boldsymbol{x}) := \arg\min_{\boldsymbol{y}\in\mathbb{R}^d} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2 + \lambda\sum_{i=1}^{d-1}|y_{i+1} - y_i|.$$

$$\textit{oscarmax: } \Pi_\Omega(\boldsymbol{x}) = P_{\Delta^d}\left(P_{OSC}(\boldsymbol{x}/\gamma)\right), \ P_{OSC}(\boldsymbol{x}) := \arg\min_{\boldsymbol{y}\in\mathbb{R}^d} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2 + \lambda\sum_{i<j}\max(|y_i|, |y_j|).$$

Here, P_{TV} and P_{OSC} indicate the proximal operators of 1-d TV and OSCAR, and can be computed **exactly** by [13] and [47], respectively. To remind the reader, P_{Δ^d} denotes the Euclidean projection onto the simplex and can be computed exactly using [34, 15]. Proposition 2 shows that we can compute fusedmax and oscarmax using the composition of two functions, for which exact non-iterative algorithms exist. This is a surprising result, since the proximal operator of the sum of two functions is not, in general, the composition of the proximal operators of each function. The proof follows by showing that the indicator function of Δ^d satisfies the conditions of [45, Corollaries 4,5].

Groups induced by P_{TV} and P_{OSC}. Let $\boldsymbol{z}^\star$ be the optimal solution of $P_{\text{TV}}(\boldsymbol{x})$ or $P_{\text{OSC}}(\boldsymbol{x})$. For P_{TV}, we denote the group of **adjacent elements with the same value** as $z_i^\star$ by $G_i^\star$, $\forall i \in [d]$. Formally, $G_i^\star = [a, b] \cap \mathbb{N}$ with $a \leq i \leq b$ where a and b are the minimal and maximal indices such that $z_i^\star = z_j^\star$ for all $j \in G_i^\star$. For P_{OSC}, we define $G_i^\star$ as the indices of **elements with the same absolute value** as $z_i^\star$, more formally $G_i^\star = \{j \in [d] \colon |z_i^\star| = |z_j^\star|\}$. Because $P_{\Delta^d}(\boldsymbol{z}^\star) = \max(\boldsymbol{z}^\star - \theta, 0)$ for some $\theta \in \mathbb{R}$, fusedmax/oscarmax either shift a group's common value or set all its elements to zero.

λ controls the trade-off between no fusion (sparsemax) and all elements fused into a single trivial group. While tuning λ may improve performance, we observe that $\lambda = 0.1$ (fusedmax) and $\lambda = 0.01$ (oscarmax) are sensible defaults that work well across all tasks and report all our results using them.

Backward computation. We already know that the Jacobian of P_{Δ^d} is the same as that of sparsemax with $\gamma = 1$. Then, by Proposition 2, if we know how to compute the Jacobians of P_{TV} and P_{OSC}, we can obtain the Jacobians of fusedmax and oscarmax by straightforward application of the chain rule. However, although P_{TV} and P_{OSC} can be computed exactly, they lack analytical expressions. We next show that we can nonetheless compute their Jacobians efficiently, without needing to solve a system.

Proposition 3 *Jacobians of P_{TV} and P_{OSC} (backward computation)*

Assume $\boldsymbol{z}^\star = P_{TV}(\boldsymbol{x})$ or $P_{OSC}(\boldsymbol{x})$ has been computed. Define the groups derived from $\boldsymbol{z}^\star$ as above.

$$\text{Then, } [J_{P_{TV}}(\boldsymbol{x})]_{i,j} = \begin{cases} \frac{1}{|G_i^\star|} & \text{if } j \in G_i^\star, \\ 0 & \text{o.w.} \end{cases} \quad \text{and } [J_{P_{OSC}}(\boldsymbol{x})]_{i,j} = \begin{cases} \frac{\operatorname{sign}(z_i^\star z_j^\star)}{|G_i^\star|} & \text{if } j \in G_i^\star \text{ and } z_i^\star \neq 0, \\ 0 & \text{o.w.} \end{cases}.$$

The proof is given in Appendix A.2. Clearly, the structure of these Jacobians permits efficient Jacobian-vector products; we discuss the computational complexity and implementation details in Appendix B. Note that P_{TV} and P_{OSC} are differentiable everywhere except at points where groups change. For these points, the same remark as for sparsemax applies, and we can use Clarke's Jacobian.

4 Experimental results

We showcase the performance of our attention mechanisms on three challenging natural language tasks: textual entailment, machine translation, and sentence summarization. We rely on available, well-established neural architectures, so as to demonstrate simple drop-in replacement of softmax with structured sparse attention; quite likely, newer task-specific models could lead to further improvement.

4.1 Textual entailment (a.k.a. natural language inference) experiments

Textual entailment is the task of deciding, given a text T and an hypothesis H, whether a human reading T is likely to infer that H is true [14]. We use the Stanford Natural Language Inference (SNLI) dataset [8], a collection of 570,000 English sentence pairs. Each pair consists of a sentence and an hypothesis, manually labeled with one of the labels ENTAILMENT, CONTRADICTION, or NEUTRAL.

We use a variant of the neural attention–based classifier proposed for this dataset by [38] and follow the same methodology as [31] in terms of implementation, hyperparameters, and grid search. We employ the CPU implementation provided in [31] and simply replace sparsemax with fusedmax/oscarmax; we observe that training time per epoch is essentially the same for each of the four attention mechanisms (timings and more experimental details in Appendix C.2).

Table 1: Textual entailment test accuracy on SNLI [8].

attention	accuracy
softmax	81.66
sparsemax	82.39
fusedmax	**82.41**
oscarmax	81.76

Table 1 shows that, for this task, fusedmax reaches the highest accuracy, and oscarmax slightly outperforms softmax. Furthermore,

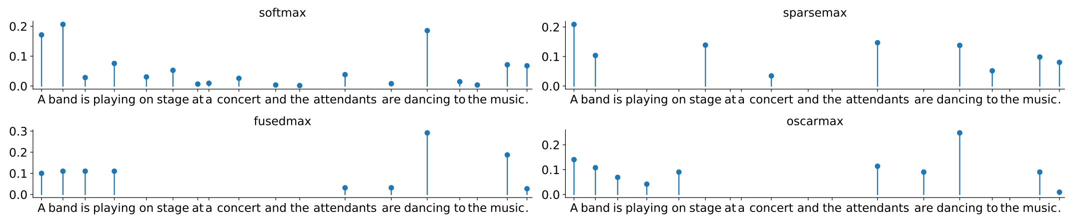

Figure 3: Attention weights when considering the contradicted hypothesis "No one is dancing."

fusedmax results in the most interpretable feature groupings: Figure 3 shows the weights of the neural network's attention to the text, when considering the hypothesis "No one is dancing." In this case, all four models correctly predicted that the text "A band is playing on stage at a concert and the attendants are dancing to the music," denoted along the x-axis, **contradicts** the hypothesis, although the attention weights differ. Notably, fusedmax identifies the meaningful segment "band is playing".

4.2 Machine translation experiments

Sequence-to-sequence neural machine translation (NMT) has recently become a strong contender in machine translation [2, 29]. In NMT, attention weights can be seen as an *alignment* between source and translated words. To demonstrate the potential of our new attention mechanisms for NMT, we ran experiments on 10 language pairs. We build on OpenNMT-py [24], based on PyTorch [37], with all default hyperparameters (detailed in Appendix C.3), simply replacing softmax with the proposed Π_Ω.

OpenNMT-py with softmax attention is optimized for the GPU. Since sparsemax, fusedmax, and oscarmax rely on sorting operations, we implement their computations on the CPU for simplicity, keeping the rest of the pipeline on the GPU. However, we observe that, even with this context switching, the number of tokens processed per second was within ¾ of the softmax pipeline. For sq-pnorm-max, we observe that the projected gradient solver used in the forward pass, unlike the linear system solver used in the backward pass, could become a computational bottleneck. To mitigate this effect, we set the tolerance of the solver's stopping criterion to 10^{-2}.

Quantitatively, we find that all compared attention mechanisms are always within 1 BLEU score point of the best mechanism (for detailed results, cf. Appendix C.3). This suggests that structured sparsity does not restrict accuracy. However, as illustrated in Figure 4, fusedmax and oscarmax often lead to more interpretable attention alignments, as well as to qualitatively different translations.

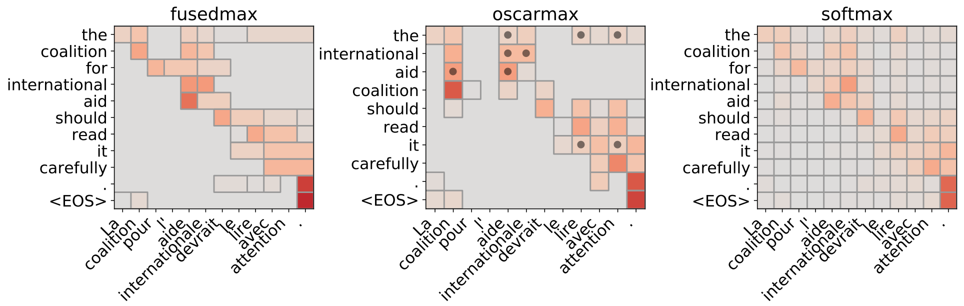

Figure 4: Attention weights for French to English translation, using the conventions of Figure 1. Within a row, weights grouped by oscarmax under the same cluster are denoted by "●". Here, oscarmax finds a slightly more natural English translation. More visualizations are given in Appendix C.3.

4.3 Sentence summarization experiments

Attention mechanisms were recently explored for sentence summarization in [39]. To generate sentence-summary pairs at low cost, the authors proposed to use the title of a news article as a noisy summary of the article's leading sentence. They collected 4 million such pairs from the Gigaword dataset and showed that this seemingly simplistic approach leads to models that generalize

Table 2: Sentence summarization results, following the same experimental setting as in [39].

	DUC 2004			Gigaword		
attention	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
softmax	27.16	9.48	24.47	35.13	17.15	32.92
sparsemax	27.69	9.55	24.96	36.04	**17.78**	33.64
fusedmax	**28.42**	**9.96**	**25.55**	**36.09**	17.62	**33.69**
oscarmax	27.84	9.46	25.14	35.36	17.23	33.03
sq-pnorm-max	27.94	9.28	25.08	35.94	17.75	33.66

surprisingly well. We follow their experimental setup and are able to reproduce comparable results to theirs with OpenNMT when using softmax attention. The models we use are the same as in §4.2.

Our evaluation follows [39]: we use the standard DUC 2004 dataset (500 news articles each paired with 4 different human-generated summaries) and a randomly held-out subset of Gigaword, released by [39]. We report results on ROUGE-1, ROUGE-2, and ROUGE-L. Our results, in Table 2, indicate that fusedmax is the best under nearly all metrics, always outperforming softmax. In addition to Figure 1, we exemplify our enhanced interpretability and provide more detailed results in Appendix C.4.

5 Related work

Smoothed max operators. Replacing the $\max$ operator by a differentiable approximation based on the $\log\text{sum}\exp$ has been exploited in numerous works. Regularizing the $\max$ operator with a squared 2-norm is less frequent, but has been used to obtain a smoothed multiclass hinge loss [41] or smoothed linear programming relaxations for maximum a-posteriori inference [33]. Our work differs from these in mainly two aspects. First, we are less interested in the $\max$ operator itself than in its gradient, which we use as a mapping from $\mathbb{R}^d$ to Δ^d. Second, since we use this mapping in neural networks trained with backpropagation, we study and compute the mapping's Jacobian (the Hessian of a regularized $\max$ operator), in contrast with previous works.

Interpretability, structure and sparsity in neural networks. Providing interpretations alongside predictions is important for accountability, error analysis and exploratory analysis, among other reasons. Toward this goal, several recent works have been relying on visualizing hidden layer activations [20, 27] and the potential for interpretability provided by attention mechanisms has been noted in multiple works [2, 38, 39]. Our work aims to fulfill this potential by providing a unified framework upon which new interpretable attention mechanisms can be designed, using well-studied tools from the field of structured sparse regularization.

Selecting contiguous text segments for model interpretations is explored in [26], where an *explanation generator* network is proposed for justifying predictions using a fused lasso penalty. However, this network is not an attention mechanism and has its own parameters to be learned. Furthemore, [26] sidesteps the need to backpropagate through the fused lasso, unlike our work, by using a stochastic training approach. In constrast, our attention mechanisms are deterministic and **drop-in replacements** for existing ones. As a consequence, our mechanisms can be coupled with recent research that builds on top of softmax attention, for example in order to incorporate soft prior knowledge about NMT alignment into attention through penalties on the attention weights [12].

A different approach to incorporating structure into attention uses the posterior marginal probabilities from a conditional random field as attention weights [23]. While this approach takes into account structural correlations, the marginal probabilities are generally dense and different from each other. Our proposed mechanisms produce sparse and clustered attention weights, a visible benefit in interpretability. The idea of deriving constrained alternatives to softmax has been independently explored for differentiable easy-first decoding [32]. Finally, sparsity-inducing penalties have been used to obtain convex relaxations of neural networks [5] or to compress models [28, 43, 40]. These works differ from ours, in that sparsity is enforced on the network **parameters**, while our approach can produce sparse and structured **outputs** from neural attention layers.

6 Conclusion and future directions

We proposed in this paper a unified regularized framework upon which new attention mechanisms can be designed. To enable such mechanisms to be used in a neural network trained by backpropagation, we demonstrated how to carry out forward and backward computations for general differentiable regularizers. We further developed two new structured attention mechanisms, *fusedmax* and *oscarmax*, and demonstrated that they enhance interpretability while achieving comparable or better accuracy on three diverse and challenging tasks: textual entailment, machine translation, and summarization.

The usefulness of a differentiable mapping from real values to the simplex or to $[0, 1]$ with sparse or structured outputs goes beyond attention mechanisms. We expect that our framework will be useful to sample from categorical distributions using the Gumbel trick [21, 30], as well as for conditional computation [6] or differentiable neural computers [18, 19]. We plan to explore these in future work.

Acknowledgements

We are grateful to André Martins, Takuma Otsuka, Fabian Pedregosa, Antoine Rolet, Jun Suzuki, and Justine Zhang for helpful discussions. We thank the anonymous reviewers for the valuable feedback.

References

[1] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*, 2017.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*, 2015.

[3] K. Ball, E. A. Carlen, and E. H. Lieb. Sharp uniform convexity and smoothness inequalities for trace norms. *Inventiones Mathematicae*, 115(1):463–482, 1994.

[4] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[5] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte. Convex neural networks. In *Proc. of NIPS*, 2005.

[6] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. In *Proc. of NIPS*, 2013.

[7] H. D. Bondell and B. J. Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR. *Biometrics*, 64(1):115–123, 2008.

[8] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*, 2015.

[9] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[10] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In *Proc. of NIPS*, 2015.

[11] F. H. Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.

[12] T. Cohn, C. D. V. Hoang, E. Vymolova, K. Yao, C. Dyer, and G. Haffari. Incorporating structural alignment biases into an attentional neural translation model. In *Proc. of NAACL-HLT*, 2016.

[13] L. Condat. A direct algorithm for 1-d total variation denoising. *IEEE Signal Processing Letters*, 20(11):1054–1057, 2013.

[14] I. Dagan, B. Dolan, B. Magnini, and D. Roth. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering*, 15(4):i–xvii, 2009.

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

[16] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. *Proc. of NIPS*, 2001.

[17] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

[18] A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. In *Proc. of NIPS*, 2014.

[19] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[20] O. Irsoy. *Deep sequential and structural neural models of compositionality*. PhD thesis, Cornell University, 2017.

[21] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. In *Proc. of ICLR*, 2017.

[22] S. M. Kakade, S. Shalev-Shwartz, and A. Tewari. Regularization techniques for learning with matrices. *Journal of Machine Learning Research*, 13:1865–1890, 2012.

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

[24] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-source toolkit for neural machine translation. *arXiv e-prints*, 2017.

[25] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL*, 2007.

[26] T. Lei, R. Barzilay, and T. Jaakkola. Rationalizing neural predictions. In *Proc. of EMNLP*, 2016.

[27] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in NLP. In *Proc. of NAACL-HLT*, 2016.

[28] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proc. of ICCVPR*, 2015.

[29] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*, 2015.

[30] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proc. of ICLR*, 2017.

[31] A. F. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*, 2016.

[32] A. F. Martins and J. Kreutzer. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proc. of EMNLP*, 2017.

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

[34] C. Michelot. 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.

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

[36] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[37] PyTorch. `http://pytorch.org`, 2017.

[38] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kocisky, and P. Blunsom. Reasoning about entailment with neural attention. In *Proc. of ICLR*, 2016.

[39] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proc. of EMNLP*, 2015.

[40] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

[41] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, 155(1):105–145, 2016.

[42] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[43] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Proc. of NIPS*, 2016.

[44] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of ICML*, 2015.

[45] Y. Yu. On decomposing the proximal map. In *Proc. of NIPS*, 2013.

[46] C. Zalinescu. *Convex analysis in general vector spaces*. World Scientific, 2002.

[47] X. Zeng and M. A. Figueiredo. Solving OSCAR regularization problems by fast approximate proximal splitting algorithms. *Digital Signal Processing*, 31:124–135, 2014.

[48] X. Zeng and F. A. Mario. The ordered weighted ℓ_1 norm: Atomic formulation, dual norm, and projections. *arXiv e-prints*, 2014.

[49] L. W. Zhong and J. T. Kwok. Efficient sparse modeling with automatic feature grouping. *IEEE transactions on neural networks and learning systems*, 23(9):1436–1447, 2012.

Supplementary material

A Proofs

A.1 Proof of Proposition 1

Recall that

$$\Pi_\Omega(\boldsymbol{x}) = \operatorname*{arg\,min}_{\boldsymbol{y}\in\Delta^d} f(\boldsymbol{y}), \quad \text{where} \quad f(\boldsymbol{y}) \coloneqq \gamma\Omega(\boldsymbol{y}) - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}.$$

At an optimal solution, we have the fixed point iteration [36, §4.2]

$$\boldsymbol{y}^* = P_{\Delta^d}(\boldsymbol{y}^\star - \nabla f(\boldsymbol{y}^\star)). \tag{2}$$

Seeing $\boldsymbol{y}^\star$ as a function of $\boldsymbol{x}$, and P_{Δ^d} and ∇f as functions of their inputs, we can apply the chain rule to (2) to obtain

$$J_{\Pi_\Omega}(\boldsymbol{x}) = J_{P_{\Delta^d}}(\boldsymbol{y}^\star - \nabla f(\boldsymbol{y}^\star))\left(J_{\Pi_\Omega}(\boldsymbol{x}) - J_{\nabla f\circ \boldsymbol{y}^\star}(\boldsymbol{x})\right). \tag{3}$$

Applying the chain rule once again to $\nabla f(\boldsymbol{y}^\star) = \gamma\nabla\Omega(\boldsymbol{y}^\star) - \boldsymbol{x}$, we obtain

$$\begin{aligned} J_{\nabla f\circ \boldsymbol{y}^\star}(\boldsymbol{x}) &= \gamma J_{\nabla\Omega}(\boldsymbol{y}^\star) J_{\Pi_\Omega}(\boldsymbol{x}) - I \\ &= \gamma H_\Omega(\boldsymbol{y}^\star) J_{\Pi_\Omega}(\boldsymbol{x}) - I. \end{aligned}$$

Plugging this into (3) and re-arranging, we obtain

$$\left(I + A(B - I)\right) J_{\Pi_\Omega} = A,$$

where we defined the shorthands

$$J_{\Pi_\Omega} \coloneqq J_{\Pi_\Omega}(\boldsymbol{x}), \quad A \coloneqq J_{P_{\Delta^d}}(\boldsymbol{y}^\star - \gamma\nabla\Omega(\boldsymbol{y}^\star) + \boldsymbol{x}) \quad \text{and} \quad B \coloneqq \gamma H_\Omega(\boldsymbol{y}^\star).$$

A.2 Proof of Proposition 3

Proof outline. Let $\boldsymbol{z}^\star = P_{\text{TV}}(\boldsymbol{x})$ or $P_{\text{OSC}}(\boldsymbol{x})$. We use the optimality conditions of P_{TV}, respectively P_{OSC} in order to express $\boldsymbol{z}^\star$ as an explicit function of $\boldsymbol{x}$. Then, obtaining the Jacobians of $P_{\text{TV}}(\boldsymbol{x})$ and $P_{\text{OSC}}(\boldsymbol{x})$ follows by application of the chain rule to the two expressions. We discuss the proof for points where P_{TV} and P_{OSC} are differentiable; on the (zero-measure) set of nondifferentiable points (i.e. where the group structure changes) we may take one of Clarke's generalized gradients [11].

Jacobian of P_{TV}.

Lemma 1 *Let $\boldsymbol{z}^\star = P_{TV}(\boldsymbol{x}) \in \mathbb{R}^d$ and $G_i^\star$ be the set of indices around i with the same value at the optimum, as defined in §3.2. Then, we have*

$$z_i^\star = \frac{\sum_{j\in G_i^\star} x_j + \lambda(s_{a_i} - s_{b_i})}{|G_i^\star|}, \tag{4}$$

where $a_i = \min G_i^\star, b_i = \max G_i^\star$ are the boundaries of segment $G_i^\star$, and

$$s_{a_i} = \begin{cases} 0 & \textit{if } a = 1, \\ \operatorname{sign}(z_{a_i-1}^\star - z_i^\star) & \textit{if } a > 1 \end{cases} \quad \textit{and} \quad s_{b_i} = \begin{cases} 0 & \textit{if } b = d, \\ \operatorname{sign}(z_i^\star - z_{b_i+1}^\star) & \textit{if } b < d \end{cases}.$$

To prove Lemma 1, we make use of the optimality conditions of the fused lasso proximity operator [17, Equation 27], which state that $\boldsymbol{z}^\star$ satisfies

$$z_j^\star - x_j + \lambda(t_j - t_{j+1}) = 0, \quad \text{where} \quad t_j \in \begin{cases} \{0\} & \text{if } i \in \{1, d\}, \\ \{\operatorname{sign}(z_j^\star - z_{j-1}^\star)\} & \text{if } z_j^\star \neq z_{j-1}^\star, \\ [-1, 1] & \text{o.w.} \end{cases} \quad \forall j \in [d]. \tag{5}$$

The optimality conditions (5) form a system with unknowns $z_j^\star, t_j$ for $j \in [d]$. To express $\boldsymbol{z}^\star$ as a function of $\boldsymbol{x}$, we shall now proceed to eliminate the unknowns t_j.

Let us focus on a particular segment $G_i^\star$. For readability, we drop the segment index i and use the shorthands $z \coloneqq z_i^\star, a \coloneqq a_i$, and $b \coloneqq b_i$. By definition, a and b satisfy

$$z_j^\star = z \quad \forall a \le j \le b, \qquad z_{a-1}^\star \ne z \text{ if } a > 1, \qquad z_{b+1}^\star \ne z \text{ if } b < d.$$

It immediately follows from the definition of t_j in (5) that

$$t_a = \begin{cases} 0 & \text{if } a = 1, \\ \operatorname{sign}(z - z_{a-1}^\star) & \text{if } a > 1 \end{cases} \quad \text{and} \quad t_{b+1} = \begin{cases} 0 & \text{if } b = d, \\ \operatorname{sign}(z_{b+1}^\star - z) & \text{if } b < d \end{cases}.$$

In other words, the unknowns t_a and t_b are already uniquely determined. To emphasize that they are known, we introduce $s_a \coloneqq t_a$ and $s_b \coloneqq t_{b+1}$, leaving t_j only unknown for $a < j \le b$.

By rearranging the optimality conditions (5) we obtain the recursion

$$\lambda t_j = x_j - z + \lambda t_{j+1} \quad \forall a \le j \le b.$$

We start with the first equation in the segment (at $j = a$), and unroll the recursion until reaching the stopping condition $j = b$.

$$\begin{aligned} \lambda s_a &= x_a - z + \lambda t_{a+1} \\ &= x_a - z + x_{a+1} - z + \cdots + x_b - z + \lambda s_b \\ &= \sum_{k=a}^{b} x_k - (b - a + 1) z + \lambda s_b \end{aligned}$$

Rearranging the terms, we obtain the expression

$$z = \frac{\sum_{k=a}^{b} x_k + \lambda(s_b - s_a)}{b - a + 1}.$$

Applying this calculation to each segment in $\boldsymbol{z}^\star$ yields the desired result. □

The proof of Proposition 3 follows by applying the chain rule to (4), noting that the groups $G_i^\star$ are constant within a neighborhood of $\boldsymbol{x}$ (observation also used for OSCAR in [7]). Therefore, for P_{TV},

$$\frac{\partial z_i^\star}{\partial x_j} = \frac{1}{|G_i^\star|} \left(\sum_{k \in G_i^\star} \frac{\partial x_k}{\partial x_j} + \lambda \left(\frac{\partial s_b}{\partial x_j} - \frac{\partial s_a}{\partial x_j} \right) \right).$$

Since s_b and s_a are either constant or sign functions w.r.t. $\boldsymbol{x}$, their partial derivatives are 0, and thus

$$\frac{\partial z_i^\star}{\partial x_j} = \begin{cases} \frac{1}{|G_i^\star|} & \text{if } j \in G_i^\star, \\ 0 & \text{o.w.} \end{cases}.$$

Jacobian of P_{OSC}.

Lemma 2 *([47, Theorem 1], [49, Proposition 3]) Let $\boldsymbol{z}^\star = P_{OSC}(\boldsymbol{x}) \in \mathbb{R}^d$ and $G_i^\star$ be the set of indices around i with the same value at the optimum: $G_i^\star = \{j \in [d] : |z_i^\star| = |z_j^\star|\}$. Then, we have*

$$z_i^\star = \operatorname{sign}(x_i) \max \left(\frac{\sum_{j \in G_i^\star} |x_j|}{|G_i^\star|} - w_i, 0 \right), \tag{6}$$

where $w_i = \lambda \left(d - \frac{u_i + v_i}{2} \right), \quad u_i = \left| \{ j \in [d] : |z_j^\star| < |z_i^\star| \} \right|, \quad v_i = u_i + |G_i^\star|.$

Lemma 2 is a simple reformulation of Theorem 1, part *ii* from [47]. With the same observation that the induced groups do not change within a neighborhood of $\boldsymbol{x}$, we may differentiate (6) to obtain

$$\frac{\partial z_i^\star}{\partial x_j} = \begin{cases} 0 & \text{if } z_i^\star = 0, \\ \dfrac{\operatorname{sign}(x_i)}{|G_i^\star|} \displaystyle\sum_{k \in G_i^\star} \frac{\partial |x_k|}{\partial x_k} \frac{\partial x_k}{\partial x_j} - \frac{\partial w_i}{\partial x_j} & \text{o.w.} \end{cases}.$$

Noting that $\frac{\partial w_i}{\partial x_j} = 0$, as w_i is derived only from group indices and the term $\frac{\partial |x_k|}{\partial x_k} \frac{\partial x_k}{\partial x_j}$ either vanishes (when $k \neq j$) or else equals $\operatorname{sign}(x_j)$ with $x_j \neq 0$, we substitute $\operatorname{sign}(z_j^\star)$ for $\operatorname{sign}(x_j)$ [47] to get

$$\frac{\partial z_i^\star}{\partial x_j} = \begin{cases} \dfrac{\operatorname{sign}(z_i^\star z_j^\star)}{|G_i^\star|} & \text{if } j \in G_i^\star \text{ and } z_i^\star \neq 0, \\ 0 & \text{o.w.} \end{cases}.$$

B Computational complexity and implementation details

B.1 Sparsemax

Computing the forward and backward pass of sparsemax is a compositional building block in fusedmax, oscarmax, as well as in the general case; for this reason, we discuss it before the others.

Forward pass. The problem is exactly the Euclidean projection on the simplex, which can be computed exactly in worst-case $\mathcal{O}(d \log d)$ due to the required sort [31, 34, 15], or in expected $\mathcal{O}(d)$ time using a pivot algorithm similar to median finding [15]. Our implementation is based on sorting.

Backward pass. From [31] we have that the result of a Jacobian-vector product $J_{\Pi_\Omega} v$ has the same sparsity pattern as $\boldsymbol{y}^\star$. If we denote the number of nonzero elements of $\boldsymbol{x}$ by $\operatorname{nnz}(\boldsymbol{x})$, we can see that $\hat{v}$ in [31, eq. 14], and thus the Jacobian-vector product itself, can be computed in $\mathcal{O}(\operatorname{nnz}(\boldsymbol{y}^\star))$.

B.2 Fusedmax

We implement fusedmax as the composition of the fused lasso proximal operator with sparsemax.

Forward pass. We need to solve the proximal operator of fused lasso. The algorithm we use is $\mathcal{O}(d^2)$ in the worst case, but has strong performance on realistic benchmarks, close to $\mathcal{O}(d)$ [13].

Backward pass. Due to the structure of the Jacobian and the locality of fused groups, Jacobian-vector products $J_{\Pi_\Omega} v$ can be computed in $\mathcal{O}(d)$ using a simple algorithm that iterates over the output $\boldsymbol{y}^\star$ and the vector $\boldsymbol{v}$ simultaneously, averaging the elements of $\boldsymbol{v}$ whose indices map to fused elements of $\boldsymbol{y}^\star$. Since only consecutive elements can be fused, this amounts to resetting to a new group as soon as we encounter an index i such that $y_i^\star \neq y_{i-1}^\star$.

B.3 Oscarmax

We implement oscarmax as the composition of the OSCAR proximal operator with sparsemax.

Forward pass. The proximal operator of the OSCAR penalty can be computed in $\mathcal{O}(d \log d)$ as a particular case of the ordered weighted ℓ_1 (OWL) proximal operator, using an algorithm involving a sort followed by isotonic regression [48].

Backward pass. The algorithm is similar in spirit to fusedmax, but because groups can reach across non-adjacent indices, a single pass is not sufficient. With no other information other than $\mathbf{y}^\star$, the backward pass can be computed in $\mathcal{O}(d \log d)$ using a stable sort followed by a linear-time pass for finding groups. Further optimization is possible if group indices may be saved from the forward pass.

B.4 General case and sq-pnorm-max

Forward pass. For general Π_Ω we may use any projected gradient solver; we choose FISTA [4]. Each iteration requires a projection onto the simplex; in the case of sq-pnorm-max, this dominates every iteration, leading to a complexity of $\mathcal{O}(td \log d)$ where t is the number of iterations performed.

Backward pass. To compute Jacobian-vector products we solve the linear system from Proposition 1: $(I + A(B - I))\,(J_{\Pi_\Omega}\boldsymbol{v}) = A\boldsymbol{v}$. This is a $d \times d$ system, which at first sight suggests a complexity of $\mathcal{O}(d^3)$. However, we can use the structure of A to solve it more efficiently.

The matrix A is defined as $A := J_{P_{\Delta^d}}(\boldsymbol{y}^\star - \nabla f(\boldsymbol{y}^\star))$. As a sparsemax Jacobian, A is row- and column-sparse, and uniquely defined by its sparsity pattern. By splitting the system into equations corresponding to zero and nonzero rows of A, we obtain that the solution $J_{\Pi_\Omega}\boldsymbol{v}$ must have the same sparsity pattern as the row-sparsity of A, therefore we only need to solve a subset of the system. From the fixed-point iteration $\boldsymbol{y}^* = P_{\Delta^d}(\boldsymbol{y}^\star - \nabla f(\boldsymbol{y}^\star))$, we have that the row-sparsity of A is the same as the sparsity of the forward pass solution $\boldsymbol{y}^*$. The backward pass complexity is thus $\mathcal{O}(\text{nnz}(\boldsymbol{y}^*)^3)$.

C Additional experimental results

C.1 Visualizing attention mappings in 3-d

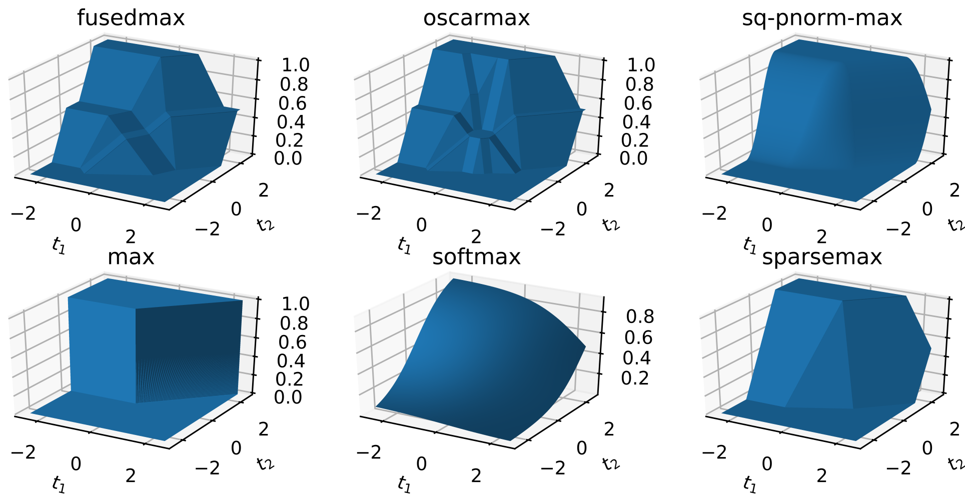

Figure 5: 3-d visualization of $\Pi_\Omega([t_1, t_2, 0])_2$ for several proposed and existing mappings Π_Ω. sq-pnorm-max with $p = 1.5$ resembles sparsemax but with smoother transitions. The proposed structured attention mechanisms, fusedmax and oscarmax, exhibit plateaus and ridges in areas where weights become fused together. We set $\gamma = 1$ and $\lambda = 0.2$.

C.2 Textual entailment results

Experimental setup. We build upon the implementation from [31], which is a slight variation of the attention model from [38], using GRUs instead of LSTMs. The GRUs encoding the premise and hypothesis have separate parameters, but the hypothesis GRU is initialized with the last state of the premise GRU. We use the same settings and methodology as [31]: we use fixed 300-dimensional GloVe vectors, we train for 200 epochs using ADAM with learning rate $3 \cdot 10^{-4}$, we use a drop-out probability of 0.1, and we choose an l_2 regularization coefficient from $\{0, 10^{-4}, 3 \cdot 10^{-4}, 10^{-3}\}$. Experiments are performed on machines with 2×Xeon X5675 3.06GHz CPUs and 96GB RAM.

Dataset and preprocessing. We use the SNLI v1 dataset [8]. We apply the minimal preprocessing from [31], skipping sentence pairs with missing labels and using the provided tokenization. This results in a training set of 549,367 sentence pairs, a development set of 9,842 sentence pairs and a test set of 9,824 sentence pairs. We report timing measurements in Table 3 and visualizations of the produced attention weights in Figure 6.

C.3 Machine translation results

Experimental setup. Because our goal is to demonstrate that our attention mechanisms can be drop-in replacements for existing ones, we focus on OpenNMT-py with default settings for all of our

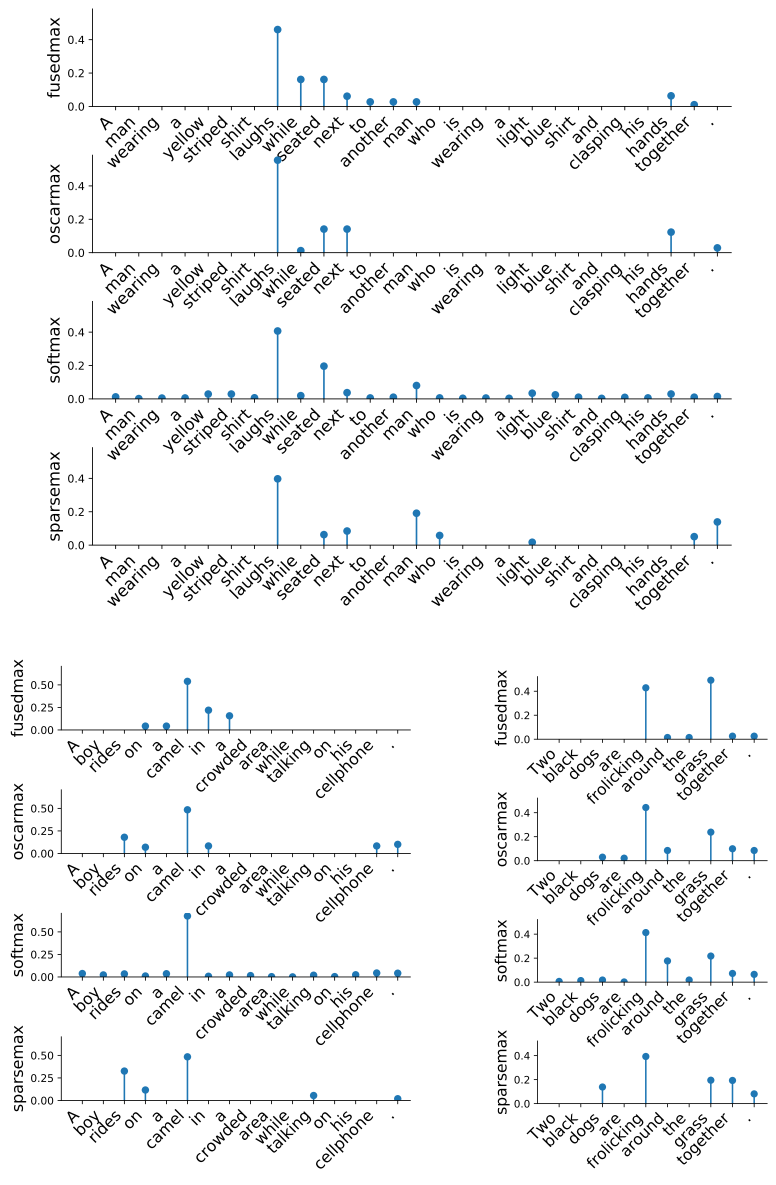

Figure 6: Attention weights on several examples also used in [38, 31]. The hypotheses considered are "Two mimes sit in complete silence." (top), "A boy is riding an animal." (left), and "Two dogs swim in the lake." (right). All attention mechanisms result in correct classifications (top: contradiction; left: entailment; right: contradiction). As can be seen, fusedmax prefers contiguous support segments even when not all weights are tied.

sequence-to-sequence experiments. These defaults are: an unidirectional LSTM, 500 dimensions for the word vectors and for the LSTM hidden representations, drop-out probability of 0.3, global attention, and input-feeding [29]. Following the default, we train our models for 13 epochs with stochastic gradient updates (batches of size 64 and initial learning rate of 1, halved every epoch after the 8th). Weights (including word embeddings) are initialized uniformly over $[-0.1, 0.1]$, and gradients are normalized to have norm 5 if their norm exceeds this value. For test scores and visualizations, we use the model snapshot at the epoch with the highest validation set accuracy. All of the experiments in this section are performed on machines equiped with Xeon E5 CPUs and Nvidia Tesla K80 GPUs.

Datasets. We employ training and test datasets from multiple sources.

attention	time per epoch
softmax	1h 26m 40s ± 51s
sparsemax	1h 24m 21s ± 54s
fusedmax	1h 23m 58s ± 50s
oscarmax	1h 23m 19s ± 50s

Table 3: Timing results for training textual entailment on SNLI, using the implementation and experimental setup from [31]. With this C++ CPU implementation, fusedmax and oscarmax are as fast as sparsemax, and all three sparse attention mechanisms are slightly faster than softmax.

- BENCHMARK: Training, validation, and test data from the NMT-Benchmark project (http://scorer.nmt-benchmark.net/). All languages have ~1M training sentence pairs, and equal validation and test sets of size 1K (French) and 2K (Italian, Dutch and Swedish).
- BENCHMARK$^+$: Training and validation data as above, but testing on all available *newstest* data. For Italian we use the 2009 data (~2.5K sentence pairs), and for French we concatenate 2009–2014 (~11K sentence pairs).
- WMT16, WMT17: Translation tasks at the first and second ACL Conferences for Machine Translation, available at http://www.statmt.org/wmt16/translation-task.html and http://www.statmt.org/wmt17/translation-task.html. Training, validation, and test sizes are, approximately, for Romanian 400K/2K/2K, for German 5.8M/6K/3K, for Finnish 2.6M/2K/2K, for Latvian 4.5M/2K/2K, and for Turkish 207K/1K/3K.

We use the preprocessing scripts from Moses [25] for tokenization, and, where needed, SGML parsing. We limit source and target vocabulary sizes to 50K lower-cased tokens and prune sentences longer than 50 tokens at training time and 100 tokens at test time. We do not perform recasing.

We report BLEU scores in Table 4 and showcase the enhanced interpretability induced by our proposed attention mechanisms in Figure 7. Timing measurements can be found in Table 5.

Table 4: Neural machine translation results: tokenized BLEU scores on test data.

	BENCHMARK				BENCHMARK$^+$		WMT16	WMT17			
	fr	it	nl	sv	fr	it	ro	de	fi	lv	tr
from English											
softmax	36.94	37.20	36.12	34.97	27.13	**24.86**	17.71	22.32	14.54	11.02	**11.95**
sparsemax	37.03	37.21	36.12	**35.09**	26.99	24.49	17.61	**22.43**	**14.85**	11.07	11.66
fusedmax	37.08	36.73	36.04	34.30	26.89	24.47	17.19	22.25	14.28	**11.27**	11.32
oscarmax	36.66	36.89	35.96	34.86	27.02	24.76	17.26	22.42	14.02	11.19	11.63
sq-pnorm-max	**37.16**	**37.39**	**36.21**	34.63	**27.25**	24.56	**17.80**	—	14.45	—	11.58
to English											
softmax	36.79	39.95	40.06	37.96	25.72	25.37	17.86	**25.82**	15.11	13.60	11.78
sparsemax	**36.91**	40.13	40.25	38.09	25.97	25.62	17.46	25.76	14.95	13.59	**12.04**
fusedmax	36.64	39.64	39.87	37.83	25.72	25.41	**18.29**	25.58	15.08	13.53	11.91
oscarmax	36.90	40.05	40.17	**38.12**	**26.13**	25.65	17.89	25.69	14.94	**13.71**	11.70
sq-pnorm-max	36.84	**40.23**	**40.48**	**38.12**	25.72	**25.70**	17.44	—	**15.20**	—	11.93

attention	time per epoch
softmax	2h
sparsemax	2h 18m
fusedmax	3h 5m
oscarmax	3h 25m
sq-pnorm-max	7h 5m

Table 5: Timing results for French-to-English translation using OpenNMT-py (all standard errors are under 2 minutes). For simplicity, all attention mechanisms, except softmax, are implemented on the CPU, thus incurring memory copies in both directions. (The rest of the pipeline runs on the GPU.) Even without special optimization, sparsemax, fusedmax, and oscarmax are practical, taking within 1.75x the training time of a softmax model on the GPU.

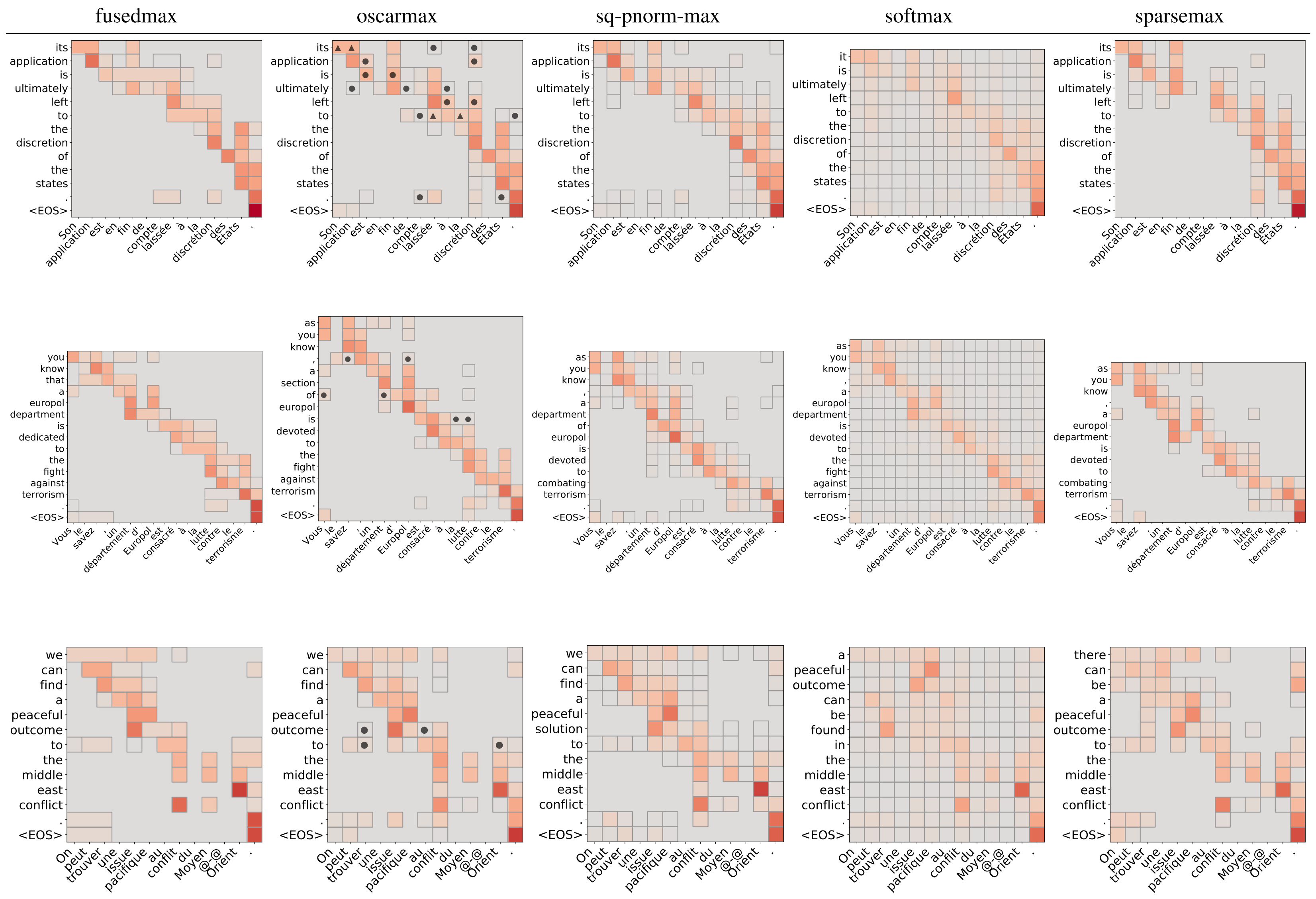

Figure 7: Attention alignment examples for French-to-English translation, following the conventions of Figure 1. “@-@" denotes a hyphen not separated by spaces. When oscarmax induces multiple clusters, we denote them using different bullets (e.g., ●, ▲, ■). Fusedmax often selects meaningful grammatical segments, such as “est consacré,” as well as determiner-noun constructions.

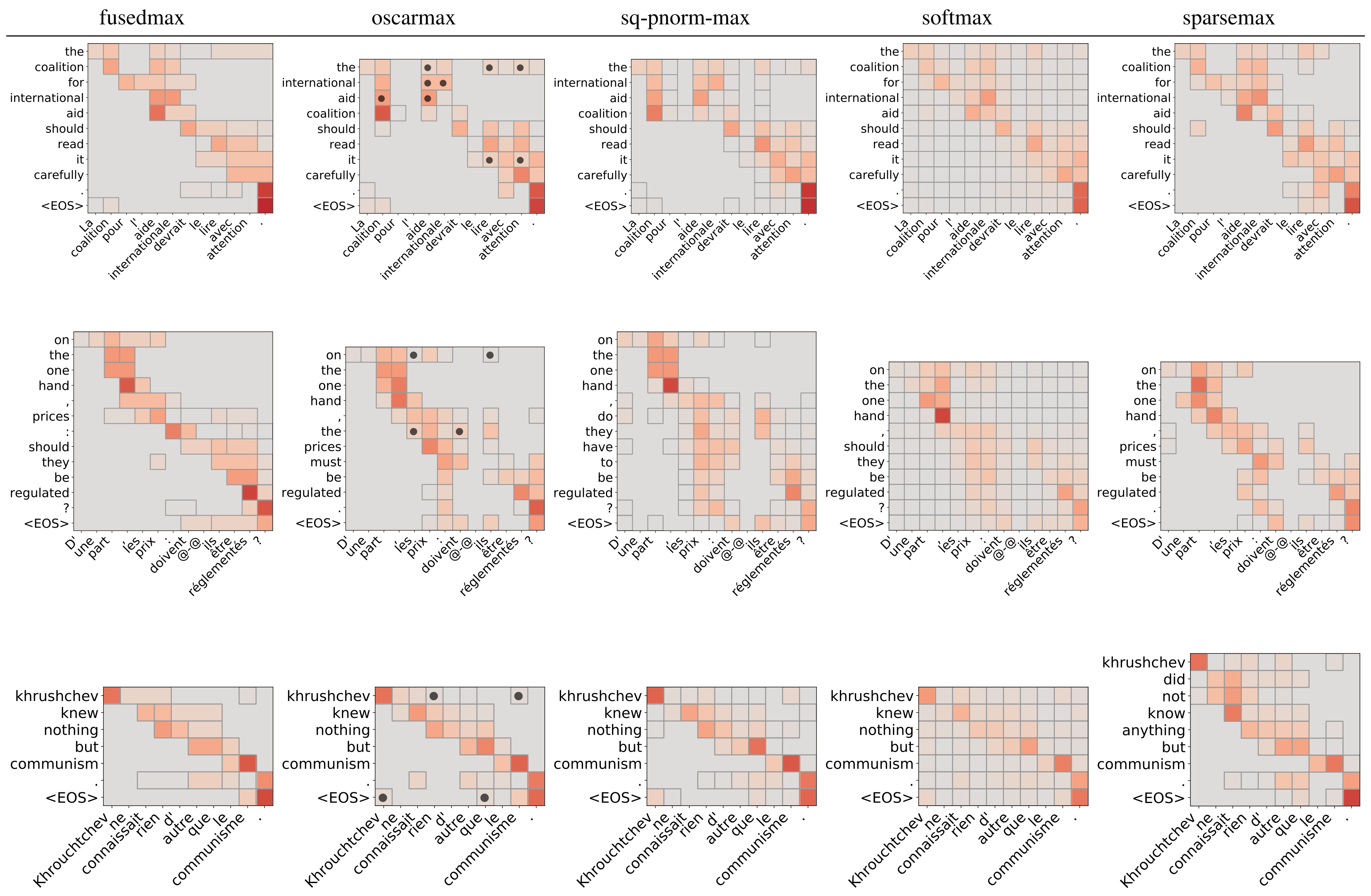

Figure 7 (continued): Further translation examples from French to English.

C.4 Sentence summarization results

Experimental setup and data. We use the exact same experimental setup and preprocessing as for machine translation, described in Appendix C.3. We use the preprocessed Gigaword sentence summarization dataset, made available by the authors of [39] at `https://github.com/harvardnlp/sent-summary`. Since, unlike [39], we do not perform any tuning on DUC-2003, we can report results on this dataset, as well. We observe that the simple sequence-to-sequence model is able to keep summaries short without any explicit constraints, informed only through training data statistics; therefore, in this section, we also report results without output truncation at 75 bytes (Table 6). We also provide precision and recall scores for ROUGE-L in Table 7. Finally, we provide attention weights plots for all studied attention mechanisms and a number of validation set examples in Figure 8.

Table 6: Sentence summarization F_1 scores for several ROUGE variations.

	Truncated				Not truncated			
attention	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-W$_{1.2}$	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-W$_{1.2}$
DUC 2003								
softmax	26.63	8.72	23.87	16.95	27.06	8.86	24.23	17.02
sparsemax	26.54	8.78	23.89	16.93	26.95	8.94	24.21	16.99
fusedmax	**27.12**	8.93	**24.39**	**17.28**	**27.48**	9.04	**24.66**	**17.30**
oscarmax	26.72	**9.08**	24.02	17.06	27.11	**9.23**	24.32	17.10
sq-pnorm-max	26.55	8.77	23.78	16.87	26.92	8.89	24.07	16.92
DUC 2004								
softmax	27.16	9.48	24.47	17.14	27.25	9.52	24.55	17.20
sparsemax	27.69	9.55	24.96	17.44	27.77	9.61	25.02	17.48
fusedmax	**28.42**	**9.96**	**25.55**	**17.78**	**28.43**	**9.96**	**25.55**	**17.79**
oscarmax	27.84	9.46	25.14	17.55	27.88	9.47	25.17	17.57
sq-pnorm-max	27.94	9.28	25.08	17.49	28.01	9.30	25.13	17.52
Gigaword								
softmax	35.13	17.15	32.92	24.17	35.01	17.10	32.77	24.00
sparsemax	36.04	**17.78**	33.64	24.69	35.97	**17.75**	33.54	**24.55**
fusedmax	**36.09**	17.62	**33.69**	24.69	**35.98**	17.60	**33.59**	24.54
oscarmax	35.36	17.23	33.03	24.25	35.26	17.20	32.92	24.10
sq-pnorm-max	35.94	17.75	33.66	**24.71**	35.86	17.73	33.54	**24.55**

Table 7: Sentence summarization: ROUGE-L precision, recall and F-scores.

	Truncated			Not truncated		
attention	P	R	F_1	P	R	F_1
DUC 2003						
softmax	29.57	20.67	23.87	30.40	20.80	24.23
sparsemax	29.59	20.58	23.89	30.37	20.68	24.21
fusedmax	**30.02**	**21.11**	**24.39**	**30.75**	**21.15**	**24.66**
oscarmax	29.64	20.78	24.02	30.40	20.87	24.32
sq-pnorm-max	29.45	20.50	23.78	30.23	20.56	24.07
DUC 2004						
softmax	30.54	21.00	24.47	30.59	21.13	24.55
sparsemax	30.99	21.57	24.96	31.03	21.64	25.02
fusedmax	**32.19**	**21.80**	**25.55**	**32.19**	**21.81**	**25.55**
oscarmax	31.89	21.46	25.14	31.91	21.51	25.17
sq-pnorm-max	31.42	21.55	25.08	31.46	21.63	25.13
Gigaword						
softmax	36.43	31.67	32.92	36.61	31.54	32.77
sparsemax	37.32	32.18	33.64	37.54	32.07	33.54
fusedmax	**37.44**	32.15	**33.69**	**37.68**	32.01	**33.59**
oscarmax	36.40	31.78	33.03	36.61	31.67	32.92
sq-pnorm-max	37.12	**32.37**	33.66	37.31	**32.26**	33.54

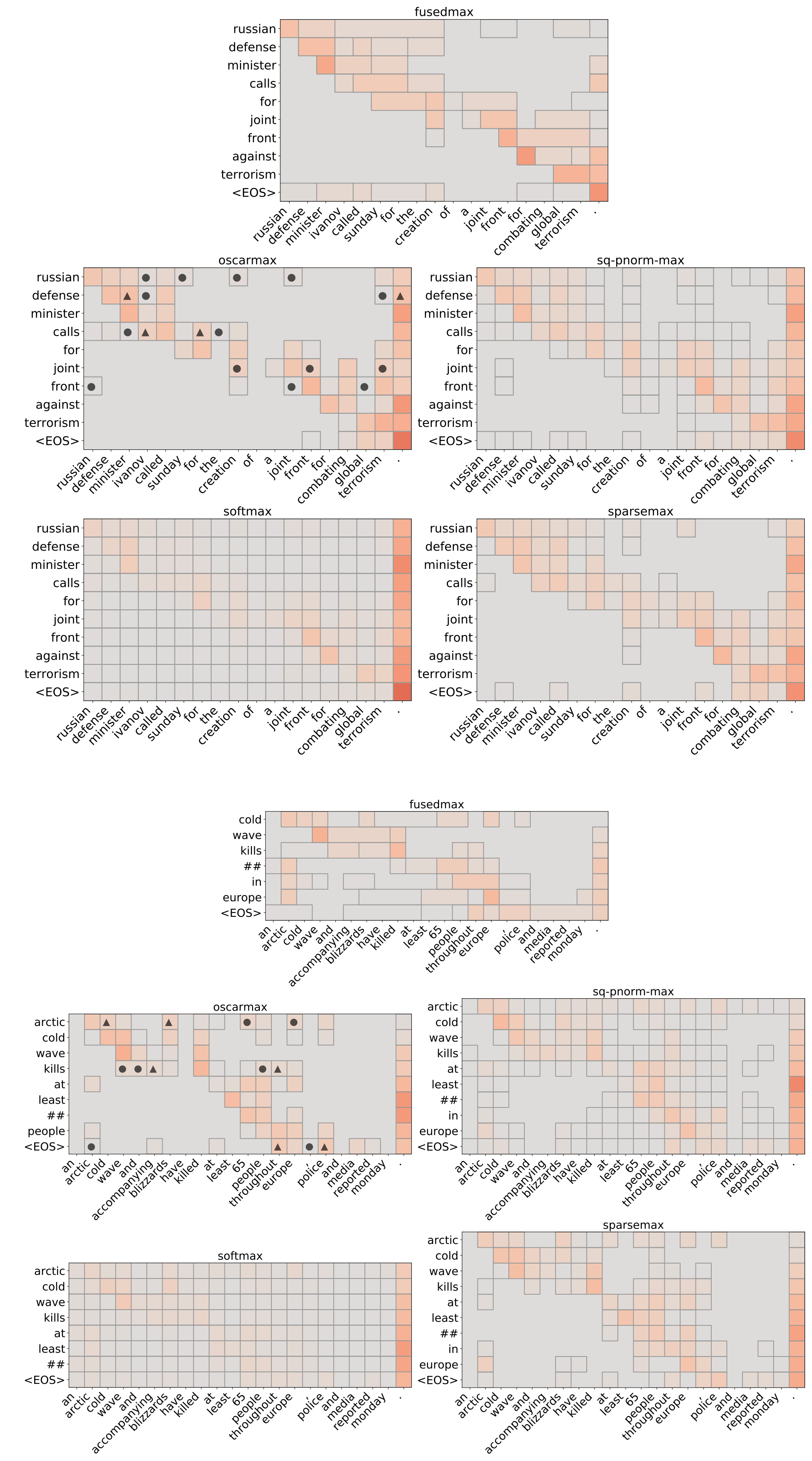

Figure 8: Summarization attention examples. The 1-d TV prior of fusedmax captures well the intuition of aligning long input spans with single expressive words, as supported by ROUGE scores.

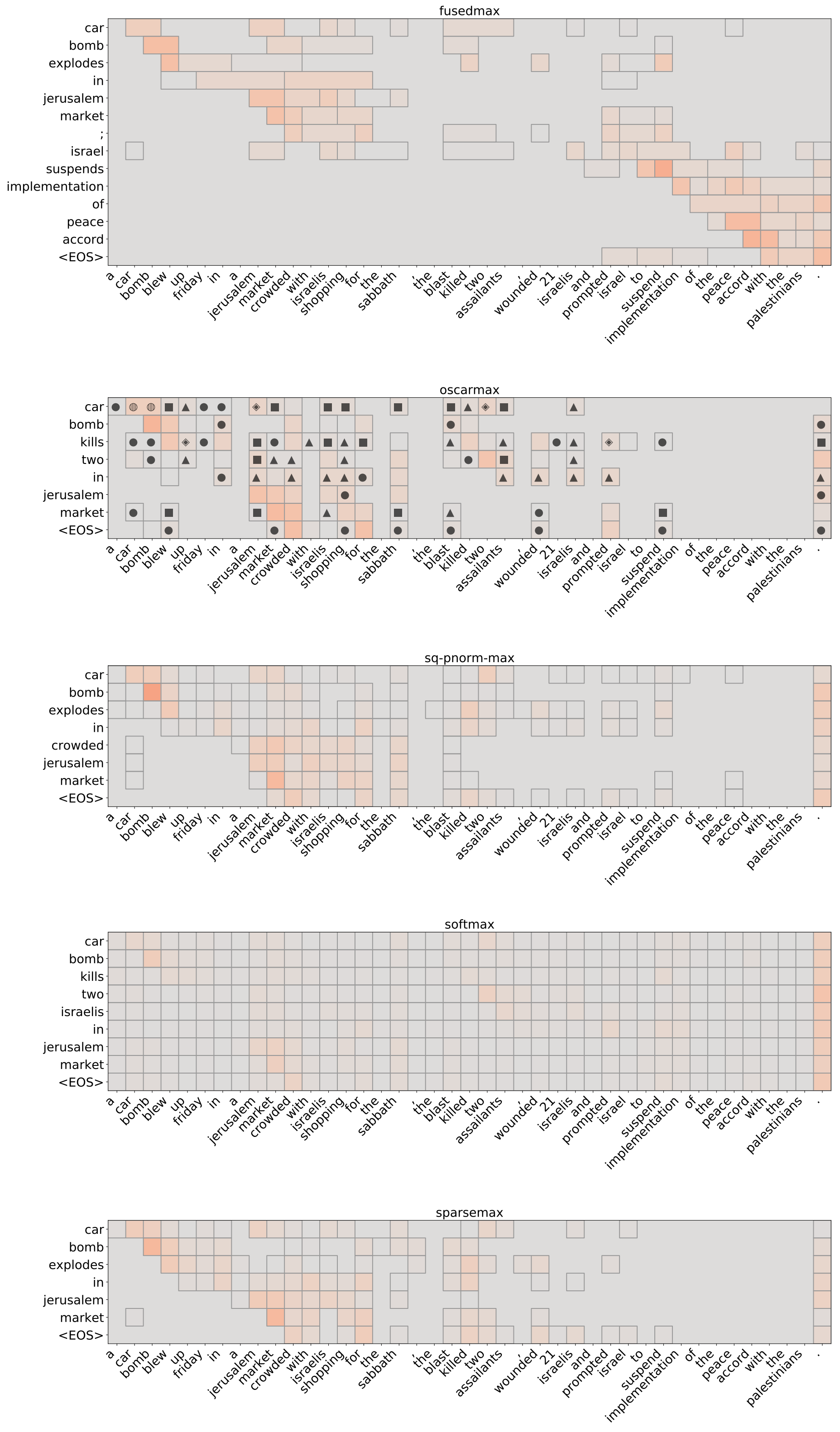

Figure 8 (continued): Summarization attention examples. Here, fusedmax recovers a longer but arguably better summary, identifying a separate but important part of the input sentence.

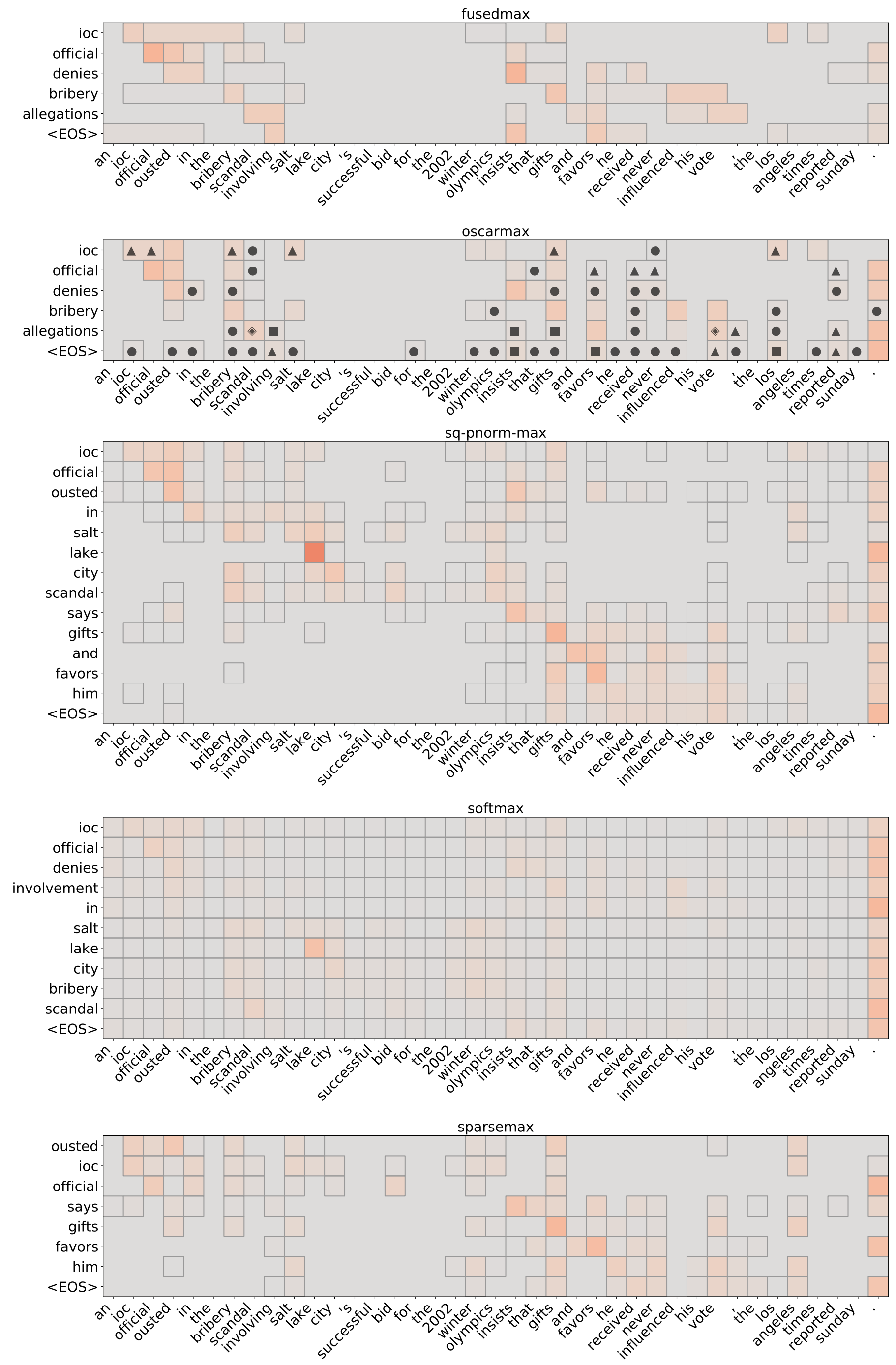

Figure 8 (continued): Summarization attention examples. Here, fusedmax and oscarmax produce a considerably shorter summary.