Modeling Structure with Undirected Neural Networks

Mihaylova, T.; Niculae, V.; Martins, A.F.T.

Publication date
2022
Document Version
Final published version
Published in
Proceedings of Machine Learning Research
License
Other

Modeling Structure with Undirected Neural Networks

Tsvetomila Mihaylova [1] **Vlad Niculae** [2] **André F.T. Martins** [1 3 4]

Abstract

Neural networks are powerful function estimators, leading to their status as a paradigm of choice for modeling structured data. However, unlike other structured representations that emphasize the modularity of the problem – *e.g.*, factor graphs – neural networks are usually monolithic mappings from inputs to outputs, with a fixed computation order. This limitation prevents them from capturing different directions of computation and interaction between the modeled variables. In this paper, we combine the representational strengths of factor graphs and of neural networks, proposing *undirected neural networks (UNNs)*: a flexible framework for specifying computations that can be performed in any order. For particular choices, our proposed models subsume and extend many existing architectures: feed-forward, recurrent, self-attention networks, auto-encoders, and networks with implicit layers. We demonstrate the effectiveness of undirected neural architectures, both unstructured and structured, on a range of tasks: tree-constrained dependency parsing, convolutional image classification, and sequence completion with attention. By varying the computation order, we show how a single UNN can be used both as a classifier and a prototype generator, and how it can fill in missing parts of an input sequence, making them a promising field for further research.

1. Introduction

Factor graphs have historically been a very appealing toolbox for representing structured prediction problems (Bakır et al., 2007; Smith, 2011; Nowozin et al., 2014), with wide applications to vision and natural language processing applications. In recent years, neural networks have taken over as the model of choice for tackling many such applications. Unlike factor graphs – which emphasize the modularity of the problem – neural networks typically work end-to-end, relying on rich representations captured at the encoder level (often pre-trained), which are then propagated to a task-specific decoder.

In this paper, we combine the representational strengths of factor graphs and neural networks by proposing **undirected neural networks** (UNNs) – a framework in which outputs are not computed by evaluating a composition of functions in a given order, but are rather obtained *implicitly* by minimizing an energy function which factors over a graph. For particular choices of factor potentials, UNNs subsume many existing architectures, including feedforward, recurrent, and self-attention neural networks, auto-encoders, and networks with implicit layers. When coupled with a coordinate descent algorithm to minimize the energy, the computation performed by an UNN is similar (but not equivalent) to a neural network sharing parameters across multiple identical layers. Since UNNs have no prescribed computation order, the exact same network can be used to predict any group of variables (outputs) given another group of variables (inputs), or vice-versa (*i.e.*, inputs from outputs) enabling new kinds of joint models. In sum, our contributions are:

- We present UNNs and show how they extend many existing neural architectures.
- We provide a coordinate descent inference algorithm, which, by an "unrolling lemma" (Lemma 1), can reuse current building blocks from feed-forward networks in a modular way.
- We develop and experiment with multiple factor graph architectures, tackling both structured and unstructured tasks, such as natural language parsing, image classification, and image prototype generation. We develop a new undirected attention mechanism and demonstrate its suitability for sequence completion.[1]

Notation. We denote vector values as a, matrix and tensor values as A, and abstract factor graph variables

[1]Instituto de Telecomunicações, Instituto Superior Técnico, Lisbon, Portugal [2]Language Technology Lab, University of Amsterdam, The Netherlands [3]LUMLIS, Lisbon ELLIS Unit, Portugal [4]Unbabel, Lisbon, Portugal. Correspondence to: Tsvetomila Mihaylova <tsvetomila.mihaylova@tecnico.ulisboa.pt>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

[1]The source code is on https://github.com/deep-spin/unn.

as A. The Frobenius inner product of two tensors with matching dimensions $A, B \in \mathbb{R}^{d_1 \times \ldots \times d_n}$ is $\langle A, B \rangle \coloneqq \sum_{i_1=1}^{d_1} \cdots \sum_{i_n=1}^{d_n} a_{i_1 \ldots i_n} b_{i_1 \ldots i_n}$. For vectors $\langle a, b \rangle = a^\top b$ and for matrices $\langle A, B \rangle = \mathrm{Tr}(A^\top B)$. The Frobenius norm is $\|A\| \coloneqq \sqrt{\langle A, A \rangle}$. Given two tensors $A \in \mathbb{R}^{c_1, \ldots, c_m}$, $B \in \mathbb{R}^{d_1, \ldots, d_n}$, their outer product is $(A \otimes B)_{i_1, \ldots, i_m, j_1, \ldots, j_n} = a_{i_1, \ldots, i_m} b_{j_1, \ldots, j_m}$. For vectors, $a \otimes b = ab^\top$. We denote the d-dimensional vector of ones as 1_d (or tensor, if d is a tuple.) The Fenchel conjugate of a function $\Psi : \mathbb{R}^d \to \mathbb{R}$ is $\Psi^*(t) \coloneqq \sup_{x \in \mathbb{R}^d} \langle x, t \rangle - \Psi(x)$. When Ψ is strictly convex, Ψ^* is differentiable, and $(\nabla \Psi^*)(t)$ is the unique maximizer $\mathrm{argmax}_{x \in \mathbb{R}^d} \langle x, t \rangle - \Psi(x)$. We denote the non-negative reals as $\mathbb{R}^d_+ \coloneqq \{x \in \mathbb{R}^d : x \geq 0\}$, and the $(d-1)$-dimensional simplex as $\triangle_d \coloneqq \{\alpha \in \mathbb{R}^d_+, \langle 1_d, \alpha \rangle = 1\}$. The Shannon entropy of a discrete distribution $y \in \triangle_d$ is $\mathcal{H}(y) \coloneqq -\sum_i y_i \log y_i$. The indicator function $\iota_{\mathcal{X}}$ is defined as $\iota_{\mathcal{X}}(x) \coloneqq 0$ if $x \in \mathcal{X}$, and $\iota_{\mathcal{X}}(x) \coloneqq +\infty$ otherwise.

2. Undirected Neural Networks

Let $\mathcal{G} = (V, F)$ be a factor graph, *i.e.*, a bipartite graph consisting of a set of variable nodes V and a set of factor nodes F, where each factor node $f \in F \subseteq 2^V$ is linked to a subset of variable nodes. Each variable node $\mathsf{X} \in V$ is associated with a representation vector $x \in \mathbb{R}^{d_\mathsf{X}}$. We define unary energies for each variable $E_\mathsf{X}(x)$, as well as higher-order energies $E_f(x_f)$, where x_f denotes the values of all variables linked to factor f. Then, an assignment defines a total energy function

$$E(x_1, \ldots, x_n) \coloneqq \sum_i E_{\mathsf{X}_i}(x_i) + \sum_f E_f(x_f). \quad (1)$$

For simple factor graphs where there is no ambiguity, we may refer to factors directly by the variables they link to. For instance, a simple fully-connected factor graph with only two variables X and Y is fully specified by $E(x, y) = E_\mathsf{X}(x) + E_\mathsf{Y}(y) + E_{\mathsf{XY}}(x, y)$.

The energy function in Eq. (1) induces preferences for certain configurations. For instance, a globally best configuration can be found by solving $\mathrm{argmin}_{x,y} E(x, y)$, while a best assignment for Y given a fixed value of X can be found by solving $\mathrm{argmin}_y E(x, y)$.[2] We may think of, or suggest using notation, that X is an *input* and Y is an *output*. However, intrinsically, factor graphs are not attached to a static notion of input and output, and instead can be used to infer any subset of variables given any other subset.

In our proposed framework of UNNs, we define the computation performed by a neural network using a factor graph, where each variable is a representation vector (*e.g.*, analogous to the output of a layer in a standard network.) We design the factor energy functions depending on the type of each variable and the desired relationships between them. Inference is performed by minimizing the joint energy with respect to all unobserved variables (*i.e.*, hidden and output values). For instance, to construct a supervised UNN, we may designate a particular variable as "input" X and another as "output" Y, alongside several hidden variables H_i, compute

$$\hat{y} = \arg\min_y \min_{h_1, \ldots, h_n} E(x, h_1, \ldots, h_n, y)\,, \quad (2)$$

and train by minimizing some loss $\ell(\hat{y}, y)$. However, UNNs are not restricted to the supervised setting or to a single input and output, as we shall explore.

While this framework is very flexible, Eq. (2) is a non-trivial optimization problem. Therefore, we focus on a class of energy functions that renders inference easier:

$$\begin{aligned} E_{\mathsf{X}_i}(x_i) &= -\langle b_{\mathsf{X}_i}, x_i \rangle + \Psi_{\mathsf{X}_i}(x_i)\,, \\ E_f(x_f) &= -\left\langle W_f, \bigotimes_{\mathsf{X}_j \in f} x_j \right\rangle, \end{aligned} \quad (3)$$

where each Ψ_{X_i} is a strictly convex regularizer, $\otimes$ denotes the outer product, and W_f is a parameter tensor of matching dimension. For pairwise factors $f = \{\mathsf{X}, \mathsf{Y}\}$, the factor energy is bilinear and can be written simply as $E_{\mathsf{XY}}(x, y) = -x^\top W y$. In factor graphs of the form given in Eq. (3), the energy is convex in each variable separately, and block-wise minimization has a closed-form expression involving the Fenchel conjugate of the regularizers. This suggests a block coordinate descent optimization strategy: given an order π, iteratively set:

$$x_{\pi_j} \leftarrow \operatorname*{argmin}_{x_{\pi_j}} E(x_1, \ldots, x_n)\,. \quad (4)$$

This block coordinate descent algorithm is guaranteed to decrease energy at every iteration and, for energies as in Eq. (3), to converge to a Nash equilibrium (Xu & Yin, 2013, Thm. 2.3); in addition, it is conveniently learning-rate free. For training, to tackle the bi-level optimization problem, we unroll the coordinate descent iterations, and minimize some loss with standard deep learning optimizers, like stochastic gradient or Adam (Kingma & Ba, 2015).

The following result, proved in Appendix A, shows that the coordinate descent algorithm (Eq. (4)) for UNNs with multilinear factor energies (Eq. (3)), corresponds to standard forward propagation on an unrolled neural network.

Lemma 1 (Unrolling Lemma)**.** *Let $\mathcal{G} = (V, F)$ be a pairwise factor graph, with multilinear higher-order energies and strictly convex unary energies, as in Eq. (3). Then, the coordinate descent updates (4) result in a chain of affine transformations (i.e., pre-activations) followed by non-linear activations, applied in the order π, yielding a traditional computation graph.*

[2] In this work, we only consider deterministic inference in factor graphs. Probabilistic models are a promising extension.

Table 1. Examples of regularizers $\Psi(h)$ corresponding to some common activation functions, where $\phi(t) = t \log t$.

$\Psi(h)$	$(\nabla\Psi^*)(t)$
$\frac{1}{2}\|h\|^2$	t
$\frac{1}{2}\|h\|^2 + \iota_{\mathbb{R}_+}(h)$	$\mathrm{relu}(t)$
$\sum_j \big(\phi(h_j) + \phi(1-h_j)\big) + \iota_{[0,1]^d}(h)$	$\mathrm{sigmoid}(t)$
$\sum_j \left(\phi\left(\frac{1+h_j}{2}\right) + \phi\left(\frac{1-h_j}{2}\right)\right) + \iota_{[-1,1]^d}(h)$	$\tanh(t)$
$-\mathcal{H}(h) + \iota_{\Delta}(h)$	$\mathrm{softmax}(t)$

We show next an undirected construction inspired by (directed) multi-layer perceptrons.

Single pairwise factor The simplest possible UNN has a pairwise factor connecting two variables X, H. We may interpret X as an input, and H either as an output (in supervised learning) or a hidden representation in unsupervised learning (Fig. 2(a)). Bilinear-convex energies as in Eq. (3) yield:

$$\begin{aligned} E_{\mathsf{XH}}(x,h) &= -\langle h, Wx\rangle\,, \\ E_{\mathsf{X}}(x) &= -\langle x, b_{\mathsf{X}}\rangle + \Psi_{\mathsf{X}}(x)\,, \\ E_{\mathsf{H}}(h) &= -\langle h, b_{\mathsf{H}}\rangle + \Psi_{\mathsf{H}}(h)\,. \end{aligned} \tag{5}$$

This resembles a Boltzmann machine with continuous variables (Smolensky, 1986; Hinton, 2007; Welling et al., 2004); however, in contrast to Boltzmann machines, we do not model joint probability distributions, but instead use factor graphs as representations of deterministic computation, more akin to computation graphs.

Given x, the updated h minimizing the energy is:

$$\begin{aligned} h_\star &= \operatorname*{argmin}_{h\in\mathbb{R}^M} -(Wx + b_{\mathsf{H}})^\top h + \Psi_{\mathsf{H}}(h) \\ &= (\nabla\Psi^*_{\mathsf{H}})(Wx + b_{\mathsf{H}}), \end{aligned} \tag{6}$$

where $\nabla\Psi^*_{\mathsf{H}}$ is the gradient of the conjugate function of Ψ_{H}. Analogously, the update for X given H is:

$$x_\star = (\nabla\Psi^*_{\mathsf{X}})(W^\top h + b_{\mathsf{X}})\,. \tag{7}$$

Other than the connection to Boltzmann machines, one round of updates of H and X in this order also describe the computation of an auto-encoder with shared encoder/decoder weights.

Table 1 shows examples of regularizers Ψ and their corresponding $\nabla\Psi^*$, In practice, we never evaluate Ψ or Ψ^*, but only $\nabla\Psi^*$, which we choose among commonly-used neural network activation functions like $\tanh$, relu, and $\mathrm{softmax}$.

Undirected multi-layer perceptron (MLP) Fig. 2(b) shows the factor graph for an undirected MLP analogous to a feed-forward one with input X, output Y, and a single hidden layer H. As in Eq. (3), we have bilinear pairwise factors

$$E_{\mathsf{XH}}(x,h) = -\langle h, Wx\rangle\,, \quad E_{\mathsf{HY}}(h,y) = -\langle y, Vh\rangle\,, \tag{8}$$

and linear-plus-convex unaries $E_{\mathsf{Z}}(x) = -\langle x, b_{\mathsf{Z}}\rangle + \Psi_{\mathsf{Z}}(x)$ for $\mathsf{Z} \in \{\mathsf{X}, \mathsf{H}, \mathsf{Y}\}$. If x is observed (fixed), coordinate-wise inference updates take the form:

$$\begin{aligned} h_\star &= (\nabla\Psi^*_{\mathsf{H}})(Wx + V^\top y + b_{\mathsf{H}})\,, \\ y_\star &= (\nabla\Psi^*_{\mathsf{Y}})(Vh + b_{\mathsf{Y}})\,. \end{aligned} \tag{9}$$

Note that E_{X} does not change anything if X is always observed. The entire algorithm can be unrolled into a directed computation graph, leading to a deep neural network with shared parameters (Fig. 1).

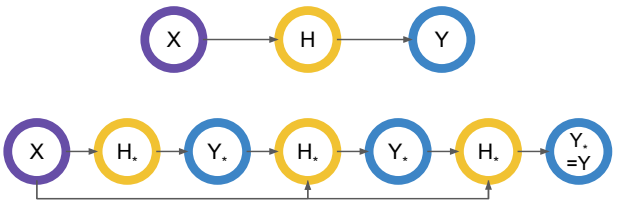

Figure 1. Unrolling the computation graph for undirected MLP with a single hidden layer. Top: MLP with one hidden layer. Bottom: Unrolled graph for UNN with $k = 3$ iterations.

The regularizers Ψ_{H} and Ψ_{Y} may be selected based on what we want $\nabla\Psi^*$ to look like, and the constraints or domains of the variables. For instance, if Y is a multiclass classification output, we may pick Ψ_{Y} such that $\nabla\Psi^*_{\mathsf{Y}}$ be the softmax function, and Ψ_{H} to induce a relu nonlinearity. Initializing $y^{(0)} = 0$ and performing a single iteration of updating H followed by Y results in a standard MLP with a single hidden layer (see also Fig. 1). However, the UNN point of view lets us decrease energy further by performing multiple iterations, as well as use the same model to infer any variables given any other ones, *e.g.*, to predict x from y instead of y from x. We demonstrate this power in Sections 3 to 5.

The above constructions provide a flexible framework for defining UNNs. However, UNNs are more general and cover more popular deep learning architectures. The following constructions illustrate some such connections.

Feed-forward neural networks Any directed computation graph associated with a neural network is a particular case of an UNN. We illustrate this for a simple feed-forward network, which chains the functions $h = f(x)$ and $y = g(h)$, where $x \in \mathbb{R}^m$, $h \in \mathbb{R}^d$, $y \in \mathbb{R}^n$ are input, hidden, and output variables, and $f : \mathbb{R}^m \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}^n$ are the functions associated to each layer (*e.g.*, an affine transformation followed by a non-linearity).

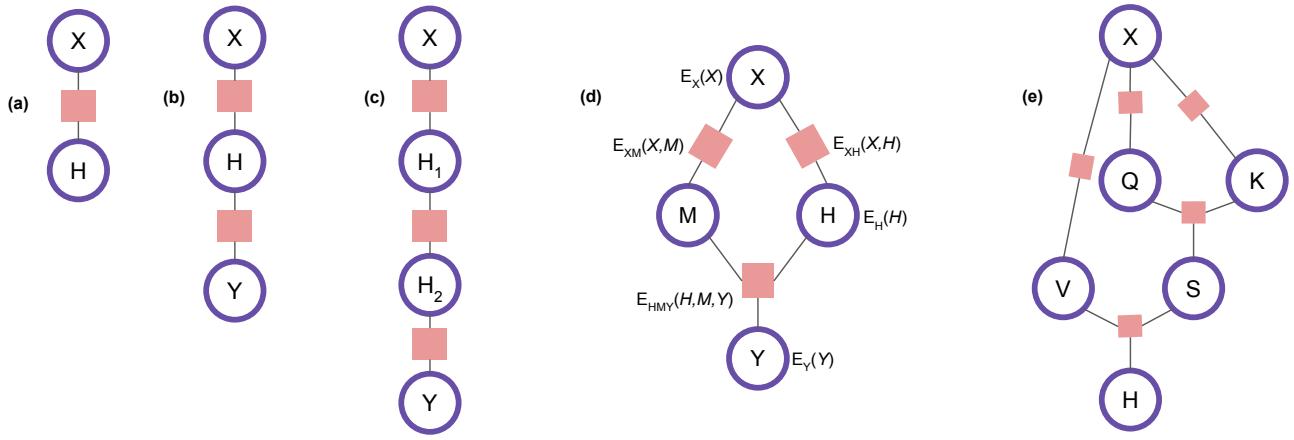

Figure 2. Factor graphs for: (a) network without intermediate layers, (b-c) undirected MLPs with one or two layers, (d) undirected biaffine dependency parser, (e) undirected self-attention. Energy labels ommitted for brevity with the exception of (d).

This factor graph is illustrated in Fig. 2(b). To see this, let $V = \{\mathsf{X}, \mathsf{H}, \mathsf{Y}\}$ and $F = \{\mathsf{XH}, \mathsf{HY}\}$ and define the energies as follows. Let $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ be any distance function satisfying $d(a, b) \geq 0$, with equality iff $a = b$; for example $d(a, b) = \|a - b\|$. Let all the unary energies be zero and define the factor energies $E_{\mathsf{XH}}(x, h) = d(h, f(x))$ and $E_{\mathsf{HY}}(h, y) = d(y, g(h))$. Then the total energy satisfies $E(x, h, y) \geq 0$, with equality iff the equations $h = f(x)$ and $y = g(h)$ are satisfied – therefore, the energy is minimized (and becomes zero) when $y = g(f(x))$, matching the corresponding directed computation graph. This can be generalized for an arbitrary deterministic neural network. This way, we can form UNNs that are partly directed, partly undirected, as the whole is still an UNN. We do this in our experiments in Section 5, where we fine-tune a pretrained BERT model appended to a UNN for parsing.

Implicit layers UNNs include networks with implicit layers (Duvenaud et al., 2020), a paradigm which, in contrast with feed-forward layers, does not specify how to compute the output from the input, but rather specifies conditions that the output layer should specify, often related to minimizing some function, *e.g.*, computing a layer h_{i+1} given a previous layer h_i involves solving a possibly difficult problem $\operatorname{argmin}_h f(h_i, h)$. Such a function f can be directly interpreted as an energy in our model, *i.e.*, $E_{\mathsf{H}_i\mathsf{H}_{i+1}} = f$.

3. Image Classification and Visualization

Unlike feed-forward networks, where the processing order is hard-coded from inputs X to outputs Y, UNNs support processing in any direction. We can thus use the same trained network both for classification as well as for generating prototypical examples for each class. We demonstrate this on the MNIST dataset of handwritten digits (Deng, 2012), showcasing convolutional UNN layers.

The architecture is shown in Fig. 2(c) and has the following variables: the image X, the class label Y and two hidden layer variables $\mathsf{H}_{\{1,2\}}$. Unlike the previous examples, the two pairwise energies involving the image and the hidden layers are convolutional, *i.e.*, linear layers with internal structure:

$$\begin{aligned} E_{\mathsf{XH}_1}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle\,, \\ E_{\mathsf{H}_1\mathsf{H}_2}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle\,, \end{aligned} \tag{10}$$

where $\mathcal{C}_{1,2}$ are linear cross-correlation operators with stride two and filter weights $W_1 \in \mathbb{R}^{32\times1\times6\times6}$ and $W_2 \in \mathbb{R}^{64\times32\times4\times4}$. The last layer is fully connected:

$$E_{\mathsf{H}_2\mathsf{Y}}(H_2, y) = -\langle V,\ y \otimes H_2\rangle\,. \tag{11}$$

The unary energies for the hidden layers contain standard (convolutional) bias term along with the binary entropy term $\Psi_{\tanh}(H)$ such that $\nabla\Psi^*_{\tanh}(t) = \tanh(t)$ (see Table 1). Note that X is no longer a constant when generating X given Y, therefore it is important to specify the unary energy E_{X}. Since pixel values are bounded, we set $\mathsf{E}_X(x) = \Psi_{\tanh}(x)$. Initializing H_1, H_2, and y with zeroes and updating them once blockwise in this order yields exactly a feed-forward convolutional neural network. As our network is undirected, we may propagate information in multiple passes, proceeding in the order $\mathsf{H}_1, \mathsf{H}_2, \mathsf{Y}, \mathsf{H}_2$ iteratively. The update for H_1 involves a convolution of X and a deconvolution of H_2; we defer the other updates to Appendix F:

$$(H_1) \overset{+}{\underset{\star}{=}} \tanh(\mathcal{C}_1(X; W_1) + \mathcal{C}_2^\top(H_2; W_2) + b_1 \otimes 1_{d_1})\,, \tag{12}$$

where $b_1 \in \mathbb{R}^{32}$ are biases for each filter, and $d_1 = 12 \times 12$ is the convolved image size. To generate digit prototype X from a given class $c \in \{1, \ldots, 10\}$, we may set $y = e_c$, initialize the other variables at zero (including X), and solve $\hat{X} = \operatorname{argmin}_X \min_{H_{1,2}} E(X, H_1, H_2, y)$ by coordinate descent in the reverse order $\mathsf{H}_2, \mathsf{H}_1, \mathsf{X}, \mathsf{H}_1$ iteratively.

Table 2. MNIST accuracy with convolutional UNN.

ITERATIONS	ACCURACY
$k = 1, \gamma = 0$ (BASELINE)	98.80
$k = 1$	98.75
$k = 2$	98.74
$k = 3$	**98.83**
$k = 4$	98.78
$k = 5$	98.69

We train our model jointly for both tasks. For each labeled pair (X, y) from the training data, we first predict $\hat{y}$ given X, then separately predict $\hat{X}$ given y. The incurred loss is a weighted combination $\ell(x, y) = \ell_f(y, \hat{y}) + \gamma \ell_b(X, \hat{X})$, where ℓ_f is a 10-class cross-entropy loss, and ℓ_b is a binary cross-entropy loss averaged over all 28×28 pixels of the image. We use $\gamma = .1$ and an Adam learning rate of .0005.

The classification results are shown in Table 2. The model is able to achieve high classification accuracy, and multiple iterations lead to a slight improvement. This result suggests the reconstruction loss for X can also be seen as a regularizer, as the same model weights are used in both directions. The more interesting impact of multiple energy updates is the image prototype generation. In Fig. 3 we show the generated digit prototypes after several iterations of energy minimization, as well as for models with a single iteration. The networks trained as UNNs produce recognizable digits, and in particular the model with more iterations learns to use the additional computation to produce clearer pictures. As for the baseline, we may interpret it as an UNN and apply the same process to extract prototypes, but this does not result in meaningful digits (Fig. 3c). Note that our model is not a generative model – in that experiment, we are not sampling an image according to a probability distribution, rather we are using energy minimization deterministically to pick a prototype of a digit given its class.

Comparison of UNN to Unconstrained Model As per Fig. 1, an unrolled UNN can be seen as a feed-forward network with a specific architecture and with weight tying. To confirm the benefit of the UNN framework, we compare against an unconstrained model, *i.e.*, with the same architecture but separate, untied weights for each unrolled layer. We use as a base the model described in forward-only mode and train a model with 2 to 5 layers with different weights instead of shared weights as in the case with the UNN. Depending on the number of layers, we cut the number of parameters in each layer, in order to obtain models with the same number of parameters as the UNN for fair comparison. The results from the experiment are described in Table 3.

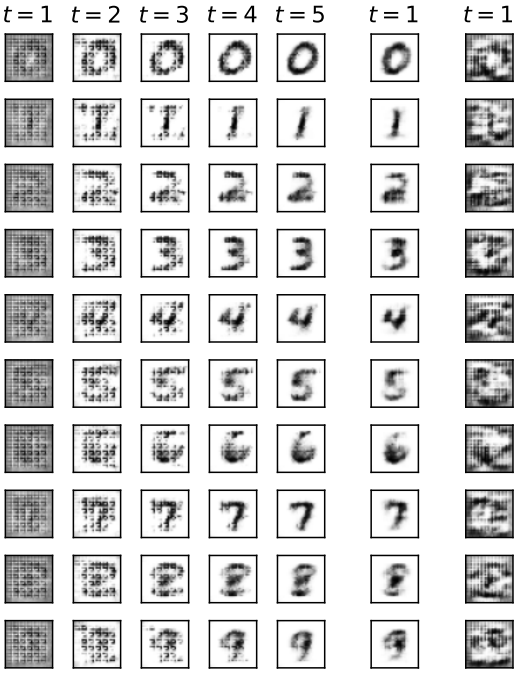

Figure 3. Digit prototypes generated by convolutional UNN. (a) best UNN ($k = 5, \gamma = .1$), (b) single iteration UNN ($k = 1, \gamma = .1$), (c): standard convnet ($k = 1, \gamma = 0$).

Table 3. Comparison of UNN with an unconstrained model with the same number of layers as the UNN iterations. The number of parameters of the UNN and the unconstrained model are roughly the same.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
ACC.	98.80	98.76	98.45	98.32	97.39
# PARAMS	50026	51220	51651	53608	51750

4. Undirected Attention Mechanism

Attention (Bahdanau et al., 2014; Vaswani et al., 2017) is a key component that enables neural networks to handle variable-length sequences as input. In this section, we propose an undirected attention mechanism (Fig. 2(e)). We demonstrate this model on the task of completing missing values in a sequence of dynamic length n, with the variable X serving as both input and output, taking values $X \in \mathbb{R}^{d \times n}$, queries, keys and values taking values $Q, K, V \in \mathbb{R}^{n \times d}$, and attention weights $S \in (\triangle_n)^n$, where d is a fixed hidden layer size. Finally, H is an induced latent sequence representation, with values $H \in \mathbb{R}^{n \times d}$. The only trainable parameters are $W_\mathsf{Q}, W_\mathsf{K}, W_\mathsf{V} \in \mathbb{R}^{d \times d}$, and the input embeddings. We model scaled dot-product attention given with $\mathrm{softmax}(d^{-\frac{1}{2}} QK^\top)V$. For all variables except S, we

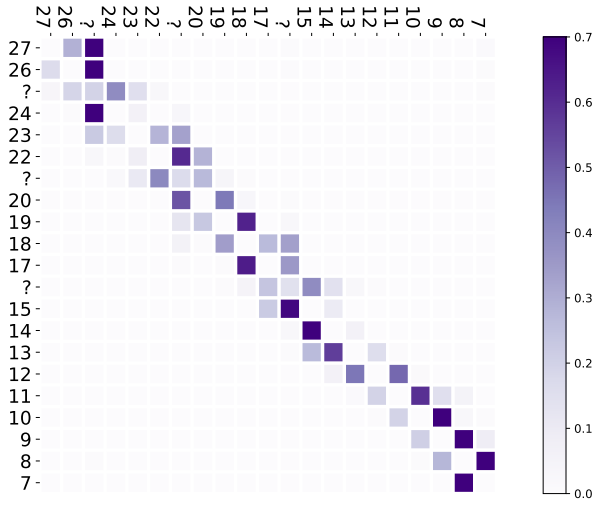

Figure 4. Undirected self-attention (one "forward-backward" pass) identifies an off-diagonal pattern, allowing generalization.

set $E_\cdot(\cdot) = \frac{1}{2}\|\cdot\|^2$. For the attention weights, we use $E_{\mathsf{S}}(S) = -\sqrt{d}\sum_{i=1}^{n}\mathcal{H}(S_i)$. The higher-order energies are:

$$\begin{aligned} E_{\mathsf{XQ}}(X,Q) &= -\langle Q, W_{\mathsf{Q}}(X+P)\rangle\,, \\ E_{\mathsf{XK}}(X,K) &= -\langle K, W_{\mathsf{K}}(X+P)\rangle\,, \\ E_{\mathsf{XV}}(X,V) &= -\langle V, W_{\mathsf{V}}(X+P)\rangle\,, \\ E_{\mathsf{QKS}}(Q,K,S) &= -\langle S, QK^\top\rangle\,, \\ E_{\mathsf{VSH}}(V,S,H) &= -\langle H, SV\rangle\,, \end{aligned} \tag{13}$$

where P is a matrix of sine and cosine positional embeddings of same dimensions as X (Vaswani et al., 2017).

Minimizing the energy yields the blockwise updates:

$$\begin{aligned} Q_\star &= W_{\mathsf{Z}}(X+P) + SK\,, \\ K_\star &= W_{\mathsf{K}}(X+P) + S^\top Q\,, \\ V_\star &= W_{\mathsf{V}}(X+P) + S^\top H\,, \\ S_\star &= \operatorname{softmax}\left(d^{-1/2}(QK^\top + VH^\top)\right)\,, \\ H_\star &= SV\,, \\ \bar{X}_\star &= \bar{V}W_{\mathsf{V}} + \bar{Q}W_{\mathsf{Q}} + \bar{K}W_{\mathsf{K}}, \end{aligned} \tag{14}$$

where $\bar{X}$ denotes only the rows of X corresponding to the masked (missing) entries.

Provided zero initialization, updating in the order $(\mathsf{V}/\mathsf{Q}/\mathsf{K}), \mathsf{S}, \mathsf{H}$ corresponds exactly to a forward pass in a standard self-attention. However, in an UNN, our expressions allow backward propagation back toward X, as well as iterating to an equilibrium. To ensure that one round of updates propagates information through all the variables, we employ the "forward-backward" order $\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}, \mathsf{S}, \mathsf{V}, \mathsf{K}, \mathsf{Q}, \bar{\mathsf{X}}$.

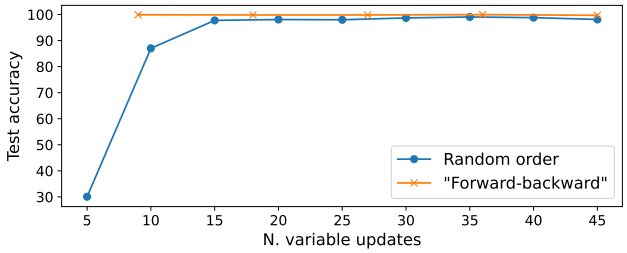

Figure 5. Comparison of the test accuracy) for models with random and "forward-backward" order of variable updates. Markers indicate one full iteration.

We evaluate the performance of the undirected attention with a toy task of sequence completion. We generate a toy dataset of numerical sequences between 1 and 64, of length at least 8 and at most 25, in either ascending or descending consecutive order. We mask out up to 10% of the tokens and generate all possible sequences, splitting them into training and test sets with around 706K and 78K instances. Undirected self-attention is applied to the input sequence. Note that because of the flexibility of the architecture, the update of the input variable X does not differ from the updates of the remaining variables, because each variable update corresponds to one step of coordinate descent. The model incurs a cross-entropy loss for the missing elements of the sequence and the parameters are updated using Adam with learning rate 10^{-4}. The hidden dimension is $d = 256$, and gradients with magnitude beyond 10 are clipped.

Undirected attention is able to solve this task, reaching over 99.8% test accuracy, confirming viablity. Figure 4 shows the attention weights for a model trained with $k = 1$. More attention plots are shown in Appendix E.

Impact of update order. The "forward-backward" order is not the only possible order of updates in an UNN. In Fig. 5 we compare its performance against a randomized coordinate descent strategy, where at each round we pick a permutation of $\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}$. In both cases, $\bar{\mathsf{X}}$ is updated at the end of each iteration. (Note that the "forward-backward" order performs almost twice the number of updates per iteration.) While the "forward-backward" order performs well even with a single pass, given sufficient iterations, random order updates can reach the same performance, suggesting that hand-crafting a meaningful order helps but is not necessary. Further analysis is reported in Appendix B.

5. Structured UNNs for Dependency Parsing

The concept of UNN can be applied to structured tasks – all we need to do is to define *structured factors*, as shown next.

We experiment with a challenging structured prediction task from natural language processing: unlabeled, non-projective

dependency parsing (Kübler et al., 2009). Given a sentence with n words, represented as a matrix $X \in \mathbb{R}^{r \times n}$ (where r is the embedding size), the goal is to predict the syntactic relations as a *dependency tree*, *i.e.*, a spanning tree which has the words as nodes. The output can be represented as a binary matrix $Y \in \mathbb{R}^{n \times n}$, where the $(i, j)^{\text{th}}$ entry indicates if there is a directed arc $i \to j$ connecting the i^{th} word (the *head*) and j^{th} word (the *modifier*). Fig. 6 shows examples of dependency trees produced by this model. We use a probabilistic model where the output Y can more broadly represent a probability distribution over trees, represented by the matrix of arc marginals induced by this distribution (illustrated in Appendix E.)

Biaffine parsing. A successful model for dependency parsing is the biaffine one (Dozat & Manning, 2016; Kiperwasser & Goldberg, 2016). This model first computes head representations $H \in \mathbb{R}^{d \times n}$ and modifier representations $M \in \mathbb{R}^{d \times n}$, via a neural network that takes X as input – here, d denotes the hidden dimension of these representations. Then, it computes a score matrix as $Z = H^\top V M \in \mathbb{R}^{n \times n}$, where $V \in \mathbb{R}^{d \times d}$ is a parameter matrix. Entries of Z can be interpreted as scores for each candidate arc. From Z, the best tree is found by the Chu-Liu-Edmonds maximum spanning arborescence algorithm (Chu & Liu, 1965; Edmonds, 1967), and probabilities and marginals by the matrix-tree theorem (Koo et al., 2007; Smith & Smith, 2007; McDonald & Satta, 2007; Kirchhoff, 1847).

UNN for parsing. We now construct an UNN with the same building blocks as this biaffine model, leading to the factor graph in Fig. 2(d). The variable nodes are $\{\mathsf{X}, \mathsf{H}, \mathsf{M}, \mathsf{Y}\}$, and the factors are $\{\mathsf{XH}, \mathsf{XM}, \mathsf{HMY}\}$. Given parameter weight matrices $V, W_\mathsf{H}, W_\mathsf{M} \in \mathbb{R}^{d \times d}$ and biases $b_\mathsf{H}, b_\mathsf{M} \in \mathbb{R}^d$, we use multilinear factor energies as follows:

$$\begin{aligned} E_\mathsf{XH}(X, H) &= -\langle H, W_\mathsf{H} X \rangle \,, \\ E_\mathsf{XM}(X, M) &= -\langle M, W_\mathsf{M} X \rangle \,, \\ E_\mathsf{YHM}(Y, H, M) &= -\langle Y, H^\top V M \rangle \,. \end{aligned} \tag{15}$$

For H and M, we use the ReLU regularizer,

$$\begin{aligned} E_\mathsf{H}(H) &= -\langle b_\mathsf{H} \otimes 1_n, H \rangle + \frac{1}{2}\|H\|^2 + \iota_{\geq 0}(H) \\ E_\mathsf{M}(M) &= -\langle b_\mathsf{M} \otimes 1_n, M \rangle + \frac{1}{2}\|M\|^2 + \iota_{\geq 0}(M) \,. \end{aligned} \tag{16}$$

For Y, however, we employ a structured entropy regularizer:

$$E_\mathsf{Y}(Y) = -\mathcal{H}_\mathcal{M}(Y) + \iota_\mathcal{M}(Y) \,, \tag{17}$$

where $\mathcal{M} = \operatorname{conv}(\mathcal{Y})$ is the marginal polytope (Wainwright & Jordan, 2008; Martins et al., 2009), the convex hull of the adjacency matrices of all valid non-projective dependency trees (Fig. 9), and $\mathcal{H}_\mathcal{M}(Y)$ is the maximal entropy over all distribution over trees with arc marginals Y:

$$\mathcal{H}_\mathcal{M}(Y) \coloneqq \max_{\alpha \in \triangle_{|\mathcal{Y}|}} \mathcal{H}(\alpha) \text{ s.t. } \mathbb{E}_{A \sim \alpha}[A] = Y \,. \tag{18}$$

Derivation of block coordinate descent updates. To minimize the total energy, we iterate between updating H, M and Y k times, similar to the unstructured case.

The updates for the heads and modifiers work out to:

$$\begin{aligned} H_\star &= \operatorname{relu}(W_\mathsf{H} X + b_\mathsf{H} \otimes 1_n + V M Y^\top) \,, \\ M_\star &= \operatorname{relu}(W_\mathsf{M} X + b_\mathsf{M} \otimes 1_n + V^\top H Y) \,. \end{aligned} \tag{19}$$

For Y, however, we must solve the problem

$$Y_\star = \operatorname*{argmin}_{Y \in \mathcal{M}} -\langle Y, H^\top V M \rangle - \mathcal{H}_\mathcal{M}(Y) \,. \tag{20}$$

This combinatorial optimization problem corresponds to *marginal inference* (Wainwright & Jordan, 2008), a well-studied computational problem in structured prediction that appears in all probabilistic models. While generally intractable, for non-projective dependency parsing it may be computed in time $\mathcal{O}(n^3)$ via the aforementioned matrix-tree theorem, the same algorithm required to compute the structured likelihood loss.[3]

With zero initialization, the first iteration yields the same hidden representations and output as the biaffine model, assuming the updates are performed in the order described. The extra terms involving $VMY^\top$ and $V^\top HY$ enable the current prediction for Y to influence neighboring words, which leads to a more expressive model overall.

Experiments. We test the architecture on several datasets from Universal Dependencies 2.7 (Zeman et al., 2020), covering different language families and dataset size: Afrikaans (AfriBooms), Arabic (PADT), Czech (PDT), English (Partut), Hungarian (Szeged), Italian (ISDT), Persian (Seraji), Portuguese (Bosque), Swedish (Talbanken), and Telugu (MTG). Performance is measured by three metrics:

- Unlabeled attachment score (UAS): a fine-grained, arc-level accuracy metric.
- Modifier list accuracy: the percentage of head words for which *all* modifiers were correctly predicted. For example, in Fig. 6, the baseline correctly predicts all modifiers for the words *perspectiva, abre, longos*, but not for the words *aplicações, prazo*.
- Exact match: the percentage of sentences for which the entire parse tree is correct; the harshest of the metrics.

[3] During training, the matrix-tree theorem can be invoked only once to compute both the update to Y as well as the gradient of the loss, since $\nabla \log p(\mathsf{Y} = Y_\text{true}) = Y_\text{true} - \hat{Y}$.

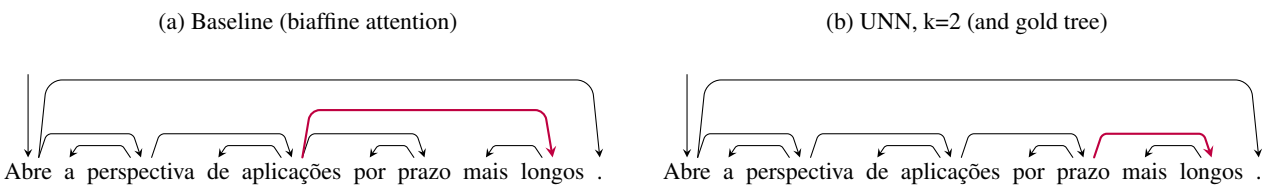

Figure 6. Examples of dependency trees produced by the parsing model for a sentence in Portuguese. The baseline model (a) erroneously assigns the noun *aplicações* as the syntactic head of the adjective *longos*. The UNN with $k = 2$ iterations (b) matches the gold parse tree for this sentence, eventually benefiting from the structural information propagated back from the node Y after the first iteration.

The latter, coarser measures can give more information whether the model is able to learn global relations, not just how to make local predictions correctly (*i.e.*, when only prediction of the arcs is evaluated).

Our architecture is as follows: First, we pass the sentence through a BERT model (BERT-BASE-MULTILINGUAL-CASED, fine-tuned during training, as directed networks can be added as components to UNNs, as mentioned in Section 2) and get the word representations of the last layer. These representations are the input x in the UNN model. Then, we apply the parsing model described in this section. The baseline ($k = 1$) corresponds to a biaffine parser using BERT features. The learning rate for each language is chosen via grid search for highest UAS on the validation set for the baseline model. We searched over the values $\{0.1, 0.5, 1, 5, 10\} \times 10^{-5}$. In the experiments, we use 10^{-5} for Italian and 5×10^{-5} for the other languages. We employ dropout regularization, using the same dropout mask for each variable throughout the inner coordinate descent iterations, so that dropped values do not leak.

The results from the parsing experiments are displayed in Table 4. The numbers in the table show results on the test set for the highest validation accuracy epoch. We see that some of the languages seem to benefit from the iterative procedure of UNNs (CS, HU, TE), while others do not (EN, AF), and little difference is observed in the remaining languages. In general, the baseline ($k = 1$) seems to attain higher accuracies in UAS (individual arcs), but most of the languages have overall more accurate structures (as measured by modifier list accuracy and by exact match) for $k > 1$. Fig. 6 illustrates with one example in Portuguese.

6. Related Work

Besides the models mentioned in Section 2 which may be regarded as particular cases of UNNs, other models and architectures, next described, bear relation to our work.

Probabilistic modeling of joint distributions Our work draws inspiration from the well-known Boltzmann machines and Hopfield networks (Ackley et al., 1985; Smolensky, 1986; Hopfield, 1984). We consider deterministic networks whose desired configurations minimize an energy function which decomposes as a factor graph. In contrast, many other works have studied probabilistic energy-based models (EBM) defined as Gibbs distributions, as well as efficient methods to learn those distributions and to sample from them (Ngiam et al., 2011; Du & Mordatch, 2019). Similar to how our convolutional UNN can be used for multiple purposes in Section 3, Grathwohl et al. (2020a) reinterprets standard discriminative classifiers $p(\mathsf{Y}|\mathsf{X})$ as an EBM of a joint distribution $p(\mathsf{X}, \mathsf{Y})$. Training stochastic EBMs requires Monte Carlo sampling or auxiliary networks (Grathwohl et al., 2020b); in contrast, our deterministic UNNs, more aligned conceptually with deterministic EBMs (LeCun et al., 2006), eschew probabilistic modeling in favor of more direct training. Moreover, our UNN architectures closely parallel feed-forward networks and reuse their building blocks, uniquely bridging the two paradigms.

Structured Prediction Energy Networks (SPENs) We saw in Section 5 that UNNs can handle structured outputs. An alternative framework for expressive structured prediction is given by SPENs (Belanger & McCallum, 2016). Most SPEN inference strategies require gradient descent, often with higher-order gradients for learning (Belanger et al., 2017), or training separate inference networks (Tu et al., 2020). UNNs in contrast, are well suited for coordinate descent inference: a learning-rate free algorithm with updates based on existing neural network building blocks. An undirected variant of SPENs would be similar to the MLP factor graph in Fig. 2(b), but with X and Y connected to a joint, higher-order factor, rather than via a chain $\mathsf{X} - \mathsf{H} - \mathsf{Y}$.

Universal transformers and Hopfield networks In Section 4 we show how we can implement self-attention with UNNs. Performing multiple energy updates resembles – but is different from – transformers (Vaswani et al., 2017) with shared weights between the layers. Our perspective of minimizing UNNs with coordinate descent using a fixed schedule and this unrolling is similar (but not exactly the same due to the skip connections) to having deeper neural networks which shared parameters for each layer. Such an architecture is the Universal Transformer (Dehghani et al., 2018), which applies a recurrent neural network to the trans-

Table 4. Structured UNN parsing results. Columns show the number of UNN iterations. The best result per row is rendered in bold.

LANGUAGE	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
UNLABELED ATTACHMENT SCORE					
AF	**89.09**	88.98	88.40	87.77	88.46
AR	**85.62**	84.94	84.22	83.69	83.63
CS	93.79	**93.83**	93.82	93.60	93.77
EN	91.96	91.86	91.09	**91.99**	91.51
FA	**83.41**	83.27	82.95	83.37	83.27
HU	85.11	**85.77**	84.47	85.13	84.09
IT	**94.76**	94.43	94.35	94.59	94.45
PT	96.99	97.00	96.83	**97.06**	96.90
SW	**91.42**	90.92	91.30	91.08	90.98
TE	89.72	89.72	**90.00**	88.45	87.75
MODIFIER LIST ACCURACY					
AF	**74.10**	72.60	72.90	71.78	72.01
AR	**70.44**	69.29	68.41	68.08	68.19
CS	84.46	84.82	**84.93**	84.12	84.49
EN	79.08	77.73	75.20	78.90	**79.44**
FA	64.80	**66.75**	65.28	66.67	65.85
HU	64.13	**66.07**	64.37	62.91	64.13
IT	**85.32**	83.59	83.71	83.94	84.05
PT	90.10	**90.69**	90.39	90.66	90.49
SW	**79.07**	78.37	78.52	78.60	78.24
TE	72.87	72.87	**73.68**	66.80	65.99
EXACT MATCH					
AF	**37.70**	33.88	34.43	33.88	32.79
AR	19.44	19.29	18.36	**19.91**	18.36
CS	59.17	60.76	**60.92**	59.42	59.84
EN	**48.59**	44.37	40.14	43.66	44.37
FA	21.52	22.15	22.78	**24.68**	23.42
HU	21.13	23.40	**24.15**	23.40	21.51
IT	**64.93**	63.54	62.85	63.89	64.24
PT	73.24	**74.86**	73.89	74.43	74.11
SW	**54.62**	52.38	54.13	53.94	52.67
TE	75.69	77.08	**79.17**	71.53	70.14

former encoder and decoder. Recent work (Ramsauer et al., 2020) shows that the self-attention layers of transformers can be regarded as the update rule of a Hopfield network with continuous states (Hopfield, 1984). This leads to a "modern Hopfield network" with continuous states and an update rule which ensures global convergence to stationary points of the energy (local minima or saddle points). Like that model, UNNs also seek local minima of an energy function, albeit with a different goal.

Deep models as graphical model inference. This line of work defines neural computation via approximate inference in graphical models. Domke (2012) derives backpropagating versions of gradient descent, heavy-ball and LBFGS. They require as input only routines to compute the gradient of the energy with respect to the domain and parameters. Domke (2012) studies learning with unrolled gradient-based inference in general energy models. UNNs, in contrast, allow efficient, learning rate free, block-coordinate optimization by design. An exciting line of work derives unrolled architectures from inference in specific generative models (Hershey et al., 2014; Li & Zemel, 2014; Lawson et al., 2019)—a powerful construction at the cost of more challenging optimization. The former is closest to our strategy, but by starting from probabilistic models the resulting updates are farther from contemporary deep learning (*e.g.*, convolutions, attention). In contrast, UNNs can reuse successful implementations, modular pretrained models, as well as structured factors, as we demonstrate in our parsing experiments. We believe that our UNN construction can shed new light over probabilistic inference models as well, uncovering deeper connections between the paradigms.

7. Conclusions and Future Work

We presented UNNs – a structured energy-based model which combines the power of factor graphs and neural networks. At inference time, the model energy is minimized with a coordinate descent algorithm, allowing reuse of existing building blocks in a modular way with guarantees of decreasing the energy at each step. We showed how the proposed UNNs subsume many existing architectures, conveniently combining supervised and unsupervised/self-supervised learning, as demonstrated on the three tasks.

We hope our first steps in this work will spark multiple directions of future work on undirected networks. One promising direction is on probabilistic UNNs with Gibbs sampling, which have the potential to bring our modular architectures to generative models. Another direction is to consider alternate training strategies for UNNs. Our strategy of converting UNNs to unrolled neural networks, enabled by Lemma 1, makes gradient-based training easy to implement, but alternate training strategies, perhaps based on equilibrium conditions or dual decomposition, hold promise. Promising directions could be using this framework for dealing with missing data or learning the joint probability distribution.

Acknowledgements

We would like to thank Mário Figueiredo, Caio Corro and the DeepSPIN team for helpful discussions. TM and AM are supported by the European Research Council (ERC StG DeepSPIN 758969) and by the Fundação para a Ciência e Tecnologia through contracts PTDC/CCI-INF/4703/2021 (PRELUNA) and UIDB/50008/2020. VN is partially supported by the Hybrid Intelligence Centre, a 10-year programme funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research (https://hybrid-intelligence-centre.nl/).

References

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A learning algorithm for boltzmann machines. *Cognitive science*, 9 (1):147–169, 1985.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bakır, G., Hofmann, T., Smola, A. J., Schölkopf, B., and Taskar, B. *Predicting structured data*. MIT press, 2007.

Belanger, D. and McCallum, A. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992. PMLR, 2016.

Belanger, D., Yang, B., and McCallum, A. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pp. 429–439. PMLR, 2017.

Chu, Y.-J. and Liu, T.-H. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. Universal transformers. In *International Conference on Learning Representations*, 2018.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Domke, J. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326. PMLR, 2012.

Dozat, T. and Manning, C. D. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

Du, Y.-l. and Mordatch, I. Implicit generation and modeling with energy based models. In *NeurIPS*, 2019.

Duvenaud, D., Kolter, J. Z., and Johnson, M. Deep implicit layers tutorial-neural odes, deep equilibirum models, and beyond. *Neural Information Processing Systems Tutorial*, 2020.

Edmonds, J. Optimum branchings. *J. Res. Nat. Bur. Stand.*, 71B:233–240, 1967.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., and Swersky, K. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*, 2020a.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., and Zemel, R. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *ICML*. PMLR, 2020b.

Hershey, J. R., Roux, J. L., and Weninger, F. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

Hinton, G. E. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10): 3088–3092, 1984.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kiperwasser, E. and Goldberg, Y. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016. doi: 10.1162/tacl_a_00101. URL https://www.aclweb.org/anthology/Q16-1023.

Kirchhoff, G. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

Koo, T., Globerson, A., Carreras Pérez, X., and Collins, M. Structured prediction models via the matrix-tree theorem. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 141–150, 2007.

Kübler, S., McDonald, R., and Nivre, J. Dependency parsing. *Synthesis lectures on human language technologies*, 1(1):1–127, 2009.

Lawson, J., Tucker, G., Dai, B., and Ranganath, R. Energy-inspired models: Learning with sampler-induced distributions. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/28659414dab9eca0219dd592b8136434-Paper.pdf.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Li, Y. and Zemel, R. S. Mean-field networks. *ArXiv*, abs/1410.5884, 2014.

Martins, A. F., Smith, N. A., and Xing, E. P. Polyhedral outer approximations with application to natural language parsing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 713–720, 2009.

McDonald, R. and Satta, G. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pp. 121–132, 2007.

Ngiam, J., Chen, Z., Koh, P. W., and Ng, A. Y. Learning deep energy models. In *ICML*, 2011.

Nowozin, S., Gehler, P. V., Jancsary, J., and Lampert, C. H. *Advanced Structured Prediction*. MIT Press, 2014.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Smith, D. A. and Smith, N. A. Probabilistic models of non-projective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 132–140, 2007.

Smith, N. A. Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274, 2011.

Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

Tu, L., Pang, R. Y., and Gimpel, K. Improving joint training of inference networks and structured prediction energy networks. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pp. 62–73, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.spnlp-1.8. URL https://aclanthology.org/2020.spnlp-1.8.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Wainwright, M. J. and Jordan, M. I. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

Welling, M., Rosen-Zvi, M., and Hinton, G. Exponential family harmoniums with an application to information retrieval. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, 2004.

Xu, Y. and Yin, W. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

Zeman, D., Nivre, J., Abrams, M., Ackermann, E., Aepli, N., Aghaei, H., Agić, Ž., Ahmadi, A., Ahrenberg, L., Ajede, C. K., Aleksandravičiūtė, G., Alfina, I., Antonsen, L., Aplonova, K., Aquino, A., Aragon, C., Aranzabe, M. J., Arnardóttir, Ĥ., Arutie, G., Arwidarasti, J. N., Asahara, M., Ateyah, L., Atmaca, F., Attia, M., Atutxa, A., Augustinus, L., Badmaeva, E., Balasubramani, K., Ballesteros, M., Banerjee, E., Bank, S., Barbu Mititelu, V., Basmov, V., Batchelor, C., Bauer, J., Bedir, S. T., Bengoetxea, K., Berk, G., Berzak, Y., Bhat, I. A., Bhat, R. A., Biagetti, E., Bick, E., Bielinskienė, A., Bjarnadóttir, K., Blokland, R., Bobicev, V., Boizou, L., Borges Völker, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Boyd, A., Brokaitė, K., Burchardt, A., Candito, M., Caron, B., Caron, G., Cavalcanti, T., Cebiroğlu Eryiğit, G., Cecchini, F. M., Celano, G. G. A., Čéplö, S., Cetin, S., Çetinoğlu, Ö., Chalub, F., Chi, E., Cho, Y., Choi, J., Chun, J., Cignarella, A. T., Cinková, S., Collomb, A., Çöltekin, Ç., Connor, M., Courtin, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Derin, M. O., de Souza, E., Diaz de Ilarraza, A., Dickerson, C., Dinakaramani, A., Dione, B., Dirix, P., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eckhoff, H., Eli, M., Elkahky, A., Ephrem, B., Erina, O., Erjavec, T., Etienne, A., Evelyn, W., Facundes, S., Farkas, R., Fernanda, M., Fernandez Alcalde, H., Foster, J., Freitas, C., Fujita, K., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Gerardi, F. F., Gerdes, K., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., González Saavedra, B., Griciūtė, B., Grioni, M., Grobol, L., Grūzītis, N., Guillaume, B., Guillot-Barbance, C., Güngör, T., Habash, N., Hafsteinsson, H., Hajič, J., Hajič jr., J., Hämäläinen, M., Hà Mỹ, L., Han, N.-R., Hanifmuti, M. Y., Hardwick, S., Harris, K., Haug, D., Heinecke, J., Hellwig, O., Hennig, F., Hladká, B., Hlaváčová, J., Hociung, F., Hohle, P., Huber, E., Hwang, J., Ikeda, T., Ingason, A. K., Ion, R., Irimia, E., Ishola, Ọ., Jelínek, T., Johannsen, A., Jónsdóttir, H., Jørgensen, F., Juutinen, M., K, S., Kaşıkara, H., Kaasen, A., Kabaeva, N., Kahane, S., Kanayama, H., Kanerva, J., Katz, B., Kayadelen, T., Kenney, J., Kettnerová, V., Kirchner, J., Klementieva, E., Köhn, A., Köksal, A., Kopacewicz, K., Korkiakangas, T., Kotsyba, N., Kovalevskaitė, J., Krek, S., Krishnamurthy, P., Kwak, S., Laippala, V., Lam, L., Lambertino, L., Lando, T., Larasati, S. D., Lavrentiev, A., Lee, J., Lê Hồng, P., Lenci, A., Lertpradit, S., Leung, H., Levina, M., Li, C. Y., Li, J., Li, K., Li, Y., Lim,

K., Lindén, K., Ljubešić, N., Loginova, O., Luthfi, A., Luukko, M., Lyashevskaya, O., Lynn, T., Macketanz, V., Makazhanov, A., Mandl, M., Manning, C., Manurung, R., Mărănduc, C., Mareček, D., Marheinecke, K., Martínez Alonso, H., Martins, A., Mašek, J., Matsuda, H., Matsumoto, Y., McDonald, R., McGuinness, S., Mendonça, G., Miekka, N., Mischenkova, K., Misirpashayeva, M., Missilä, A., Mititelu, C., Mitrofan, M., Miyao, Y., Mojiri Foroushani, A., Moloodi, A., Montemagni, S., More, A., Moreno Romero, L., Mori, K. S., Mori, S., Morioka, T., Moro, S., Mortensen, B., Moskalevskyi, B., Muischnek, K., Munro, R., Murawaki, Y., Müürisep, K., Nainwani, P., Nakhlé, M., Navarro Horñiacek, J. I., Nedoluzhko, A., Nešpore-Bērzkalne, G., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikaido, Y., Nikolaev, V., Nitisaroj, R., Nourian, A., Nurmi, H., Ojala, S., Ojha, A. K., Olúòkun, A., Omura, M., Onwuegbuzia, E., Osenova, P., Östling, R., Øvrelid, L., Özateş, Ş. B., Özgür, A., Öztürk Başaran, B., Partanen, N., Pascual, E., Passarotti, M., Patejuk, A., Paulino-Passos, G., Peljak-Łapińska, A., Peng, S., Perez, C.-A., Perkova, N., Perrier, G., Petrov, S., Petrova, D., Phelan, J., Piitulainen, J., Pirinen, T. A., Pitler, E., Plank, B., Poibeau, T., Ponomareva, L., Popel, M., Pretkalniņa, L., Prévost, S., Prokopidis, P., Przepiórkowski, A., Puolakainen, T., Pyysalo, S., Qi, P., Rääbis, A., Rademaker, A., Rama, T., Ramasamy, L., Ramisch, C., Rashel, F., Rasooli, M. S., Ravishankar, V., Real, L., Rebeja, P., Reddy, S., Rehm, G., Riabov, I., Rießler, M., Rimkutė, E., Rinaldi, L., Rituma, L., Rocha, L., Rögnvaldsson, E., Romanenko, M., Rosa, R., Roșca, V., Rovati, D., Rudina, O., Rueter, J., Rúnarsson, K., Sadde, S., Safari, P., Sagot, B., Sahala, A., Saleh, S., Salomoni, A., Samardžić, T., Samson, S., Sanguinetti, M., Särg, D., Saulīte, B., Sawanakunanon, Y., Scannell, K., Scarlata, S., Schneider, N., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shen, M., Shimada, A., Shirasu, H., Shohibussirri, M., Sichinava, D., Sigurdsson, E. F., Silveira, A., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Skachedubova, M., Smith, A., Soares-Bastos, I., Spadine, C., Steingrímsson, S., Stella, A., Straka, M., Strickland, E., Strnadová, J., Suhr, A., Sulestio, Y. L., Sulubacak, U., Suzuki, S., Szántó, Z., Taji, D., Takahashi, Y., Tamburini, F., Tan, M. A. C., Tanaka, T., Tella, S., Tellier, I., Thomas, G., Torga, L., Toska, M., Trosterud, T., Trukhina, A., Tsarfaty, R., Türk, U., Tyers, F., Uematsu, S., Untilov, R., Urešová, Z., Uria, L., Uszkoreit, H., Utka, A., Vajjala, S., van Niekerk, D., van Noord, G., Varga, V., Villemonte de la Clergerie, E., Vincze, V., Wakasa, A., Wallenberg, J. C., Wallin, L., Walsh, A., Wang, J. X., Washington, J. N., Wendt, M., Widmer, P., Williams, S., Wirén, M., Wittern, C., Woldemariam, T., Wong, T.-s., Wróblewska, A., Yako, M., Yamashita, K., Yamazaki, N., Yan, C., Yasuoka, K., Yavrumyan, M. M., Yu, Z., Žabokrtský, Z., Zahra, S., Zeldes, A., Zhu, H., and Zhuravleva, A. Universal dependencies 2.7, 2020. URL http://hdl.handle.net/11234/1-3424. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

A. Proof of Lemma 1

We provide a more general proof for multilinear factor potentials, of which bilinear potentials are a special case. Let $\mathcal{G} = (V, F)$ be the factor graph underlying the UNN, with energy function $E(x_1, \ldots, x_n) = \sum_i E_{\mathsf{X}_i}(x_i) + \sum_f E_f(x_f)$. We assume $E_{\mathsf{X}_i}(x_i) = -b_i^\top x_i + \Psi_{\mathsf{X}_i}(x_i)$ for each $\mathsf{X}_i \in V$, with Ψ_{X_i} convex, and $E_f(x_f) = -\langle W_f, \otimes_{j \in f} x_j \rangle$ for each higher order factor $f \in F$ (multilinear factor energy), where $\otimes$ is the outer product, and W_f is a parameter tensor of matching dimension. For pairwise factors $f = \{\mathsf{X}_i, \mathsf{X}_j\}$, the factor energy is bilinear and can be written simply as $E_f(x_i, x_j) = -x_i^\top W_f x_j$.

The (block) coordinate descent algorithm updates each representation $x_i \in V$ sequentially, leaving the remaining representations fixed. Let $F(\mathsf{X}_i) = \{f \in F : \mathsf{X}_i \in f\} \subseteq F$ denote the set of factors X_i is linked to. The updates can be written as:

$$\begin{aligned}
(x_i)_\star &= \arg\min_{x_i} E_{\mathsf{X}_i}(x_i) + \sum_{f \in F(\mathsf{X}_i)} E_f(x_f) \\
&= \arg\min_{x_i} \Psi_{\mathsf{X}_i}(x_i) \underbrace{-b_i^\top x_i - \sum_{f \in F(\mathsf{X}_i)} \langle W_f, \bigotimes_{j \in f} x_j \rangle}_{-z_i^\top x_i} \\
&= (\nabla \Psi^*_{\mathsf{X}_i})(z_i),
\end{aligned} \tag{21}$$

where z_i is a pre-activation given by

$$z_i = \left(\sum_{f \in F(\mathsf{X}_i)} \rho_i(W_f) \bigotimes_{j \in f, j \neq i} x_j \right) + b_i, \tag{22}$$

and ρ_i is the linear operator that reshapes and rolls the axis of W_f corresponding to x_i to the first position. If all factors are pairwise, the update is more simply:

$$(x_i)_\star = (\nabla \Psi^*_{\mathsf{X}_i}) \left(\sum_{f = \{\mathsf{X}_i, \mathsf{X}_j\} \in F(\mathsf{X}_i)} \rho_i(W_f) x_j + b_i \right), \tag{23}$$

where ρ_i is either the identity or the transpose operator. The update thus always consists in applying a (generally non-linear) transformation $\nabla \Psi^*_{\mathsf{X}_i}$ to an affine transformation of the neighbors of X_i in the graph (that is, the variables that co-participate in some factor).

Therefore, given any topological order of the variable nodes in V, running k iterations of the coordinate descent algorithm following that topological order is equivalent to performing forward propagation in an (unrolled) directed acyclic graph, where each node applies affine transformations on input variables followed by the activation function $\nabla \Psi^*_{\mathsf{X}_i}$.

B. Analysis of Order and Number of Variable Updates

For one of the experiments - undirected self-attention, we analyze how the order of variable updates and the number of update passes during training affect the model performance.

Order of variable updates. In Section 4 we showed that one pass of the "forward-backward" order or variable updates $(\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}, \mathsf{S}, \mathsf{V}, \mathsf{K}, \mathsf{Q}, \hat{\mathsf{X}})$ performs well enough for the of sequence completion. Since the flexibility of our model does not limit us to a specific order, we compare it to a random order of updating the variables (a permutation of $\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}$; $\hat{\mathsf{X}}$ is always updated last). One pass over the "forward-backward" order performs nine variable updates, and one pass over the random order - five. In Fig. 5 we show how the two ways of order perform for different number of variable updates (for example, 2 passes over the "forward-backward" model equal 18 variable updates, and over the random model - 10). The "forward-backward" order performs best, but the random order can achieve similar performance after enough number of updates.

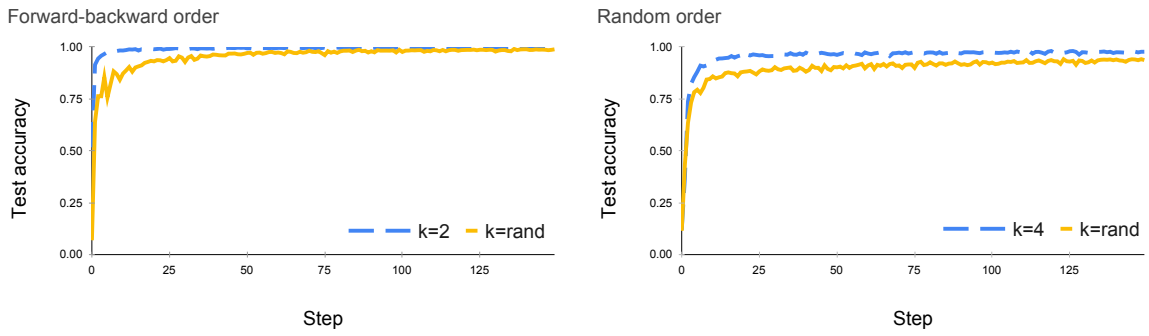

Figure 7. Learning curves for random number of variable update passes - for "forward-backward" and random order of operation updates.

Number of Energy Update Iterations In addition to comparing the number of energy update iterations k, we also try setting a random number of updates during training. Instead of specifying a fixed number of iterations k, we take a random k between 1 and 5 and train the model with it. We evaluate the performance on inference with $k = 3$ (the average value). In Fig. 7 we compare the performance of the best model trained with random number of iterations k with the best performing models trained with fixed k. As the plots show, the model trained with a random number of iterations performs on par with the best models with fixed k, but takes more time to train.

C. Analysis of Alternative Initialization Strategies

In addition to the zero initialization for the output variable y, we also experiment with two more initialization strategies - random and uniform initialization. We perform this comparison for the MNIST experiment from Section 3. For the random initialization, we initialize y with random numbers from a uniform distribution on the interval $[0, 1)$ and apply $\mathrm{softmax}$. For the uniform initialization, we assign equal values summing to one. We compare to the zero initialization strategy on the MNIST forward-backward experiment with $\gamma = .1$. The results are presented in Table 5. Random initialization shows promise, but the differences are small, and zero-init has the advantage of clearer parallels to the feed-forward case, so we report that and use it throughout all other experiments. The alternative initialization strategies can be further explored in further work.

Table 5. Comparison of different initialization strategies for the MNIST experiment from Section 3.

INITIALIZATION	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
ZERO	98.75	98.74	98.83	98.78	98.69
RANDOM	98.77	98.83	98.90	98.85	98.70
UNIFORM	98.76	98.74	98.83	98.78	98.68

D. Results for forward-only UNN for MNIST

In addition to the results for forward-backward training of the UNN, we also report results from training the UNN only in forward mode with $\gamma = 0$, i.e. when the model is trained for image classification only. The results are in Table 6.

E. Additional visualizations

Undirected self-attention weights. In Fig. 8 we show an example of the weights of the undirected self-attention described in Section 4. The attention weights are the values of the variable S calculated in the forward and backward pass.

Dependency tree packed representation Figure 9 shows an example of how the trees are represented in the output of the model in Section 5.

Table 6. MNIST accuracy with convolutional UNN in forward-only mode (*i.e.* $\gamma = 0$).

ITERATIONS	ACCURACY
$k = 1, \gamma = 0$ (BASELINE)	98.80
$k = 2$	**98.82**
$k = 3$	98.75
$k = 4$	98.74
$k = 5$	98.69

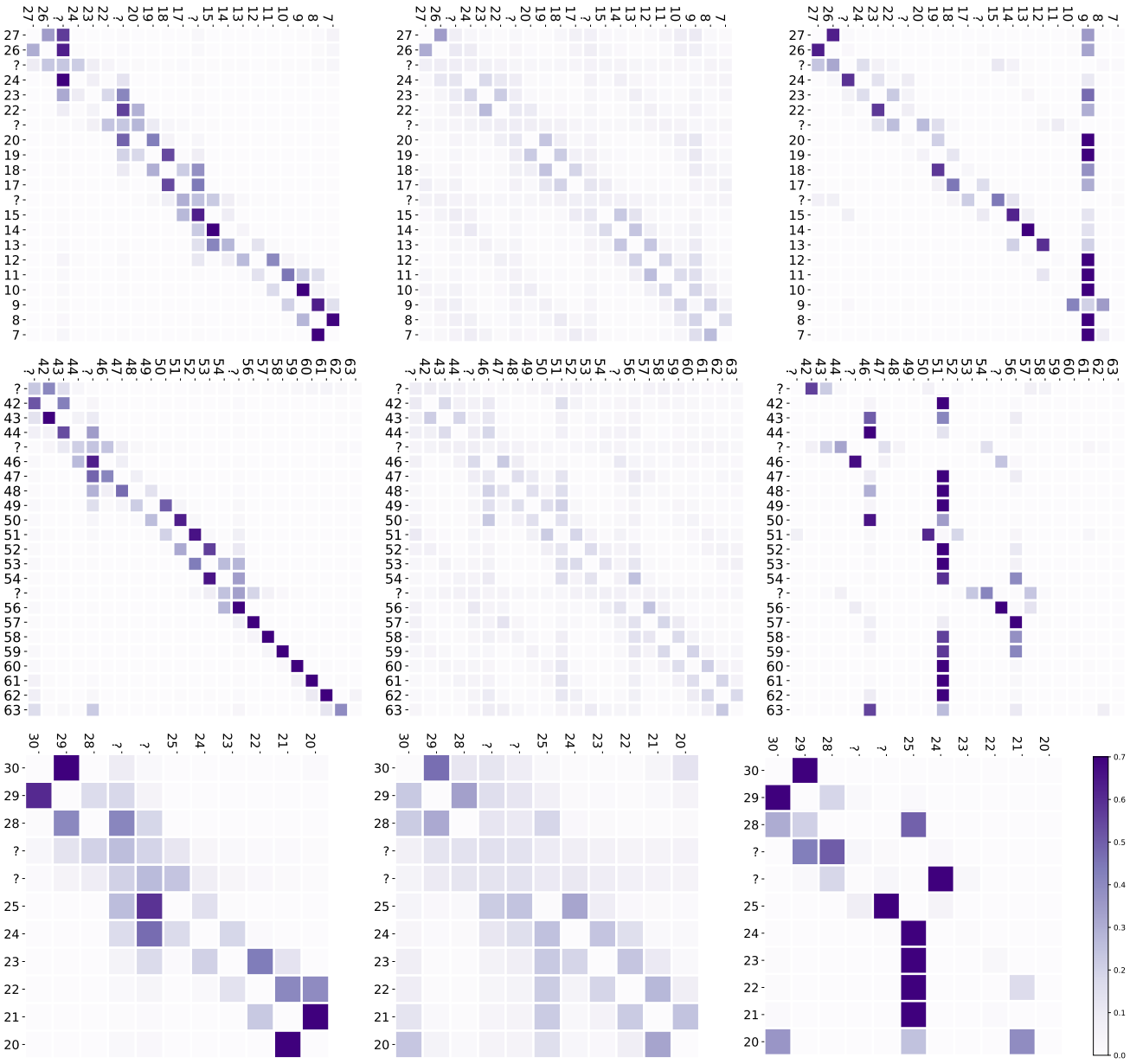

Figure 8. Example of the self-attention weights for models trained with $k = 1$ (left) and $k = 2$ after one iteration (middle) and two iterations (right). For $k = 2$, the model is more like an unrolled two-layer attention mechanism, with the first step identifying an off-diagonal pattern and the latter pooling information into an arbitrary token.

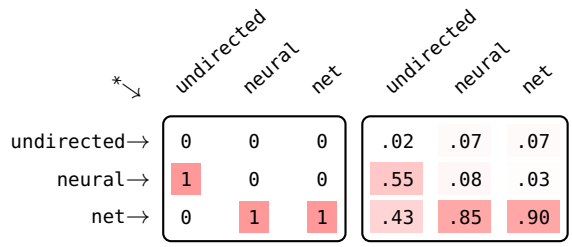

Figure 9. "Packed" matrix representation of a dependency tree (left) and dependency arc marginals (right). Each element corresponds to an arc h→m, and the diagonal corresponds to the arcs from the root, ∗→m. The marginals, computed via the matrix-tree theorem, are the structured counterpart of softmax, and correpond to arc probabilities.

F. Derivation of updates for convolutional UNN

The two-layer convolutional UNN is defined by the pairwise energies

$$\begin{aligned} E_{\mathsf{X}\mathsf{H}_1}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle\,, \\ E_{\mathsf{H}_1\mathsf{H}_2}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle\,, \\ E_{\mathsf{H}_2\mathsf{Y}}(H_2, y) &= -\langle y, V H_2\rangle\,, \end{aligned} \tag{24}$$

and the unary energies

$$\begin{aligned} E_{\mathsf{X}}(X) &= \frac{1}{2}\|X\|_2^2\,, \\ E_{\mathsf{H}_1}(H_1) &= -\langle H_1, b_1 \otimes 1_{d_1}\rangle + \Psi_{\tanh}(H_1)\,, \\ E_{\mathsf{H}_2}(H_2) &= -\langle H_2, b_2 \otimes 1_{d_2}\rangle + \Psi_{\tanh}(H_2)\,, \\ E_{\mathsf{Y}}(y) &= -\langle b, y\rangle - \mathcal{H}(y)\,. \end{aligned} \tag{25}$$

Above, $\mathcal{C}_1$ and $\mathcal{C}_2$ are are linear cross-correlation (convolution) operators with stride two and filter weights $W_1 \in \mathbb{R}^{32\times 1\times 6\times 6}$ and $W_2 \in \mathbb{R}^{64\times 32\times 4\times 4}$, and $b_1 \in \mathbb{R}^{32}$ and $b_2 \in \mathbb{R}^{64}$ are vectors of biases for each convolutional filter. The hidden activations have dimension $H_1 \in \mathbb{R}^{32\times(d_1)}$ and $H_2 \in \mathbb{R}^{64\times(d_2)}$, where d_1 and d_2 are tuples that depend on the input image size; for MNIST, $X \in \mathbb{R}^{1\times 28\times 28}$ leading to $d_1 = 12\times 12$ and $d_2 = 5\times 5$.

To derive the energy updates, we use the fact that a real linear operator $\mathcal{A}$ interacts with the Frobenius inner product as:

$$\langle P, \mathcal{A}(Q)\rangle = \langle Q, \mathcal{A}^\top(P)\rangle\,, \tag{26}$$

where $\mathcal{A}^\top$ is the transpose, or adjoint, operator.[4] If $\mathcal{C}$ is a convolution (*i.e.*, `torch.conv2d`) then $\mathcal{C}^\top$ is a deconvolution (*i.e.*, `torch.conv_transpose2d`) with the same filters. We then have

$$\begin{aligned} E_{\mathsf{X}\mathsf{H}_1}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle = -\langle X, \mathcal{C}_1^\top(H_1; W_1)\rangle\,, \\ E_{\mathsf{H}_1\mathsf{H}_2}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle = -\langle H_1, \mathcal{C}_2^\top(H_2; W_2)\rangle\,. \end{aligned} \tag{27}$$

Adding up all energies and ignoring the constant terms in each update, we get

$$\begin{aligned} X_\star &= \operatorname*{argmin}_X -\langle X, \mathcal{C}_1^\top(H_1, W_1)\rangle + \Psi_{\tanh}(X) \\ &= \tanh(\mathcal{C}_1^\top(H_1, W_1))\,, \\ (H_1)_\star &= \operatorname*{argmin}_{H_1} -\langle H_1, \mathcal{C}_1(X, W_1)\rangle - \langle H_1, \mathcal{C}_2^\top(H_2, W_2)\rangle - \langle H_1, b_1 \otimes 1_{d_1\times d_1}\rangle + \Psi_{\tanh(H_1)} \\ &= \tanh(\mathcal{C}_1(X, W_1) + \mathcal{C}_2^\top(H_2, W_2) + b_1 \otimes 1_{d_1\times d_1})\,, \\ (H_2)_\star &= \operatorname*{argmin}_{H_2} -\langle H_2, \mathcal{C}_2(H_1, W_2)\rangle - \langle H_2, \sigma_y(V)y\rangle - \langle H_2, b_2 \otimes 1_{d_2\times d_2}\rangle + \Psi_{\tanh(H_2)} \\ &= \tanh(\mathcal{C}_2(H_1, W_2) + \sigma_y(V)y + b_2 \otimes 1_{d_2\times d_2})\,, \\ y_\star &= -\langle y, V H_2\rangle - \langle y, b\rangle - \mathcal{H}y \\ &= \operatorname{softmax}(V H_2 + b)\,. \end{aligned} \tag{28}$$

[4] This generalizes the observation that $p^\top A q = q^\top A^\top p$.

Note that in our case, $H_2 \in \mathbb{R}^{64\times5\times5}$, $V \in \mathbb{R}^{10\times64\times5\times5}$ and thus $VH_2 \in \mathbb{R}^{10}$ is a tensor contraction (*e.g.*, `torch.tensordot(V, H_2, dims=3)`). The σ_y linear operator – opposite of ρ from Lemma 1 – rolls the axis of V corresponding to y to the *last* position, such that $\sigma_y(V) \in \mathbb{R}^{64\times5\times5\times10}$, the tensor analogue of a transposition (*e.g.*, `torch.permute(V, (1, 2, 3, 0))`.)