Learning Energy-Based Approximate Inference Networks for Structured Applications in NLP

Lifu Tu

July 15, 2020

This is part of my thesis proposal. It describes the previous related work, the motivation of my work, and my recent work on energy-based inference network. I would like to thanks my advisor Kevin Gimpel for helpful comments and my collaborators. I welcome any comments you may have regarding the content. If you spot a missing reference or have relevant work to be mentioned. Do let me know. lifu [at] ttic [dot] edu

1 Introduction

1.1 Structured Prediction

Many tasks(e.g., sequence labeling, semantic role labeling, parsing, machine translation) in natural language processing involve predicting structured outputs. It is crucial to model the correlations between the structured output. Researchers are increasingly applying deep representation learning to these problems, but the structured component of these approaches is usually quite simplistic. In many applications, we could predict each sub-component of the structured output independently given the inputs. However, this may have substantially lower accuracy than an approach that models the interactions between the structured outputs. Due to the exponentially-large space of candidate outputs, it is computational challenging to jointly predict all components of the structured outputs.

1.2 Frameworks for Structured Prediction

There are several approaches for structured prediction: local classifiers, discriminative structure models and energy-based models.

Local Classifiers: Assume we have the features $F(\boldsymbol{x}) = (F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), \ldots, F_n(\boldsymbol{x}))$ for a given sequence $\boldsymbol{x}$. These could be a hand-engineered set of feature functions or by the way of a learned deep neural network, such as BiLSTM [Hochreiter and Schmidhuber, 1997]. For the local classifiers, the outputs are conditionally independent given the features:

$$\log P(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_i \log P(y_i \mid F_i(\boldsymbol{x}))$$

It is natural to use the $P(y_i \mid F_i(\boldsymbol{x}))$ to predict the tag at the position i. It is done with a trivial operations that computes the argmax of a vector. According to the above, we could see that the local classifiers are easy to train and do inference with. However, because of the independence assumptions , the expressive power of models could be limited. And it is hard to guarantee that the decoded output is a valid sequence, for example, a valid B-I-O. We can observe that the local classifiers completely ignore the current label when predicting the next label.

Discriminative Structure Models: In discriminative strucure models, the score function decomposes additively across parts. Each part is a sub-component of input/output pair. Table 1 shows four different discriminative structure models, which are widely used before.

- transition-based structured prediction: the joint conditional is modeled as the product of locally normalized probability distribution over all positions. However, during training, the true previous label is always used. This could cause mismatch between training and test time, which is exposure bias [Ranzato et al., 2016].

	modeling	learning
transition-based	$P(\boldsymbol{y} \mid \boldsymbol{x}) = \Pi_t P(y_t \mid x, y_{t-1})$ locally normalized	$\max_\Theta \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} \log P_\Theta(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$ Previous gold label is used during training.
CRF	The joint distribution $P(\boldsymbol{y} \mid \boldsymbol{x})$ is defined by neural field; it solves the label bias problem	$\max_\Theta \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} \log P_\Theta(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$
perceptron	The score function S is usually linear weighted sum of the features, $S(x, y) = W^\top f(\boldsymbol{x}, \boldsymbol{y})$	$\min_\Theta \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} [\max_{\boldsymbol{y}} (S_\Theta(\boldsymbol{x}_i, \boldsymbol{y}) - -S_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i))]_+$
large margin	The score function S is usually linear weighted sum of the features, $S(x, y) = W^\top f(\boldsymbol{x}, \boldsymbol{y})$	$\min_\Theta \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} [\max_{\boldsymbol{y}} (\triangle(\boldsymbol{y}, \boldsymbol{y}_i) + S_\Theta(\boldsymbol{x}_i, \boldsymbol{y}) - S_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i))]_+$

Table 1: Comparisons of different discriminative models. $\mathcal{D}$ is the set of training pairs, $[f]_+ = \max(0, f)$, and $\triangle(\boldsymbol{y}, \boldsymbol{y}')$ is a structured **cost** function that returns a nonnegative value indicating the difference between $\boldsymbol{y}$ and $\boldsymbol{y}'$.

- conditional random fields [Lafferty et al., 2001](CRF): Pros: probabilistic interpretation (builds on graphical models), extends to latent variable models, flexible regularization. It solves the label bias problem. It has the efficient training and decoding based on dynamic programming for linear-chain CRF. However, it could be computationally expensive given a large label space. And the inference could be challenging for a general CRF framework.
- Structured Perceptron [Collins, 2002] describe an algorithm for training discriminative models, for example CRF. Usually Viterbi algorithm or other algorithms are used rather than an exhausive search in the exponentially large label space.
- Large Margin Structured Classifiers: learning general functional dependencies between arbitrary input and output spaces is one of the key challenges in computational intelligence. Taskar et al. [2004], Tsochantaridis et al. [2005] presents structured support vector machine(SSVM) that solves the optimization problem in polynomial time with exponential, number of constraints.

There has been a lot of work on using neural networks to define the potential functions in the discriminative structure models, e.g., neural CRF [Passos et al., 2014], RNN-CRF [Huang et al., 2015, Lample et al., 2016], CNN-CRF [Collobert et al., 2011] etc. However the potential functions are still limited in size.

Energy-based Models: Energy-based modeling [LeCun et al., 2006] associates a scalar measure $E(\boldsymbol{x}, \boldsymbol{y})$ of compatibility to each configuration of input $\boldsymbol{x}$ and output variables $\boldsymbol{y}$. Belanger and McCallum [2016] formulated deep energy-based models for structured prediction, which they called structured prediction energy networks (SPENs). SPENs use arbitrary neural networks to define the scoring function over input/output pairs. Compared with discriminative structure models, it is a deep structure model. For discriminative structure models, the potential functions are still limited in size. The dependence of their expressivity and scalability on the structured output is limited. SPENs, by contrast, do not place any limits on the size of the potential functions. The structured prediction energy networks is more flexible framework for structured prediction.

1.3 The Difficulties of Energy-Based Models

This deep architecture captures dependencies between labels that would lead to intractable graphical models. This flexibility of the deep energy-based models leads to challenges for **learning** and **inference**. For the discriminative training method, it is expensive for the "cost-augmented" inference step:

$$\max_{\boldsymbol{y}} (\triangle(\boldsymbol{y}, \boldsymbol{y}') - E_\Theta(\boldsymbol{x}, \boldsymbol{y}))$$

where $\triangle(\boldsymbol{y}, \boldsymbol{y}')$ is a structured **cost** function that returns a nonnegative value indicating the difference between $\boldsymbol{y}$ and $\boldsymbol{y}'$. E_Θ is a function parameterized by Θ that uses a functional architecture to compute a scalar energy for an input/output pair. This step is to find the $\boldsymbol{y}$ with low energy but with high cost.

And for the decoding, it is hard to jointly predict the label sequence for a task with complex structured components:

$$\min_{\boldsymbol{y}} E_\Theta(\boldsymbol{x}, \boldsymbol{y})$$

It maybe intractable to solve the above two optimization problems.

The original work on SPENs used gradient descent for structured inference [Belanger and McCallum, 2016, Belanger et al., 2017]. However, it is hard to guarantee the convergence for gradient descent inference. Furthermore, a lot of iterations could be needed for the convergence. Both of these could slow down the inference step and decrease the performance.

1.4 The New Approach for Training and Inference in Energy-Based Models

Prior work with SPENs used gradient descent for inference, relaxing the structured output to a set of continuous variables and then optimizing the energy with respect to them. We replace this use of gradient descent with a neural network trained to approximate structured argmax inference. The "inference network" outputs continuous values that we treat as the output structure. We develop large-margin training criteria for joint training of the structured energy function and inference network. According to our experiments results, we report speed-ups of 10-60x compared to Belanger et al. [2017] while also improving accuracy on multi-label classification. For sequence labeling with simple structured energies,our approach performs comparably to exact inference while being much faster at test time.

1.5 The Benefits of Learning Energy-Based Inference network

As stated above, the flexibility of the deep energy-based models leads to challenges for **learning** and **inference**. However, following are the potential benefits of Learning Energy-Based Inference network.

- Modeling complex structured components: for example, sequence labeling tasks usually learn a linear-chain CRFs that only learn the weight between successive labels and neural machine translation systems use unstructured training of local factors. For the energy model, it could capture the arbitrary dependence, especially the long-range dependency. For the generation, we could use for reducing the repetition, high BLEU score or semantic similarity with different energy terms.

- The energy could be used to improve the inference (even though exact inference maybe intractable, the approximate inference could be used.). For example, neural machine translation systems use unstructured training of local factors followed by beam search for test-time inference. However, the beam search algorithm could not be use if the score could not be decomposed from left to right. For example, the neural generation system [Li et al., 2016] could rerank the outputs from the n-best list by linearly combining the forward score $p(\boldsymbol{y} \mid \boldsymbol{x})$ and the “reverse score” $p(\boldsymbol{x} \mid \boldsymbol{y})$, where the latter comes from a separately-trained seq2seq model. The score function(energy) is potentially useful for the generation with the approximate inference network.

1.6 Overview and Contributions

In Section 3, we review previous two standard inference methods: gradient descent for inference and Viterbi algorithm. In Section 4 we introduce our proposed method, which replace this use of gradient descent with a neural network trained to approximate structured argmax inference. In Section 5, we demonstrate that inference networks achieve a better speed/accuracy/search error trade-off than gradient descent, while also being faster than exact inference at similar accuracy levels. In Section 6 and Section 7, we also develop large-margin training objectives to jointly train deep energy functions and inference networks. In Section 8, inference network is applied for non-autoregressive machine translation model training. We achieve state-of-the-art non-autoregressive results on the IWSLT 2014 DE-EN and WMT 2016 RO-EN datasets, approaching the performance of autoregressive models. Section **??**, several future research directiona are proposed.

2 RELATED WORK

2.1 Energy-Based Models

More recently, structured models have been combined with deep nets [Passos et al., 2014, Huang et al., 2015, Lample et al., 2016, Collobert et al., 2011, Hu et al., 2019, Mostajabi et al., 2018, Hwang et al., 2019, Graber et al., 2018a,b, Zhang et al., 2019],However the potential functions are still limited. To address the shortcoming, energy-based models are proposed, for instance, SPENs [Belanger and McCallum, 2016] and GSPEN [Graber and Schwing, 2019]. They do not allow for the explicit specification of output structure.

Recently, Grathwohl et al. [2020] also demonstrate that energy based training of the joint distribution improves calibration, robustness.

Although energy-based models have the strong ability to model the complex structured components, it have had limited application in NLP due to the computational challenges involved in learning and inference in extremely large search spaces. Previous, the partition function is needed with effectively estimating and sampling for training energy-based model [Hinton, 2002, Teh et al., 2003]. In Belanger and McCallum [2016] and Section 6 and Section 7, we use margin-based methods for energy training, which find the negative example in the cost-augmented inference step. However, it could be time-consuming and intractable. In Section 6 and Section 7, we use an inference network to approximate this inference step. In Wang and Ou [2018], Bakhtin et al. [2020], they use noise-contrastive estimation [Gutmann and Hyvarinen, 2010] for the energy training with some negative examples. It usually assumes that the model have "self-normalized" outputs.

2.2 Adversarial Training

Our training methods are reminiscent of other alternating optimization problems like that underlying generative adversarial networks (GANs; Goodfellow et al. 2014, Salimans et al. 2016, Zhao et al. 2016, Arjovsky et al. 2017). GANs are based on a minimax game and have a value function that one agent (a discriminator D) seeks to maximize and another (a generator G) seeks to minimize.

Progress in training GANs has come largely from overcoming learning difficulties by modifying loss functions and optimization, and GANs have become more successful and popular as a result. Notably, Wasserstein GANs [Arjovsky et al., 2017] provided the first convergence measure in GAN training using Wasserstein distance. To compute Wasserstein distance, the discriminator uses weight clipping, which limits network capacity. Weight clipping was subsequently replaced with a gradient norm constraint [Gulrajani et al., 2017]. Miyato et al. [2018] proposed a novel weight normalization technique called spectral normalization. These methods may be applicable to the similar optimization problems solved in learning SPENs. By their analysis, a log loss discriminator converges to a degenerate uniform solution. When using hinge loss, we can get a non-degenerate discriminator while matching the data distribution [Dai et al., 2017, Zhao et al., 2016]. Our formulation is closer to this hinge loss version of the GAN.

2.3 Approximate Inference

Since we train a single inference network for an entire dataset, our approach is also related to "**amortized inference**" [Srikumar et al., 2012, Gershman and Goodman, 2014, Paige and Wood, 2016, Chang et al., 2015]. Such methods precompute or save solutions to subproblems for faster overall computation. Our inference networks likely devote more modeling capacity to the most frequent substructures in the data. A kind of inference network is used in variational autoencoders [Kingma and Welling, 2013] to approximate posterior inference in generative models.

Our approach is also related to **knowledge distillation** [Ba and Caruana, 2014, Hinton et al., 2015], which refers to strategies in which one model (a "student") is trained to mimic another (a "teacher"). Typically, the teacher is a larger, more accurate model but which is too computationally expensive to use at test time. Urban et al. [2016] train shallow networks using image classification data labeled by an ensemble of deep teacher nets. Geras et al. [2016] train a convolutional network to mimic an LSTM for speech recognition. Others have explored knowledge distillation for sequence-to-sequence learning [Kim and Rush, 2016] and parsing [Kuncoro et al., 2016]. It has been empirically observed that distillation can improve generalization, Mobahi et al. [2020] provides a theoretical analysis of distillation when the teacher and student architectures are identical. In our methods, there is no limitation for model size of "student" and "teacher".

Our methods are also related to work in structured prediction that seeks to approximate structured models with factorized ones, e.g., mean-field approximations in graphical models [Koller and Friedman, 2009, Krähenbühl and Koltun, 2011]. Like our use of inference networks, there have been efforts in designing differentiable approximations of combinatorial search procedures [Martins and Kreutzer, 2017, Goyal et al., 2018] and structured losses for training with them [Wiseman and Rush, 2016]. Since we relax discrete output variables to be continuous, there is also a connection to recent work that focuses on structured prediction with continuous valued output variables [Wang et al., 2016]. They also propose a formulation that yields an alternating optimization problem, but it is based on proximal methods.

There are other settings in which **gradient descent** is used for inference, e.g., image generation applications like DeepDream [Mordvintsev et al., 2015] and neural style transfer [Gatys et al., 2015], as well as machine translation [Hoang et al., 2017]. In these and related settings, gradient descent has started to be replaced by inference networks, especially for image transformation tasks [Johnson et al., 2016, Li and Wand, 2016]. Our results below provide more evidence for making this transition. An alternative to what we pursue here would be to obtain an easier convex optimization problem for inference via input convex neural networks [Amos et al., 2017].

2.4 Non-Autoregressive Machine Translation

Non-autoregressive neural machine translation began with the work of Gu et al. [2018], who found benefit from using knowledge distillation [Hinton et al., 2015], and in particular sequence-level distilled outputs [Kim and Rush, 2016]. Subsequent work has narrowed the gap between non-autoregressive and autoregressive translation, including multi-iteration refinements [Lee et al., 2018, Ghazvininejad et al., 2019, Saharia et al., 2020, Kasai et al., 2020] and rescoring with autoregressive models [Kaiser et al., 2018, Wei et al., 2019, Ma et al., 2019, Sun et al., 2019]. Ghazvininejad et al. [2020] and Saharia et al. [2020] proposed aligned cross entropy or latent alignment models and achieved the best results of all non-autoregressive models without refinement or rescoring. We propose training inference networks with autoregressive energies and outperform the best purely non-autoregressive methods.

Another related approach trains an "actor" network to manipulate the hidden state of an autoregressive neural MT system [Gu et al., 2017, Chen et al., 2018, Zhou et al., 2020] in order to bias it toward outputs with better BLEU scores. This work modifies the original pretrained network rather than using it to define an energy for training an inference network.

3 BACKGROUND

We denote the space of inputs by $\mathcal{X}$. For a given input $\boldsymbol{x} \in \mathcal{X}$, we denote the space of legal structured outputs by $\mathcal{Y}(\boldsymbol{x})$. We denote the entire space of structured outputs by $\mathcal{Y} = \cup_{\boldsymbol{x}\in}\mathcal{Y}(\boldsymbol{x})$. Structure prediction energy network(SPEN) [Belanger and McCallum, 2016] defines an **energy function** $E_\Theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ parameterized by Θ that uses a functional architecture to compute a scalar energy for an input/output pair. The energy function can be an arbitrary function of the entire input/output pair, such as a deep neural network. Given the energy function, the inference step is to find the output with lowest energy:

$$\hat{\boldsymbol{y}} = \operatorname*{argmin}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} E_\Theta(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

However, solving Eq. (1) requires combinatorial algorithms because $\mathcal{Y}$ is a discrete structured space. This becomes intractable when E_Θ does not decompose into a sum over small "parts" of $\boldsymbol{y}$. Belanger and McCallum [2016] relax this problem by allowing the discrete vector $\boldsymbol{y}$ to be continuous.

Gradient Descent for Inference. Belanger and McCallum [2016] uses gradient descent for inference. To use gradient descent (GD) for structured inference, researchers typically relax the output space from a discrete, combinatorial space to a continuous one and then use gradient descent to solve the following optimization problem:

$$\operatorname*{argmin}_{\boldsymbol{y}\in\mathcal{Y}_R(\boldsymbol{x})} E_\Theta(\boldsymbol{x}, \boldsymbol{y})$$

where $\mathcal{Y}_R$ is the relaxed continuous output space. For sequence labeling, $\mathcal{Y}_R(\boldsymbol{x})$ consists of length-$|\boldsymbol{x}|$ sequences of probability distributions over output labels. To obtain a discrete labeling for evaluation, the most probable label at each position is returned.

We actually perform gradient descent in an even more relaxed output space $\mathcal{Y}_{R'}(\boldsymbol{x})$ which consists of length-$|\boldsymbol{x}|$ sequences of vectors, where each vector $\mathbf{y}_t \in \mathbb{R}^L$, L is the label set size. When computing the energy, we use a softmax transformation on each $\mathbf{y}_t$, solving the following optimization problem with gradient descent:

$$\operatorname*{argmin}_{\boldsymbol{y}\in\mathcal{Y}_{R'}(\boldsymbol{x})} E_\Theta(\boldsymbol{x}, \mathrm{softmax}(\boldsymbol{y})) \tag{2}$$

where the softmax operation above is applied independently to each vector $\mathbf{y}_t$ in the output structure $\boldsymbol{y}$.

Viterbi Algorithm Viterbi algorithm [Viterbi, 1967] is a dynamic programming algorithm for the finding the most likely sequence. In HMM or CRF, the conditional probability $\log p(\boldsymbol{y} \mid \boldsymbol{x})$ could be decomposed in a similar way.

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{i=1}^{|x|} score_1(y_i, y_{i-1}) + score_2(y_i, \boldsymbol{x})$$

here $score_1(y_i, y_{i-1})$ is a bigram score between the label y_i and y_{i-1}, $score_2(y_i, \boldsymbol{x})$ is a uniary score at position i with label y_i. Particularly, in HMM, $score_1(y_i, y_{i-1}) = \log p_\eta(y_i \mid y_{i-1})$, and $score_2(y_i, \boldsymbol{x}) = \log p_\tau(x_i \mid y_i)$. The inference in HMMs or CRF is done with the following optimization:

$$Classifier(\boldsymbol{x}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \sum_{i=1}^{|x|} score_1(y_i, y_{i-1}) + score_2(y_i, \boldsymbol{x}) \tag{3}$$

The above optimization problem could be solved with the dynamic programming algorithm. We set a variable $V(m, y')$, which means the probability of sequence starting with label y' at the position m. Then we have:

$$\begin{aligned} V(1, \hat{y}) =& score_1(\hat{y}, \langle s \rangle) + score_2(\hat{y}, \boldsymbol{x}) \\ V(m, \hat{y}) =& max_{y'}(score_1(\hat{y}, y') + score_2(\hat{y}, \boldsymbol{x}) + V(m-1, y')) \end{aligned}$$

$\langle s \rangle$ is the start sequence symbol.The second equation could be done recursively. If we consider that the last symbol is the end symbol $\langle /s \rangle$, then the output sequence $\boldsymbol{y}_{|\boldsymbol{x}|}$ is:

$$\underset{y'}{\operatorname{argmax}}\, score_1(< /s >, y') + V(|\boldsymbol{x}|, y')$$

And $\boldsymbol{y}_{|\boldsymbol{x}|-1}$, $\boldsymbol{y}_{|\boldsymbol{x}|-2}$,..., $\boldsymbol{y}_2$, $\boldsymbol{y}_1$ are got recursively. The time complexity is $\mathcal{O}(nL^2)$, where n is the sequence length and L is the size of the label space.

4 Inference Networks

Previous work [Belanger and McCallum, 2016] relaxed $\boldsymbol{y}$ from a discrete to a continuous vector and used gradient descent for inference. We also relax $\boldsymbol{y}$ but we use a different strategy to approximate inference. In Tu and Gimpel [2018] we define an **inference network** $\mathbf{A}_\Psi(\boldsymbol{x})$ parameterized by Ψ and train it with the goal that

$$\mathbf{A}_\Psi(\boldsymbol{x}) \approx \underset{\boldsymbol{y} \in \mathcal{Y}_R(\boldsymbol{x})}{\operatorname{argmin}} E_\Theta(\boldsymbol{x}, \boldsymbol{y}) \tag{4}$$

Given an energy function E_Θ and a dataset X of inputs, we solve the following optimization problem:

$$\hat{\Psi} \leftarrow \underset{\Psi}{\operatorname{argmin}} \sum_{\boldsymbol{x} \in X} E_\Theta(\boldsymbol{x}, \mathbf{A}_\Psi(\boldsymbol{x})) \tag{5}$$

The architecture of $\mathbf{A}_\Psi$ will depend on the task. For Multiple Label Classification(MLC), the same set of labels is applicable to every input, so $\boldsymbol{y}$ has the same length for all inputs. So, we can use a feed-forward network for $\mathbf{A}_\Psi$ with a vector output, treating each dimension as the prediction for a single label. For sequence labeling, each $\boldsymbol{x}$ (and therefore each $\boldsymbol{y}$) can have a different length, so we must use a network architecture for $\mathbf{A}_\Psi$ that permits different lengths of predictions. We use an RNN that returns a vector at each position of $\boldsymbol{x}$. We interpret this vector as a probability distribution over output labels at that position.

We note that the output of $\mathbf{A}_\Psi$ must be compatible with the energy function, which is typically defined in terms of the original discrete output space $\mathcal{Y}$. This may require generalizing the energy function to be able to operate both on elements of $\mathcal{Y}$ and $\mathcal{Y}_R$.

5 Benchmarking Approximate Inference Methods

In this section, I will introduce how to apply our method on several tasks and compare with several other inference method: Viterbi, Gradient descent inference.

The input space $\mathcal{X}$ is now the set of all sequences of symbols drawn from a vocabulary. For an input sequence $\boldsymbol{x}$ of length N, where there are L possible output labels for each position in $\boldsymbol{x}$, the output space $\mathcal{Y}(\boldsymbol{x})$ is $[L]^N$, where the notation $[q]$ represents the set containing the first q positive integers. We define $\boldsymbol{y} = \langle y_1, y_2, .., y_N \rangle$ where each y_i ranges over possible output labels, i.e., $y_i \in [L]$.

When defining our energy for sequence labeling, we take inspiration from bidirectional LSTMs (BLSTMs; Hochreiter and Schmidhuber 1997) and conditional random fields (CRFs; Lafferty et al. 2001). A "linear chain" CRF uses two types of features: one capturing the connection between an output label and $\boldsymbol{x}$ and the other capturing the dependence between neighboring output labels. We use a BLSTM to compute feature representations for $\boldsymbol{x}$. We use $f(\boldsymbol{x}, t) \in \mathbb{R}^d$ to denote the "input feature vector" for position t, defining it to be the d-dimensional BLSTM hidden vector at t.

The CRF energy function is the following:

$$E_\Theta(\boldsymbol{x}, \boldsymbol{y}) = -\left(\sum_t U_{y_t}^\top f(\boldsymbol{x}, t) + \sum_t W_{y_{t-1}, y_t} \right) \tag{6}$$

where $U_i \in \mathbb{R}^d$ is a parameter vector for label i and the parameter matrix $W \in \mathbb{R}^{L \times L}$ contains label pair parameters. The full set of parameters Θ includes the U_i vectors, W, and the parameters of the BLSTM. The above energy only permits **discrete** $\boldsymbol{y}$. However, the general energy which permits **continuous** $\boldsymbol{y}$ is needed. Now, I will discuss the continuous version of the above energy.

For sequence labeling tasks, given an input sequence $\boldsymbol{x} = \langle x_1, x_2, ..., x_{|\boldsymbol{x}|} \rangle$, we wish to output a sequence $\boldsymbol{y} = \langle \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{|\boldsymbol{x}|} \rangle \in \mathcal{Y}(\boldsymbol{x})$. Here $\mathcal{Y}(\boldsymbol{x})$ is the structured output space for $\boldsymbol{x}$. Each label $\mathbf{y}_t$ is represented as an L-dimensional one-hot vector where L is the number of labels.

For the general case that permits relaxing $\boldsymbol{y}$ to be **continuous**, we treat each y_t as a vector. It will be one-hot for the ground truth $\boldsymbol{y}$ and will be a vector of label probabilities for relaxed $\boldsymbol{y}$'s. Then the general energy function is:

$$E_\Theta(\boldsymbol{x}, \boldsymbol{y}) = -\left(\sum_t \sum_{i=1}^{L} y_{t,i} \left(U_i^\top f(\boldsymbol{x}, t) \right) + \sum_t \boldsymbol{y}_{t-1}^\top W \boldsymbol{y}_t \right) \tag{7}$$

where $y_{t,i}$ is the ith entry of the vector y_t. In the discrete case, this entry is 1 for a single i and 0 for all others, so this energy reduces to Eq. (6) in that case. In the continuous case, this scalar indicates the probability of the tth position being labeled with label i.

For the label pair terms in this general energy function, we use a bilinear product between the vectors $\boldsymbol{y}_{t-1}$ and $\boldsymbol{y}_t$ using parameter matrix W, which also reduces to Eq. (6) when they are one-hot vectors.

5.1 Experimental Setup

We perform experiments on three tasks: Twitter part-of-speech tagging (POS) [Gimpel et al., 2011, Owoputi et al., 2013] and, named entity recognition (NER) [Tjong Kim Sang and De Meulder, 2003], and CCG supersense tagging (CCG) [Hockenmaier and Steedman, 2002].

For our experimental comparison, we consider two CRF variants. The first is the basic model described above, which we refer to as BLSTM-CRF. We refer to the CRF with the following three techniques (word embedding fine-tuning, character-based embeddings, dropout) as BLSTM-CRF+:

Word Embedding Fine-Tuning. We used pretrained, fixed word embeddings when using the BLSTM-CRF model, but for the more complex BLSTM-CRF+ model, we fine-tune the pretrained word embeddings during training.

Character-Based Embeddings. Character-based word embeddings provide consistent improvements in sequence labeling [Lample et al., 2016, Ma and Hovy, 2016]. In addition to pretrained word embeddings, we produce a character-based embedding for each word using a character convolutional network like that of Ma and Hovy [2016]. The filter size is 3 characters and the character embedding dimensionality is 30. We use max pooling over the character sequence in the word and the resulting embedding is concatenated with the word embedding before being passed to the BLSTM.

Dropout. We also add dropout during training [Hinton et al., 2012]. Dropout is applied before the character embeddings are fed into the CNNs, at the final word embedding layer before the input to the BLSTM, and after the BLSTM. The dropout rate is 0.5 for all experiments.

	Inference Networks				
	CNN	BLSTM	seq2seq	Viterbi	Gradient Descent
POS	12500	1250	357	500	20
NER	10000	1000	294	360	23
CCG	6666	1923	1000	232	16

Table 2: Speed comparison of inference networks across tasks and architectures (examples/sec).

Inference Network Architectures. In our experiments, we use three options for the inference network architectures: convolutional neural networks (CNN), recurrent neural networks, sequence-to-sequence (seq2seq, Sutskever et al. 2014b) models. For seq2seq inference network, since sequence labeling tasks have equal input and output sequence lengths and a strong connection between corresponding entries in the sequences, Goyal et al. [2018] used fixed attention that deterministically attends to the ith input when decoding the ith output, and hence does not learn any attention parameters. For each, we optionally include the modeling improvements (word embedding fine-tuning, character-based embeddings, dropout) described in the above. When doing so, we append "+" to the setting's name to indicate this (e.g., infnet+).

5.2 Training Objective

For training the inference network parameters Ψ, we find that a local cross entropy loss consistently worked well for sequence labeling. We use this local cross entropy loss in this proposal, so we perform learning by solving the following:

$$\operatorname*{argmin}_{\Psi} \sum_{\langle \boldsymbol{x},\boldsymbol{y}\rangle} E_{\Theta}(\boldsymbol{x}, \mathbf{A}_{\Psi}(\boldsymbol{x})) + \lambda \ell_{\text{token}}(\boldsymbol{y}, \mathbf{A}_{\Psi}(\boldsymbol{x}))$$

where the sum is over $\langle \boldsymbol{x}, \boldsymbol{y}\rangle$ pairs in the training set. The token-level loss is defined:

$$\ell_{\text{token}}(\boldsymbol{y}, \mathbf{A}(\boldsymbol{x})) = \sum_{t=1}^{|\boldsymbol{y}|} \text{CE}(\mathbf{y}_t, \mathbf{A}(\boldsymbol{x})_t) \tag{8}$$

where $\mathbf{y}_t$ is the L-dimensional one-hot label vector at position t in $\boldsymbol{y}$, $\mathbf{A}(\boldsymbol{x})_t$ is the inference network's output distribution at position t, and CE stands for cross entropy. ℓ_{token} is the loss used in our non-structured baseline models.

5.3 Speed Comparison

Asymptotically, Viterbi takes $\mathcal{O}(nL^2)$ time, where n is the sequence length. The BLSTM and our deterministic-attention seq2seq models have time complexity $\mathcal{O}(nL)$. CNNs also have complexity $\mathcal{O}(nL)$ but are more easily parallelizable. Table 2 shows test-time inference speeds for inference networks, gradient descent, and Viterbi for the BLSTM-CRF model. We use GPUs and a minibatch size of 10 for all methods. CNNs are 1-2 orders of magnitude faster than the others. BLSTMs work almost as well as seq2seq models and are 2-4 times faster in our experiments. Viterbi is actually faster than seq2seq when L is small, but for CCG, which has $L = 400$, it is 4-5 times slower. Gradient descent is slower than the others because it generally needs many iterations (20-50) for competitive performance.

5.4 Search Error

We can view inference networks as approximate search algorithms and assess characteristics that affect search error. To do so, we train two LSTM language models (one on word sequences and one on gold label sequences) on the Twitter POS data.

We compute the difference in the BLSTM-CRF energies between the inference network output $\boldsymbol{y}_{inf}$ and the Viterbi output $\boldsymbol{y}_{vit}$ as the search error:

$$E_{\Theta}(\boldsymbol{x}, \boldsymbol{y}_{inf}) - E_{\Theta}(\boldsymbol{x}, \boldsymbol{y}_{vit}) \tag{9}$$

We compute the same search error for gradient descent. For the BLSTM inference network, Spearman's ρ between the word sequence perplexity and search error is 0.282; for the label sequence perplexity, it is 0.195. For gradient descent inference, Spearman's ρ between the word sequence perplexity and search

	N	Twitter POS Tagging Acc. (↑)	Twitter POS Tagging Energy (↓)	NER F1 (↑)	NER Energy (↓)	CCG Supertagging Acc. (↑)	CCG Supertagging Energy (↓)
gold standard		100	-159.65	100	-230.63	100	-480.07
BLSTM-CRF+/Viterbi		90.9	-163.20	91.6	-231.53	94.3	-483.09
gradient descent	10	89.2	-161.69	81.9	-227.92	65.1	-412.81
	20	90.8	-163.06	89.1	-231.17	74.6	-414.81
	30	90.8	-163.02	89.6	-231.30	83.0	-447.64
	40	90.7	-163.03	89.8	-231.34	88.6	-471.52
	50	90.8	-163.04	89.8	-231.35	90.0	-476.56
	100	-	-	-	-	90.1	-476.98
	500	-	-	-	-	90.1	-476.99
	1000	-	-	-	-	90.1	-476.99
infnet+		91.3	-162.07	90.8	-231.19	94.2	-481.32
discretized output from infnet+		91.3	-160.87	90.8	-231.34	94.2	-481.95
instance-tailored infnet+	3	91.0	-162.59	91.3	-231.32	94.3	-481.91
	5	90.9	-162.81	91.2	-231.37	94.3	-482.23
	10	91.3	-162.85	91.5	-231.39	94.3	-482.56
infnet+ as warm start for gradient descent	3	91.4	-163.06	91.4	-231.42	94.4	-482.62
	5	91.2	-163.12	91.4	-231.45	94.4	-482.64
	10	91.2	-163.15	91.5	-231.46	94.4	-482.78

Table 3: Test set results of approximate inference methods for three tasks, showing performance metrics (accuracy and F1) as well as average energy of the output of each method. The inference network architectures in the above experiments are: CNN for POS, seq2seq for NER, and BLSTM for CCG. N is the number of epochs for GD inference or instance-tailored fine-tuning.

error is 0.122; for the label sequence perplexity, it is 0.064. These positive correlations mean that for frequent sequences, inference networks and gradient descent exhibit less search error. We also note that the correlations are higher for the inference network than for gradient descent, showing the impact of amortization during learning of the inference network parameters. That is, since we are learning to do inference from a dataset, we would expect search error to be smaller for more frequent sequences, and we do indeed see this correlation.

5.5 Methods to Improve Inference Networks

To further improve the performance of an inference network for a particular test instance $\boldsymbol{x}$, we propose two novel approaches that leverage the strengths of inference networks to provide effective starting points and then use instance-level fine-tuning in two different ways.

5.5.1 Instance-Tailored Inference Networks

For each test example $\boldsymbol{x}$, we initialize an instance-specific inference network $\mathbf{A}_\Psi(\boldsymbol{x})$ using the trained inference network parameters, then run gradient descent on the following loss:

$$\underset{\Psi}{\operatorname{argmin}}\, E_\Theta(\boldsymbol{x}, \mathbf{A}_\Psi(\boldsymbol{x})) \tag{10}$$

This procedure fine-tunes the inference network parameters for a single test example to minimize the energy of its output. For each test example, the process is repeated, with a new instance-specific inference network being initialized from the trained inference network parameters.

5.5.2 Warm-Starting Gradient Descent with Inference Networks

Given a test example $\boldsymbol{x}$, we initialize $\boldsymbol{y} \in \mathcal{Y}_{R'}(\boldsymbol{x})$ using the inference network and then use gradient descent by solving Eq. 2 described in Section 3 to update $\boldsymbol{y}$. However, the inference network output is in $\mathcal{Y}_R(\boldsymbol{x})$ while gradient descent works with the more relaxed space $\mathcal{Y}_{R'}(\boldsymbol{x})$. So we simply use the logits from the inference network, which are the score vectors before the softmax operations.

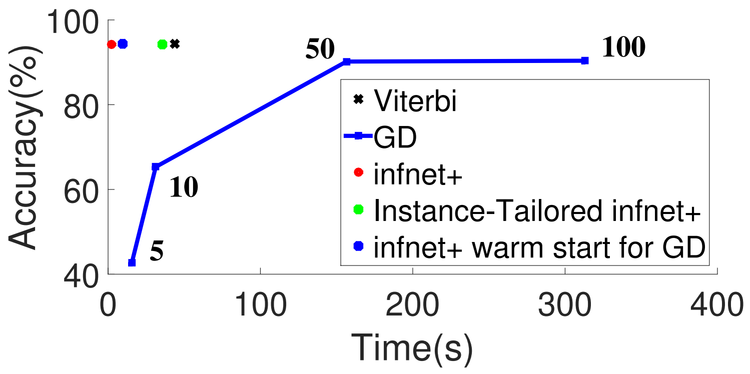


Figure 1: CCG test results for inference methods (GD = gradient descent). The x-axis is the total inference time for the test set. The numbers on the GD curve are the number of gradient descent iterations.

5.6 Speed, Accuracy, and Search Error

Table 3 compares inference methods in terms of both accuracy and energies reached during inference. For each number N of gradient descent iterations in the table, we tune the learning rate per-sentence and report the average accuracy/F1 with that fixed number of iterations. We also report the average energy reached. For inference networks, we report energies both for the output directly and when we discretize the output (i.e., choose the most probable label at each position).

Gradient Descent Across Tasks. The number of gradient descent iterations required for competitive performance varies by task. For POS, 20 iterations are sufficient to reach accuracy and energy close to Viterbi. For NER, roughly 40 iterations are needed for gradient descent to reach its highest F1 score, and for its energy to become very close to that of the Viterbi outputs. However, its F1 score is much lower than Viterbi. For CCG, gradient descent requires far more iterations, presumably due to the larger number of labels in the task. Even with 1000 iterations, the accuracy is 4% lower than Viterbi and the inference networks. Unlike POS and NER, the inference network reaches much lower energies than gradient descent on CCG, suggesting that the inference network may not suffer from the same challenges of searching high-dimensional label spaces as those faced by gradient descent.

Inference Networks Across Tasks. For POS, the inference network does not have lower energy than gradient descent with $\geq$ 20 iterations, but it does have higher accuracy. This may be due in part to our use of multi-task learning for inference networks. The discretization of the inference network outputs increases the energy on average for this task, whereas it decreases the energy for the other two tasks. For NER, the inference network reaches a similar energy as gradient descent, especially when discretizing the output, but is considerably better in F1. The CCG tasks shows the largest difference between gradient descent and the inference network, as the latter is much better in both accuracy and energy.

Instance Tailoring and Warm Starting. Across tasks, instance tailoring and warm starting lead to lower energies than infnet+. The improvements in energy are sometimes joined by improvements in accuracy, notably for NER where the gains range from 0.4 to 0.7 in F1. Warm starting gradient descent yields the lowest energies (other than Viterbi), showing promise for the use of gradient descent as a local search method starting from inference network output.

Wall Clock Time Comparison. Figure 1 shows the speed/accuracy trade-off for the inference methods, using wall clock time for test set inference as the speed metric. On this task, Viterbi is time-consuming because of the larger label set size. The inference network has comparable accuracy to Viterbi but is much faster. Gradient descent needs much more time to get close to the others but plateaus before actually reaching similar accuracy. Instance-tailoring and warm starting reside between infnet+ and Viterbi, with warm starting being significantly faster because it does not require updating inference network parameters.

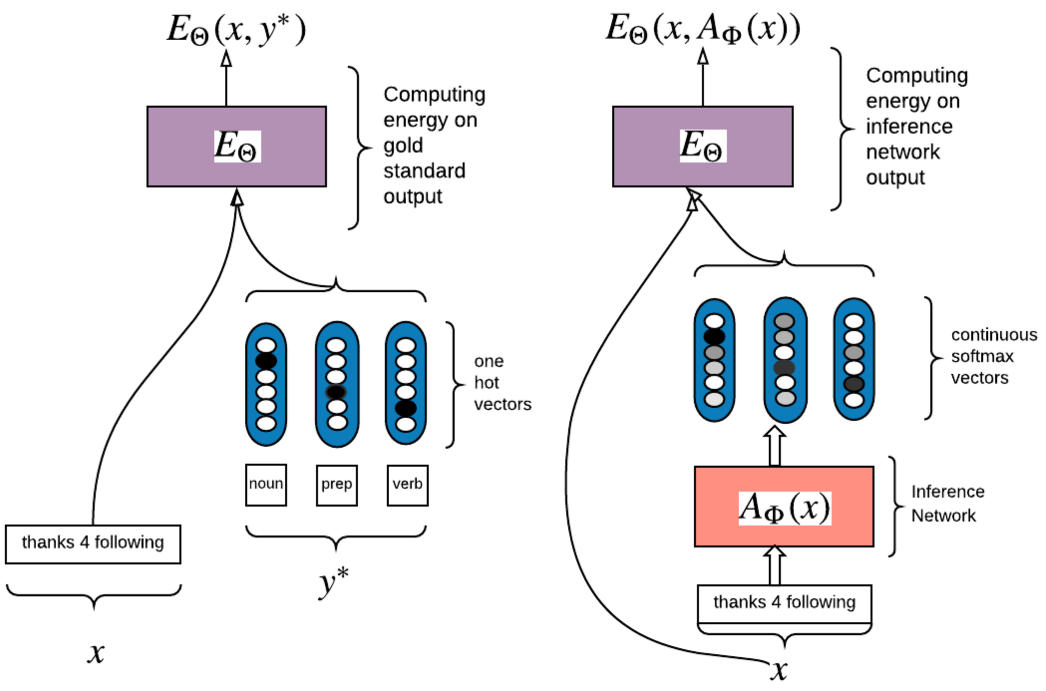


Figure 2: The architectures of inference network $\mathbf{F}_\Phi$ and energy network E_Θ.

6 SPEN Training Using Inference Networks

In Section 5, we discussed training inference networks for a pretrained, fixed energy function for sequence labeling. We now describe our completed work in joint learning of energy functions and inference networks, using the framework of SPENs.

Belanger and McCallum [2016] relaxed $\boldsymbol{y}$ from a discrete to a continuous vector and used gradient descent for inference. We also relax $\boldsymbol{y}$ but we use a different strategy to approximate inference. We define an **inference network** $\mathbf{A}_\Psi(\boldsymbol{x})$ parameterized by Ψ and train it with the goal as Equation 1. Figure 2 shows the architectures of inference network $\mathbf{F}_\Phi$ and energy network E_Θ. Given an energy function E_Θ and a dataset X of inputs, we solve the optimization problem as shown in Equation 5.

The following parts show how to train SPENs and inference networks.

6.1 Joint Training of SPENs and Inference Networks

Belanger and McCallum [2016] propose a structured hinge loss for training SPENs:

$$\min_\Theta \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} \left[\max_{\boldsymbol{y} \in \mathcal{Y}_R(\boldsymbol{x})} \left(\triangle(\boldsymbol{y}, \boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i) \right) \right]_+ \tag{11}$$

where $\mathcal{D}$ is the set of training pairs, $[f]_+ = \max(0, f)$, and $\triangle(\boldsymbol{y}, \boldsymbol{y}')$ is a structured **cost** function that returns a nonnegative value indicating the difference between $\boldsymbol{y}$ and $\boldsymbol{y}'$. This loss is often referred to as "margin-rescaled" structured hinge loss [Taskar et al., 2004, Tsochantaridis et al., 2005].

However, this loss is expensive to minimize for structured models because of the "cost-augmented" inference step ($\max_{\boldsymbol{y} \in \mathcal{Y}_R(\boldsymbol{x})}$). In prior work with SPENs, this step used gradient descent.

We replace this with a **cost-augmented inference network** $\mathbf{F}_\Phi(\boldsymbol{x})$. As suggested by the notation, the cost-augmented inference network $\mathbf{F}_\Phi$ and the inference network $\mathbf{A}_\Psi$ will typically have the same functional form, but use different parameters Φ and Ψ.

We write our new optimization problem as:

$$\min_\Theta \max_\Phi \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} [\triangle(\mathbf{F}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i, \mathbf{F}_\Phi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+ \tag{12}$$

We treat this optimization problem as a minmax game and find a saddle point for the game. Following Goodfellow et al. [2014], we implement this using an iterative numerical approach. We alternatively optimize Φ and Θ, holding the other fixed. Optimizing Φ to completion in the inner loop of training is computationally prohibitive and may lead to overfitting. So we alternate between one mini-batch for optimizing Φ and one for optimizing Θ. We also add L_2 regularization terms for Θ and Φ.

The objective for the cost-augmented inference network is:

$$\hat{\Phi} \leftarrow \underset{\Phi}{\operatorname{argmax}} [\triangle(\mathbf{F}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i, \mathbf{F}_\Phi(\boldsymbol{x})_i) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+ \tag{13}$$

That is, we update Φ so that $\mathbf{F}_\Phi$ yields an output that has low energy and high cost, in order to mimic cost-augmented inference. The energy parameters Θ are kept fixed. There is an analogy here to the generator in GANs: $\mathbf{F}_\Phi$ is trained to produce a high-cost structured output that is also appealing to the current energy function.

The objective for the energy function is:

$$\hat{\Theta} \leftarrow \underset{\Theta}{\operatorname{argmin}} [\triangle(\mathbf{A}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i, \mathbf{F}_\Phi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+ + \lambda \|\Theta\|_2^2 \tag{14}$$

That is, we update Θ so as to widen the gap between the cost-augmented and ground truth outputs. There is an analogy here to the discriminator in GANs. The energy function is updated so as to enable it to distinguish "fake" outputs produced by $\mathbf{F}_\Phi$ from real outputs $\boldsymbol{y}_i$. Training iterates between updating Φ and Θ using the objectives above.

6.2 Test-Time Inference

After training, we want to use an inference network $\mathbf{A}_\Psi$ defined in Eq. (4). However, training only gives us a cost-augmented inference network $\mathbf{F}_\Phi$. Since $\mathbf{A}_\Psi$ and $\mathbf{F}_\Phi$ have the same functional form, we can use Φ to initialize Ψ, then do additional training on $\mathbf{A}_\Psi$ as in Eq. (5) where X is the training or validation set. This step helps the resulting inference network to produce outputs with lower energy, as it is no longer affected by the cost function. Since this procedure does not use the output labels of the $\boldsymbol{x}$'s in X, it could also be applied to the test data in a transductive setting.

6.3 Variations and Special Cases

This approach also permits us to use large-margin structured prediction with slack rescaling [Tsochantaridis et al., 2005]. Slack rescaling can yield higher accuracies than margin rescaling, but requires "cost-scaled" inference during training which is intractable for many classes of output structures.

However, we can use our notion of inference networks to circumvent this tractability issue and approximately optimize the slack-rescaled hinge loss, yielding the following optimization problem:

$$\min_{\Theta} \max_{\Phi} \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} \triangle(\mathbf{F}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i)[1 - E_\Theta(\boldsymbol{x}_i, \mathbf{F}_\Phi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+ \tag{15}$$

Using the same argument as above, we can also break this into alternating optimization of Φ and Θ.

We can optimize a structured perceptron [Collins, 2002] version by using the margin-rescaled hinge loss (Eq. (21)) and fixing $\triangle(\mathbf{F}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i) = 0$. When using this loss, the cost-augmented inference network is actually a test-time inference network, because the cost is always zero, so using this loss may lessen the need to retune the inference network after training.

When we fix $\triangle(\mathbf{F}_\Phi(\boldsymbol{x}_i), \boldsymbol{y}_i) = 1$, then margin-rescaled hinge is equivalent to slack-rescaled hinge. While using $\triangle = 1$ is not useful in standard max-margin training with exact argmax inference (because the cost has no impact on optimization when fixed to a positive constant), it is potentially useful in our setting.

Consider our SPEN objectives with $\triangle = 1$:

$$[1 - E_\Theta(\boldsymbol{x}_i, \mathbf{F}_\Phi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+ \tag{16}$$

There will always be a nonzero difference between the two energies because $\mathbf{F}_\Phi(\boldsymbol{x}_i)$ will never exactly equal the discrete vector $\boldsymbol{y}_i$.

Since there is no explicit minimization over all discrete vectors $\boldsymbol{y}$, this case is more similar to a "contrastive" hinge loss which seeks to make the energy of the true output lower than the energy of a particular "negative sample" by a margin of at least 1.

	Bibtex	Bookmarks	Delicious	avg.
MLP	38.9	33.8	37.8	36.8
SPEN (BM16)	**42.2**	34.4	**37.5**	38.0
SPEN (E2E)	38.1	33.9	34.4	35.5
SPEN (InfNet)	**42.2**	**37.6**	**37.5**	**39.1**

Table 4: Test F1 when comparing methods on multi-label classification datasets.

	Training Speed (examples/sec)			Testing Speed (examples/sec)		
	Bibtex	Bookmarks	Delicious	Bibtex	Bookmarks	Delicious
MLP	21670	19591	26158	90706	92307	113750
SPEN (E2E)	551	559	383	1420	1401	832
SPEN (InfNet)	5533	5467	4667	94194	88888	112148

Table 5: Training and test-time inference speed comparison (examples/sec).

6.4 Results

In this section, we compare our approach to previous work on traing SPENs

6.4.1 Multi-Label Classification

Energy Functions for Multi-label Classification We describe the SPEN for multi-label classification (MLC) from Belanger and McCallum [2016]. Here, $\boldsymbol{x}$ is a fixed-length feature vector. We assume there are L labels, each of which can be on or off for each input, so $\mathcal{Y}(\boldsymbol{x}) = \{0,1\}^L$ for all $\boldsymbol{x}$. The energy function is the sum of two terms: $E_\Theta(\boldsymbol{x},\boldsymbol{y}) = E^{loc}(\boldsymbol{x},\boldsymbol{y}) + E^{lab}(\boldsymbol{y})$. $E^{loc}(\boldsymbol{x},\boldsymbol{y})$ is the sum of linear models:

$$E^{loc}(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{L} y_i b_i^\top F(\boldsymbol{x}) \tag{17}$$

where b_i is a parameter vector for label i and $F(\boldsymbol{x})$ is a multi-layer perceptron computing a feature representation for the input $\boldsymbol{x}$. $E^{lab}(\boldsymbol{y})$ scores $\boldsymbol{y}$ independent of $\boldsymbol{x}$:

$$E^{lab}(\boldsymbol{y}) = c_2^\top g(C_1 \boldsymbol{y}) \tag{18}$$

where c_2 is a parameter vector, g is an elementwise non-linearity function, and C_1 is a parameter matrix.

Performance Comparison to Prior Work. Table 4 shows results comparing to prior work. The MLP and "SPEN (BM16)" baseline results are taken from [Belanger and McCallum, 2016]. We obtained the "SPEN (E2E)" [Belanger et al., 2017] results by running the code available from the authors on these datasets. This method constructs a recurrent neural network that performs gradient-based minimization of the energy with respect to $\boldsymbol{y}$. They noted in their software release that, while this method is more stable, it is prone to overfitting and actually performs worse than the original SPEN. We indeed find this to be the case, as SPEN (E2E) underperforms SPEN (BM16) on all three datasets.

Our method ("SPEN (InfNet)") achieves the best average performance across the three datasets. It performs especially well on Bookmarks, which is the largest of the three. Our results use the contrastive hinge loss and retune the inference network on the development data after the energy is trained; these decisions were made based on the tuning, but all four hinge losses led to similarly strong results.

Speed Comparison. Table 5 compares training and test-time inference speed among the different methods. We only report speeds of methods that we ran.[1] The SPEN (E2E) times were obtained using code obtained from Belanger and McCallum. We suspect that SPEN (BM16) training would be comparable to or slower than SPEN (E2E).

Our method can process examples during training about 10 times as fast as the end-to-end SPEN, and 60-130 times as fast during test-time inference. In fact, at test time, our method is roughly the same speed as the MLP baseline, since our inference networks use the same architecture as the feature networks which

[1]The MLP F1 scores above were taken from Belanger and McCallum [2016], but the MLP timing results reported in Table 5 are from our own experimental replication of their results.

	validation accuracy (%)	
SPEN hinge loss	-retuning	+retuning
margin rescaling	89.1	89.3
slack rescaling	89.4	89.6
perceptron (MR, $\triangle = 0$)	89.2	89.4
contrastive ($\triangle = 1$)	88.8	89.0

Table 6: Comparison of SPEN hinge losses and showing the impact of retuning (Twitter POS validation accuracies). Inference networks are trained with the cross entropy term.

form the MLP baseline. Compared to the MLP, the training of our method takes significantly more time overall because of joint training of the energy function and inference network, but fortunately the test-time inference is comparable.

6.4.2 Sequence Labeling

Energy Functions for Sequence Labeling For sequence labeling tasks, given an input sequence $\boldsymbol{x} = \langle x_1, x_2, ..., x_{|\boldsymbol{x}|}\rangle$, we wish to output a discrete sequence. In Equation 6, the energy function only permits discrete $\boldsymbol{y}$. For the general case that permits relaxing $\boldsymbol{y}$ to be continuous, we treat each y_t as a vector. It will be one-hot for the ground truth $\boldsymbol{y}$ and will be a vector of label probabilities for relaxed $\boldsymbol{y}$'s. Then the general energy function in Equation 7.

Experimental Setup For Twitter part-of-speech (POS) tagging, we use the annotated data from Gimpel et al. [2011] and Owoputi et al. [2013] which contains $L = 25$ POS tags. For training, we combine the 1000-tweet OCT27TRAIN set and the 327-tweet OCT27DEV set. For validation, we use the 500-tweet OCT27TEST set and for testing we use the 547-tweet DAILY547 test set. We use 100-dimensional skip-gram embeddings trained on 56 million English tweets with `word2vec` [Mikolov et al., 2013].[2]

We use a BLSTM to compute the "input feature vector" $f(\boldsymbol{x}, t)$ for each position t, using hidden vectors of dimensionality $d = 100$. We also use BLSTMs for the inference networks. The output layer of the inference network is a softmax function, so at every position, the inference network produces a distribution over labels at that position. We train inference networks using stochastic gradient descent (SGD) with momentum and train the energy parameters using Adam. For $\triangle$, we use L_1 distance. We tune hyperparameters on the validation set; full details of tuning are provided in the appendix. We found that the cross entropy stabilization term worked well for this setting.

We compare to standard BLSTM and CRF baselines. We train the BLSTM baseline to minimize per-token log loss; this is often called a "BLSTM tagger". We train a CRF baseline using the energy in Eq. (6) with the standard conditional log-likelihood objective using the standard dynamic programming algorithms (forward-backward) to compute gradients during training. Further details are provided in the appendix.

Learned Pairwise Potential Matrix Figure 3 shows the learned pairwise potential matrix W in Twitter POS tagging. We can see strong correlations between labels in neighborhoods. For example, an adjective (A) is more likely to be followed by a noun (N) than a verb (V) (see row labeled "A" in the figure).

Loss Function Comparison. Table 6 shows results when comparing SPEN training objectives. We see a larger difference among losses here than for MLC tasks. When using the perceptron loss, there is no margin, which leads to overfitting: 89.4 on validation, 88.6 on test (not shown in the table). The contrastive loss, which strives to achieve a margin of 1, does better on test (89.0). We also see here that margin rescaling and slack rescaling both outperform the contrastive hinge, unlike the MLC tasks. We suspect that in the case in which each input/output has a different length, using a cost that captures length is more important.

Comparison to Standard Baselines. Table 7 compares our final tuned SPEN configuration to two standard baselines: a BLSTM tagger and a CRF. The SPEN achieves higher validation and test accuracies with faster test-time inference. While our method is slower than the baselines during training, it is faster than the CRF at test time, operating at essentially the same speed as the BLSTM baseline while being more accurate.

[2]The pretrained embeddings are the same as those used by Tu et al. [2017] and are available at `http://ttic.uchicago.edu/~lifu/`

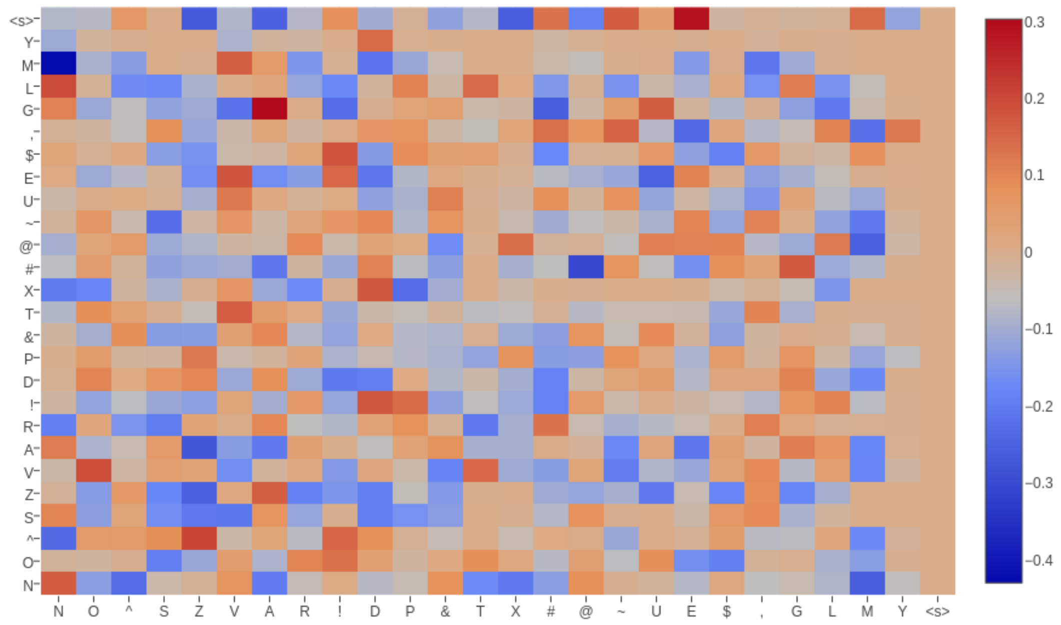


Figure 3: Learned pairwise potential matrix for Twitter POS tagging.

	validation accuracy (%)	test accuracy (%)	training speed (examples/sec)	testing speed (examples/sec)
BLSTM	88.6	88.8	385	1250
CRF	89.1	89.2	250	500
SPEN (InfNet)	89.6	89.8	125	1250

Table 7: Twitter POS accuracies of BLSTM, CRF, and SPEN (InfNet), using our tuned SPEN configuration (slack-rescaled hinge, inference network trained with cross entropy term). Though slowest to train, the SPEN matches the test-time speed of the BLSTM while achieving the highest accuracies.

6.4.3 Tag Language Model

The above results only use the pairwise energy. In order to capture long-distance dependencies in an entire sequence of labels, we define an additional energy term $E^{\text{TLM}}(\boldsymbol{y})$ based on the pretrained TLM. If the argument $\boldsymbol{y}$ consisted of one-hot vectors, we could simply compute its likelihood. However, to support relaxed $\boldsymbol{y}$'s, we need to define a more general function:

$$E^{\text{TLM}}(\boldsymbol{y}) = -\sum_{t=1}^{|\boldsymbol{y}|+1} \log(y_t^\top \,\text{TLM}(\langle y_0, ..., y_{t-1}\rangle)) \tag{19}$$

where y_0 is the start-of-sequence symbol, $y_{|\boldsymbol{y}|+1}$ is the end-of-sequence symbol, and $\text{TLM}(\langle y_0, ..., y_{t-1}\rangle)$ returns the softmax distribution over tags at position t (under the pretrained tag language model) given the preceding tag vectors. When each y_t is a one-hot vector, this energy reduces to the negative log-likelihood of the tag sequence specified by $\boldsymbol{y}$.

We define the new joint energy as the sum of the energy function in Eq. (7) and the TLM energy function in Eq. (19). During learning, we keep the TLM parameters fixed to their pretrained values, but we tune the weight of the TLM energy (over the set $\{0.1, 0.2, 0.5\}$) in the joint energy. We train SPENs with the new joint energy using the margin-rescaled hinge, training the inference network with the cross entropy term.

Setup To compute the TLM energy term, we first automatically tag unlabeled tweets, then train an LSTM language model on the automatic tag sequences. When doing so, we define the input tag embeddings to be L-dimensional one-hot vectors specifying the tags in the training sequences. This is nonstandard compared to standard language modeling. In standard language modeling, we train on observed sequences

	val. accuracy (%)	test accuracy (%)
-TLM	89.8	89.6
+TLM	89.9	90.2

Table 8: Twitter POS validation/test accuracies when adding tag language model (TLM) energy term to a SPEN trained with margin-rescaled hinge.

		predicted tags	
#	tweet (target word in bold)	-TLM	+TLM
1	... that's a t-17 , technically . does **that** count as top-25 ?	determiner	pronoun
2	... lol you know im down **like** 4 flats on a cadillac ... lol ...	adjective	preposition
3	... them who he is : he wants her to **like** him for his pers ...	preposition	verb
4	I wonder when Nic Cage is going to **film** " Another Something Something Las Vegas " .	noun	verb
5	Cut my hair , **gag** and bore me	noun	verb
6	... they had their fun , we **hd** ours ! ;) lmaooo	proper noun	verb
7	" Logic will get you from A to **B** . Imagination will take you everywhere . " - Albert Einstein .	verb	noun
8	lmao I'm not a sheep who listens to it **cos** everyone else does ...	verb	preposition
9	Noo its not cuss you have swag **andd** you wont look dumb ! ...	noun	coord. conj.

Table 9: Examples of improvements in Twitter POS tagging when using tag language model (TLM). In all of these examples, the predicted tag when using the TLM matches the gold standard.

and compute likelihoods of other fully-observed sequences. However, in our case, we train on tag sequences but we want to use the same model on sequences of tag *distributions* produced by an inference network. We train the TLM on sequences of one-hot vectors and then use it to compute likelihoods of sequences of tag distributions.

To obtain training data for training the tag language model, we run the Twitter POS tagger from Owoputi et al. [2013] on a dataset of 303K randomly-sampled English tweets. We train the tag language model on 300K tweets and use the remaining 3K for tuning hyperparameters and early stopping. We train an LSTM language model on the tag sequences using stochastic gradient descent with momentum and early stopping on the validation set. We used a dropout rate of 0.5 for the LSTM hidden layer. We tune the learning rate ($\{0.1, 0.2, 0.5, 1.0\}$), the number of LSTM layers ($\{1, 2\}$), and the hidden layer size ($\{50, 100, 200\}$).

Results Table 8 shows results.[3] Adding the TLM energy leads to a gain of 0.6 on the test set. Other settings showed more variance; when using slack-rescaled hinge, we found a small drop on test, while when simply training inference networks for a fixed, pretrained joint energy with tuned mixture coefficient, we found a gain of 0.3 on test when adding the TLM energy. We investigated the improvements and found some to involve corrections that seemingly stem from handling non-local dependencies better.

Table 9 shows examples in which our SPEN that includes the TLM appears to be using broader context when making tagging decisions. These are examples from the test set labeled by two models: the SPEN without the TLM (which achieves 89.6% accuracy, as shown in Table 8) and the SPEN with the TLM (which reaches 90.2% accuracy). In example 1, the token "that" is predicted to be a determiner based on local context, but is correctly labeled a pronoun when using the TLM. This example is difficult because of the noun/verb tag ambiguity of the next word ("count") and its impact on the tag for "that". Examples 2 and 3 show two corrections for the token "like", which is a highly ambiguous word in Twitter POS tagging. The broader context makes it much clearer which tag is intended.

The next two examples (4 and 5) are cases of noun/verb ambiguity that are resolvable with larger context. The last four examples show improvements for nonstandard word forms. The shortened form of "had" (example 6) is difficult to tag due to its collision with "HD" (high-definition), but the model with the TLM is able to tag it correctly. In example 7, the ambiguous token "b" is frequently used as a short form of

[3]The baseline results differ slightly from earlier results because we found that we could achieve higher accuracies in SPEN training by avoiding using pretrained feature network parameters for the inference network.

"be" on Twitter, and since it comes after "to" in this context, the verb interpretation is encouraged. However, the broader context makes it clear that it is not a verb and the TLM-enriched model tags it correctly. The words in the last two examples are nonstandard word forms that were not observed in the training data, which is likely the reason for their erroneous predictions. When using the TLM, we can better handle these rare forms based on the broader context. These results suggest that our method of training inference networks can be used to add rich features to structured prediction, though we leave a thorough exploration of global energies to future work.

7 Joint Parameterizations for Inference Networks

In Section 6, we jointly train the cost-augmented inference network and energy network, then do fine-tuning of the cost-augmented inference network to make it more like a test-time inference network. We propose a different loss that separates the two inference networks and trains them jointly:

$$\min_{\Theta} \frac{\lambda}{n}\sum_{i=1}^{n}\left[\max_{\boldsymbol{y}}(-E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i))\right]_+ + \frac{1}{n}\sum_{i=1}^{n}\left[\max_{\boldsymbol{y}}(\triangle(\boldsymbol{y},\boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i))\right]_+$$

The above objective contains two different inference problems, which are also the two inference problems that must be solved in structured max-margin learning, whether during training or during test-time inference. Eq. (1) shows the test-time inference problem. The other one is cost-augmented inference, defined as follows:

$$\operatorname*{argmin}_{\boldsymbol{y}'\in\mathcal{Y}(\boldsymbol{x})}(E_\Theta(\boldsymbol{x},\boldsymbol{y}) - \triangle(\boldsymbol{y}',\boldsymbol{y})) \tag{20}$$

This inference problem involves finding an output with low energy but high cost relative to the gold standard output. Thus, it is not well-aligned with the test-time inference problem. In Section 6, we used the same inference network for solving both problems, which led them to have to perform fine-tuning at test-time with a different objective. We avoid this issue by instead jointly training two inference networks, one for cost-augmented inference and the other for test-time inference:

$$\min_{\Theta}\max_{\Phi,\Psi}\sum_{\langle\boldsymbol{x}_i,\boldsymbol{y}_i\rangle\in\mathcal{D}} \underbrace{[\triangle(\mathbf{F}_\Phi(\boldsymbol{x}),\boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i,\mathbf{F}_\Phi(\boldsymbol{x})) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i)]_+}_{\text{margin-rescaled loss}} + \lambda\underbrace{[-E_\Theta(\boldsymbol{x}_i,\mathbf{A}_\Psi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i)]_+}_{\text{perceptron loss}} \tag{21}$$

We treat this optimization problem as a minmax game and find a saddle point for the game similar to Section 6 and Goodfellow et al. [2014]. We alternatively optimize Θ, Φ and Ψ.

We drop the zero truncation $(\max(0,.))$ when updating the inference network parameters to improve stability during training. This also lets us remove the terms that do not have inference networks.

When we remove the truncation at 0, the objective for the inference network parameters is:

$$\hat{\Psi},\hat{\Phi} \leftarrow \operatorname*{argmax}_{\Psi,\Phi} \triangle(\mathbf{F}_\Phi(\boldsymbol{x}),\boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i,\mathbf{F}_\Phi(\boldsymbol{x})) - \lambda E_\Theta(\boldsymbol{x}_i,\mathbf{A}_\Psi(\boldsymbol{x}_i))$$

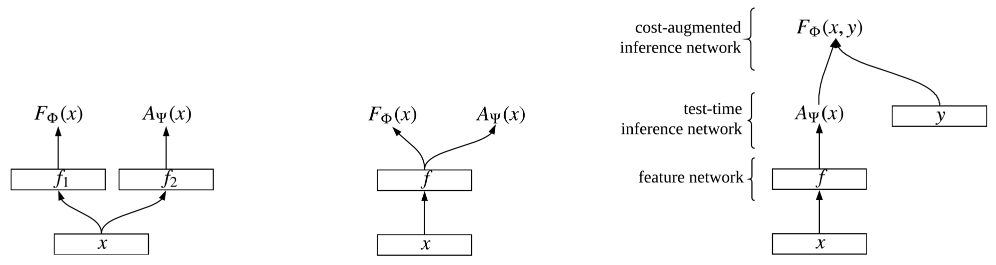
The objective for the energy function is:

$$\hat{\Theta} \leftarrow \operatorname*{argmin}_{\Theta}\big[\triangle(\mathbf{F}_\Phi(\boldsymbol{x}),\boldsymbol{y}_i) - E_\Theta(\boldsymbol{x}_i,\mathbf{F}_\Phi(\boldsymbol{x})) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i)\big]_+ + \lambda\big[-E_\Theta(\boldsymbol{x}_i,\mathbf{A}_\Psi(\boldsymbol{x}_i)) + E_\Theta(\boldsymbol{x}_i,\boldsymbol{y}_i)\big]_+$$

The new objective jointly trains the energy function E_Θ, cost-augmented inference network $\mathbf{F}_\Phi$, and test-time inference network $\mathbf{A}_\Psi$. This objective offers us several options for defining joint parameterizations of the two inference networks.

We consider three options which are visualized in Figure 4 and described below:

- (a) Separated: $\mathbf{F}_\Phi$ and $\mathbf{A}_\Psi$ are two independent networks with their own architectures and parameters as shown in Figure 4(a).
- (b) Shared: $\mathbf{F}_\Phi$ and $\mathbf{A}_\Psi$ share the feature network as shown in Figure 4(b). We consider this option because both $\mathbf{F}_\Phi$ and $\mathbf{A}_\Psi$ are trained to produce output labels with low energy. However $\mathbf{F}_\Phi$ also needs to produce output labels with high cost $\triangle$ (i.e., far away from the ground truth).



(a) separated networks (b) shared feature networks (c) stacked networks with y as extra input to F_Φ

Figure 4: Parameterizations for cost-augmented inference network $\mathbf{F}_\Phi$ and test-time inference network $\mathbf{A}_\Psi$.

- (c) Stacked: Here, the cost-augmented network is a function of the output of the test-time inference network and the gold standard output $\boldsymbol{y}$ is included as an additional input to the cost-augmented network. That is, $\mathbf{F}_\Phi = f(\mathbf{A}_\Psi(\boldsymbol{x}), \boldsymbol{y})$ where f is a parameterized function. This is depicted in Figure 4(c). Note that we block the gradient at $\mathbf{A}_\Psi$ when updating Ψ.

For the third option, we will consider multiple choices for the function f. One choice is to use an affine transform on the concatenation of the inference network and the ground truth label:

$$\mathbf{F}_\Phi(\boldsymbol{x}, \boldsymbol{y})_i = \mathrm{softmax}(W[\mathbf{A}_\Psi(\boldsymbol{x})_i; \boldsymbol{y}_i] + b)$$

where semicolon (;) denotes vertical concatenation, L is the label set size, $\boldsymbol{y}_i \in \mathbb{R}^L$ (position i of $\boldsymbol{y}$) is a one-hot vector, $\mathbf{A}_\Psi(\boldsymbol{x})_i$ and $\mathbf{F}_\Phi(\boldsymbol{x})_i$ are position i of $\mathbf{A}_\Psi$ and $\mathbf{F}_\Phi$, and W is a $2L$ by L parameter matrix. Another choice of f is a BiLSTM:

$$\mathbf{F}_\Phi(\boldsymbol{x}, \boldsymbol{y})_i = \mathrm{BiLSTM}([\mathbf{A}_\Psi(\boldsymbol{x}); \boldsymbol{y}])$$

We could have $\boldsymbol{y}$ as input to the other architectures, but we limit our search to these three options. One motivation for these parameterizations is to reduce the total number of parameters in the procedure. Generally, the number of parameters is expected to decrease when moving from option (a) to (b), and when moving from (b) to (c). We will compare the three options empirically in our experiments, in terms of both accuracy and number of parameters.

Another motivation, specifically for the third option, is to distinguish the two inference networks in terms of their learned functionality. With all three parameterizations, the cost-augmented network will be trained to produce an output that differs from the ground truth, due to the presence of the $\triangle(\mathbf{F}_\Phi(\boldsymbol{x}), \boldsymbol{y}_i)$ term. However, in section 6 we found that the trained cost-augmented network was barely affected by fine-tuning for the test-time inference objective. This suggests that the cost-augmented network was mostly acting as a test-time inference network by the time of convergence. With the third parameterization above, however, we explicitly provide the ground truth output $\boldsymbol{y}$ to the cost-augmented network, permitting it to learn to change the predictions of the test-time network in appropriate ways to improve the energy function. We will explore this effect quantitatively and qualitatively below in our experiments.

7.1 Results and Analysis

Effect of Removing Truncation. Table 10 shows results for the margin-rescaled and perceptron losses when considering the removal of zero truncation and its interaction with the use of the local CE term. Training fails for both tasks when using zero truncation without the CE term. Removing truncation makes learning succeed and leads to effective models even without using CE. However, when using the local CE term, truncation has little effect on performance. The importance of CE in Section 6 is likely due to the fact that truncation was being used.

Effect of Local CE. The local cross entropy (CE) term is useful for both tasks, though it appears more helpful for tagging. This may be because POS tagging is a more local task. Regardless, for both tasks, the inclusion of the CE term speeds convergence and improves training stability. For example, on NER, using the CE term reduces the number of epochs chosen by early stopping from $\sim$100 to $\sim$25. On Twitter POS Tagging, using the CE term reduces the number of epochs chosen by early stopping from $\sim$150 to $\sim$60.

	zero trunc.	CE	POS acc (%)	NER F1 (%)	NER+ F1 (%)
margin-rescaled	yes	no	13.9	3.91	3.91
	no	no	87.9	85.1	88.6
	yes	yes	89.4*	85.2*	89.5*
	no	yes	89.4	85.2	89.5
perceptron	no	no	88.2	84.0	88.1
	no	yes	88.6	84.7	89.0

Table 10: Test set results for Twitter POS tagging and NER of several SPEN configurations. Results with * correspond to the setting of Section 6.

	POS				NER				NER+
	acc (%)	$\|T\|$	$\|I\|$	speed	F1 (%)	$\|T\|$	$\|I\|$	speed	F1 (%)
BiLSTM	88.8	166K	166K	–	84.9	239K	239K	–	89.3
SPENs with inference networks in Section 6:									
margin-rescaled	89.4	333K	166K	–	85.2	479K	239K	–	89.5
perceptron	88.6	333K	166K	–	84.4	479K	239K	–	89.0
SPENs with inference networks, compound objective, CE, no zero truncation (this paper):									
separated	89.7	500K	166K	66	85.0	719K	239K	32	89.8
shared	89.8	339K	166K	78	85.6	485K	239K	38	90.1
stacked	**89.8**	**335K**	**166K**	**92**	**85.6**	**481K**	**239K**	**46**	**90.1**

Table 11: Test set results for Twitter POS tagging and NER. $|T|$ is the number of trained parameters; $|I|$ is the number of parameters needed during the inference procedure. Training speeds (examples/second) are shown for joint parameterizations to compare them in terms of efficiency. Best setting (highest performance with fewest parameters and fastest training) is in boldface.

Effect of Compound Objective and Joint Parameterizations. The compound objective is the sum of the margin-rescaled and perceptron losses, and outperforms them both (see Table 11). Across all tasks, the shared and stacked parameterizations are more accurate than the previous objectives. For the separated parameterization, the performance drops slightly for NER, likely due to the larger number of parameters. The shared and stacked options also have fewer parameters to train than the separated option, and the stacked version processes examples at the fastest rate during training.

The left part of Table 12 shows how the performance of the test-time inference network $\mathbf{A}_\Psi$ and the cost-augmented inference network $\mathbf{F}_\Phi$ vary when using the new compound objective. The differences between $\mathbf{F}_\Phi$ and $\mathbf{A}_\Psi$ are larger than in the baseline configuration, showing that the two are learning complementary functionality.

With the stacked parameterization, the cost-augmented network $\mathbf{F}_\Phi$ receives as an additional input the gold standard label sequence, which leads to the largest differences as the cost-augmented network can explicitly favor incorrect labels.[4]

The right part of Table 12 shows qualitative differences between the two inference networks. On the POS development set, we count the differences between the predictions of $\mathbf{A}_\Psi$ and $\mathbf{F}_\Phi$ when $\mathbf{A}_\Psi$ makes the correct prediction.[5] The most frequent combinations show that $\mathbf{F}_\Phi$ tends to output tags that are highly confusable with those output by $\mathbf{A}_\Psi$. For example, it often outputs proper noun when the gold standard is common noun or vice versa. It also captures the noun-verb ambiguity and ambiguities among adverbs, adjectives, and prepositions.

[4]We also tried a BiLSTM in the final layer of the stacked parameterization but results were similar to the simpler affine architecture, so we only report results here with the affine architecture.

[5]We used the stacked parameterization.

		POS	NER
		$\mathbf{A}_\Psi - \mathbf{F}_\Phi$	$\mathbf{A}_\Psi - \mathbf{F}_\Phi$
margin-rescaled		0.2	0
compound	separated	2.2	0.4
	shared	1.9	0.5
	stacked	**2.6**	**1.7**

test-time ($\mathbf{A}_\Psi$)	cost-augmented ($\mathbf{F}_\Phi$)
common noun	proper noun
proper noun	common noun
common noun	adjective
proper noun	proper noun + possessive
adverb	adjective
preposition	adverb
adverb	preposition
verb	common noun
adjective	verb

Table 12: Top: differences in accuracy/F1 between test-time inference networks $\mathbf{A}_\Psi$ and cost-augmented networks $\mathbf{F}_\Phi$ (on development sets). The "margin-rescaled" row uses a SPEN with the local CE term and without zero truncation, where $\mathbf{A}_\Psi$ is obtained by fine-tuning $\mathbf{F}_\Phi$ as done in Section 6. Bottom: most frequent output differences between $\mathbf{A}_\Psi$ and $\mathbf{F}_\Phi$ on the development set.

8 Energy-Based Inference Networks for Non-Autoregressive Machine Translation

In this section, inference network is applied for non-autoregressive machine translation model training. Our approach, which we call ENGINE (ENerGy-based Inference NEtworks), achieves state-of-the-art non-autoregressive results on the IWSLT 2014 DE-EN and WMT 2016 RO-EN datasets, approaching the performance of autoregressive models.

8.1 Generalized Energy and Inference Network for NMT

Most neural machine translation (NMT) systems model the conditional distribution $p_\Theta(\boldsymbol{y} \mid \boldsymbol{x})$ of a target sequence $\boldsymbol{y} = \langle y_1, y_2, ..., y_T\rangle$ given a source sequence $\boldsymbol{x} = \langle x_1, x_2, ..., x_{T_s}\rangle$, where each y_t comes from a vocabulary $\mathcal{V}$, y_T is $\langle eos\rangle$, and y_0 is $\langle bos\rangle$. It is common in NMT to define this conditional distribution using an "autoregressive" factorization [Sutskever et al., 2014a, Bahdanau et al., 2015, Vaswani et al., 2017]:

$$\log p_\Theta(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{t=1}^{|\boldsymbol{y}|} \log p_\Theta(y_t \mid \boldsymbol{y}_{0:t-1}, \boldsymbol{x})$$

This model can be viewed as an energy-based model [LeCun et al., 2006] by defining the **energy function** $E_\Theta(\boldsymbol{x}, \boldsymbol{y}) = -\log p_\Theta(\boldsymbol{y} \mid \boldsymbol{x})$. Given trained parameters Θ, test time inference seeks to find the translation for a given source sentence $\boldsymbol{x}$ with the lowest energy: $\hat{\boldsymbol{y}} = \text{argmin}_{\boldsymbol{y}}\ E_\Theta(\boldsymbol{x}, \boldsymbol{y})$.

Finding the translation that minimizes the energy involves combinatorial search. We train **inference networks** to perform this search approximately. The idea of this approach is to replace the test time combinatorial search typically employed in structured prediction with the output of a network trained to produce approximately optimal predictions as shown in Section 5 and Section 6. More formally, we define an inference network $\mathbf{A}_\Psi$ which maps an input $\boldsymbol{x}$ to a translation $\boldsymbol{y}$ and is trained with the goal that $\mathbf{A}_\Psi(\boldsymbol{x}) \approx \text{argmin}_{\boldsymbol{y}}\ E_\Theta(\boldsymbol{x}, \boldsymbol{y})$.

Specifically, we train the inference network parameters Ψ as follows (assuming Θ is pretrained and fixed):

$$\widehat{\Psi} = \underset{\Psi}{\text{argmin}} \sum_{\langle \boldsymbol{x}, \boldsymbol{y}\rangle \in \mathcal{D}} E_\Theta(\boldsymbol{x}, \mathbf{A}_\Psi(\boldsymbol{x})) \tag{22}$$

where $\mathcal{D}$ is a training set of sentence pairs. The network architecture of $\mathbf{A}_\Psi$ can be different from the architectures used in the energy function. In this paper, we combine an autoregressive energy function with a non-autoregressive inference network. By doing so, we seek to combine the effectiveness of the autoregressive energy with the fast inference speed of a non-autoregressive network.

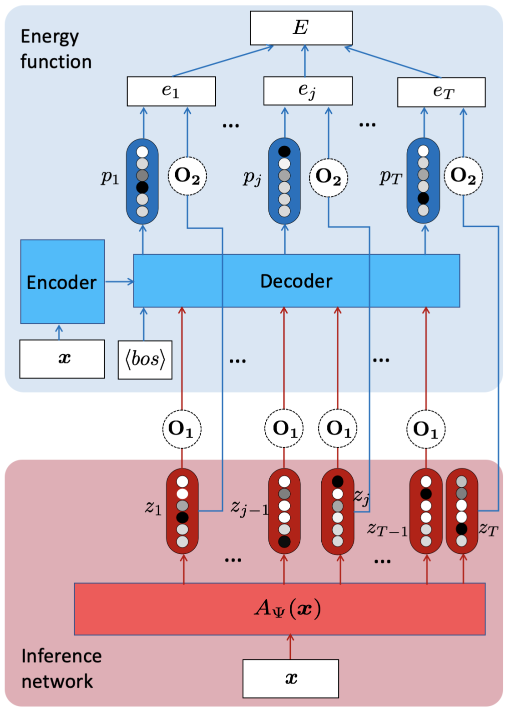


Figure 5: The model for learning test-time inference networks for NAT-NMT when the energy function $E_\Theta(\boldsymbol{x}, \boldsymbol{y})$ is a pretrained seq2seq model with attention.

In order to allow for gradient-based optimization of the inference network parameters Ψ, we now define a more general family of energy functions for NMT. First, we change the representation of the translation $\boldsymbol{y}$ in the energy, redefining $\boldsymbol{y} = \langle \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{|\boldsymbol{y}|} \rangle$ as a sequence of ***distributions*** over words instead of a sequence of words.

In particular, we consider the generalized energy

$$E_\Theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{|\boldsymbol{y}|} e_t(\boldsymbol{x}, \boldsymbol{y}) \tag{23}$$

where

$$e_t(\boldsymbol{x}, \boldsymbol{y}) = -\boldsymbol{y}_t^\top \log p_\Theta(\cdot \mid \boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{x}). \tag{24}$$

We use the $\cdot$ notation in $p_\Theta(\cdot \mid \ldots)$ above to indicate that we may need the full distribution over words. Note that by replacing the $\boldsymbol{y}_t$ with one-hot distributions we recover the original energy.

In order to train an inference network to minimize this energy, we simply need a network architecture that can produce a sequence of word distributions, which is satisfied by recent non-autoregressive NMT models [Ghazvininejad et al., 2019]. However, because the distributions involved in the original energy are one-hot, it may be advantageous for the inference network too to output distributions that are one-hot or approximately so. We will accordingly view inference networks as producing a sequence of T logit vectors $\boldsymbol{z}_t \in \mathbb{R}^{|\mathcal{V}|}$, and we will consider two operators $\mathbf{O_1}$ and $\mathbf{O_2}$ that will be used to map these $\boldsymbol{z}_t$ logits into distributions for use in the energy. Figure 5 provides an overview of our approach, including this generalized energy function, the inference network, and the two operators $\mathbf{O_1}$ and $\mathbf{O_2}$.

8.2 Choices for Operators

The choices we consider for $\mathbf{O_1}$ and $\mathbf{O_2}$, which we present generically for operator $\mathbf{O}$ and logit vector $\boldsymbol{z}$, are shown in Table 13, and described in more detail below. Some of these $\mathbf{O}$ operations are not differentiable, and so the Jacobian matrix $\frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ must be approximated during learning; we show the approximations we use in Table 13 as well.

We consider five choices for each $\mathbf{O}$:

(a) **SX**: softmax. Here $\mathbf{O}(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z})$; no Jacobian approximation is necessary.

	$\mathbf{O}(\boldsymbol{z})$	$\frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$
SX	$\boldsymbol{q}$	$\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{z}}$
STL	$\textit{onehot}(\text{argmax}(\boldsymbol{z}))$	$\boldsymbol{I}$
SG	$\textit{onehot}(\text{argmax}(\tilde{\boldsymbol{q}}))$	$\frac{\partial \tilde{\boldsymbol{q}}}{\partial \tilde{\boldsymbol{z}}}$
ST	$\textit{onehot}(\text{argmax}(\boldsymbol{q}))$	$\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{z}}$
GX	$\tilde{\boldsymbol{q}}$	$\frac{\partial \tilde{\boldsymbol{q}}}{\partial \tilde{\boldsymbol{z}}}$

Table 13: Let $\mathbf{O}(\boldsymbol{z}) \in \Delta^{|\mathcal{V}|-1}$ be the result of applying an $\mathbf{O_1}$ or $\mathbf{O_2}$ operation to logits $\boldsymbol{z}$ output by the inference network. Also let $\tilde{\boldsymbol{z}} = \boldsymbol{z} + \boldsymbol{g}$, where $\boldsymbol{g}$ is Gumbel noise, $\boldsymbol{q} = \text{softmax}(\boldsymbol{z})$, and $\tilde{\boldsymbol{q}} = \text{softmax}(\tilde{\boldsymbol{z}})$. We show the Jacobian (approximation) $\frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ we use when computing $\frac{\partial \ell_{\text{loss}}}{\partial \boldsymbol{z}} = \frac{\partial \ell_{\text{loss}}}{\partial \mathbf{O}(\boldsymbol{z})} \frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$, for each $\mathbf{O}(\boldsymbol{z})$ considered.

(b) **STL**: straight-through logits. Here $\mathbf{O}(\boldsymbol{z}) = \textit{onehot}(\text{argmax}_i\, \boldsymbol{z})$. $\frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ is approximated by the identity matrix $\boldsymbol{I}$ (see Bengio et al. [2013]).

(c) **SG**: straight-through Gumbel-Softmax. Here $\mathbf{O}(\boldsymbol{z}) = \textit{onehot}(\text{argmax}_i\, \text{softmax}(\boldsymbol{z} + \boldsymbol{g}))$, where g_i is Gumbel noise.[6] $\frac{\partial \mathbf{O}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ is approximated with $\frac{\partial\, \text{softmax}(\boldsymbol{z}+\boldsymbol{g})}{\partial \boldsymbol{z}}$ [Jang et al., 2016].

(d) **ST**: straight-through. This setting is identical to SG with $\boldsymbol{g} = \mathbf{0}$ (see Bengio et al. [2013]).

(e) **GX**: Gumbel-Softmax. Here $\mathbf{O}(\boldsymbol{z}) = \text{softmax}(\boldsymbol{z} + \boldsymbol{g})$, where again g_i is Gumbel noise; no Jacobian approximation is necessary.

$\mathbf{O_1} \setminus \mathbf{O_2}$	SX	STL	SG	ST	GX
SX	**55 (20.2)**	256 (0)	56 (19.6)	**55 (20.1)**	55 (19.6)
STL	97 (14.8)	164 (8.2)	94 (13.7)	95 (14.6)	190 (0)
SG	82 (15.2)	206 (0)	81 (14.7)	82 (15.0)	83 (13.5)
ST	81 (14.7)	170 (0)	81 (14.4)	80 (14.3)	83 (13.7)
GX	**53 (19.8)**	201 (0)	56 (18.3)	54 (19.6)	55 (19.4)

(a) seq2seq AR energy, BiLSTM inference networks

$\mathbf{O_1} \setminus \mathbf{O_2}$	SX	STL	SG	ST	GX
SX	**80 (31.7)**	133 (27.8)	81 (31.5)	**80 (31.7)**	81 (31.6)
STL	186 (25.3)	133 (27.8)	95 (20.0)	97 (30.1)	180 (26.0)
SG	98 (30.1)	133 (27.8)	95 (30.1)	97 (30.0)	97 (29.8)
ST	98 (30.2)	133 (27.8)	95 (30.0)	97 (30.1)	97 (30.0)
GX	81 (31.5)	133 (27.8)	81 (31.2)	81 (31.5)	81 (31.4)

(b) transformer AR energy, CMLM inference networks

Table 14: Comparison of operator choices in terms of energies (BLEU scores) on the IWSLT14 DE-EN dev set with two energy/inference network combinations. Oracle lengths are used for decoding. $\mathbf{O_1}$ is the operation for feeding inference network outputs into the decoder input slots in the energy. $\mathbf{O_2}$ is the operation for computing the energy on the output. Each row corresponds to the same $\mathbf{O_1}$, and each column corresponds to the same $\mathbf{O_2}$.

	IWSLT14 DE-EN		WMT16 RO-EN	
	Energy (↓)	BLEU (↑)	Energy (↓)	BLEU (↑)
baseline	153.54	8.28	175.94	9.47
distill	112.36	14.58	205.71	5.76
ENGINE	51.98	19.55	64.03	21.69

Table 15: Test results of non-autoregressive models when training with the references ("baseline"), distilled outputs ("distill"), and energy ("ENGINE"). Oracle lengths are used for decoding. Here, ENGINE uses BiLSTM inference networks and pretrained seq2seq AR energies. ENGINE outperforms training on both the references and a pseudocorpus.

[6] $g_i = -\log(-\log(u_i))$ and $u_i \sim \textbf{Uniform}(0, 1)$.

	IWSLT14 DE-EN		WMT16 RO-EN	
	# iterations		# iterations	
	1	10	1	10
CMLM	28.11	33.39	28.20	33.31
ENGINE	31.99	33.17	33.16	34.04

Table 16: Test BLEU scores of non-autoregressive models using no refinement (# iterations = 1) and using refinement (# iterations = 10). Note that the # iterations = 1 results are purely non-autoregressive. ENGINE uses a CMLM as the inference network architecture and the transformer AR energy. The length beam size is 5 for CMLM and 3 for ENGINE.

8.3 Experimental Setup

8.3.1 Datasets

We evaluate our methods on two datasets: IWSLT14 German (DE) $\rightarrow$ English (EN) and WMT16 Romanian (RO) $\rightarrow$ English (EN). All data are tokenized and then segmented into subword units using byte-pair encoding [Sennrich et al., 2016]. We use the data provided by Lee et al. [2018] for RO-EN.

8.3.2 Autoregressive Energies

We consider two architectures for the pretrained autoregressive (AR) energy function. The first is an autoregressive sequence-to-sequence (seq2seq) model with attention [Luong et al., 2015]. The encoder is a two-layer BiLSTM with 512 units in each direction, the decoder is a two-layer LSTM with 768 units, and the word embedding size is 512. The second is an autoregressive transformer model [Vaswani et al., 2017], where both the encoder and decoder have 6 layers, 8 attention heads per layer, model dimension 512, and hidden dimension 2048.

8.3.3 Inference Network Architectures

We choose two different architectures: a BiLSTM "tagger" (a 2-layer BiLSTM followed by a fully-connected layer) and a conditional masked language model (CMLM; Ghazvininejad et al., 2019), a transformer with 6 layers per stack, 8 attention heads per layer, model dimension 512, and hidden dimension 2048. Both architectures require the target sequence length in advance; methods for handling length are discussed in Sec. 8.3.4. For baselines, we train these inference network architectures as non-autoregressive models using the standard per-position cross-entropy loss. For faster inference network training, we initialize inference networks with the baselines trained with cross-entropy loss in our experiments.

The baseline CMLMs use the partial masking strategy described by Ghazvininejad et al. [2019]. This involves using some masked input tokens and some provided input tokens during training. At test time, multiple iterations ("refinement iterations") can be used for improved results [Ghazvininejad et al., 2019]. Each iteration uses partially-masked input from the preceding iteration. We consider the use of multiple refinement iterations for both the CMLM baseline and the CMLM inference network.[7]

8.3.4 Predicting Target Sequence Lengths

Non-autoregressive models often need a target sequence length in advance [Lee et al., 2018]. We report results both with oracle lengths and with a simple method of predicting it. We follow Ghazvininejad et al. [2019] in predicting the length of the translation using a representation of the source sequence from the encoder. The length loss is added to the cross-entropy loss for the target sequence. During decoding, we select the top $k = 3$ length candidates with the highest probabilities, decode with the different lengths in parallel, and return the translation with the highest average of log probabilities of its tokens.

8.4 Results

Effect of choices for $\mathbf{O_1}$ and $\mathbf{O_2}$. Table 14 compares various choices for the operations $\mathbf{O_1}$ and $\mathbf{O_2}$. For subsequent experiments, we choose the setting that feeds the whole distribution into the energy function

[7]The CMLM inference network is trained with full masking (no partial masking like in the CMLM baseline). However, since the CMLM inference network is initialized using the CMLM baseline, which is trained using partial masking, the CMLM inference network is still compatible with refinement iterations at test time.

	IWSLT14 DE-EN	WMT16 RO-EN
Autoregressive (Transformer)		
Greedy Decoding	33.00	33.33
Beam Search	34.11	34.07
Non-autoregressive		
Iterative Refinement [Lee et al., 2018]	-	$25.73^{\dagger}$
NAT with Fertility [Gu et al., 2018]	-	$29.06^{\dagger}$
CTC [Libovický and Helcl, 2018]	-	$24.71^{\dagger}$
FlowSeq [Ma et al., 2019]	$27.55^{\dagger}$	$30.44^{\dagger}$
CMLM [Ghazvininejad et al., 2019]	28.25	$28.20^{\dagger}$
Bag-of-ngrams-based loss [Shao et al., 2020]	-	$29.29^{\dagger}$
AXE CMLM [Ghazvininejad et al., 2020]	-	$31.54^{\dagger}$
Imputer-based model [Saharia et al., 2020]	-	$31.7^{\dagger}$
ENGINE (ours)	**31.99**	**33.16**

Table 17: BLEU scores on two datasets for several non-autoregressive methods. The inference network architecture is the CMLM. For methods that permit multiple refinement iterations (CMLM, AXE CMLM, ENGINE), one decoding iteration is used (meaning the methods are purely non-autoregressive). †Results are from the corresponding papers.

($\mathbf{O_1}$ = SX) and computes the loss with straight-through ($\mathbf{O_2}$ = ST). Using Gumbel noise in $\mathbf{O_2}$ has only minimal effect, and rarely helps. Using ST instead also speeds up training by avoiding the noise sampling step.

Training with distilled outputs vs. training with energy. We compared training non-autoregressive models using the references, distilled outputs, and as inference networks on both datasets. Table 15 shows the results when using BiLSTM inference networks and seq2seq AR energies. The inference networks improve over training with the references by 11.27 BLEU on DE-EN and 12.22 BLEU on RO-EN. In addition, inference networks consistently improve over non-autoregressive networks trained on the distilled outputs.

Impact of refinement iterations. Ghazvininejad et al. [2019] show improvements with multiple refinement iterations. Table 16 shows refinement results of CMLM and ENGINE. Both improve with multiple iterations, though the improvement is much larger with CMLM. However, even with 10 iterations, ENGINE is comparable to CMLM on DE-EN and outperforms it on RO-EN.

Comparison to other NAT models. Table 17 shows 1-iteration results on two datasets. To the best of our knowledge, ENGINE achieves state-of-the-art NAT performance: 31.99 on IWSLT14 DE-EN and 33.16 on WMT16 RO-EN. In addition, ENGINE achieves comparable performance with the autoregressive NMT model.

References

B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *Proc. of ICML*, 2017.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in NIPS*, 2014.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

A. Bakhtin, Y. Deng, S. Gross, M. Ott, M. Ranzato, and A. Szlam. Energy-based models for text, 2020.

D. Belanger and A. McCallum. Structured prediction energy networks. In *Proceedings of the 33rd International Conference on Machine Learning - Volume 48*, ICML'16, pages 983–992, 2016.

D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. In *Proc. of ICML*, 2017.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

K.-W. Chang, S. Upadhyay, G. Kundu, and D. Roth. Structural learning with amortized inference. In *Proc. of AAAI*, 2015.

Y. Chen, V. O. Li, K. Cho, and S. Bowman. A stable and effective learning strategy for trainable greedy decoding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 380–390, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1035. URL https://www.aclweb.org/anthology/D18-1035.

M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, 2002.

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

Z. Dai, A. Almahairi, B. Philip, E. Hovy, and A. Courville. Calibrating energy-based generative adversarial networks. In *Proc. of ICLR*, 2017.

L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

K. J. Geras, A. rahman Mohamed, R. Caruana, G. Urban, S. Wang, O. Aslan, M. Philipose, M. Richardson, and C. Sutton. Blending LSTMs into CNNs. In *Proc. of ICLR (workshop track)*, 2016.

S. Gershman and N. Goodman. Amortized inference in probabilistic reasoning. In *Proc. of the Cognitive Science Society*, 2014.

M. Ghazvininejad, O. Levy, Y. Liu, and L. Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6111–6120, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://www.aclweb.org/anthology/D19-1633.

M. Ghazvininejad, V. Karpukhin, L. Zettlemoyer, and O. Levy. Aligned cross entropy for non-autoregressive machine translation. *arXiv preprint arXiv:2004.01655*, 2020.

K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-speech tagging for Twitter: annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, Portland, Oregon, USA, June 2011.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in NIPS*, 2014.

K. Goyal, G. Neubig, C. Dyer, and T. Berg-Kirkpatrick. A continuous relaxation of beam search for end-to-end training of neural sequence models. In *Proc. of AAAI*, 2018.

C. Graber and A. G. Schwing. Graph Structured Prediction Energy Networks. In *Proc. NeurIPS*, 2019.

C. Graber, O. Meshi, and A. Schwing. Deep structured prediction with nonlinear output transformations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6320–6331. Curran Associates, Inc., 2018a. URL http://papers.nips.cc/paper/7869-deep-structured-prediction-with-nonlinear-output-transformations.pdf.

C. Graber, O. Meshi, and A. G. Schwing. Deep Structured Prediction with Nonlinear Output Transformations. In *Proc. NeurIPS*, 2018b.

W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkxzx0NtDB.

J. Gu, K. Cho, and V. O. Li. Trainable greedy decoding for neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1968–1978, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1210. URL https://www.aclweb.org/anthology/D17-1210.

J. Gu, J. Bradbury, C. Xiong, V. O. Li, and R. Socher. Non-autoregressive neural machine translation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. 2017. URL http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf.

M. Gutmann and A. Hyvarinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2015.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14 (8):1771–1800, 2002. URL http://www.ncbi.nlm.nih.gov/pubmed/12180402.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

C. D. V. Hoang, G. Haffari, and T. Cohn. Towards decoding as continuous optimisation in neural machine translation. In *Proc. of EMNLP*, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

J. Hockenmaier and M. Steedman. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*, 2002.

K. Hu, Z. Ou, M. Hu, and J. Feng. Neural crf transducers for sequence labeling. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2997–3001, 2019.

Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015. URL http://arxiv.org/abs/1508.01991.

J.-J. Hwang, T.-W. Ke, J. Shi, and S. X. Yu. Adversarial structure matching for structured prediction tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4056–4065, 2019.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. of ECCV*, 2016.

L. Kaiser, S. Bengio, A. Roy, A. Vaswani, N. Parmar, J. Uszkoreit, and N. Shazeer. Fast decoding in sequence models using discrete latent variables. In *International Conference on Machine Learning*, pages 2395–2404, 2018.

J. Kasai, J. Cross, M. Ghazvininejad, and J. Gu. Parallel machine translation with disentangled context transformer, 2020.

Y. Kim and A. M. Rush. Sequence-level knowledge distillation. In *Proc. of EMNLP*, 2016.

D. Kingma and M. Welling. Auto-encoding variational Bayes. *CoRR*, abs/1312.6114, 2013.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. 2009.

P. Krähenbühl and V. Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in NIPS*, 2011.

A. Kuncoro, M. Ballesteros, L. Kong, C. Dyer, and N. A. Smith. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proc. of EMNLP*, 2016.

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, 2001. ISBN 1-55860-778-1.

G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California, June 2016.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

J. Lee, E. Mansimov, and K. Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. URL https://www.aclweb.org/anthology/D18-1149.

C. Li and M. Wand. Precomputed real-time texture synthesis with Markovian generative adversarial networks. *CoRR*, abs/1604.04382, 2016.

J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1014. URL https://www.aclweb.org/anthology/N16-1014.

J. Libovický and J. Helcl. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1336. URL https://www.aclweb.org/anthology/D18-1336.

T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany, Aug. 2016.

X. Ma, C. Zhou, X. Li, G. Neubig, and E. Hovy. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4281–4291, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1437. URL https://www.aclweb.org/anthology/D19-1437.

A. F. T. Martins and J. Kreutzer. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proc. of EMNLP*, 2017.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in NIPS*, 2013.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

H. Mobahi, M. Farajtabar, and P. L. Bartlett. Self-distillation amplifies regularization in hilbert space. *CoRR*, abs/2002.05715, 2020. URL https://arxiv.org/abs/2002.05715.

A. Mordvintsev, C. Olah, and M. Tyka. DeepDream-a code example for visualizing neural networks. *Google Research*, 2015.

M. Mostajabi, M. Maire, and G. Shakhnarovich. Regularizing deep networks by modeling and predicting label structure. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proc. of NAACL*, 2013.

B. Paige and F. Wood. Inference networks for sequential Monte Carlo in graphical models. In *Proc. of ICML*, 2016.

A. Passos, V. Kumar, and A. McCallum. Lexicon infused phrase embeddings for named entity resolution. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 78–86, Ann Arbor, Michigan, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-1609. URL https://www.aclweb.org/anthology/W14-1609.

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

C. Saharia, W. Chan, S. Saxena, and M. Norouzi. Non-autoregressive machine translation with latent alignments. *arXiv preprint arXiv:2004.07437*, 2020.

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. 2016. URL http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162.

C. Shao, J. Zhang, Y. Feng, F. Meng, and J. Zhou. Minimizing the bag-of-ngrams difference for non-autoregressive neural machine translation. In *AAAI*, 2020.

V. Srikumar, G. Kundu, and D. Roth. On amortizing inference cost for structured prediction. In *Proc. of EMNLP*, 2012.

Z. Sun, Z. Li, H. Wang, D. He, Z. Lin, and Z. Deng. Fast structured decoding for sequence models. In *Advances in Neural Information Processing Systems 32*, pages 3016–3026. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8566-fast-structured-decoding-for-sequence-models.pdf.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in NIPS*, pages 3104–3112, 2014a.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. 2014b.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in NIPS*, 2004.

Y. W. Teh, M. Welling, S. Osindero, and G. E. Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260, Dec 2003.

E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of CONLL*, 2003.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2005.

L. Tu and K. Gimpel. Learning approximate inference networks for structured prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

L. Tu, K. Gimpel, and K. Livescu. Learning to embed words in context for syntactic tasks. In *Proc. of RepL4NLP*, 2017.

G. Urban, K. J. Geras, S. Ebrahimi Kahou, O. Aslan, S. Wang, R. Caruana, A.-r. Mohamed, M. Philipose, and M. Richardson. Do deep convolutional nets really need to be deep? *arXiv preprint arXiv:1603.05691*, 2016.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 13(2):260–269, 1967. URL http://dblp.uni-trier.de/db/journals/tit/tit13.html#Viterbi67.

B. Wang and Z. Ou. Improved training of neural trans-dimensional random field language models with dynamic noise-contrastive estimation. 2018.

S. Wang, S. Fidler, and R. Urtasun. Proximal deep structured models. In *Advances in NIPS*, 2016.

B. Wei, M. Wang, H. Zhou, J. Lin, and X. Sun. Imitation learning for non-autoregressive neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1304–1312, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1125. URL https://www.aclweb.org/anthology/P19-1125.

S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Proc. of EMNLP*, 2016.

Z. Zhang, X. Ma, and E. Hovy. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5592–5598, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1562. URL https://www.aclweb.org/anthology/P19-1562.

J. J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *CoRR*, 2016.

C. Zhou, J. Gu, and G. Neubig. Understanding knowledge distillation in non-autoregressive machine translation. In *International Conference on Learning Representations (ICLR)*, April 2020. URL https://arxiv.org/abs/1911.02727.