

Benchmarking Approximate Inference Methods for Neural Structured Prediction

Lifu Tu Kevin Gimpel

Toyota Technological Institute at Chicago, Chicago, IL, 60637, USA

{lifut, kgimpel}@ttic.edu

Abstract

Exact structured inference with neural network scoring functions is computationally challenging but several methods have been proposed for approximating inference. One approach is to perform gradient descent with respect to the output structure directly (Belanger and McCallum, 2016). Another approach, proposed recently, is to train a neural network (an “inference network”) to perform inference (Tu and Gimpel, 2018). In this paper, we compare these two families of inference methods on three sequence labeling datasets. We choose sequence labeling because it permits us to use exact inference as a benchmark in terms of speed, accuracy, and search error. Across datasets, 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. We find further benefit by combining inference networks and gradient descent, using the former to provide a warm start for the latter.¹

1 Introduction

Structured prediction models commonly involve complex inference problems for which finding exact solutions is intractable (Cooper, 1990). There are generally two ways to address this difficulty. One is to restrict the model family to those for which inference is feasible. For example, state-of-the-art methods for sequence labeling use structured energies that decompose into label-pair potentials and then use rich neural network architectures to define the potentials (Collobert et al., 2011; Lample et al., 2016; Ma and Hovy, 2016, *inter alia*). Exact dynamic programming algorithms like the Viterbi algorithm can be used for inference.

¹Code is available at github.com/lifut-tu/BenchmarkingApproximateInference

The second approach is to retain computationally-intractable scoring functions but then use approximate methods for inference. For example, some researchers relax the structured output space from a discrete space to a continuous one and then use gradient descent to maximize the score function with respect to the output (Belanger and McCallum, 2016). Another approach is to train a neural network (an “inference network”) to output a structure in the relaxed space that has high score under the structured scoring function (Tu and Gimpel, 2018). This idea was proposed as an alternative to gradient descent in the context of structured prediction energy networks (Belanger and McCallum, 2016).

In this paper, we empirically compare exact inference, gradient descent, and inference networks for three sequence labeling tasks. We train conditional random fields (CRFs) for sequence labeling with neural networks used to define the potentials. We choose a scoring function that permits exact inference via Viterbi so that we can benchmark the approximate methods in terms of search error in addition to speed and accuracy. We consider three families of neural network architectures to serve as inference networks: convolutional neural networks (CNNs), recurrent neural networks (RNNs), and sequence-to-sequence models with attention (seq2seq; Sutskever et al., 2014; Bahdanau et al., 2015). We also use multi-task learning while training inference networks, combining the structured scoring function with a local cross entropy loss.

Our empirical findings can be summarized as follows. Gradient descent works reasonably well for tasks with small label sets and primarily local structure, like part-of-speech tagging. However, gradient descent struggles on tasks with long-distance dependencies, even with small label set sizes. For tasks with large label set sizes, inference networks and Viterbi perform comparably,

with Viterbi taking much longer. In this regime, it is difficult for gradient descent to find a good solution, even with many iterations.

In comparing inference network architectures, (1) CNNs are the best choice for tasks with primarily local structure, like part-of-speech tagging; (2) RNNs can handle longer-distance dependencies while still offering high decoding speeds; and (3) seq2seq networks consistently work better than RNNs, but are also the most computationally expensive.

We also compare search error between gradient descent and inference networks and measure correlations with input likelihood. We find that inference networks achieve lower search error on instances with higher likelihood (under a pretrained language model), while for gradient descent the correlation between search error and likelihood is closer to zero. This shows the impact of the use of dataset-based learning of inference networks, i.e., they are more effective at amortizing inference for more common inputs.

Finally, we experiment with two refinements of inference networks. The first fine-tunes the inference network parameters for a single test example to minimize the energy of its output. The second uses an inference network to provide a warm start for gradient descent. Both lead to reductions in search error and higher accuracies for certain tasks, with the warm start method leading to a better speed/accuracy trade-off.

2 Sequence Models

For sequence labeling tasks, given an input sequence $\mathbf{x} = \langle x_1, x_2, \dots, x_{|\mathbf{x}|} \rangle$, we wish to output a sequence $\mathbf{y} = \langle y_1, y_2, \dots, y_{|\mathbf{x}|} \rangle \in \mathcal{Y}(\mathbf{x})$. Here $\mathcal{Y}(\mathbf{x})$ is the structured output space for \mathbf{x} . Each label y_t is represented as an L -dimensional one-hot vector where L is the number of labels.

Conditional random fields (CRFs; Lafferty et al., 2001) form one popular class of methods for structured prediction, especially for sequence labeling. We define our structured energy function to be similar to those often used in CRFs for sequence labeling:

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

where $y_{t,i}$ is the i th entry of the vector \mathbf{y}_t . In the standard discrete-label setting, each \mathbf{y}_t is a

one-hot vector, but this energy is generalized to be able to use both discrete labels and continuous relaxations of the label space, which we will introduce below. Also, we use $f(\mathbf{x}, t) \in \mathbb{R}^d$ to denote the “input feature vector” for position t , $\mathbf{u}_i \in \mathbb{R}^d$ is a label-specific parameter vector used for modeling the local scoring function, and $\mathbf{W} \in \mathbb{R}^{L \times L}$ is a parameter matrix learned to model label transitions. For the feature vectors we use a bidirectional long short-term memory (BLSTM; Hochreiter and Schmidhuber, 1997), so this forms a BLSTM-CRF (Lample et al., 2016; Ma and Hovy, 2016).

For training, we use the standard conditional log-likelihood objective for CRFs, using the forward and backward dynamic programming algorithms to compute gradients. For a given input \mathbf{x} at test time, prediction is done by choosing the output with the lowest energy:

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

The Viterbi algorithm can be used to solve this problem exactly for the energy defined above.

2.1 Modeling Improvements: BLSTM-CRF+

For our experimental comparison, we consider two CRF variants. The first is the basic model described above, which we refer to as BLSTM-CRF. Below we describe three additional techniques that we add to the basic model. We will refer to the CRF with these three techniques as BLSTM-CRF+. Using these two models permits us to assess the impact of model complexity and performance level on the inference method comparison.

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.

3 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:

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

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

There are multiple settings in which gradient descent has been used for structured inference, e.g., image generation (Johnson et al., 2016), structured prediction energy networks (Belanger and McCallum, 2016), and machine translation (Hoang et al., 2017). Gradient descent has the advantage of simplicity. Standard autodifferentiation toolkits can be used to compute gradients of the energy with respect to the output once the output space has been relaxed. However, one challenge is maintaining constraints on the variables being optimized. Therefore, we actually perform gradient descent in an even more relaxed output space $\mathcal{Y}_{R'}(\mathbf{x})$ which consists of length- $|\mathbf{x}|$ sequences of vectors, where each vector $\mathbf{y}_t \in \mathbb{R}^L$. When computing the energy, we use a softmax transformation on each \mathbf{y}_t , solving the following optimization problem with gradient descent:

$$\operatorname{argmin}_{\mathbf{y} \in \mathcal{Y}_{R'}(\mathbf{x})} E_{\Theta}(\mathbf{x}, \operatorname{softmax}(\mathbf{y})) \quad (1)$$

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

4 Inference Networks

Tu and Gimpel (2018) define an **inference network** (“infnet”) $\mathbf{A}_{\Psi} : \mathcal{X} \rightarrow \mathcal{Y}_R$ and train it with the goal that

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

where \mathcal{Y}_R is the relaxed continuous output space as defined in Section 3. For sequence labeling, for example, an inference network \mathbf{A}_{Ψ} takes a sequence \mathbf{x} as input and outputs a distribution over labels for each position in \mathbf{x} . Below we will consider three families of neural network architectures for \mathbf{A}_{Ψ} .

For training the inference network parameters Ψ , Tu and Gimpel (2018) explored stabilization and regularization terms and found that a local cross entropy loss consistently worked well for sequence labeling. We use this local cross entropy loss in this paper, so we perform learning by solving the following:

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

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

$$\ell_{\text{token}}(\mathbf{y}, \mathbf{A}(\mathbf{x})) = \sum_{t=1}^{|\mathbf{y}|} \text{CE}(\mathbf{y}_t, \mathbf{A}(\mathbf{x})_t) \quad (2)$$

where \mathbf{y}_t is the L -dimensional one-hot label vector at position t in \mathbf{y} , $\mathbf{A}(\mathbf{x})_t$ is the inference network’s output distribution at position t , and CE stands for cross entropy. We will give more details on how ℓ_{token} is defined for different inference network families below. It is also the loss used in our non-structured baseline models.

4.1 Inference Network Architectures

We now describe options for inference network architectures for sequence labeling. For each, we optionally include the modeling improvements described in Section 2.1. When doing so, we append “+” to the setting’s name to indicate this (e.g., infnet+).

4.1.1 Convolutional Neural Networks

CNNs are frequently used in NLP to extract features based on symbol subsequences, whether words or characters (Collobert et al., 2011; Kalchbrenner et al., 2014; Kim, 2014;

Kim et al., 2016; Zhang et al., 2015). CNNs use filters that are applied to symbol sequences and are typically followed by some sort of pooling operation. We apply filters over a fixed-size window centered on the word being labeled and do not use pooling. The feature maps $f_n(\mathbf{x}, t)$ for $(2n + 1)$ -gram filters are defined:

$$f_n(\mathbf{x}, t) = g(\mathbf{W}_n[\mathbf{v}_{x_{t-n}}; \dots; \mathbf{v}_{x_{t+n}}] + \mathbf{b}_n)$$

where g is a nonlinearity, \mathbf{v}_{x_t} is the embedding of word x_t , and \mathbf{W}_n and \mathbf{b}_n are filter parameters. We consider two CNN configurations: one uses $n = 0$ and $n = 1$ and the other uses $n = 0$ and $n = 2$. For each, we concatenate the two feature maps and use them as input to the softmax layer over outputs. In each case, we use H filters for each feature map.

4.1.2 Recurrent Neural Networks

For sequence labeling, it is common to use a BLSTM that runs over the input sequence and produces a softmax distribution over labels at each position in the sequence. We use this “BLSTM tagger” as our RNN inference network architecture. The parameter H refers to the size of the hidden vectors in the forward and backward LSTMs, so the full dimensionality passed to the softmax layer is $2H$.

4.1.3 Sequence-to-Sequence Models

Sequence-to-sequence (seq2seq; Sutskever et al. 2014) models have been successfully used for many sequential modeling tasks. It is common to augment models with an attention mechanism that focuses on particular positions of the input sequence while generating the output sequence (Bahdanau et al., 2015). 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 i th input when decoding the i th output, and hence does not learn any attention parameters. It is shown as follows:

$$P(\mathbf{y}_t \mid \mathbf{y}_{<t}, \mathbf{x}) = \text{softmax}(\mathbf{W}_s[\mathbf{h}_t, \mathbf{s}_t])$$

where \mathbf{s}_t is the hidden vector at position t from a BLSTM run over \mathbf{x} , \mathbf{h}_t is the decoder hidden vector at position t , and \mathbf{W}_s is a parameter matrix. The concatenation of the two hidden vectors is used to produce the distribution over labels.

When using this inference network, we redefine the local loss to the standard training criterion for seq2seq models, namely the sum of the log losses for each output conditioned on the previous outputs in the sequence. We always use the previous predicted label as input (as used in “scheduled sampling,” Bengio et al., 2015) during training because it works better for our tasks. In our experiments, the forward and backward encoder LSTMs use hidden dimension H , as does the LSTM decoder. Thus the model becomes similar to the BLSTM tagger except with conditioning on previous labeling decisions in a left-to-right manner.

We also experimented with the use of beam search for both the seq2seq baseline and inference networks and did not find much difference in the results. Also, as alternatives to the deterministic position-based attention described above, we experimented with learned local attention (Luong et al., 2015) and global attention, but they did not work better on our tasks.

4.2 Methods to Improve Inference Networks

To further improve the performance of an inference network for a particular test instance \mathbf{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.

4.2.1 Instance-Tailored Inference Networks

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

$$\underset{\Psi}{\text{argmin}} E_{\Theta}(\mathbf{x}, \mathbf{A}_\Psi(\mathbf{x})) \quad (3)$$

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.

4.2.2 Warm-Starting Gradient Descent with Inference Networks

Given a test example \mathbf{x} , we initialize $\mathbf{y} \in \mathcal{Y}_{R'}(\mathbf{x})$ using the inference network and then use gradient descent by solving Eq. 1 described in Section 3 to update \mathbf{y} . However, the inference network output is in $\mathcal{Y}_R(\mathbf{x})$ while gradient descent works with the

more relaxed space $\mathcal{Y}_{R'}(x)$. So we simply use the logits from the inference network, which are the score vectors before the softmax operations.

5 Experimental Setup

We perform experiments on three tasks: Twitter part-of-speech tagging (POS), named entity recognition (NER), and CCG supersense tagging (CCG).

5.1 Datasets

POS. We use the annotated data from Gimpel et al. (2011) and Owoputi et al. (2013) which contains 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 the 100-dimensional skip-gram embeddings from Tu et al. (2017) which were trained on a dataset of 56 million English tweets using `word2vec` (Mikolov et al., 2013). The evaluation metric is tagging accuracy.

NER. We use the CoNLL 2003 English data (Tjong Kim Sang and De Meulder, 2003). There are four entity types: PER, LOC, ORG, and MISC. There is a strong local dependency between neighboring labels because this is a labeled segmentation task. We use the BIOES tagging scheme, so there are 17 labels. We use 100-dimensional pre-trained GloVe (Pennington et al., 2014) embeddings. The task is evaluated with micro-averaged F1 score using the `conlleval` script.

CCG. We use the standard splits from CCG-bank (Hockenmaier and Steedman, 2002). We only keep sentences with length less than 50 in the original training data when training the CRF. The training data contains 1,284 unique labels, but because the label distribution has a long tail, we use only the 400 most frequent labels, replacing the others by a special tag `*`. The percentages of `*` in train/development/test are 0.25/0.23/0.23%. When the gold standard tag is `*`, the prediction is always evaluated as incorrect. We use the same GloVe embeddings as in NER. Because of the compositional nature of supertags, this task has more non-local dependencies. The task is evaluated with per-token accuracy.

5.2 Training and Tuning

For the optimization problems mentioned below, we use stochastic gradient descent with momentum as the optimizer. Full details of hyperparameter tuning are in the appendix.

Local Baselines. We consider local (non-structured) baselines that use the same architectures as the inference networks but train using only the local loss ℓ_{token} .

Structured Baselines. We train the BLSTM-CRF and BLSTM-CRF+ models with the standard conditional log-likelihood objective. We tune hyperparameters on the development sets.

Gradient Descent for Inference. We use gradient descent for structured inference by solving Eq. 1. We randomly initialize $\mathbf{y} \in \mathcal{Y}_{R'}(x)$ and, for N iterations, we compute the gradient of the energy with respect to \mathbf{y} , then update \mathbf{y} using gradient descent with momentum, which we found to generally work better than constant step size. We tune N and the learning rate via instance-specific oracle tuning, i.e., we choose them separately for each input to maximize performance (accuracy or F1 score) on that input. Even with this oracle tuning, we find that gradient descent struggles to compete with the other methods.

Inference Networks. To train the inference networks, we first train the BLSTM-CRF or BLSTM-CRF+ model with the standard conditional log-likelihood objective. The hidden sizes H are tuned in that step. We then fix the energy function and train the inference network \mathbf{A}_Ψ using the combined loss from Section 4.

For instance-tailored inference networks and when using inference networks as a warm start for gradient descent, we tune the number of epochs N and the learning rate on the development set, and report the performance on the test set, using the same values of N and the learning rate for all test examples.

6 BLSTM-CRF Results

This first section of results uses the simpler BLSTM-CRF modeling configuration. In Section 7 below we present results with the stronger BLSTM-CRF+ configuration and also apply the same modeling improvements to the baselines and inference networks.

	Twitter POS Tagging			NER			CCG Supertagging		
	CNN	BLSTM	seq2seq	CNN	BLSTM	seq2seq	CNN	BLSTM	seq2seq
local baseline	89.6	88.0	88.9	79.9	85.0	85.3	90.6	92.2	92.7
infnet	89.9	89.5	89.7	82.2	85.4	86.1	91.3	92.8	92.9
gradient descent	89.1			84.4			89.0		
Viterbi	89.2			87.2			92.4		

Table 1: Test results for all tasks. Inference networks, gradient descent, and Viterbi are all optimizing the BLSTM-CRF energy. Best result per task is in bold.

Table 1 shows test results for all tasks and architectures. The inference networks use the same architectures as the corresponding local baselines, but their parameters are trained with both the local loss and the BLSTM-CRF energy, leading to consistent improvements. CNN inference networks work well for POS, but struggle on NER and CCG compared to other architectures. BLSTMs work well, but are outperformed slightly by seq2seq models across all three tasks. Using the Viterbi algorithm for exact inference yields the best performance for NER but is not best for the other two tasks.

It may be surprising that an inference network trained to mimic Viterbi would outperform Viterbi in terms of accuracy, which we find for the CNN for POS tagging and the seq2seq inference network for CCG. We suspect this occurs for two reasons. One is due to the addition of the local loss in the inference network objective; the inference networks may be benefiting from this multi-task training. [Edunov et al. \(2018\)](#) similarly found benefit from a combination of token-level and sequence-level losses. The other potential reason is beneficial inductive bias with the inference network architecture. For POS tagging, the CNN architecture is clearly well-suited to this task given the strong performance of the local CNN baseline. Nonetheless, the CNN inference network is able to improve upon both the CNN baseline and Viterbi.

Hidden Sizes. For the test results in Table 1, we did limited tuning of H for the inference networks based on the development sets. Figure 1 shows the impact of H on performance. Across H values, the inference networks outperform the baselines. For NER and CCG, seq2seq outperforms the BLSTM which in turn outperforms the CNN.

Tasks and Window Sizes. Table 2 shows that CNNs with smaller windows are better for POS, while larger windows are better for NER and CCG. This suggests that POS has more local dependencies among labels than NER and CCG.

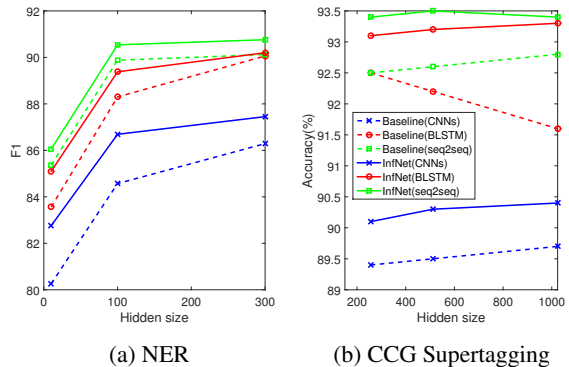


Figure 1: Development results for inference networks with different architectures and hidden sizes (H).

		{1,3}-gram	{1,5}-gram
POS	local baseline	89.2	88.7
	infnet	89.6	89.0
NER	local baseline	84.6	85.4
	infnet	86.7	86.8
CCG	local baseline	89.5	90.4
	infnet	90.3	91.4

Table 2: Development results for CNNs with two filter sets ($H = 100$).

6.1 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 3 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.

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

Table 3: Speed comparison of inference networks across tasks and architectures (examples/sec).

6.2 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 also compute the difference in the BLSTM-CRF energies between the inference network output \mathbf{y}_{inf} and the Viterbi output \mathbf{y}_{vit} as the search error: $E_{\Theta}(\mathbf{x}, \mathbf{y}_{inf}) - E_{\Theta}(\mathbf{x}, \mathbf{y}_{vit})$. 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 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.

7 BLSTM-CRF+ Results

We now compare inference methods when using the improved modeling techniques described in Section 2.1 (i.e., the setting we called BLSTM-CRF+). We use these improved techniques for all models, including the CRF, the local baselines, gradient descent, and the inference networks. When training inference networks, both the inference network architectures and the structured energies use the techniques from Section 2.1. So, when referring to inference networks in this section, we use the name infnet+.

The results are shown in Table 4. With a more powerful local architecture, structured prediction is less helpful overall, but inference networks still

	POS	NER	CCG
local baseline	91.3	90.5	94.1
infnet+	91.3	90.8	94.2
gradient descent	90.8	89.8	90.4
Viterbi	90.9	91.6	94.3

Table 4: Test results with BLSTM-CRF+. For local baseline and inference network architectures, we use CNN for POS, seq2seq for NER, and BLSTM for CCG.

	F1
local baseline (BLSTM)	90.3
infnet+ (1-layer BLSTM)	90.7
infnet+ (2-layer BLSTM)	91.1
Viterbi	91.6

Table 5: NER test results (for BLSTM-CRF+) with more layers in the BLSTM inference network.

improve over the local baselines on 2 of 3 tasks.

POS. As in the BLSTM-CRF setting, the local CNN baseline and the CNN inference network outperform Viterbi. This is likely because the CRFs use BLSTMs as feature networks, but our results show that CNN baselines are consistently better than BLSTM baselines on this task. As in the BLSTM-CRF setting, gradient descent works quite well on this task, comparable to Viterbi, though it is still much slower.

NER. We see slightly higher BLSTM-CRF+ results than several previous state-of-the-art results (cf. 90.94; Lample et al., 2016 and 91.37; Ma and Hovy, 2016). The stronger BLSTM-CRF+ configuration also helps the inference networks, improving performance from 90.5 to 90.8 for the seq2seq architecture over the local baseline. Though gradient descent reached high accuracies for POS tagging, it does not perform well on NER, possibly due to the greater amount of non-local information in the task.

While we see strong performance with infnet+, it still lags behind Viterbi in F1. We consider additional experiments in which we increase the number of layers in the inference networks. We use a 2-layer BLSTM as the inference network and also use weight annealing of the local loss hyperparameter λ , setting it to $\lambda = e^{-0.01t}$ where t is the epoch number. Without this annealing, the 2-layer inference network was difficult to train. The weight annealing was helpful for encouraging the inference network to focus more on the non-local information in the energy function rather than the token-level loss. As shown in Table 5, these changes yield an improvement of 0.4 in F1.

N	Twitter POS Tagging		NER		CCG Supertagging		
	Acc. (\uparrow)	Energy (\downarrow)	F1 (\uparrow)	Energy (\downarrow)	Acc. (\uparrow)	Energy (\downarrow)	
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 6: 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.

CCG. Our BLSTM-CRF+ reaches an accuracy of 94.3%, which is comparable to several recent results (93.53, Xu et al., 2016; 94.3, Lewis et al., 2016; and 94.50, Vaswani et al., 2016). The local baseline, the BLSTM inference network, and Viterbi are all extremely close in accuracy. Gradient descent struggles here, likely due to the large number of candidate output labels.

7.1 Speed, Accuracy, and Search Error

Table 6 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, pre-

sumably 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 ≥ 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.

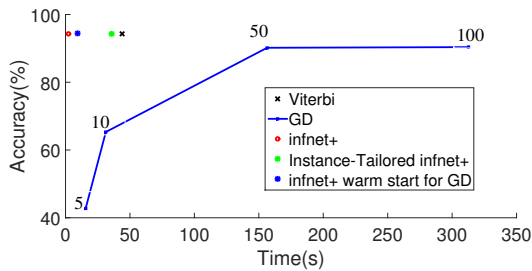


Figure 2: CCG test results for inference methods (GD = gradient descent). The x-axis is the total inference time for the test set. The numbers on the GD curve are the number of gradient descent iterations.

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 2 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.

8 Related Work

The most closely related prior work is that of [Tu and Gimpel \(2018\)](#), who experimented with RNN inference networks for sequence labeling. We compared three architectural families, showed the relationship between optimal architectures and downstream tasks, compared inference networks to gradient descent, and proposed novel variations.

We focused in this paper on sequence labeling, in which CRFs with neural network potentials have emerged as a state-of-the-art approach ([Lample et al., 2016](#); [Ma and Hovy, 2016](#); [Strubell et al., 2017](#); [Yang et al., 2018](#)). Our results suggest that inference networks can provide a feasible way to speed up test-time inference over Viterbi without much loss in performance. The benefits of inference networks may be coming in part from multi-task training; [Edunov et al. \(2018\)](#) similarly found benefit from combining token-

level and sequence-level losses.

We focused on structured prediction in this paper, but inference networks are useful in other settings as well. For example, it is common to use a particular type of inference network to approximate posterior inference in neural approaches to latent-variable probabilistic modeling, such as variational autoencoders ([Kingma and Welling, 2013](#)) and, more closely related to this paper, variational sequential labelers ([Chen et al., 2018](#)). In such settings, [Kim et al. \(2018\)](#) have found benefit with instance-specific updating of inference network parameters, which is related to our instance-level fine-tuning. There are also connections between structured inference networks and amortized structured inference ([Srikumar et al., 2012](#)) as well as methods for neural knowledge distillation and model compression ([Hinton et al., 2015](#); [Ba and Caruana, 2014](#); [Kim and Rush, 2016](#)).

Gradient descent is used for inference in several settings, e.g., structured prediction energy networks ([Belanger and McCallum, 2016](#)), image generation applications ([Mordvintsev et al., 2015](#); [Gatys et al., 2015](#)), finding adversarial examples ([Goodfellow et al., 2015](#)), learning paragraph embeddings ([Le and Mikolov, 2014](#)), and machine translation ([Hoang et al., 2017](#)). Gradient descent has started to be replaced by inference networks in some of these settings, such as image transformation ([Johnson et al., 2016](#); [Li and Wand, 2016](#)). Our results provide more evidence that gradient descent can be replaced by inference networks or improved through combination with them.

9 Conclusion

We compared several methods for approximate inference in neural structured prediction, finding that inference networks achieve a better speed/accuracy/search error trade-off than gradient descent. We also proposed instance-level inference network fine-tuning and using inference networks to initialize gradient descent, finding further reductions in search error and improvements in performance metrics for certain tasks.

Acknowledgments

We would like to thank Ke Li for suggesting experiments that combine inference networks and gradient descent, the anonymous reviewers for their feedback, and NVIDIA for donating GPUs used in this research.

References

- Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of International Conference on Learning Representations (ICLR)*.
- David Belanger and Andrew McCallum. 2016. Structured prediction energy networks. In *Proceedings of the 33rd International Conference on Machine Learning - Volume 48, ICML'16*, pages 983–992.
- Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1171–1179.
- Mingda Chen, Qingming Tang, Karen Livescu, and Kevin Gimpel. 2018. Variational sequential labelers for semi-supervised learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 215–226.
- Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12.
- Gregory F. Cooper. 1990. The computational complexity of probabilistic inference using Bayesian belief networks (research note). *Artif. Intell.*, 42(2-3).
- Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc’Aurelio Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 355–364, New Orleans, Louisiana.
- Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *CoRR*, abs/1508.06576.
- Kevin Gimpel, Nathan Schneider, Brendan O’Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanagan, and Noah A. Smith. 2011. 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.
- Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of International Conference on Learning Representations (ICLR)*.
- Kartik Goyal, Graham Neubig, Chris Dyer, and Taylor Berg-Kirkpatrick. 2018. A continuous relaxation of beam search for end-to-end training of neural sequence models. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, New Orleans, Louisiana.
- Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*.
- Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.
- Cong Duy Vu Hoang, Gholamreza Haffari, and Trevor Cohn. 2017. Towards decoding as continuous optimisation in neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 146–156, Copenhagen, Denmark.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.
- Julia Hockenmaier and Mark Steedman. 2002. 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*.
- Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of European Conference on Computer Vision (ECCV)*.
- Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland.
- Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar.
- Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI’16*, pages 2741–2749. AAAI Press.
- Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the*

- 2016 *Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas.
- Yoon Kim, Sam Wiseman, Andrew Miller, David Sontag, and Alexander Rush. 2018. Semi-amortized variational autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2678–2687, Stockholm, Sweden.
- Diederik Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *CoRR*, abs/1312.6114.
- John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. 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.
- Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. 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.
- Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14.
- Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California.
- Chuan Li and Michael Wand. 2016. Precomputed real-time texture synthesis with Markovian generative adversarial networks. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 702–716.
- Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. 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.
- Xuezhe Ma and Eduard Hovy. 2016. 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.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119.
- Alexander Mordvintsev, Christopher Olah, and Mike Tyka. 2015. DeepDream—a code example for visualizing neural networks. *Google Research*.
- Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390, Atlanta, Georgia.
- Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.
- Vivek Srikumar, Gourab Kundu, and Dan Roth. 2012. On amortizing inference cost for structured prediction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1114–1124, Jeju Island, Korea.
- Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2670–2680, Copenhagen, Denmark.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. 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.
- Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.
- Lifu Tu and Kevin Gimpel. 2018. Learning approximate inference networks for structured prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*.
- Lifu Tu, Kevin Gimpel, and Karen Livescu. 2017. Learning to embed words in context for syntactic tasks. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 265–275, Vancouver, Canada.
- Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, pages 232–237, San Diego, California.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Expected F-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 210–220, San Diego, California.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3879–3889, Santa Fe, New Mexico, USA.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 649–657.

A Appendix

Local Baselines. We consider local (non-structured) baselines that use the same architectures as the inference networks but train using only the local loss ℓ_{token} . We tune the learning rate ($\{5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$). We train on the training set, use the development sets for tuning and early stopping, and report results on the test sets.

Structured Baselines. We train the BLSTM-CRF and BLSTM-CRF+ models with the standard conditional log-likelihood objective. We tune hyperparameters on the development sets. The tuned BLSTM hidden size H for BLSTM-CRF is 100 for POS/NER and 512 for CCG; for BLSTM-CRF+ the tuned hidden size is 100 for POS, 200 for NER, and 400 for CCG.

Gradient Descent for Inference. For the number of epochs N , we consider values in the set $\{5, 10, 20, 30, 40, 50, 100, 500, 1000\}$. For each N , we tune the learning rate over the set $\{1e^4, 5e^3, 1e^3, 500, 100, 50, 10, 5, 1\}$. These learning rates may appear extremely large when we are accustomed to choosing rates for empirical risk minimization, but we generally found that the most effective learning rates for structured inference are orders of magnitude larger than those effective for learning. To provide as strong performance as possible for the gradient descent method, we tune N and the learning rate via oracle tuning, i.e., we choose them separately for each input to

maximize performance (accuracy or F1 score) on that input.

Inference Networks. To train the inference networks, we first train the BLSTM-CRF or BLSTM-CRF+ model with the standard conditional log-likelihood objective. The hidden sizes H are tuned in that step. We then fix the energy function and train the inference network \mathbf{A}_Ψ using the combined loss from Section 4. We tune the learning rate over the set $\{5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$ for the inference network and the local loss weight λ over the set $\{0.2, 0.5, 1, 2, 5\}$. We use early stopping on the development sets and report the results on the test sets using the trained inference networks.