Fast and Accurate Entity Recognition with Iterated Dilated Convolutions

Emma Strubell **Patrick Verga** **David Belanger** **Andrew McCallum**
College of Information and Computer Sciences
University of Massachusetts Amherst
{strubell, pat, belanger, mccallum}@cs.umass.edu

Abstract

Today when many practitioners run basic NLP on the entire web and large-volume traffic, faster methods are paramount to saving time and energy costs. Recent advances in GPU hardware have led to the emergence of bi-directional LSTMs as a standard method for obtaining per-token vector representations serving as input to labeling tasks such as NER (often followed by prediction in a linear-chain CRF). Though expressive and accurate, these models fail to fully exploit GPU parallelism, limiting their computational efficiency. This paper proposes a faster alternative to Bi-LSTMs for NER: Iterated Dilated Convolutional Neural Networks (ID-CNNs), which have better capacity than traditional CNNs for large context and structured prediction. Unlike LSTMs whose sequential processing on sentences of length N requires $O(N)$ time even in the face of parallelism, ID-CNNs permit fixed-depth convolutions to run in parallel across entire documents. We describe a distinct combination of network structure, parameter sharing and training procedures that enable dramatic 14-20x test-time speedups while retaining accuracy comparable to the Bi-LSTM-CRF. Moreover, ID-CNNs trained to aggregate context from the entire document are even more accurate while maintaining 8x faster test time speeds.

1 Introduction

In order to democratize large-scale NLP and information extraction while minimizing our environmental footprint, we require fast, resource-efficient methods for sequence tagging tasks such as part-of-speech tagging and named entity recognition (NER). Speed is not sufficient of course: they must also be expressive enough to tolerate the tremendous lexical variation in input data.

The massively parallel computation facilitated by GPU hardware has led to a surge of successful neural network architectures for sequence labeling (Ling et al., 2015; Ma and Hovy, 2016; Chiu and Nichols, 2016; Lample et al., 2016). While these models are expressive and accurate, they fail to fully exploit the parallelism opportunities of a GPU, and thus their speed is limited. Specifically, they employ either recurrent neural networks (RNNs) for feature extraction, or Viterbi inference in a structured output model, both of which require sequential computation across the length of the input.

Instead, parallelized runtime independent of the length of the sequence saves time and energy costs, maximizing GPU resource usage and minimizing the amount of time it takes to train and evaluate models. Convolutional neural networks (CNNs) provide exactly this property (Kim, 2014; Kalchbrenner et al., 2014). Rather than composing representations incrementally over each token in a sequence, they apply filters in parallel across the entire sequence at once. Their computational cost grows with the number of layers, but not the input size, up to the memory and threading limitations of the hardware. This provides, for example, audio generation models that can be trained in parallel (van den Oord et al., 2016).

Despite the clear computational advantages of CNNs, RNNs have become the standard method for composing deep representations of text. This is because a token encoded by a bidirectional RNN will incorporate evidence from the entire input sequence, but the CNN's representation is limited by

the effective input width[1] of the network: the size of the input context which is observed, directly or indirectly, by the representation of a token at a given layer in the network. Specifically, in a network composed of a series of stacked convolutional layers of convolution width w, the number r of context tokens incorporated into a token's representation at a given layer l, is given by $r = l(w-1)+1$. The number of layers required to incorporate the entire input context grows linearly with the length of the sequence. To avoid this scaling, one could pool representations across the sequence, but this is not appropriate for sequence labeling, since it reduces the output resolution of the representation.

In response, this paper presents an application of *dilated convolutions* (Yu and Koltun, 2016) for sequence labeling (Figure 1). For dilated convolutions, the effective input width can grow exponentially with the depth, with no loss in resolution at each layer and with a modest number of parameters to estimate. Like typical CNN layers, dilated convolutions operate on a sliding window of context over the sequence, but unlike conventional convolutions, the context need not be consecutive; the dilated window skips over every dilation width d inputs. By stacking layers of dilated convolutions of exponentially increasing dilation width, we can expand the size of the effective input width to cover the entire length of most sequences using only a few layers: The size of the effective input width for a token at layer l is now given by $2^{l+1}-1$. More concretely, just four stacked dilated convolutions of width 3 produces token representations with a n effective input width of 31 tokens – longer than the average sentence length (23) in the Penn TreeBank.

Our overall *iterated dilated CNN* architecture (ID-CNN) repeatedly applies the same block of dilated convolutions to token-wise representations. This parameter sharing prevents overfitting and also provides opportunities to inject supervision on intermediate activations of the network. Similar to models that use logits produced by an RNN, the ID-CNN provides two methods for performing prediction: we can predict each token's label independently, or by running Viterbi inference in a chain structured graphical model.

In experiments on CoNLL 2003 and OntoNotes 5.0 English NER, we demonstrate significant speed gains of our ID-CNNs over various recurrent models, while maintaining similar F1 performance. When performing prediction using independent classification, the ID-CNN consistently outperforms a bidirectional LSTM (Bi-LSTM), and performs on par with inference in a CRF with logits from a Bi-LSTM (Bi-LSTM-CRF). As an extractor of per-token logits for a CRF, our model out-performs the Bi-LSTM-CRF. We also apply ID-CNNs to entire documents, where independent token classification is as accurate as the Bi-LSTM-CRF while decoding almost 8× faster. The clear accuracy gains resulting from incorporating broader context suggest that these models could similarly benefit many other context-sensitive NLP tasks which have until now been limited by the computational complexity of existing context-rich models.[2]

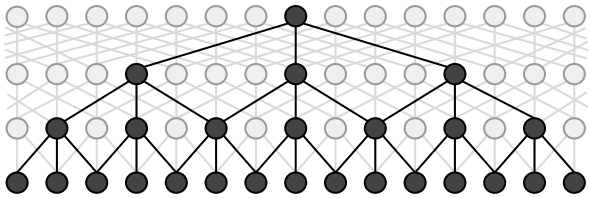

Figure 1: A dilated CNN block with maximum dilation width 4 and filter width 3. Neurons contributing to a single highlighted neuron in the last layer are also highlighted.

2 Background

2.1 Conditional Probability Models for Tagging

Let $x = [x_1, \ldots, x_T]$ be our input text and $y = [y_1, \ldots, y_T]$ be per-token output tags. Let D be the domain size of each y_i. We predict the most likely y, given a conditional model $P(y|x)$.

This paper considers two factorizations of the conditional distribution. First, we have

$$P(y|x) = \prod_{t=1}^{T} P(y_t|F(x)), \qquad (1)$$

where the tags are conditionally independent given some features for x. Given these features, $O(D)$ prediction is simple and parallelizable across the

[1] What we call *effective input width* here is known as the *receptive field* in the vision literature, drawing an analogy to the visual receptive field of a neuron in the retina.

[2] Our implementation in TensorFlow (Abadi et al., 2015) is available at: https://github.com/iesl/dilated-cnn-ner

length of the sequence. However, feature extraction may not necessarily be parallelizable. For example, RNN-based features require iterative passes along the length of x.

We also consider a linear-chain CRF model that couples all of y together:

$$P(y|x) = \frac{1}{Z_x} \prod_{t=1}^{T} \psi_t(y_t|F(x))\psi_p(y_t, y_{t-1}), \quad (2)$$

where ψ_t is a local factor, ψ_p is a pairwise factor that scores consecutive tags, and Z_x is the partition function (Lafferty et al., 2001). To avoid overfitting, ψ_p does not depend on the timestep t or the input x in our experiments. Prediction in this model requires global search using the $O(D^2T)$ Viterbi algorithm.

CRF prediction explicitly reasons about interactions among neighboring output tags, whereas prediction in the first model compiles this reasoning into the feature extraction step (Liang et al., 2008). The suitability of such compilation depends on the properties and quantity of the data. While CRF prediction requires non-trivial search in output space, it can guarantee that certain output constraints, such as for IOB tagging (Ramshaw and Marcus, 1999), will always be satisfied. It may also have better sample complexity, as it imposes more prior knowledge about the structure of the interactions among the tags (London et al., 2016). However, it has worse computational complexity than independent prediction.

3 Dilated Convolutions

CNNs in NLP are typically one-dimensional, applied to a sequence of vectors representing tokens rather than to a two-dimensional grid of vectors representing pixels. In this setting, a convolutional neural network layer is equivalent to applying an affine transformation, W_c to a sliding window of width r tokens on either side of each token in the sequence. Here, and throughout the paper, we do not explicitly write the bias terms in affine transformations. The convolutional operator applied to each token x_t with output c_t is defined as:

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t\pm k}, \quad (3)$$

where $\oplus$ is vector concatenation.

Dilated convolutions perform the same operation, except rather than transforming adjacent inputs, the convolution is defined over a wider effective input width by skipping over δ inputs at a time, where δ is the dilation width. We define the dilated convolution operator:

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t\pm k\delta}. \quad (4)$$

A dilated convolution of width 1 is equivalent to a simple convolution. Using the same number of parameters as a simple convolution with the same radius (i.e. W_c has the same dimensionality), the $\delta > 1$ dilated convolution incorporates broader context into the representation of a token than a simple convolution.

3.1 Multi-Scale Context Aggregation

We can leverage the ability of dilated convolutions to incorporate global context without losing important local information by stacking dilated convolutions of increasing width. First described for pixel classification in computer vision, Yu and Koltun (2016) achieve state-of-the-art results on image segmentation benchmarks by stacking dilated convolutions with exponentially increasing rates of dilation, a technique they refer to as *multi-scale context aggregation*. By feeding the outputs of each dilated convolution as the input to the next, increasingly non-local information is incorporated into each pixel's representation. Performing a dilation-1 convolution in the first layer ensures that no pixels within the effective input width of any pixel are excluded. By doubling the dilation width at each layer, the size of the effective input width grows exponentially while the number of parameters grows only linearly with the number of layers, so a pixel representation quickly incorporates rich global evidence from the entire image.

4 Iterated Dilated CNNs

Stacked dilated CNNs can easily incorporate global information from a whole sentence or document. For example, with a radius of 1 and 4 layers of dilated convolutions, the effective input width of each token is width 31, which exceeds the average sentence length (23) in the Penn TreeBank corpus. With a radius of size 2 and 8 layers of dilated convolutions, the effective input width exceeds 1,000 tokens, long enough to encode a full newswire document.

Unfortunately, simply increasing the depth of stacked dilated CNNs causes considerable overfitting in our experiments. In response, we present Iterated Dilated CNNs (ID-CNNs), which instead apply the same small stack of dilated convolutions multiple times, each iterate taking as input the result of the last application. Repeatedly employing the same parameters in a recurrent fashion provides both broad effective input width and desirable generalization capabilities. We also obtain significant accuracy gains with a training objective that strives for accurate labeling after each iterate, allowing follow-on iterations to observe and resolve dependency violations.

4.1 Model Architecture

The network takes as input a sequence of T vectors $\mathbf{x_t}$, and outputs a sequence of per-class scores $\mathbf{h_t}$, which serve either as the local conditional distributions of Eqn. (1) or the local factors ψ_t of Eqn. (2).

We denote the jth dilated convolutional layer of dilation width δ as $D_\delta^{(j)}$. The first layer in the network is a dilation-1 convolution $D_1^{(0)}$ that transforms the input to a representation $\mathbf{i_t}$:

$$\mathbf{i_t} = D_1^{(0)}\mathbf{x_t} \tag{5}$$

Next, L_c layers of dilated convolutions of exponentially increasing dilation width are applied to $\mathbf{i_t}$, folding in increasingly broader context into the embedded representation of $\mathbf{x_t}$ at each layer. Let $r()$ denote the ReLU activation function (Glorot et al., 2011). Beginning with $\mathbf{c_t}^{(0)} = \mathbf{i_t}$ we define the stack of layers with the following recurrence:

$$\mathbf{c_t}^{(j)} = r\left(D_{2^{L_c-1}}^{(j-1)}\mathbf{c_t}^{(j-1)}\right) \tag{6}$$

and add a final dilation-1 layer to the stack:

$$\mathbf{c_t}^{(L_c+1)} = r\left(D_1^{(L_c)}\mathbf{c_t}^{(L_c)}\right) \tag{7}$$

We refer to this stack of dilated convolutions as a *block* $B(\cdot)$, which has output resolution equal to its input resolution. To incorporate even broader context without over-fitting, we avoid making B deeper, and instead iteratively apply B L_b times, introducing no extra parameters. Starting with $\mathbf{b_t}^{(1)} = B\left(\mathbf{i_t}\right)$:

$$\mathbf{b_t}^{(k)} = B\left(\mathbf{b_t}^{(k-1)}\right) \tag{8}$$

We apply a simple affine transformation W_o to this final representation to obtain per-class scores for each token $\mathbf{x_t}$:

$$\mathbf{h_t}^{(L_b)} = W_o\mathbf{b_t}^{(L_b)} \tag{9}$$

4.2 Training

Our main focus is to apply the ID-CNN an encoder to produce per-token logits for the first conditional model described in Sec. 2.1, where tags are conditionally independent given deep features, since this will enable prediction that is parallelizable across the length of the input sequence. Here, maximum likelihood training is straightforward because the likelihood decouples into the sum of the likelihoods of independent logistic regression problems for every tag, with natural parameters given by Eqn. (9):

$$\frac{1}{T}\sum_{t=1}^{T}\log P(y_t \mid \mathbf{h_t}^{(L_b)}) \tag{10}$$

We can also use the ID-CNN as logits for the CRF model (Eqn. (2)), where the partition function and its gradient are computed using the forward-backward algorithm.

We next present an alternative training method that helps bridge the gap between these two techniques. Sec. 2.1 identifies that the CRF has preferable sample complexity and accuracy since prediction directly reasons in the space of structured outputs. In response, we compile some of this reasoning in output space into ID-CNN feature extraction. Instead of explicit reasoning over output labels during inference, we train the network such that each block is predictive of output labels. Subsequent blocks learn to correct dependency violations of their predecessors, refining the final sequence prediction.

To do so, we first define predictions of the model after each of the L_b applications of the block. Let $\mathbf{h_t}^{(k)}$ be the result of applying the matrix W_o from (9) to $\mathbf{b_t}^{(k)}$, the output of block k. We minimize the average of the losses for each application of the block:

$$\frac{1}{L_b}\sum_{k=1}^{L_b}\frac{1}{T}\sum_{t=1}^{T}\log P(y_t \mid \mathbf{h_t}^{(k)}). \tag{11}$$

By rewarding accurate predictions after each application of the block, we learn a model where later blocks are used to refine initial predictions.

The loss also helps reduce the vanishing gradient problem (Hochreiter, 1998) for deep architectures. Such an approach has been applied in a variety of contexts for training very deep networks in computer vision (Romero et al., 2014; Szegedy et al., 2015; Lee et al., 2015; Gülçehre and Bengio, 2016), but not to our knowledge in NLP.

We apply dropout (Srivastava et al., 2014) to the raw inputs $\mathbf{x_t}$ and to each block's output $\mathbf{b_t}^{(b)}$ to help prevent overfitting. The version of dropout typically used in practice has the undesirable property that the randomized predictor used at train time differs from the fixed one used at test time. Ma et al. (2017) present *dropout with expectation-linear regularization*, which explicitly regularizes these two predictors to behave similarly. All of our best reported results include such regularization. This is the first investigation of the technique's effectiveness for NLP, including for RNNs. We encourage its further application.

5 Related work

The state-of-the art models for sequence labeling include an inference step that searches the space of possible output sequences of a chain-structured graphical model, or approximates this search with a beam (Collobert et al., 2011; Weiss et al., 2015; Lample et al., 2016; Ma and Hovy, 2016; Chiu and Nichols, 2016). These outperform similar systems that use the same features, but independent local predictions. On the other hand, the greedy *sequential prediction* (Daumé III et al., 2009) approach of Ratinov and Roth (2009), which employs lexicalized features, gazetteers, and word clusters, outperforms CRFs with similar features.

LSTMs (Hochreiter and Schmidhuber, 1997) were used for NER as early as the CoNLL shared task in 2003 (Hammerton, 2003; Tjong Kim Sang and De Meulder, 2003). More recently, a wide variety of neural network architectures for NER have been proposed. Collobert et al. (2011) employ a one-layer CNN with pre-trained word embeddings, capitalization and lexicon features, and CRF-based prediction. Huang et al. (2015) achieved state-of-the-art accuracy on part-of-speech, chunking and NER using a Bi-LSTM-CRF. Lample et al. (2016) proposed two models which incorporated Bi-LSTM-composed character embeddings alongside words: a Bi-LSTM-CRF, and a greedy stack LSTM which uses a simple shift-reduce grammar to compose words into labeled entities. Their Bi-LSTM-CRF obtained the state-of-the-art on four languages without word shape or lexicon features. Ma and Hovy (2016) use CNNs rather than LSTMs to compose characters in a Bi-LSTM-CRF, achieving state-of-the-art performance on part-of-speech tagging and CoNLL NER without lexicons. Chiu and Nichols (2016) evaluate a similar network but propose a novel method for encoding lexicon matches, presenting results on CoNLL and OntoNotes NER. Yang et al. (2016) use GRU-CRFs with GRU-composed character embeddings of words to train a single network on many tasks and languages.

In general, distributed representations for text can provide useful generalization capabilities for NER systems, since they can leverage unsupervised pre-training of distributed word representations (Turian et al., 2010; Collobert et al., 2011; Passos et al., 2014). Though our models would also likely benefit from additional features such as character representations and lexicons, we focus on simpler models which use word-embeddings alone, leaving more elaborate input representations to future work.

In these NER approaches, CNNs were used for low-level feature extraction that feeds into alternative architectures. Overall, end-to-end CNNs have mainly been used in NLP for sentence classification, where the output representation is lower resolution than that of the input Kim (2014); Kalchbrenner et al. (2014); Zhang et al. (2015); Toutanova et al. (2015). Lei et al. (2015) present a CNN variant where convolutions adaptively skip neighboring words. While the flexibility of this model is powerful, its adaptive behavior is not well-suited to GPU acceleration.

Our work draws on the use of dilated convolutions for image segmentation in the computer vision community (Yu and Koltun, 2016; Chen et al., 2015). Similar to our block, Yu and Koltun (2016) employ a *context-module* of stacked dilated convolutions of exponentially increasing dilation width. Dilated convolutions were recently applied to the task of speech generation (van den Oord et al., 2016), and concurrent with this work, Kalchbrenner et al. (2016) posted a pre-print describing the similar ByteNet network for machine translation that uses dilated convolutions in the encoder and decoder components. Our basic model architecture is similar to that of the ByteNet encoder, except that the inputs to our model are tokens and

not bytes. Additionally, we present a novel loss and parameter sharing scheme to facilitate training models on much smaller datasets than those used by Kalchbrenner et al. (2016). We are the first to use dilated convolutions for sequence labeling.

The broad effective input width of the ID-CNN helps aggregate document-level context. Ratinov and Roth (2009) incorporate document context in their greedy model by adding features based on tagged entities within a large, fixed window of tokens. Prior work has also posed a structured model that couples predictions across the whole document (Bunescu and Mooney, 2004; Sutton and McCallum, 2004; Finkel et al., 2005).

6 Experimental Results

We describe experiments on two benchmark English named entity recognition datasets. On CoNLL-2003 English NER, our ID-CNN performs on par with a Bi-LSTM not only when used to produce per-token logits for structured inference, but the ID-CNN with greedy decoding also performs on-par with the Bi-LSTM-CRF while running at more than 14 times the speed. We also observe a performance boost in almost all models when broadening the context to incorporate entire documents, achieving an average F1 of 90.65 on CoNLL-2003, out-performing the sentence-level model while still decoding at nearly 8 times the speed of the Bi-LSTM-CRF.

6.1 Data and Evaluation

We evaluate using labeled data from the CoNLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003) and OntoNotes 5.0 (Hovy et al., 2006; Pradhan et al., 2006). Following previous work, we use the same OntoNotes data split used for co-reference resolution in the CoNLL-2012 shared task (Pradhan et al., 2012). For both datasets, we convert the IOB boundary encoding to BILOU as previous work found this encoding to result in improved performance (Ratinov and Roth, 2009). As in previous work we evaluate the performance of our models using segment-level micro-averaged F1 score. Hyperparameters that resulted in the best performance on the validation set were selected via grid search. A more detailed description of the data, evaluation, optimization and data pre-processing can be found in the Appendix.

6.2 Baselines

We compare our **ID-CNN** against strong LSTM and CNN baselines: a **Bi-LSTM** with local decoding, and one with CRF decoding (**Bi-LSTM-CRF**). We also compare against a non-dilated CNN architecture with the same number of convolutional layers as our dilated network (**4-layer CNN**) and one with enough layers to incorporate an effective input width of the same size as that of the dilated network (**5-layer CNN**) to demonstrate that the dilated convolutions more effectively aggregate contextual information than simple convolutions (i.e. using fewer parameters). We also compare our document-level ID-CNNs to a baseline which does not share parameters between blocks (**noshare**) and one that computes loss only at the last block, rather than after every iterated block of dilated convolutions (**1-loss**).

We do not compare with deeper or more elaborate CNN architectures for a number of reasons: 1) Fast train and test performance are highly desirable for NLP practitioners, and deeper models require more computation time 2) more complicated models tend to over-fit on this relatively small dataset and 3) most accurate deep CNN architectures repeatedly up-sample and down-sample the inputs. We do not compare to stacked LSTMs for similar reasons — a single LSTM is already slower than a 4-layer CNN. Since our task is sequence labeling, we desire a model that maintains the token-level resolution of the input, making dilated convolutions an elegant solution.

6.3 CoNLL-2003 English NER

6.3.1 Sentence-level prediction

Table 1 lists F1 scores of models predicting with sentence-level context on CoNLL-2003. For models that we trained, we report F1 and standard deviation obtained by averaging over 10 random restarts. The Viterbi-decoding Bi-LSTM-CRF and ID-CNN-CRF and greedy ID-CNN obtain the highest average scores, with the ID-CNN-CRF outperforming the Bi-LSTM-CRF by 0.11 points of F1 on average, and the Bi-LSTM-CRF out-performing the greedy ID-CNN by 0.11 as well. Our greedy ID-CNN outperforms the Bi-LSTM and the 4-layer CNN, which uses the same number of parameters as the ID-CNN, and performs similarly to the 5-layer CNN which uses more parameters but covers the same effective input width. All CNN models out-perform the Bi-

Model	F1
Ratinov and Roth (2009)	86.82
Collobert et al. (2011)	86.96
Lample et al. (2016)	90.33
Bi-LSTM	89.34 ± 0.28
4-layer CNN	89.97 ± 0.20
5-layer CNN	90.23 ± 0.16
ID-CNN	90.32 ± 0.26
Collobert et al. (2011)	88.67
Passos et al. (2014)	90.05
Lample et al. (2016)	90.20
Bi-LSTM-CRF (re-impl)	90.43 ± 0.12
ID-CNN-CRF	**90.54** ± **0.18**

Table 1: F1 score of models observing sentence-level context. No models use character embeddings or lexicons. Top models are greedy, bottom models use Viterbi inference .

LSTM when paired with greedy decoding, suggesting that CNNs are better token encoders than Bi-LSTMs for independent logistic regression. When paired with Viterbi decoding, our ID-CNN performs on par with the Bi-LSTM, showing that the ID-CNN is also an effective token encoder for structured inference.

Our ID-CNN is not only a better token encoder than the Bi-LSTM but it is also faster. Table 2 lists relative decoding times on the CoNLL development set, compared to the Bi-LSTM-CRF. We report decoding times using the fastest batch size for each method.[3]

The ID-CNN model decodes nearly 50% faster than the Bi-LSTM. With Viterbi decoding, the gap closes somewhat but the ID-CNN-CRF still comes out ahead, about 30% faster than the Bi-LSTM-CRF. The most vast speed improvements come when comparing the greedy ID-CNN to the Bi-LSTM-CRF – our ID-CNN is more than 14 times faster than the Bi-LSTM-CRF at test time, with comparable accuracy. The 5-layer CNN, which observes the same effective input width as the ID-CNN but with more parameters, performs at about the same speed as the ID-CNN in our experiments. With a better implementation of dilated convolutions than currently included in TensorFlow, we would expect the ID-CNN to be notably faster than the 5-layer CNN.

[3]For each model, we tried batch sizes $b = 2^i$ with $i = 0...11$. At scale, speed should increase with batch size, as we could compose each batch of as many sentences of the same length as would fit in GPU memory, requiring no padding and giving CNNs and ID-CNNs even more of a speed advantage.

Model	Speed
Bi-LSTM-CRF	1×
Bi-LSTM	9.92×
ID-CNN-CRF	1.28×
5-layer CNN	12.38×
ID-CNN	14.10×

Table 2: Relative test-time speed of sentence models, using the fastest batch size for each model.[5]

Model	w/o DR	w/ DR
Bi-LSTM	88.89 ± 0.30	**89.34** ± 0.28
4-layer CNN	89.74 ± 0.23	**89.97** ± 0.20
5-layer CNN	89.93 ± 0.32	**90.23** ± 0.16
Bi-LSTM-CRF	90.01 ± 0.23	**90.43** ± 0.12
4-layer ID-CNN	89.65 ± 0.30	**90.32** ± 0.26

Table 3: Comparison of models trained with and without expectation-linear dropout regularization (DR). DR improves all models.

We emphasize the importance of the dropout regularizer of Ma et al. (2017) in Table 3, where we observe increased F1 for every model trained with expectation-linear dropout regularization. Dropout is important for training neural network models that generalize well, especially on relatively small NLP datasets such as CoNLL-2003. We recommend this regularizer as a simple and helpful tool for practitioners training neural networks for NLP.

6.3.2 Document-level prediction

In Table 4 we show that adding document-level context improves every model on CoNLL-2003. Incorporating document-level context further improves our greedy ID-CNN model, attaining 90.65 average F1. We believe this model sees greater improvement with the addition of document-level context than the Bi-LSTM-CRF due to the ID-CNN learning a feature function better suited for representing broad context, in contrast with the Bi-LSTM which, though better than a simple RNN at encoding long memories of sequences, may reach its limit when provided with sequences more than 1,000 tokens long such as entire documents.

We also note that our combination of training objective (Eqn. 11) and tied parameters (Eqn.

[5]Our ID-CNN could see up to 18× speed-up with a less naive implementation than is included in TensorFlow as of this writing.

Model	F1
4-layer ID-CNN (sent)	90.32 ± 0.26
Bi-LSTM-CRF (sent)	90.43 ± 0.12
4-layer CNN × 3	90.32 ± 0.32
5-layer CNN × 3	90.45 ± 0.21
Bi-LSTM	89.09 ± 0.19
Bi-LSTM-CRF	90.60 ± 0.19
ID-CNN	**90.65 ± 0.15**

Table 4: F1 score of models trained to predict document-at-a-time. Our greedy ID-CNN model performs as well as the Bi-LSTM-CRF.

Model	F1
ID-CNN noshare	89.81 ± 0.19
ID-CNN 1-loss	90.06 ± 0.19
ID-CNN	**90.65 ± 0.15**

Table 5: Comparing ID-CNNs with 1) back-propagating loss only from the final layer (**1-loss**) and 2) untied parameters across blocks (**noshare**)

8) more effectively learns to aggregate this broad context than a vanilla cross-entropy loss or deep CNN back-propagated from the final neural network layer. Table 5 compares models trained to incorporate entire document context using the document baselines described in Section 6.2.

In Table 6 we show that, in addition to being more accurate, our ID-CNN model is also much faster than the Bi-LSTM-CRF when incorporating context from entire documents, decoding at almost 8 times the speed. On these long sequences, it also tags at more than 4.5 times the speed of the greedy Bi-LSTM, demonstrative of the benefit of our ID-CNNs context-aggregating computation that does not depend on the length of the sequence.

6.4 OntoNotes 5.0 English NER

We observe similar patterns on OntoNotes as we do on CoNLL. Table 7 lists overall F1 scores of our models compared to those in the existing literature. The greedy Bi-LSTM out-performs the lexicalized greedy model of Ratinov and Roth (2009), and our ID-CNN out-performs the Bi-LSTM as well as the more complex model of Durrett and Klein (2014) which leverages the parallel co-reference annotation available in the OntoNotes corpus to predict named entities jointly with entity linking and co-reference. Our greedy model is out-performed by the Bi-LSTM-CRF reported in Chiu and Nichols (2016) as well as our own re-implementation, which appears to be the new state-of-the-art on this dataset.

Model	Speed
Bi-LSTM-CRF	1×
Bi-LSTM	4.60×
ID-CNN	7.96×

Table 6: Relative test-time speed of document models (fastest batch size for each model).

Model	F1	Speed
Ratinov and Roth (2009)[6]	83.45	
Durrett and Klein (2014)	84.04	
Chiu and Nichols (2016)	86.19 ± 0.25	
Bi-LSTM-CRF	86.99 ± 0.22	1×
Bi-LSTM-CRF-Doc	86.81 ± 0.18	1.32×
Bi-LSTM	83.76 ± 0.10	24.44×
ID-CNN-CRF (1 block)	86.84 ± 0.19	1.83×
ID-CNN-Doc (3 blocks)	85.76 ± 0.13	21.19×
ID-CNN (3 blocks)	85.27 ± 0.24	13.21×
ID-CNN (1 block)	84.28 ± 0.10	26.01×

Table 7: F1 score of sentence and document models on OntoNotes.

The gap between our greedy model and those using Viterbi decoding is wider than on CoNLL. We believe this is due to the more diverse set of entities in OntoNotes, which also tend to be much longer – the average length of a multi-token named entity segment in CoNLL is about one token shorter than in OntoNotes. These long entities benefit more from explicit structured constraints enforced in Viterbi decoding. Still, our ID-CNN outperforms all other greedy methods, achieving our goal of learning a better token encoder for structured prediction.

Incorporating greater context significantly boosts the score of our greedy model on OntoNotes, whereas the Bi-LSTM-CRF performs more poorly. In Table 7, we also list the F1 of our ID-CNN model and the Bi-LSTM-CRF model trained on entire document context. For the first time, we see the score decrease when more context is added to the Bi-LSTM-CRF model, though the ID-CNN, whose sentence model a lower score than that of the Bi-LSTM-CRF, sees an increase. We believe the decrease in the Bi-LSTM-CRF model occurs because of the

[6]Results as reported in Durrett and Klein (2014) as this data split did not exist at the time of publication.

nature of the OntoNotes dataset compared to CoNLL-2003: CoNLL-2003 contains a particularly high proportion of ambiguous entities,[7] perhaps leading to more benefit from document context that helps with disambiguation. In this scenario, adding the wider context may just add noise to the high-scoring Bi-LSTM-CRF model, whereas the less accurate dilated model can still benefit from the refined predictions of the iterated dilated convolutions.

7 Conclusion

We present iterated dilated convolutional neural networks, fast token encoders that efficiently aggregate broad context without losing resolution. These provide impressive speed improvements for sequence labeling, particularly when processing entire documents at a time. In the future we hope to extend this work to NLP tasks with richer structured output, such as parsing.

Acknowledgments

We thank Subhransu Maji and Luke Vilnis for helpful discussions, and Brendan O'Connor, Yoav Goldberg, the UMass NLP reading group and many anonymous reviewers for constructive comments on various drafts of the paper. We are also grateful to Guillaume Lample for sharing his pretrained word embeddings. This work was supported in part by the Center for Intelligent Information Retrieval, in part by DARPA under agreement number FA8750-13-2-0020, in part by Defense Advanced Research Agency (DARPA) contract number HR0011-15-2-0036, in part by the National Science Foundation (NSF) grant number DMR-1534431, and in part by the National Science Foundation (NSF) grant number IIS-1514053. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow.org*.

Razvan Bunescu and Raymond J. Mooney. 2004. Collective information extraction with relational markov networks. In *ACL*, pages 439–446.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2015. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*.

Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

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(Aug):2493–2537.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28 (NIPS)*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325.

Greg Durrett and Dan Klein. 2014. A joint model for entity analysis: Coreference, typing and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *AISTATS*.

Çalar Gülçehre and Yoshua Bengio. 2016. Knowledge matters: Importance of prior information for optimization. *Journal of Machine Learning Research*, 17(8):1–32.

James Hammerton. 2003. Named entity recognition with long short-term memory. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL*, pages 172–175. Association for Computational Linguistics.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[7] According to the ACL Wiki page on CoNLL-2003: "The corpus contains a very high ratio of metonymic references (city names standing for sport teams)"

Sepp Hochreiter and J urgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

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*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*.

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)*, pages 282–289.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL*.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Chen-Yu Lee, Saining Xie, Patrick W Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-supervised nets. In *AISTATS*, volume 2, page 5.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding cnns for text: non-linear, non-consecutive convolutions. *Empirical Methods in Natural Language Processing*.

Percy Liang, Hal Daumé III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *Proceedings of the 25th international conference on Machine learning*, pages 592–599. ACM.

Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, and Silvio Amir. 2013. Not all contexts are created equal: Better word representations with variable attention. In *EMNLP*. Association for Computational Linguistics.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *EMNLP*.

Ben London, Bert Huang, and Lise Getoor. 2016. Stability and generalization in structured prediction. *Journal of Machine Learning Research*, 17(222):1–52.

Xuezhe Ma, Yingkai Gaom, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. 2017. Dropout with expectation-linear regularization. In *ICLR*.

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*, page 10641074.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. In *CoNLL*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Bj orkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2006. Towards robust linguistic analysis using ontonotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Proceedings of the Joint Conference on EMNLP and CoNLL: Shared Task*, pages 1–40.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Charles Sutton and Andrew McCallum. 2004. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning*.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

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-Volume 4*, pages 142–147. Association for Computational Linguistics.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509. Association for Computational Linguistics.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Annual Meeting of the Association for Computational Linguistics*.

Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. 2016. Multi-task cross-lingual sequence tagging from scratch. In *arXiv preprint arXiv:1603.06270*.

Fisher Yu and Vladlen Koltun. 2016. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS)*.

A Appendix

A.1 Optimization and data pre-processing

Our models are trained end-to-end using back-propagation and mini-batched Adam (Kingma and Ba, 2015) SGD. We use dropout regularization (Srivastava et al., 2014) on the input embeddings and final dilation layer of each block, along with the dropout regularizer described in Ma et al. (2017) using a single Monte Carlo sample for each training example. We also found word dropout (Dai and Le, 2015; Lample et al., 2016) crucial for learning a high-quality representation for out-of-vocabulary words. We used the modified version of identity initialization (Le et al., 2015) reported by Yu and Koltun (2016) to initialize our dilated layers, which we found to perform the best in initial experiments compared to orthogonal and Xavier initialization (Glorot and Bengio, 2010). Since our models use the same number of filters in each dilated layer, this initialization simplifies to setting the parameters corresponding to the central token to the identity matrix, and all other parameters (corresponding to left and right context) to zero. All other layers (embeddings, projections) were initialized using normally distributed Xavier initialization.

As in previous work, we found that initializing the word embedding lookup table with pre-trained embeddings was vital to achieve good performance. In initial experiments, we found the 100-dimensional skip-n-gram (Ling et al., 2013) embeddings of Lample et al. (2016) to outperform the 50-dimensional word embeddings of Collobert et al. (2011), and so we use these 100-dimensional embeddings in all experiments. We concatenate a 5-dimensional word shape vector based on whether the token was all capitalized, not capitalized, first-letter capitalized or contained a capital letter. We preprocessed the data by replacing all digits with 0, but did not lowercase thus our embeddings are case-sensitive.

We use the parameters of the trained sentence models to initialize the parameters of the document models in order to significantly speed up the rate of convergence of the document models.

A.2 Data details

Entities in the CoNLL-2003 corpus are labeled with one of four types: PER, ORG, LOC or MISC, with a fairly even distribution over the four entity types. OntoNotes contains a larger and more diverse set of 19 different entity types, adding: ORDINAL, PRODUCT, NORP, WORK_OF_ART, LANGUAGE, MONEY, PERCENT, CARDINAL, GPE, TIME, DATE, FAC, LAW, EVENT and QUANTITY. The OntoNotes corpus also covers a wider range of text genres, including telephone conversations, web text, broadcast news and translated documents, whereas the CoNLL-2003 text covers only newswire. The combined entity types and boundary encodings result in 17 possible output labels in the CoNLL-2003 corpus and 74 labels in the OntoNotes corpus. The sizes of the two corpora in terms of documents, sentences, tokens and entities are given in Table 8.

Data		Train	Dev	Test
CoNLL-2003	Tok	204,567	51,578	46,666
	Sent	14,041	3,250	3,453
	Doc	945	215	230
	Ent	23,499	5,942	5,648
OntoNotes 5.0	Tok	1,088,503	147,724	152,728
	Sent	59,924	8,528	8,262
	Doc	2,483	319	322
	Ent	81,828	11,066	11,257

Table 8: Statistics of NER datasets

A.3 Evaluation

To select hyperparameters, we iteratively perform grid search over increasingly fine-grained settings of dropout, learning rate, Adam β_2 and ϵ parameters, gradient clipping threshold, number of dilated layers, number of repeated blocks, regularizer penalty and batch size. Since we found the variance in score between runs to vary significantly, in the last iteration of grid search, we ran each setting of parameters three times and averaged their scores on the validation set. Of these, we ran the top ten settings ten times, and took the parameters which averaged the highest F1 on the development set, and report scores on the test set using these parameters. Note that we do not in the final stage include the development set as training data as has been done in some previous work, and so do not directly compare to results from other papers which do so.

We evaluate test-time speed using our top-performing trained models. All timing experiments were run on a nVidia Titan X GPU with a 2.4GHz Intel Xeon CPU. We do not include data

loading, preprocessing or feature hashing in our timing since this is exactly the same across all models. Reported is the time it takes for each model to produce a sequence of labels given a sequence of integers representing the words and their capitalization. After a burn-in run to account for caching and GPU data I/O, we run each model 20 times over the development set and average these times. We do this for batch sizes ranging from 1 to 10,000.