Named Entity Recognition with Capsule Network

Matthew Potts UC Berkeley
Dave Huber UC Berkeley
Legg Yeung UC Berkeley
{mepotts, dhuber, yeunghoman}@berkeley.edu

Abstract

Dynamic Routing Between Capsules Sabour et al. (2017) showed Capsule Network (CapsNet) to achieve state of the art performance on MNIST classification. A simple 3-layer CapsNet gives noticeably lower test error than a standard Convolutional Neural Network (CNN) with similar depth. We posit that the mathematical and architectural properties which made CapsNet superior for digit recognition can extend to named entity recognition (NER). Keeping the network architectures close to that of Sabour et al., we compare CapsNet against CNN on the CoNLL-2003 Shared Task. Our 3-layer CapsNet achieves an F_1 rate of 0.874%, compared to CNN F_1 rate of 0.861%. An in-depth error analysis shows that CapsNet is more sensitive to the part-to-whole structures and boundaries in multi-word entities than CNN.

1 Introduction

For both computer vision and natural language processing tasks, CNNs have been widely used in detection and recognition tasks because they are fast to train and able to extract implicit features automatically. For image classification tasks, CNNs can abstract pixel level intensity values to multiple feature maps that describe properties such as edges and shades. Similarly, for natural language processing tasks, CNNs have been used to abstract word level embeddings to multiple feature maps that capture implicit N-gram level signals. These feature maps are then fed into several feed-forward layers with ReLU non-linearity, before a Softmax classifier is applied to generate normalized probabilities corresponding to a set target classes.

In *Dynamic Routing Between Capsules*, Sabour et al. suggested CapsNet as a better alternative to CNNs for image entity recognition tasks. Both CapsNet and CNNs can take 2-dimensional values as input and make prediction for the most likely target class. For MNIST digit classification, keeping the network depth similar, CapsNet achieved a low test error of 0.25% compared to standard CNN of 0.39%. This result begs the question of whether similar performance gain can be achieved on NER tasks. Three major advantages of CapsNet over CNNs motivated the experiments in *Dynamic Routing Between Capsules*. Firstly, vector capsule outputs in CapsNet can better retain useful information about "pose" than scalar max-pooling outputs in CNNs. Secondly, dynamic routing in CapsNet can better direct lower level capsule outputs to higher level capsules (e.g. entities) by agreement. Thirdly, because each lower level capsule can route their outputs to multiple higher level capsules, CapsNet can recognize highly overlapping entities. Since NER tasks can benefit from retaining more word level information such as position and tags, composing multiple-word entities from adjacent words, and understanding competing entity tags for any particular word, we posit that CapsNet intuitions can be transferred to replace CNNs in NER tasks as well.

Our main contribution lies in extending and validating this CapsNet over CNN intuition for an NER task. We compare these 2 networks using the CoNLL-2003 dataset. We provide a walk through of our models in terms of architecture, test evaluation and an extensive error analysis, which shows that ...(TBD in final paper).

2 Background

The formulation of our experiment is inspired by earlier research on named entity recognition in natural language processing and digit recognition in computer vision.

2.1 Dataset

We compare our models by their NER performance on the CoNLL-2003 English corpus, which was taken from Reuters news stories between August 1996 and August 1997. The CoNLL-2003 Shared Task on this benchmark dataset was attempted by 16 teams of researchers using a wide range of modeling paradigms including Maximum Entropy Models, Hidden Markov Models, Support Vector Machines, Conditional Random Fields, Neural Nets and combined models. Our experiment instead focuses on CapsNet and CNN only. In terms of input features, each line in each data file contains four fields : the word, its part-of-speech-tag, its chunk tag and its named entity which is the target. The research teams in the CoNLL-2003 Shared Task tended to use extra information such as gazetteers, unannotated data and externally developed name entity recognizers. In our experiment, because a large number of parameters have to be trained for CapsNet and CNN, we limit our input features to word level embeddings, part-of-speech tags and chunk tags. For evaluation purposes, three data files containing the training, development and test set are available. In the The CoNLL-2003 Shared Task [1], the lowest performing LSTM model (Hammerton, 2003) achieved a precision of 69.09%, recall of 53.26%, and F_1 rate of 60.15%. The highest performing combined classifier (Florian et al., 2003) achieved a precision of 88.99%, recall of 88.54%, and F_1 rate of 88.76%. We adopt the same performance metrics to compare CapsNet and CNN, and observe that our best performing CapsNet would have ranked third while a CNN of the same depth would have ranked fourth in the CoNLL-2003 Shared Task.

2.2 State of the art NER model

State of the art performance on the CoNLL-2003 English dataset was achieved in *Named Entity Recognition with Bidirectional LSTM-CNNs* (Chiu and Nichols, 2016), with a F_1 rate of 91.62%. The researchers fed 50-dimensional word level embeddings, CNN-extracted character features, randomly initialized 25-dimensional character embeddings, capitalization features, external lexicon DBpedia and additional character features into a two-level, stacked Bidirectional LSTM. For word embeddings, the researchers tested random and other pre-trained embeddings such as GloVe which our models use as well. Because the convolutional, dynamic routing and decoder feed-forward layers in CapsNet require a larger number of trainable parameters relative to recurrent cells in LSTMs, our input features are not as extensive compared to Chiu and Nichols (2016). Within our scope of investigation, we focus primarily on comparing CapsNet and CNN on the CoNLL-2003 dataset, and leave the opportunity to explore hybrid architecture and extensive input features for an extension research in the near future.

2.3 Encoding primary capsules

The initial success of CapsNet depended largely on the model's ability to encapsulate inverse graphics information within its internal states. In *Transforming Auto-encoders*(Hinton et al, 2011), the researchers proposed computing inverse graphics from MNIST pixels using transforming autoencoders and use the generated vectors as instantiation parameters of primary layer capsules in a full Capsule Network. This autoencoder is composed of hidden layers and probability gates and the intermediate states it outputs can be routed to higher level capsules, or decoded to reconstruct the original digits. Our research is partly built on this idea, but adapted to accommodate sliding windows of word token features instead of 2-dimensional image pixels. Also, we use only convolutions and a non-linear squash function to instantiate the primary capsules.

2.4 Existing CapsNet architecture

We keep the architecture of ours CapsNet and CNN models as close as possible to those presented by Sabour et al.(2017), with the input and target formats adapted to word level representations and named entity classes found in the CoNLL-2003 dataset. The CapsNet model has one convolutional layer, two dynamically routed capsule layers and one fully connected layer, and the CNN model has three convolutional layers and two fully connected layers. Following Sabour et al., we also implement a decoder to reconstruct

[1] The baseline predictions were produced by a system which only identifies entities for tokens which has a unique named entity label in the training data. In other words, this system memorize the tokens with real named entities it has seen in the training set and apply its memory on the development and testing data. If a token was part of more than one named entity, then the system would select the named entity that spans the longest number of tokens. This simple baseline has a precision of 71.91%, recall of 50.90% and F_1 rate of 59.61% – less than one percent short of the lowest performing of the 16 models in the CoNLL-2003 Shared Task.

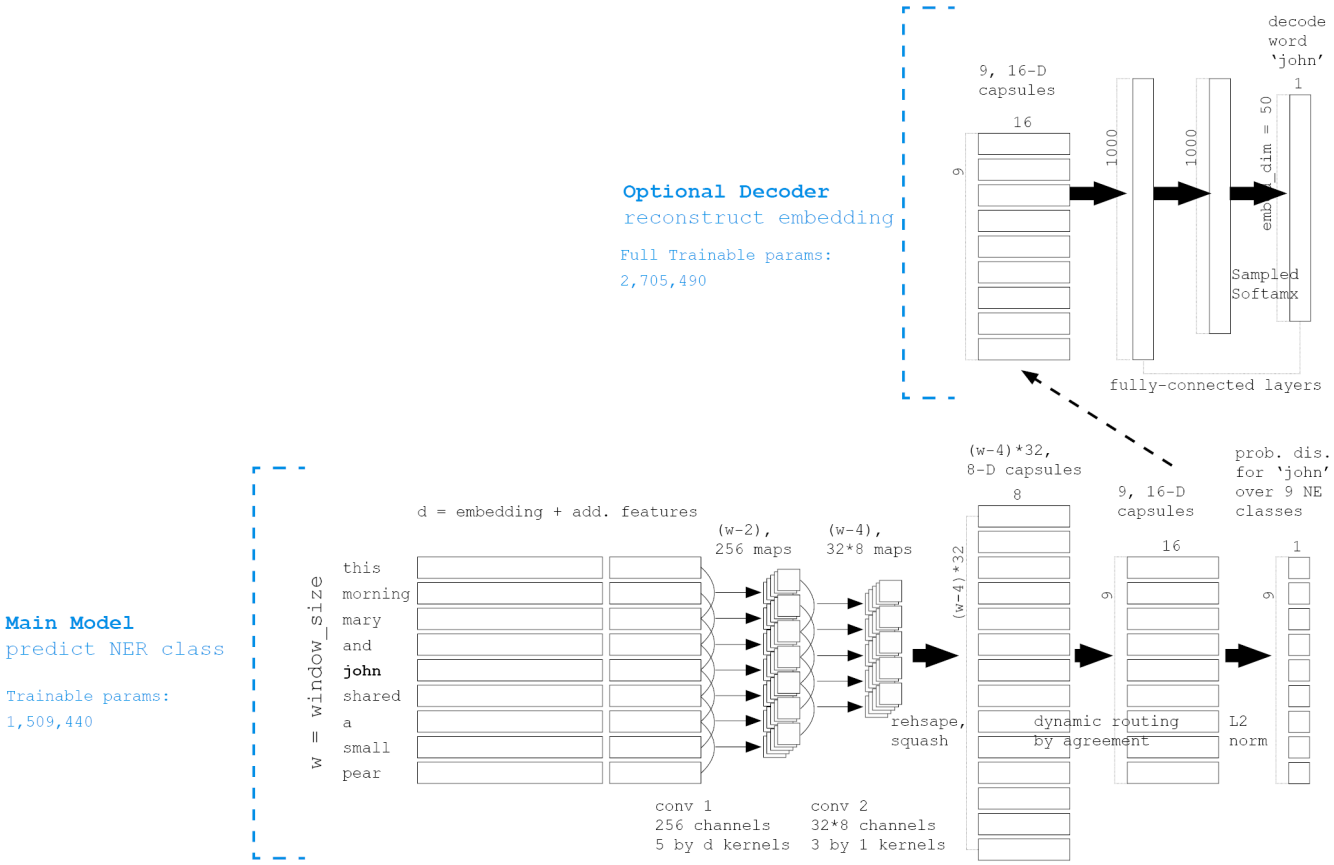

Figure 1: CapsNet Architecture for Named Entity Recognition

the input targets. This decoder act as a regularizer to avoid over-fitting and encourage our model to retain semantic features in deeper layers of the model.

3 Methods

We established a CapsNet model and a CNN of similar depth as its baseline model for an apple-to-apple comparison.

3.1 Core Features

Our CapsNet and CNN models both use the Stanford's GloVe embeddings trained on 6 billion words from Wikipedia and Web text (Pennington et al. 2014). We explored both retrainable and non-retrainable word embeddings. Each word is represented by a long, 100-dimensional vector concatenated from 50-dimensional GloVe embedding, 45-dimensional part-of-speech features, 5-dimensional capitalization features (`allCaps`, `upperInitial`, `lowercase`, `mixedCaps`, `noinfo`). Both models ingest words in windows of length 11, with the word of interest placed at the center of the window.

3.2 Core Architecture – CapsNet

A diagram of our CapsNet architecture is provided in Figure 1. In the main model, a size-11 sliding window with the target word in the center ingest concatenated embeddings, part-of-speech and chunk tags. `conv1` has 256 channels, each of size 5by100[2], stride 1 and ReLU activation. This layer convert word level features into more implicit local 5-gram features. At this stage, the feature maps have shape $(7, 1)$. They are then passed into `conv2` of 32by8 channels, each of size 3by1, again stride 1 and ReLU activation. At this stage, the feature maps have shape $(5, 1)$. They are then reshaped into 160 capsules each of 8-dimensions. A non-linearity function is applied on all capsules to squash their individual vector lengths back to the $[0, 1]$ range. This process generates the primary capsule outputs.

Then, we compute these $i = 160$ primary capsule outputs to j second layer capsules outputs,

[2]Embedding size is 50, 1-hot vector for PoS tags is 50, and 1-hot vector for Capitalization info is 5. Together these vectors are concatenated to a total length of 100.

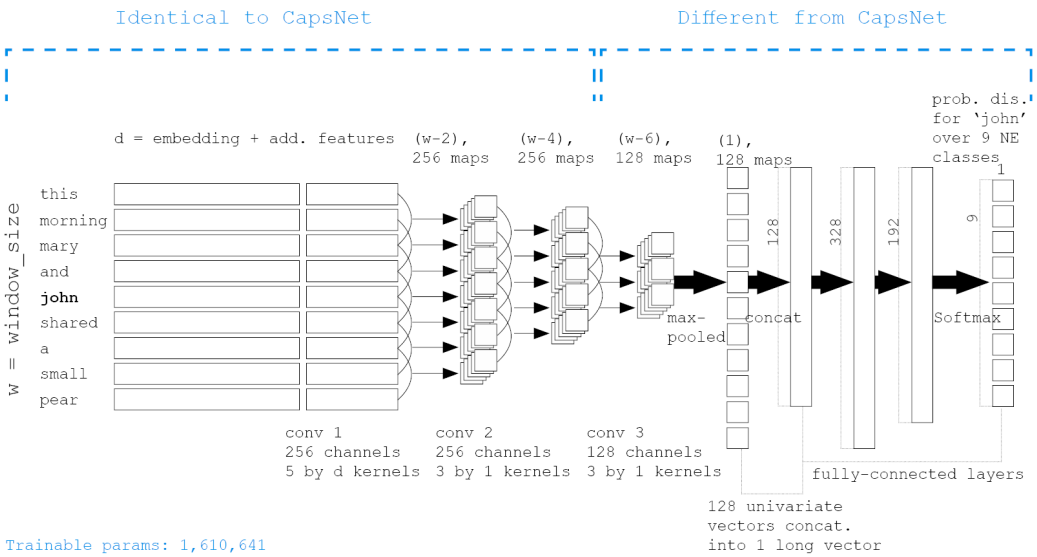

Figure 2: CNN Baseline Architecture for Named Entity Recognition

this mathematical procedure closely follows that of Sabour et al (2017). The second layer capsules each correspond to a named entity class so there are $j = 9$ capsules which we set each capsule to be 16-dimensional. After an elaborate inter-layer capsule prediction and 3-iteration dynamic routing process[3], the probability that the current word of interest belong to a particular named entity class j is now represented by the scalar length, a L-2 norm, of the corresponding second layer capsule vector. We back-propagate the same margin loss[4]proposed by Sabour et al (2017) during training. Since it allows multiple digits in MNIST recognition, we posit that it also allows overlapping entities on a particular word in named entity recognition.

As a regularization method, We implement a decoder structure that reconstruct the target word, the center word of the input window. Our decoder flatten final capsule layer outputs into a long vector of length 144, and then mask it by the gold named entity label at training or predicted named entity label at prediction. The resultant vector propagates through two feed-forward layers to reconstruct the 50-dimensional embedding of the target word. Cosine proximity loss is back-propagated during training. This decoder can be turned off optionally. The main model has a total of $1,509,440$ trainable parameters, and the full model with decoder has $2,705,490$ trainable parameters.

3.3 Baseline Architecture – CNN

A diagram of our baseline CNN architecture is provided in Figure 2. The architecture is a standard CNN with three convolutional layers. All layers up to `conv2` are identical to the CapsNet. We switch out the construction of capsules and dynamic routing layers for `conv3` (128 channels, size 3, stride 1 and ReLU activation), a max-pooling layer and two feed-forward layers of size 328 and 192. The output state is then connected to a 9-class Softmax layer and cross-entropy loss is back-propagated during training. In total, this model has $1,610,641$ parameters.

4 Results

We performed experiments on model architecture using the training and development sets, before evaluating three selected models on the test set.

4.1 Architecture Experimentation

We experimented with various architectural features on the development set, including decoder,

[3]Adapting equations (1,2,3) from Sabour et al. (2017), this process begins with multiplying primary capsule output $\mathbf{u}_i$ by weight matrix $\mathbf{W}_{ij}$ to give "prediction vectors" $\hat{\mathbf{u}}_{j|i}$. We then compute a weighted sum of $\hat{\mathbf{u}}_{j|i}$ corresponding to each named entity class j by coupling with coefficients c_{ij} and summing over all i. This weighted sum is annotated as $\mathbf{s}_j$. Finally, $\mathbf{s}_j$ is then squashed to give the second layer capsule output $\mathbf{v}_j$. An 3-iteration routing process is used to update c_{ij}. In the first iteration, we initialize $b_{ij} = 0$ to represent the log probabilities that capsule i should be coupled to j, then compute a softmax of b_{ij} to give the first iteration of c_{ij}. In the following iterations, we update b_{ij} by a delta of $\hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$ to compute $\mathbf{s}_j$ and $\mathbf{v}_j$ again.

[4]Adapting equation (4) from Sabour et al. (2017), margin loss L_k for digit class k is adapted to named entity class k. Total loss per example sums L_k over all 9 NER classes.

3 window sizes, re-trainable[5]GloVe or randomly initialized embeddings, and features additional to word embeddings. The major hyper-parameters we considered include convolution kernel sizes, number of channels, hidden layer sizes and dropout rates. Of the ten of forty-two best performing models on the development set, four has decoders; eight have window size 11 rather than 9 or 7; all use re-trainable embeddings; four use part-of-speech and capitalization features. CapsNet models makes up all of the top twenty-three models rather than CNNs. It seems that larger window size and re-trainable GloVe embeddings improve performance in general, while the additional parameters needed for the decoder and additional features cancels out their advantages. CNNs seem generally inferior to CapsNets in terms of F_1 rate but converges with fewer epochs. Table 1 highlights some models from this process.

#	Mod	Dec	Win	Emb	+Fea	$F_{\beta=1}$
1	CAP	off	11	glove	yes	92.24
2	CAP	off	11	glove	no	92.15
3	CAP	on	7	glove	no	92.11
24	CAP	off	9	glove	yes	90.93
26	CAP	off	11	rand.	yes	90.79
27	CNN	off	11	glove	no	90.59
33	CNN	off	9	glove	no	89.49

Table 1: Development set performance in %

4.2 Test Set Performance

We selected three models for test set evaluation. The first one is the top performing model on the development set, a CapsNet with no decoder, window size 11, retrainable GloVe embedding and additional word features. The second model is a variant of the first with a decoder. The third model is a CNN variant of the first with no decoder option. Table 2 shows that the first model performed best with a precision of 87.59%, recall of 87.33% and F_1 rate of 87.46%. However, the best CapsNet model only lead its CNN variant by 1.36% while the state-of-the-art model leads the best CapsNet model by 4.16%. A more in-depth analysis is necessary to investigate if the marginal performance difference between CapsNet and CNN models is meaningful in terms of language understanding.

[5]An earlier series of experiments show that re-trainable GloVe embeddings always performed better than non-trainable GloVe embeddings on this dataset.

Model	Precision	Recall	$F_{\beta=1}$
Chiu (2016)	91.39	91.85	91.62
CapsNet	87.59	87.33	87.46
CapsNet+decoder	86.40	87.17	86.78
CNN Baseline	85.93	86.23	86.08
CoNLL Baseline	71.91	50.90	59.61

Table 2: Test set performance in %

5 Error analysis

We make detailed comparisons of our CapsNet and CNN models through key performance metrics, language examples of erroneous predictions and target proximities from reconstructed embeddings.

5.1 Model Predictions overview

5.2 Error analysis scope

We investigate the errors through categories and rank the worst examples by descending order of KL-divergence. These categories include overall, per NER gold label, hallucinations [6], missed NER labels [7], NER label mismatches [8]. For each noteworthy error, we study both CapsNet and CNN predictions to provide the following insights. In addition, we compare the reconstructed embeddings against the input target word embeddings.

5.2.1 Common Weaknesses

5.2.2 Common Strengths

5.2.3 Capsule Routing versus Max-pooling

5.2.4 Decoder Analysis

6 Concluding Remarks

7 Acknowledgements

test

References

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[6]When the gold NER label shows no NER "O" class but a trained model predicts a NER non-"O" class.

[7]When the gold NER label correspond to a NER non-"O" class but a trained model predicts no NER "O" class.

[8]When both the gold label and trained model predicts a NER non-"O" class but there is a NER class mismatch.