Named Entity Recognition as a sequence labeling problem

Anonymous ACL submission

1 Introduction

Named Entity Recognition(NER) is a fundamental NLP task where the objective is to identify the named entities in a piece of text. In this work, we focus on designing powerful features and models for NER, which leads to reasonably good accuracy without the need of a complex model. We tried our approach on CONLL-2003 NER dataset which has four class of named entities: person, organization, location, and miscellaneous.

2 Information Encoding

2.1 Word Encoding

(a) Word Indicator BOW style 1-hot representation in the vocab space. Extremely **sparse** nature makes learning difficult and slow.
(b) SVD Word Embedding To combat with sparsity we take a k-rank $k = 300$ SVD of the BOW representation of training set offline.
(c) Pre-trained Word Embedding Glove pre-trained embedding per word

2.2 Context Encoding

We designed two types of context features:
(a) window context average of previous , current and next word encoding with different lengths $n \in [3, 5]$
(b) one sided context Average word encoding of either prev or next n words $n \in [1, 2]$

2.3 Syntactic Information Encoding

(a) POS Indicator 1-hot sparse representation of the Parts of speech of the token.
(b) Case Indicator We considered two types of casing indicators.
(a) Set if If the first char of the token is upper case and it's not the first word,
(b) Set if all the chars of the current token are uppercase.
n-gram We also used some n-gram features $n \in [1, 3]$ which should give some context information as well.

2.4 Tag information

While designing the emission features we also designed features not only per token in the vocabulary but also we constructed these features for each token-tag combination i.e. taking the tag as an input to the feature space. This is particularly useful since it helps understanding the transitions from one tag to another. We will discuss more on the quantitative observations in 3.

3 Experiments and Results

The approaches we tried can be classified into two broad categories.

3.1 Model Free Feature

In these setting we compute the features offline and then pass it as an input to the NER model.

3.1.1 Multi Layer Perceptron(MLP)

We used indicator features and word embeddings as input to a 3 layer Feed-forward neural network with ELU activation. More importantly, we trained it to predict 1 token at a time. While this does not perform really well as compared to some of the other models especially sequential models we tried but we do see some improved accuracy when the model is trained with both the individual word embedding as well as the average context embedding of its surrounding. Table 1 refers to this model as MLP:¡Feature Combinations¿

3.1.2 Hidden Markov Model(HMM)

The other model that we tried is a hidden markov model which is sequential model and learns the hidden states from the observation. Here, the transition matrix was calculated based on training data and fed to the forward algorithm which uses viterbi decoding to compute the hidden states (tags) given the observations (tokens) based on the esmiated transition matrix. This model performs quite well due to the sequential nature of viterbi decoding. However, since it does not update the potential paths it fails to perform really well compared to the likes of CRF or LSTM based models as can be compared in Table 1 referred as HMM.

3.1.3 Baseline Conditional Random Field(CRF)

CRFs are extremely popular class of graphical models, which was the state of the art until the recent advances of deep sequential models. Similar to HMMs CRF in its core also leverages viterbi decoding in a slightly modified way to compute the best possible transition path of hidden states. While HMMs are dependent on the estimated transition probabilities from data, CRFs are able to learn the weights during training which makes it much more powerful as compared to HMM. We tried traininf both the transition and the emission weights separately. While both perform closely , but letting the optimizer update both the weights leads to a slightly better performing model. The baseline CRF was trained using pre-computed emission features and letting it to update both transition and emission weight matrices. This would give something around 85% accuracy on CONLL dataset table 1 refers to this model as CRF: EM.

Model	Features	Prec	Recall	F1
HMM	Freq Count	85.44	69.90	76.89
MLP	WE	68.27	55.54	61.25
	WE+ CW1 + PI + UI	73.46	78.66	75.97
CRF	EM	85.93	85.16	85.54
	LSTM-WE	87.59	86.54	**87.06**
	MLP-WE	83.78	84.98	84.37

Table 1: Performance on CoNLL 2003 Shared Task on Named Entity Recognition (dev set) for different feature combinations. Legends: W: word, E: Embedding, C: Context, L: Left, R: Right, I: Indicator, P: Part of Speech, U: Uppercase, E: Emission ex: CW1 is context window of length 1 LSTM and MLP as feature implies o/p of LSTM trained with WE is used as an input to the model and trained jointly

3.2 Joint Feature Learning

Since CRF seems to be a very strong model, we tried to further improve it it in conjunction with other models. More specifically, we tried to learn the feature space using a different model and feeding that to another model and training them jointly.

3.2.1 MLP - CRF

We take our 3 layer Feed-forward neural network with ELU activation fed with pretrained word embeddings. More specifically, per sentence we construct a n*300 dimensional space where each word is replaced with a 300 dimensional word embedding. And then, we use the last layer of the model as the input to the CRF. And, we train them joinlty using standard optimizers. Using MLP as emission feature extraction did not help boosting the performance much most likely because of the carefully crafted features per tag level were much stronger than learnt using just pretrained word embeddings. **An importnat observation is that NER is highly sensitive to context**, while the same MLP performs really poor alone, with context features it performs much better. However, if we use MLP as an input to a CRF while excluding context features and entirely relying on MLP to learn it fails.

3.2.2 LSTM-CRF

This important observation about context motivated us to learn the feature space using a more powerful sequential model. Here we used single layer bidirectional LSTM initialized with pretrained word embedding as the embed layer. We use the last time-step of LSTM as the emission feature and feed this as an input to the CRF. This increases the accuracy much further as shown in table 1 and the most powerful model we have come up with.

4 Additional Experimental Setup

4.1 Modified Scaled Loss Function

One particular challenging aspect for this particular task was the imbalance between the classes. In particular there are several Unknown or unseen In these imbalanced settings it is very easy to learn a classifier that predicts a large number of tags corresponding to the most frequent class. To combat this problem we modified the loss function using a scaling factor per label to up-weight the minority class.

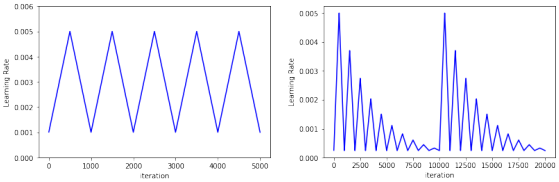
$$\alpha_k = \frac{card(y == k)}{\sum_{i=1}^{n}(card(y = i) - (card(y = k)}$$

4.2 Cyclically Annealed Learning Rate

Due to the non-convex nature of the optimization surface of DNNs, gradient based algorithms like stochastic gradient descent (SGD) are prone

to getting trapped in suboptimal local minima or saddle points. Adaptive learning rates like Adam and Adagrad try to solve this problem by tuning learning rate based on the magnitude of gradients. While in most cases they find a reasonably good solution, they usually can't explore the entire gradient landscape. Periodic random initialization or warm restarts often helps in better exploration of the landscape while showed that letting the learning rate(LR) vary cyclicaly (Cyclic learning rate(CLR)) 1a helps in faster convergence. Recently (Acharya et al., 2019) proposed a simulated annealing inspired LR update policy we call Cyclically Annealed Learning Rate (CALR). CALR is described in algorithm 1, (Fig. 1b). In addition to varying LR in a CLR style triangular windows, CALR exponentially decays the upper bound LR_{UB} of the triangular window from its initial value. When the learning rate decreases to a lower bound LR_{LB}, we increase the upper bound to its initial value LR_{UB} and continue with the LR updates. This exponential decay ensures slow steps on the optimization landscape. Intuitively, the proposed warm restarts, motivated by simulated annealing, make CALR more likely to escape from local minima or saddle points and enables better exploration of the parameter space. CALR should have similar but less aggressive effect as random initialization or warm restarts of LR. While exponentially decayed small cyclic windows let CALR carefully explore points local to the current solution, cyclic temperature increase helps it jump out and explore regions around other nearby local minima.

Throughout all the experiments we have used CALR in conjunction with Adam resulting in much faster convergence while the accuracy was not significantly improved.



(a) CLR (b) CALR

Figure 1: Comparison of CLR and CALR update Policy

Algorithm 1 Cyclically Annealed Learning Rate Schedule

procedure CALR(Iteration, Step Size, LR_{LB}, LR_{UB})
  $LR_{UB} \leftarrow LR_{UB}(init)$
  $LR_{LB} \leftarrow LR_{LB}(init)$
  for each EPOCH **do**
    $LR_{UB} \leftarrow LR_{UB} \times \exp(Decay)$
    if $LR_{UB} \leq LR_{LB}$ **then**
      $LR_{UB} \leftarrow LR_{UB}(init)$
    for each Training BATCH **do**
      LR←CLR(Iteration,StepSize, LR_{LB}, LR_{UB})
  return LR

procedure CLR(Iteration, Step Size, LR_{LB}, LR_{UB})
▷ this procedure implements a triangular window of width StepSize and height $LR_{UB} - LR_{LB}$
  $Bump \leftarrow \frac{LR_{UB}-LR_{LB}}{StepSize}$
  $Cycle \leftarrow (Iteration) \bmod (2 \times StepSize)$
  if $Cycle < StepSize$ **then**
    $LR \leftarrow LR_{LB} + (Cycle)$
  else
    $LR \leftarrow LR_{UB} - (Cycle - StepSize) \times Bump$
  return LR

4.3 Handling Invalid Transitions

It is very common in a NER setting to get invalid transitions especially in a multi-label sequence tagging setup.In order to bias the model to not learn invalid transitions we introduced high penalty term for invalid transitions. One simple way to do this was to set a very high negative weight associated with the transition matrix during initialization.

References

Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6196–6203.