Exploiting the Matching Information in the Support Set for Few Shot Event Classification

Viet Dac Lai[1], Franck Dernoncourt[2], and Thien Huu Nguyen[1]

[1] Department of Computer and Information Science, University of Oregon, USA
[2] Adobe Research, USA
vietl@cs.uoregon.edu, dernonco@adobe.com, thien@cs.uoregon.edu

Abstract. The existing event classification (EC) work primarily focuses on the traditional supervised learning setting in which models are unable to extract event mentions of new/unseen event types. Few-shot learning has not been investigated in this area although it enables EC models to extend their operation to unobserved event types. To fill in this gap, in this work, we investigate event classification under the few-shot learning setting. We propose a novel training method for this problem that extensively exploit the support set during the training process of a few-shot learning model. In particular, in addition to matching the query example with those in the support set for training, we seek to further match the examples within the support set themselves. This method provides more training signals for the models and can be applied to every metric-learning-based few-shot learning methods. Our extensive experiments on two benchmark EC datasets show that the proposed method can improve the best reported few-shot learning models by up to 10% on accuracy for event classification.

Keywords: Event classification · Auxiliary Loss · Few-shot learning.

1 Introduction

Event Classification (EC) is an important task of Information Extraction (IE) in Natural Language Processing (NLP). The target of EC is to classify the event mentions for some set of event types (i.e., classes). Event mentions are often associated with some words/phrases that are responsible to trigger the corresponding events in the sentences. For example, consider the following two sentences:

(1) *The companies **fire** the employee who wrote anti-diversity memo.*
(2) *The troops were ordered to cease **fire***

In these examples, an EC system should be able to classify the word "*fire*" in the two above sentences as an *Employment-Termination* event and an *Attack* event, respectively. As demonstrated by the examples, a notable challenge in EC is that the similar surface forms of the words might convey different events depending on the context. Two main methods have been employed for EC. The first approach explores linguistic features (e.g., syntactic and semantic properties) to train statistical models [9]. The second approach, on the other hand,

focuses on developing deep neural network models (e.g., convolutional neural network (CNN) and recurrent neural network (RNN)) to automatically learn effective features from large scale datasets [5, 13]. Due to the development of the deep learning models, the performance for EC has been improved significantly [19, 17, 16, 14, 23].

The current EC models mainly employ the traditional supervised learning setting [19, 17] where the set of event types for classification has been predetermined. However, once a model is trained on the datasets with the given set of event types, it is unable to detect event mentions of unseen event types. To extend EC to new event types, a common solution is to annotate additional training data for such new event types and re-train the models, which is extremely expensive. It is thus desirable to formalize EC in the few-shot learning setting where the systems need to learn to recognize event mentions for new event types from a handful of examples. This is, in fact, closer to how humans learn to do tasks and make the EC models more applicable in practice. However, to our knowledge, there has been no prior work on few-shot learning for EC.

In few-shot learning, we are given a support set and a query instance. The support set contains examples from a set of classes (e.g. events in EC). A learning model needs to predict the class, to which the query instance belongs, among the classes presented in the support set. This is done based on the matching information between the query example and those in the support set. To apply this setting to extract the examples of some new type, we need to collect just a few examples of the new type and add them to the support set to form a new class. Afterward, whenever we need to predict whether a new example has the new type or not, we can set it as the query example and perform the models in this setting.

In practice, we often have some existing datasets (denoted by D) with examples for some pre-defined types. The previous work on few-shot learning has thus exploited such datasets to simulate the aforementioned few-shot learning setting to train the models [26]. Basically, in each episode of the training process, a subset of the types in D is sampled for which a few examples are selected for each type to serve as the support set. Some other examples are also chosen from the remaining examples of each sampled type to establish the query points. The models would then be trained to correctly map the query examples to their corresponding types in the support set based on the context matching of the examples [7].

One potential issue with this training procedure is that the training signals for the models only come from the matching information between the query examples and the examples in the support set. The available matching information between the examples in the support set themselves is not yet explored in the existing few-shot learning work [28, 26], especially for the NLP tasks [7]. While this approach can be acceptable for the tasks in computer vision, it might not be desirable for NLP applications, especially for EC. Overall, datasets in NLP are much smaller than those in computer vision, thus limiting the variety of the context for training purposes. The ignorance of the matching information for the

examples in the support set might cause inefficiency in using the training data for EC where the models cannot fully exploit the available information and fail to achieve good performance. Consequently, in this work, we propose to simultaneously exploit the matching information between the examples in the support set and between the query examples with the examples in the support set to train the few-shot learning models for EC. This is done by adding additional terms in the loss function (i.e., the auxiliary losses) to capture the matching knowledge between the examples in the support set. We expect that this new training technique can better utilize the training data and improve the performance of few-shot learning in EC.

We extensively apply the proposed training method on different metric learning models for few-shot learning on two benchmark EC datasets. The experiments show that the new training technique can significantly improve all the considered few-shot learning methods over the two datasets with a large performance gap. In summary, the contribution of this work includes: (i) for the first time in the literature, we study the few-shot learning problem for event Classification, (ii) we propose a novel training technique for the few-shot learning models based on metric learning. The proposed training method exploits the matching information between the examples in the support set as additional training signals, and (iii) we achieve the state-of-the-art performance for EC on the few-shot learning setting, functioning as the baselines for the future research in this area.

2 Related Work

Early studies in event classification mainly focus on designing linguistic features [1, 9, 12] for statistical models. Due to the development of deep learning, many advanced network architectures have been investigated to advance the event classification accuracy [5, 19, 17, 18, 21, 13, 22]. However, none of them investigates the few-shot learning problem for EC as we do in this work. Although some recent studies have considered a related setting where event types are augmented with some keywords [3, 24, 11], these works do not explicitly examine the few-shot learning setting as we do in this work. Some other efforts on zero-shot learning for event classification [8] are also related to our work in this paper.

Few-shot learning facilitates the models to learn effective latent features without large scale data. The early studies apply transfer learning to fine-tune the pre-trained models, exploiting the latent information from the common classes with adequate instances [4, 2]. Metric learning, on the other hand, learns to model the distance distribution among the observed classes [10, 28, 26]. Recently, the idea of a fast learner that can generalize to a new concept quickly is introduced in meta-learning [25, 6]. Among these methods, metric-learning is more explainable and easier to train and implement compared to transfer learning and meta-learning. Notably, the prototypical networks in metric learning achieve state-of-the-art performance on several FSL benchmarks and show its robustness against noisy data [26, 7]. Although many FSL methods are proposed for

image recognition [10, 28, 26, 6, 25], there have been few studies investigating this setting for NLP problems [7, 29].

3 Methodology

3.1 Notation

The task of few-shot event classification is to predict the event type of a query example x given a support set S and a set of event type $T = \{t_1, t_2, \ldots, t_N\}$ (N is the number of event types). In few-shot learning, S contains a few examples for each event type in T. For convenience, we denote the support set as:

$$\begin{aligned} S = &\{(s_1^1, a_1^1, t_1), \ldots, (s_1^{K_1}, a_1^{K_1}, t_1) \\ &\ldots \\ &(s_N^1, a_N^1, t_N), \ldots, (s_N^{K_N}, a_N^{K_N}, t_N)\}, \end{aligned} \tag{1}$$

where (s_i^j, a_i^j, t_i) indicates that the a_i^j-th word in the sentence s_i^j is the trigger word of an event mention with the event type t_i, and $K_1, K_2, \ldots, K_N$ are the numbers of examples in the support set for each type $t_1, t_2, \ldots, t_N$ respectively. For simplicity, we use $w_1, w_2, \ldots, w_l$ to represent the word sequence for some sentence with length l in this work.

Similarly, the query example x can also be represented by $x = (q, p, t)$ where q, p and t represent the query sentence, the position of the trigger word in the sentence, and the true event type for this event mention respectively. Note that $t \in T$ is only provided in the training time and the models need to predict this event type in the test time.

In practice, the numbers of support examples in S (i.e., $K_1, \ldots, K_N$) may vary. However, to ease the processing and speed up the training process with GPU, similar to recent studies in FSL [7], we employ the N-way K-shot FSL setting. In this setting, the numbers of instances per class in the support set are equal ($K_1 = \ldots = K_N = K > 1$) and small ($K \in \{5, 10\}$).

Note that to evaluate the few-shot learning models for EC, we would need the training data D_{train} and the test data D_{test}. For few-shot learning, it is crucial that the sets of event types in D_{train} and D_{test} are disjoint. The event type set T in each episode would then be a sample of the sets of event types in D_{train} or D_{test}, depending on the training and evaluation time respectively. Also, as mentioned in the introduction, in one episode of the training process, a set of query examples (i.e., the query set) would be sampled so it involves the similar event types T as the support set, and the examples for each type in the query set would be different from those in the support set. At the test time, the classification accuracy of the models over all the examples in the test set would be evaluated.

3.2 Few-shot Learning for Event Classification

The few-shot learning framework for EC in this work follows the typical metric learning structures in the prototypical networks [26, 7], involving three major components: instance encoder, prototypical module, classifier module.

Instance encoder Given a sentence $s = \{w_1, w_2, \ldots, w_l\}$ and the position of the trigger word a (i.e., w_a is the trigger word of the event mention in s and (s, a) can belong to an example in S or the query example), following the common practice in EC [19, 5], we first convert each word $w_i \in s$ into a real-valued vector to facilitate the neural computation in the following steps. In particular, in this work, we represent each word w_i using the concatenation of the following two vectors:

- The pre-trained word embedding of w_i: this vector is expected to capture the hidden syntactic and semantic information for w_i [15].
- The position embedding of w_i: this vector is obtained by mapping its relative distance to the trigger word w_a (i.e., $i - a$) to an embedding vector in the position embedding table. The position embedding table is initialized randomly and updated during the training process of the models. The purpose of the position embedding vectors is to explicitly inform the models of the position of the trigger word in the sentence [5].

After converting w_i into a representation vector e_i, the input sentence s becomes a sequence of representation vectors $E = e_1, e_2, \ldots, e_l$. Based on this sequence of vectors, a neural network architecture f would be used to transform E into an overall representation vector v to encode the input example (s, m) (i.e., $v = f(s, m)$). In this work, we investigate two network architectures for the encoding function f, i.e., one early architecture for EC based on CNN and one recent popular architecture for NLP based on Transformers:

CNN encoder: This model applies the temporal convolution operation with some window size k and multiple filters over the input vector sequence E, producing a hidden vector for each position in the input sentence. Such hidden vectors are then aggregated via the max-pooling operation to obtain the overall representation vector v for (s, m) [5, 7].

Transformer encoder: This is an advanced model to encode sequences of vectors based on attention mechanism without recurrent neural network [27]. The transformer encoder involves multiple layers; each of them consumes the sequence of hidden vectors from the previous layer to generate the sequence of hidden vectors for the current layer. The first layer would take E as the input while the hidden vector sequence returned by the last layer (i.e., the vector at the position a of the trigger word) would be used to constitute the overall representation vector v in this case. Each layer in the transformer encoder is composed of two sublayers (i.e., a multi-head self-attention layer and a feed-forward layer) augmented with a residual connection around them [27].

Prototypical module The prototypical module aims to compute a single prototype vector to represent each class in T of the support set. In this work, we consider two versions of this prototypical module in the literature. The first version is from the original prototypical networks [26]. It simply obtains the prototype vector c_i for a class t_i using the average of the representation vectors of the examples with the event type t_i in the support set S:

$$\mathbf{c}^i = \frac{1}{K} \sum_{(s_i^j, a_i^j, t_i) \in S} f(s_i^j, a_i^j) \tag{2}$$

The second version, on the other hand, comes from the hybrid attention-based prototypical networks [7]. The prototype vector is a weighted sum of the representation vectors of the examples in the support set. The example weights (i.e., the attention weights) are determined by the similarity of the examples in the support set with respect to the query example $x = (q, p, t)$:

$$\begin{aligned} \mathbf{c}^i &= \sum_{(s_i^j, a_i^j, t_i) \in S} \alpha_{ij} f(s_i^j, a_i^j) \\ \text{where } \alpha_{ij} &= \frac{\exp(b_{ij})}{\sum_{(s_i^k, a_i^k, t_i) \in S} \exp(b_{ik})} \\ b_{ij} &= \sigma(f(s_i^j, a_i^j) \odot f(q, p)) \end{aligned} \tag{3}$$

In this formula, $\odot$ is the element-wise multiplication and *sum* is the summation operation done over all the dimensions of the input vector.

Classifier module In this module, we compute the probability distribution over the possible types for x in T using the distances from the query example $x = (q, p, t)$ to the prototypes of the classes/event types T in the support set:

$$P(y = t_i | x, S) = \frac{\exp(-d(f(q, p), \mathbf{c}^i))}{\sum_{j=1}^{N} \exp(-d(f(q, p), \mathbf{c}^j))} \tag{4}$$

where d is a distance function, and $\mathbf{c}^i$ and $\mathbf{c}^j$ are the prototype vectors obtained in either Equation (2) or Equation (3).

In this paper, we consider three popular distance functions in different few-shot learning models using metric learning:

- Cosine similarity in matching networks (called **Matching**) [28]
- Euclidean distance in the prototypical networks. Depending on whether the prototype vectors are computed with Equation 2 or 3, we have two variations of this distance function, called as **Proto** [26], and **Proto+Att** (i.e., in hybrid attention-based prototypical networks [7]) respectively.
- Learnable distance function using convolutional neural networks in relation networks (called **Relation**)

Given the probability distribution $P(y|x, S)$, the typical way to train the few shot learning framework is to optimize the negative log-likelihood function for x (with t as the ground-truth event type for x) [26, 7]:

$$L_{query}(x, S) = -\log P(y = t|x, S) \quad (5)$$

Matching the examples in the support set The typical loss function for few-shot learning in Equation 5 aims to learn by matching the query example x with the examples in the support set S via the prototype vectors. An issue with this mechanism is it only employs the matching signals between the query example and the support examples for training. This can be acceptable for large datasets (e.g., in computer vision) where many examples can play the role of the query examples to provide sufficient training signals for the learning process. However, for EC, the available datasets are often small (e.g., the ACE 2005 dataset with only about a few thousands of annotated event mentions), making the sole reliance on the query examples for training signals less efficient. In other words, the few-shot learning framework might not be trained well with the limited data for the query matching for EC. Consequently, in this work, we propose to introduce more training signals for few-shot learning for EC by additionally exploiting the matching information among the examples in the support set themselves. In particular, as there are multiple examples (although only a few) per class/type in the support set, we select a subset of such examples for each type in S and enforce the models to be able to match such the selected examples to their corresponding types in the remaining support set.

Formally, let $S_i = \{(s_i^1, a_i^1, t_i), \ldots, (s_i^K, a_i^K, t_i)\} \forall 1 \leq i \leq N$ so $S = S_1 \cup S_2 \ldots \cup S_N$. Let Q be some integer that is less than K (i.e., $1 \leq Q < K$). For each type t_i, we randomly select Q examples from S_i (called the auxiliary query examples), forming the auxiliary query set S_i^Q (i.e., $S_i^Q \subset S_i$, $|S_i^Q| = Q$). The remaining set of S_i is then denoted by $S_i^S = S_i \setminus S_i^Q$. We unify the sets S_i^S to constitute an auxiliary support set S^S while the union of S_i^Q serves as the auxiliary query set: $S^S = S_1^S \cup S_2^S \cup \ldots \cup S_N^S, S^Q = S_1^Q \cup S_2^Q \cup \ldots \cup S_N^Q$.

Given the auxiliary support set S^S, we seek to enhance the training signals for the few-shot models by matching the examples in the auxiliary query set S^Q with S^S. Specifically, we first use the same networks in the instance encoder and prototypical modules to compute the auxiliary prototypes for the classes in T of the auxiliary support set S^S. For each auxiliary example $z = (s_z, a_z, t_z) \in S^Q$ (s_z, a_z and t_z are the sentence, the trigger word position and the event type in z respectively), we use the network in the classifier module to obtain the probability distribution $P(.|z, S^S)$ over the possible event types for z based on the auxiliary support set S^S. Afterward, we enforce that the models can correctly predict the event types for all the examples in the auxiliary query sets S_i^Q given the support set S^S by introducing the auxiliary loss function:

$$L_{aux}(S) = -\sum_{i=1}^{N} \sum_{z=(s_z,a_z,t_i) \in S_i^Q} \log P(y = t_i|z, S^S) \quad (6)$$

Eventually, the overall loss function to be optimized to train the models in this work is: $L(x, S) = L_{query}(x, S) + \lambda L_{aux}(S)$ where λ is a trade-off parameter between the main loss function and the auxiliary loss function. For convenience, we call the training method with the auxiliary loss function for few shot learning in this section *LoLoss* (i.e., *leave-out loss*) in the following experiments.

4 Experiments

4.1 Datasets

We evaluate all the models in this study on the ACE 2005. ACE 2005 involves 33 event subtypes which are categorized into 8 event types: *Business, Contact, Conflict, Justice, Life, Movement, Personnel, and Transaction*. The TAC KBP dataset, on the other hand, contains 38 event subtypes for 9 event types. Due to the larger numbers of the event subtypes, we will use the subtypes in these datasets as the classes for our few-shot learning problem.

As we want to maximize the numbers of examples in the training data, for each dataset (i.e., ACE 2005 or TAC KBP 2015), we choose the event subtypes in 4 event types that have the least number of examples in total and split at the ratio 1:1 into the test and development classes. Following this heuristics to select the classes, the event types used for training data in ACE 2005 involve *Business*, *Contact*, *Conflict*, and *Justice* while the event types for testing and development data are *Life*, *Movement*, *Personnel*, and *Transaction*. For TAC KBP 2015, the training classes include *Business*, *Contact*, *Conflict*, *Justice*, and *Manufacture* while the test and development classes consist of *Life*, *Movement*, *Personnel*, and *Transaction*. Finally, due to the intention to follow the prior work on few-shot learning with 10 examples per class in the support set and 5 examples per class in the query set for training [7], we remove the examples of any subtypes whose have less than 15 examples in the training, test and development sets of the datasets.

4.2 Hyper-Parameters

Similar to the prior work [7], we evaluate all the models using N-way K-shot FSL settings with $N, K \in \{5, 10\}$. For training, we avoid feeding the same set of event subtypes in every batch to make training batches more diverse. Thus, following [7], we sample 20 event subtypes for each training batch while still keeping either 5 or 10 classes in the test time.

We initialize the word embeddings using the pre-trained GloVe embeddings with 300 dimensions. The word embeddings are updated during the training time as in [20]. We also randomly initialize the position embedding vectors with 50 dimensions. The other parameters are selected based on the development data of the datasets, leading to similar parameters for both ACE 2005 and TAC KBP 2015. In particular, the CNN encoder contains a single CNN layer with window size 3 and 250 filters. We manage to use this simple CNN encoder to have a

fair comparison with the previous study [7]. The Transformer encoder contains 2 layers with a context size of 512 and 10 heads in the attention mechanism. The number of examples per class in the auxiliary query sets Q is set to 2 while the trade-off parameter λ in the loss function is 0.1. We use stochastic gradient descent with a learning rate of 0.001 to optimize the models.

4.3 Results

Table 1 shows the accuracy of the models (i.e., Matching, Proto, Proto+Att, and Relation) on the ACE 2005 test dataset, using the CNN encoder and Transformer encoder. There are several observations from the table. First, comparing the instance encoders, it is clear that the transformer encoder is significantly better than the CNN encoder across all the possible few-shot learning models and settings for EC. Second, comparing the few-shot learning models, the prototypical networks significantly outperform Matching and Relation with a large performance gap across all the settings. Among the prototypical networks, Proto+Att achieves better performance than Proto, thus confirming the benefits of the attention-based mechanism for the prototypical module. Third, comparing the pairs (5-way 5-shot vs 5-way 10-shot) and (10-way 5 shot vs 10 way 10 shot), we see that the performance of the models would be almost always better with larger K (i.e., the number of examples per class in the support set) on different settings, consistent with the natural intuition about the benefit of having more examples for training.

FSL Setting	**5 way 5 shot**	**5 way 10 shot**	**10 way 5 shot**	**10 way 10 shot**	**5 way 5 shot**	**5 way 10 shot**	**10 way 5 shot**	**10 way 10 shot**
	CNN Encoder				Transformer Encoder			
Matching	45.81	49.01	30.41	35.66	71.83	76.51	61.2	66.79
Matching+LoLoss	**51.78**	**52.64**	**32.48**	**39.15**	**78.13**	**83.42**	**68.91**	**75.30**
Proto	70.92	74.40	57.59	62.67	78.07	82.64	68.77	74.99
Proto+LoLoss	**76.98**	**82.19**	**66.92**	**73.63**	**81.27**	**86.20**	**73.07**	**79.63**
Proto+Att	72.26	74.22	57.28	64.36	80.77	83.96	72.78	77.97
Proto+Att+LoLoss	**76.93**	**75.59**	**67.54**	**66.70**	**83.38**	**87.20**	**76.03**	**81.79**
Relation	36.33	33.75	24.21	18.04	51.22	55.47	36.98	39.89
Relation+LoLoss	**37.86**	**38.52**	**25.99**	**23.47**	**54.74**	**56.60**	**39.74**	**41.69**

Table 1. Accuracy of event classification on ACE-2005 dataset. **+LoLoss** indicates the use of the auxiliary loss.

Most importantly, we see that training the models with the LoLoss procedure would significantly improve the models' performance. This is true across different few-shot learning models, N-way K-shot settings, and encoder choices. The results clearly demonstrate the effectiveness of the proposed training procedure to exploit the matching information between examples in the support set for few-shot learning for EC. For simplicity, we only focus on the best few-shot learning models (i.e., the prototypical networks) and the Transformer encoder under 5-way 5-shot and 10-way 10-shot in the following analysis. Even though

we show the results in fewer settings and models in table 2 and 3, the same trends are observed for the other models and settings as well.

Table 2 additionally reports the accuracy of Transformer-based models on the TAC KBP 2015 dataset. As we can see from the table, most of our observations for the ACE 2005 dataset still hold for TAC KBP 2015, once again confirming the advantages of the proposed LoLoss technique in this work.

Model	**5 way 5 shot**	**10 way 10 shot**
Matching	72.78	65.55
Matching+LoLoss	**75.58**	**68.53**
Proto	78.08	73.23
Proto+LoLoss	**78.88**	**74.82**
Proto+Att	75.35	71.28
Proto+Att+LoLoss	**79.93**	**76.37**
Relation	50.97	34.91
Relation+LoLoss	**51.65**	**35.13**

Table 2. Accuracy of the models with the Transformer encoder on the TAC-KBP test dataset. +LoLoss indicates the use of the auxiliary loss.

4.4 Robustness against noise

In this section, we seek to evaluate the robustness of the few-shot learning models against the possible noise in the training data. In particular, in each training episode where a set of examples is sampled for each type in T to form the query set Q, we simulate the noisy data by randomly selecting a portion of the examples in Q for label perturbation. Essentially, for each example in the selected subset of Q, we change its original label to another random one in T, making it a noisy example with an incorrect label. By varying the size of the selected portion in Q for label perturbation, we can control the level of noise in the training process for FSL in EC.

Noise rate	**Model**	**5 way 5 shot**	**10 way 10 shot**
20%	Proto+Att	70.08	59.55
	Proto+Att+LoLoss	**74.61**	**64.66**
30%	Proto+Att	67.38	57.08
	Proto+Att+LoLoss	**72.45**	**62.65**
50%	Proto+Att	60.50	50.67
	Proto+Att+LoLoss	**65.29**	**55.21**

Table 3. The accuracy on the ACE-2005 test set with different noise rates.

Table 3 shows the accuracy of the Proto+Att model on the ACE 2005 test set that employs the Transformer encoder with or without the LoLoss training procedure for different noise rates. As we can see from the table, the introduction of noisy data would, in general, degrade the accuracy of the models (i.e., comparing the cells in Table 3 with the Proto+Att based model in Table 1). However, over different noise rates and N way K shot settings, the Proto+Att model trained

with LoLoss is still always significantly better than those without LoLoss. The performance gap is substantial that is at least 4.5% over different settings. In fact, we see that LoLoss can improve Proto+Att in the noisy setting (i.e., at least 4.5%) more significantly than those in the setting without noisy data (i.e., at most 3.3% on the 5 way 5 shot and 10 way 10 shot settings in Table 1). Such evidence further confirms the effectiveness and robustness against noisy data of LoLoss for few-shot learning due to its exploitation of the matching information between the examples in the support set.

5 Conclusion

In this paper, we perform the first study on few-shot learning for event classification. We investigate different metric learning methods for this problem, featuring the typical prototypical network framework with several choices for the instance encoder (i.e., CNN and Transformer). In addition, we propose a novel technique, called LoLoss, to train the few-shot learning models for EC based on the matching information for the examples in the support set. The proposed LoLoss technique is applied to different few-shot learning methods for different datasets and settings that altogether help to significantly improve the performance of the baseline models. In the future, we plan to examine LoLoss for few-shot learning for other NLP and vision problems (e.g., relation extraction, image classification).

Acknowledgments

This research has been supported in part by Vingroup Innovation Foundation (VINIF) in project code VINIF.2019.DA18 and Adobe Research Gift. This research is also based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Pro

References

1. Ahn, D.: The stages of event extraction. In: Proceedings of the Workshop on Annotating and Reasoning about Time and Events. pp. 1–8 (2006)
2. Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. In: Proceedings of ICML workshop on unsupervised and transfer learning (2012)
3. Bronstein, O., Dagan, I., Li, Q., Ji, H., Frank, A.: Seed-based event trigger labeling: How far can event descriptions get us? In: ACL-IJCNLP (2015)
4. Caruana, R.: Learning many related tasks at the same time with backpropagation. In: NIPS (1995)
5. Chen, Y., Xu, L., Liu, K., Zeng, D., Zhao, J.: Event extraction via dynamic multi-pooling convolutional neural networks. In: ACL-IJCNLP (2015)
6. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML (2017)

7. Gao, T., Han, X., Liu, Z., Sun, M.: Hybrid attention-based prototypical networks for noisy few-shot relation classification. In: AAAI (2019)
8. Huang, L., Ji, H., Cho, K., Voss, C.R.: Zero-shot transfer learning for event extraction. In: ACL. pp. 2160–2170 (2018)
9. Ji, H., Grishman, R.: Refining event extraction through cross-document inference. In: ACL (2008)
10. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: ICML deep learning workshop. vol. 2 (2015)
11. Lai, V.D., Nguyen, T.: Extending event detection to new types with learning from keywords. In: Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019) (2019)
12. Li, Q., Ji, H., Hong, Y., Li, S.: Constructing information networks using one single model. In: EMNLP (2014)
13. Liu, S., Chen, Y., Liu, K., Zhao, J.: Exploiting argument information to improve event detection via supervised attention mechanisms. In: ACL (2017)
14. Lu, W., Nguyen, T.H.: Similar but not the same: Word sense disambiguation improves event detection via neural representation matching. In: EMNLP (2018)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS (2013)
16. Nguyen, T.H., , Meyers, A., Grishman, R.: New york university 2016 system for kbp event nugget: A deep learning approach. In: TAC (2016e)
17. Nguyen, T.H., Cho, K., Grishman, R.: Joint event extraction via recurrent neural networks. In: NAACL (2016)
18. Nguyen, T.H., Fu, L., Cho, K., Grishman, R.: A two-stage approach for extending event detection to new types via neural networks. In: Proceedings of the 1st ACL Workshop on Representation Learning for NLP (RepL4NLP) (2016b)
19. Nguyen, T.H., Grishman, R.: Event detection and domain adaptation with convolutional neural networks. In: ACL-IJCNLP (2015)
20. Nguyen, T.H., Grishman, R.: Relation extraction: Perspective from convolutional neural networks. In: Proceedings of the 1st NAACL Workshop on Vector Space Modeling for NLP (VSM) (2015a)
21. Nguyen, T.H., Grishman, R.: Modeling skip-grams for event detection with convolutional neural networks. In: EMNLP (2016d)
22. Nguyen, T.H., Grishman, R.: Graph convolutional networks with argument-aware pooling for event detection. In: AAAI (2018a)
23. Nguyen, T.M., Nguyen, T.H.: One for all: Neural joint modeling of entities and events. In: AAAI (2019)
24. Peng, H., Song, Y., Roth, D.: Event detection and co-reference with minimal supervision. In: EMNLP (2016)
25. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., Lillicrap, T.: Meta-learning with memory-augmented neural networks. In: ICML (2016)
26. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: NIPS (2017)
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: NIPS (2017)
28. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In: NIPS (2016)
29. Yu, M., Guo, X., Yi, J., Chang, S., Potdar, S., Cheng, Y., Tesauro, G., Wang, H., Zhou, B.: Diverse few-shot text classification with multiple metrics. arXiv preprint arXiv:1805.07513 (2018)