Digital Investigation 18 (2016) 33–49

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Detecting predatory conversations in social media by deep Convolutional Neural Networks

Mohammadreza Ebrahimi [a], [*], Ching Y. Suen [b], Olga Ormandjieva [b]

[a] Centre of Pattern Recognition and Machine Intelligence, Concordia University, EV 11.155, 1455 de Maisonneuve West, Montreal H3G 1M8, Quebec, Canada
[b] Department of Computer Science & Software Engineering, Concordia University, Canada

ARTICLE INFO

Article history:
Received 24 February 2016
Received in revised form 29 June 2016
Accepted 1 July 2016
Available online 5 July 2016

Keywords:
Online predator identification
Convolutional neural network
Predatory conversation
Support vector machine
Deep learning
Word embedding
Language model

ABSTRACT

Automatic identification of predatory conversations in chat logs helps the law enforcement agencies act proactively through early detection of predatory acts in cyberspace. In this paper, we describe the novel application of a deep learning method to the automatic identification of predatory chat conversations in large volumes of chat logs. We present a classifier based on Convolutional Neural Network (CNN) to address this problem domain. The proposed CNN architecture outperforms other classification techniques that are common in this domain including Support Vector Machine (SVM) and regular Neural Network (NN) in terms of classification performance, which is measured by F_1-score. In addition, our experiments show that using existing pre-trained word vectors are not suitable for this specific domain. Furthermore, since the learning algorithm runs in a massively parallel environment (i.e., general-purpose GPU), the approach can benefit a large number of computation units (neurons) compared to when CPU is used. To the best of our knowledge, this is the first time that CNNs are adapted and applied to this application domain.

Introduction

Investigating large corpora of chat logs in forensic studies in order to identify online predators cannot be done manually. The task is critical for early identification of the potential predators who abuse children and juveniles. We address the task as Online Predator Identification (OPI). The PAN-2012 competition (Inches and Crestani, 2012) demonstrated that using machine learning and text mining techniques can be of significant help to the manual investigation process done by forensic investigators. It showed that applying text mining process can decrease the search space by flagging the most potential predatory conversations. This translates directly into a *binary text classification* problem:

Let $D \subset X \times Y$ be the data set that contains the conversations, where $X = \{\chi_1, \chi_2, \ldots, \chi_n\}$ is the set of conversations so that $\chi_i = (x_1, x_2, \ldots, x_m)^T$ is an m-dimension feature vector for ith conversation. Also, let $Y = \{p, np\}$ be the set of class labels in binary classification problem in which non-predatory and predatory conversations are denoted by np and p respectively. The goal is to assign the right label from Y to each conversation.

After having several consultation sessions with experts in Sûreté du Québec, the corresponding police department responsible for combating this type of crime, we present an empirical approach to solve the mentioned classification problem. Specifically, we use a special deep learning method called Convolutional Neural Network (CNN) to present an accurate classifier for the online predator

[*] Corresponding author.
E-mail addresses: mo_ebrah@encs.concordia.ca (M. Ebrahimi), suen@cse.concordia.ca (C.Y. Suen), ormandj@cse.concordia.ca (O. Ormandjieva).

http://dx.doi.org/10.1016/j.diin.2016.07.001
1742-2876/© 2016 Elsevier Ltd. All rights reserved.

identification domain. Then, we compare the performance of our model to the current methods. We show that the proposed CNN architecture improves the performance of the classification in terms of a widely accepted indicator for supervised learning algorithms, F_1-measure.

Our CNN architecture convincingly outperforms Support Vector Machines (SVM) and also the traditional Neural Networks (NN). In addition, inspired by Johnson's approach (Johnson and Zhang, 2015a), we learn the word representation internally rather than using general pre-trained word vectors such as word2vec (Mikolov et al., 2013). We show that better results can be achieved by using this approach compared to when an existing pre-trained word vector representation is used (Section Investigating the effect of convolution). The results of our work can be used as a good starting point for other researchers or practitioners who are interested in the application of deep learning methods in this application domain.

In the background section, we first provide a brief introduction to common deep neural network algorithms in text classification. Then we narrow the topic down to the application of CNN in the domain of text classification and OPI. This section ends with introducing the features of common deep learning software tools that can guide researchers to select the appropriate one based on their needs. In Section CNN for OPI, we describe the proposed CNN architecture and hyper parameter tuning. Section Experiments covers the results obtained from applying CNNs to this domain. Section Conclusion and discussion is devoted to our best practices and the discussion about our experiments in this work, and finally Section Future work describes our future works.

Background

This section provides the basic knowledge about deep neural-network-based methods for text and more specifically Convolutional Neural Network, which is the main topic of this paper. It starts with the time line of applying classification algorithms to the OPI in the past and ends with a brief introduction to corresponding frameworks that are commonly used in practice.

Online predator identification with classification

One of the earliest attempts for applying classifiers to the OPI problem was accomplished by Pendar (2007) who used a weighted k-Nearest Neighbor (k-NN) classifier to recognize predators from underage victims. The first empirical system with the goal of determining predatory messages in chat logs was ChatCoder1 (Kontostathis, 2009) which was followed by ChatCoder2 in 2011 (Mcghee et al., 2011). The system used a rule-based approach in conjunction with decision trees and k-NN classifier as an instance-based learning method.

Later, classification algorithms that have been used to address the OPI problem in the past, cover a wide range including Entropy-based Classification (Eriksson and Karlgren, 2012), k-Nearest Neighbor (Kang et al., 2012), traditional Neural Networks (Villatoro-Tello et al., 2012), and Support Vector Machine (Morris, 2013).

Later, Escalante et al. (2013) proposed using chained-classifiers based on adapting a psychological hypothesis that underscores three stages employed by predators to approach the victim. The work suggests that adopting psycholinguistic hypotheses can improve the classification accuracy.

Bogdanova et al. (2014), enriched the lexical features by adding high-level features such as emotions, neuroticism, and psychological aspects. Also, Cano et al. (2014) incorporated sentiment features, psycholinguistic features, and discourse patterns as additional features to the original bag-of-words model in order to improve the classification accuracy. Finally, Ebrahimi et al. (2016) dealt with the OPI problem from anomaly detection perspective by using one-class SVM classification algorithm. In this work, we tackle the problem by applying a deep learning approach to this area and we compare the results with the most successful commonly used classification algorithms (i.e., traditional neural network and support vector machine).

Deep neural-network-based methods for text

Deep learning has emerged as a hot trend in artificial intelligence and machine learning. The paradigm is called deep learning because of the fact that there is a hierarchy of numerous layers in the main model and each layer encodes a level of abstraction in the training data. Using these models has been proven to be more efficient than the simple data mining and machine learning models. Since some deep learning methods, such as autoencoders and recurrent neural networks have generative/discriminative or supervised/unsupervised variants, it is not straightforward to define a rigid taxonomy for deep learning methods.

Deep neural-network-based methods encompass the notion of a traditional neural network, which is characterized by defining affine transformations on a set of random variables and applying a non-linearity afterward (Bengio, 2009). This means that assuming the input as the first layer h^0, the output of layer h^k is defined by the following formula (Bengio, 2009).

$$h^k = \sigma(b^k + W^k h^{k-1}) \qquad (1)$$

in which σ is a non-linear function, b^k is the bias (offset) for layer k and W^k is the weight matrix between layers k and k-1. Very often, the non-linear function σ is *sigmoid*, *tanh* or *rectifier* function. Although there are other activation functions, these three types are the most common ones and are defined in Equations (2)–(4).

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \qquad (2)$$

$$sig(x) = \frac{1}{1 + e^{-x}} \qquad (3)$$

$$rec(x) = Max(0,x) \simeq \ln(1+e^x) \quad (4)$$

We focus on the neural-network-based architectures that are common in text mining and NLP (Socher et al., 2013; Johnson and Zhang, 2015a; İrsoy and Cardie, 2014; Mikolov et al., 2010; Kim, 2014). These models include Recursive Neural Networks, Recurrent Neural Networks, Long Short-Term Memory (LSTM), and Convolutional Neural Networks. Here we describe the gist of each of these algorithms and we will delve into more details about Convolutional Neural Networks in Section Deep neural-network -based methods for text, since it is the approach that is used in this work.

Recursive Neural Networks have unique characteristics that make them suitable for structured data such as natural language processing tasks that often can be represented by parse trees (Socher et al., 2011). In this architecture, the computation units (i.e., neurons) are usually arranged in a tree structure. The learning phase leverages a variant of the backpropagation approach called “backpropagation through structure”. Socher et al. (2013) have proposed a variant of this model for accomplishing sentiment analysis and improved the state of the art by 5.4% in positive/negative sentiment classification.

Recurrent Neural Networks are designed for processing variable-sized sequential input data in which the observed input affects the probability of observing subsequent inputs. Unlike the bag-of-words model that considers each word independently, Recurrent Neural Networks can deal with this type of data efficiently (Mikolov et al., 2010). That is, the generated output at time *t* depends on the output that has been generated by the network in time stamp *t-1*. This behavior mimics the notion of temporal memory in these systems. This is an important property that makes these models significantly more efficient in processing sentences as inputs. As a good example, sequential data analysis can be applied to opinion mining from textual sentences. İrsoy and Cardie (2014) show that applying recurrent neural network outperforms the state-of-the-art method (a variant of Conditional Random Field). The interesting point in their work is that the performance has been achieved without using standard hand-curated sentiment lexicon and syntactical analysis required in previous successful approaches (İrsoy and Cardie, 2014). The learning algorithm for Recurrent Neural Network is known as *Backpropagation Through Time* and is basically a variant of traditional backpropagation that also takes the partial derivatives into account in each of the previous time stamps. Considering so many time stamps requires calculating a large number of gradients. LSTM (Hochreiter and Schmidhuber, 1997) is a special type of recurrent neural network used to alleviate the problem of vanishing gradients. For a detailed description about the above-mentioned deep models refer to (Goodfellow et al., 2016).

Another important notion of using deep learning in natural language is known as *word representation* or *word embedding*. It is different from traditional bag-of-words document analysis in the sense that the feature vectors are not simply comprised of the frequency of each word that exists in the document. Instead, each single word has its own vector representation and the final feature set of a document is obtained by concatenation of these vector representations (also known as embeddings) to form a feature matrix. The first word embedding was introduced by Rumelhart et al. (1986) and it was a technique that represented features (words) based on backpropagation errors. Recently, many word embedding techniques (also known as language models) have been proposed in the literature. Among these methods, *skip-gram* method, proposed by Mikolov et al. (2013), turned out to be significantly efficient compared to the others. The implementation of the method is called word2vec and has been excessively used by researchers recently. A similar variant has also been one provided for obtaining the representation of sentences and paragraphs in (Le and Mikolov, 2014). Later, Pennington et al. (2014) introduced Global Vectors (GloVe) that outperformed word2vec representation in some applications.

We dedicate the rest of this section to CNNs. We describe the usage of this model in detail in Section CNN for OPI when we see it in practice applied to the OPI domain.

Convolutional Neural Networks for text classification

Being inspired by biological visual system, CNNs have been successfully used in various image processing tasks (LeCun et al., 2010). They have been recently applied to the text mining and natural language domain and revealed promising results that could push forward the performance of the state of the art on several datasets. Zhang and Wallace (2015) have authored a holistic guide on designing CNNs for sentence classification. They depict a general CNN for a binary sentence classification task with just one convolution and one pooling layer. Since this architecture is almost common in all CNN applications for text classification, we describe it here to help the reader obtain a basic understanding.

The input is the concatenation of word embeddings in the form of a matrix. Then the whole input is divided into an arbitrary number of regions. Each region can have its own filters. One can think of filters as a linear transformation of the features. Then each filter is applied on the input sentence matrix to form a variable-sized vector. This is called convolution in neural network literature because literally applying the filter removes some of the connections from the network which leads to having a partially connected network in the corresponding layer.

The inherent problem with the output of convolution layer is that the output is variable-sized. The standard solution to solve this issue is having a *pooling* layer right after the convolution layer. This layer simply aggregates each output vector as a singular value by simply getting the maximum or average among the elements of each feature vector. However, this is not the only reason for having the pooling layer. Pooling also acts as a sub sampling procedure, which enables the CNN to capture the aspects and characteristics of the data that are more abstract. As a result, each layer captures features that are more abstract than the previous layer. Finally, the output layer is usually a fully-connected layer with softmax activation function or it can be another multi–layer network which operates on fix-sized feature vectors in order to produce the final classification results (in this case positive/negative).

Using CNNs, Collobert et al. (2011) have proposed a unified general-purpose language framework that is capable of performing a variety of NLP tasks including part-of-speech tagging, chunking, named entity recognition and semantic role labeling.

This common architecture utilizes a specific word embedding approach in which the first layer encodes words by using their index in a general dictionary. Then, it is trained by using backpropagation to obtain the feature vectors. Afterward, the output of this layer is fed to the convolution layer. The architecture has two modes of operation: *window approach* and *sentence approach*. In the former approach, the number of words that are being fed to the first layer are fixed by the user, while in the latter, the number of words are variable depending on the size of each sentence.

Another successful application of CNNs to the test classification has been introduced by Kim (Kim, 2014). He used the word2vec word embedding in order to obtain the feature vectors and then applied the original architecture of CNN in order to carry out seven NLP tasks mainly related to sentence classification and sentiment analysis. Surprisingly, this model has pushed the state of the art in four of the tasks just by having one convolution layer, one pooling layer and a softmax classifier. Fig. 1 depicts this architecture (Kim, 2014).

A more complicated variant of CNNs called Dynamic CNNs has been introduced for sentence classification by Kalchbrenner et al. (2014). The model enhances the pooling mechanism and is able to form feature graphs for each sentence to capture the semantic relations.

Johnson and Zhang (2015a) proposed a novel supervised approach for using CNNs without any pre-trained embeddings upfront. They have also introduced a semi-supervised setting (Johnson and Zhang, 2015b). We describe this method in greater detail when we utilize it in our proposed CNN architecture in Section Proposed CNN architecture.

Deep learning tools and frameworks

A number of deep learning tools have been developed, each with their own strengths and weaknesses. Choosing the appropriate tool for the task is often critical for improving model accuracy. Table 1 summarizes and compares some of the characteristics of deep learning framework/tools as well as their pros and cons. Since we aimed to specifically use Convolutional Neural Networks on relatively large documents, we chose ConText2.0 as an out of the box library to help us try the different CNN architectures.

CNN for OPI

As already mentioned, CNNs have led to state-of-the-art results in image processing tasks and recently they have gained attention in the field of text mining (Bengio, 2009; Schmidhuber, 2015). This motivated us to adapt a special CNN architecture to this application domain, which led to higher classification accuracy compared to common methods.

Applying CNNs

Convolutional Neural Networks can be used as a binary classifier in order to accomplish the above-mentioned task. In general, the input vector $\mathbf{x}$ is segmented into m region vectors $r_0(\mathbf{x})$, $r_1(\mathbf{x})$, …, $r_m(\mathbf{x})$. There is at least one convolution layer followed by a pooling layer in a CNN. The computation units in the convolution layer are not fully connected to the input elements (unlike in original Neural Networks). This happens due to the fact that the

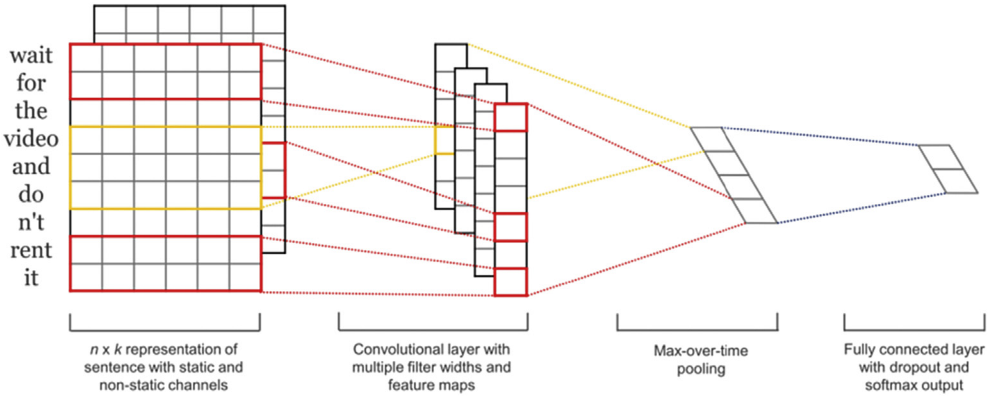

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Fig. 1. CNN-based sentence classification model using word2vec embedding and max-pooling (Kim, 2014).

Table 1
Comparison of deep learning frameworks.

Framework	Language	Algorithm coverage	Developer	Characteristics
Caffe[1]	C++/Cuda	CNNs	Berkeley Vision and Learning Center	• Highly efficient for image processing with ConvNets • Specific to image and machine vision
Theano[2]/ PyLearn2	Python	• Restricted Boltzmann Machines • Stacked Denoising Autoencoders • CNNs	LISA lab at the University of Montreal	• General-purpose • requires symbolic math expressions
Torch[3]	Lua Script	• Restricted Boltzmann Machines • Stacked Denoising Autoencoders • CNNs	Facebook and Google	• Matlab-like script and Intuitive in usage • General-purpose • Learning curve for Lua language
ConText[4]	C++/Cuda	• Supervised CNN • Semi-supervised CNN	(Johnson and Zhang, 2015a)	• High performance • Specific to document classification • Easy-to-use bash script support • Runs on NVIDIA GPU
DL4J[5]	Java and Scala	• Restricted Boltzman Machines • CNNs • Recursive Nets • Recurrent Nets • Deep-belief Nets • Stacked Denoising Autoencoders • Deep Autoencoders	Sky Mind Company	• Faster than python • General-purpose • Transparent parallelism • Work with Hadoop and Spark • Slower development speed compared to scripting languages
CNTK[6]	C++/Cuda	• CNNs • Recurrent Nets • Long Short Term Memory Networks (LSTMs)	Microsoft	• Graphical User Interface • Runs on multiple GPUs on multiple machines
TensorFlow	C++/python	• Multi-purpose	Google	• Multi-platform (CPU, GPU and Mobile Device) • General purpose

[1] http://caffe.berkeleyvision.org/.
[2] http://deeplearning.net/software/theano/.
[3] http://torch.ch/.
[4] http://riejohnson.com/cnn_download.html.
[5] http://deeplearning4j.org/.
[6] https://github.com/Microsoft/CNTK.

convolution layer operates on different regions of the input. The pooling layer is a sub-sampling layer that provides a higher-level abstraction of feature in each convolution layer. The most common pooling methods are max-pooling and average pooling. According to (Zhang and Wallace, 2015) max-pooling usually outperforms average-pooling for text classification.

The learning algorithm uses backpropagation in order to calculate the gradients and tries to minimize the loss function, which is usually square, logistic, or cross entropy loss. The square loss is one of the most commonly used loss functions and is defined as:

$$L(y,f(\mathrm{x})) = C(y - f(\mathrm{x}))^2 \tag{5}$$

in which x is the input vector, y denotes the actual class label assigned to the input vector, and *f*(x*)* is the classifier output, and C is a constant normally set to 0.5 or 1.

Holding the same notation, the logistic loss is defined as:

$$L(y,f(\mathrm{x})) = \log\left(1 + e^{-yf(\mathrm{x})}\right) \tag{6}$$

Finally, the Cross entropy loss is defined as:

$$L(y,f(\mathrm{x})) = -y\ \ln(f(\mathrm{x})) - (1-y)\ln(1 - f(\mathrm{x})) \tag{7}$$

We discuss the suitable architecture of CNN, which can be used in the OPI domain successfully.

Proposed CNN architecture

In choosing the appropriate architecture for identifying the predatory conversations, the designer deals with three major decisions that we address here.

- *Recurrent Neural Networks vs. CNNs*

As already mentioned in Section Background, Recurrent Neural Networks can be trained efficiently on a short window of words or on short sentences. However, when the input sequence consists of multiple sentences or even multiple paragraphs (as in OPI domain), the training of Recurrent Neural Networks becomes intractable. Both RNNs and CNNs can be utilized for the identification of online predators depending on the ultimate goal of the analysis. However, in our use case (identification of predatory conversations), we deal with relatively long documents as our input sequence (tens of sentences in average). As a result, Recurrent Neural Networks cannot be used due to aforementioned problem, while CNNs do not suffer from this issue.

- *Pre-trained word embeddings vs. internal word embeddings*

Pre-trained word embeddings, obtained by algorithms such as word2vec and GloVe, are trained on general web documents. That can explain why using word2vec leads to good results in general domains, such as sentiment analysis and topic classification. However, in a specific domain like OPI, using the existing pre-trained word vectors, learned from Google News dataset or other general web documents, does not lead to high classification accuracy due to the existence of a large number of out-of-vocabulary words. As an example, consider the following short messages in a chat conversation from PAN-2012 dataset: "*r u der?*" or "*I got ur msg thoe* …". The pre-trained word vectors based on news or other general web documents probably lack words such as "der" and "ur". However, we expect that relatively higher classification performance can be achieved by applying Johnson's approach (Johnson and Zhang, 2015a) in which CNN directly uses the input data and learns the word representations internally. We assess this hypothesis by applying both approaches on PAN-2012 data set in Section Experiments and we compare a CNN architecture that uses pre-trained word vectors to the one that learns the representations internally. Based on the results of this experiment, in the presented architecture, we feed the feature vectors directly to the convolution layer to learn them internally (see Fig. 2).

- *Bag-of-words feature encoding vs. one-hot feature encoding*

If the designer decides not to use word embeddings, as in our case, s/he should consider another way of feature representation to feed the input text into the convolution layer. Basically, there are two main approaches:

(1) *Bag-of-words variants*: Assuming that the number of features (words) in the corpus is denoted by *n*, a simple way is the binary representation of each region in which the presence or absence of a feature is represented by *1* and *0* respectively. As an example, Let ***D*** be a short document containing the sequence "*r u there?*". Then we can define 3 overlapping regions of size 2 in a way that $\boldsymbol{R_1} = [10 \ldots 0100]^T$ represents 'r u', $\boldsymbol{R_2} = [00 \ldots 0110]^T$ represents 'u there', and finally $\boldsymbol{R_3} = [0 \ldots 0011]^T$ represents 'there ?'. As observed, in each region vector there are two *1*s and all the remaining n-2 features are 0. Of course, one can also use the normalized frequency of words instead of only 0 and 1. One of the main drawbacks of this approach is the fact that it does not count the order of words at all.

(2) *Sequential concatenation of one-hot vectors:* This encoding method was introduced by Johnson and Zhang (2015a) and Johnson and Zhang (2015b) in order to take the words order into account. In this approach, the region sequences are concatenated to form the feature vector representing a document. Considering the document ***D*** again, the three overlapping regions '*r u*', '*u there*' and '*there ?*' would be represented by $\boldsymbol{R_1} = [10 \ldots 0|010 \ldots 0]^T$,

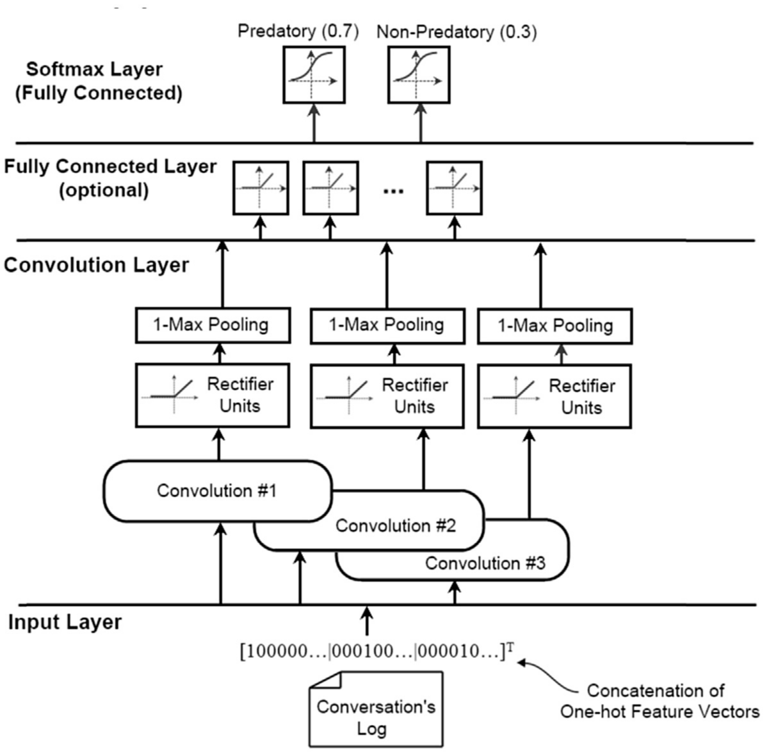


Fig. 2. The proposed CNN Architecture used for OPI.

$\boldsymbol{R_2} = [010 \ldots 0|0 \ldots 10]^T$, and $\boldsymbol{R_3} = [0 \ldots 10|0 \ldots 01]^T$ respectively. As can be seen, each vector has two parts separated by a pipe (for the sake of visualization). Each part corresponds to one token in the region and contains only a single *1* for that token in the whole vocabulary. For example, the first part of $\boldsymbol{R_1}$ represents token ‘r’. The drawback of this approach is that it makes the feature space extremely sparse. But according to Johnson and Zhang's report (Johnson and Zhang, 2015a), if the implementation leverages the sparse matrix vector calculations as in (Johnson, 2016), it can lead to significant classification results in many cases. After trying both approaches, we observed that the best results were obtained by utilizing the second approach in our case. Fig. 2 illustrates the proposed CNN architecture described above.

After deciding about the topology of the CNN, we can focus on setting the hyper parameters. We describe the hyper parameters' setting and the resultant outcome in the next section.

Experiments

Environmental settings

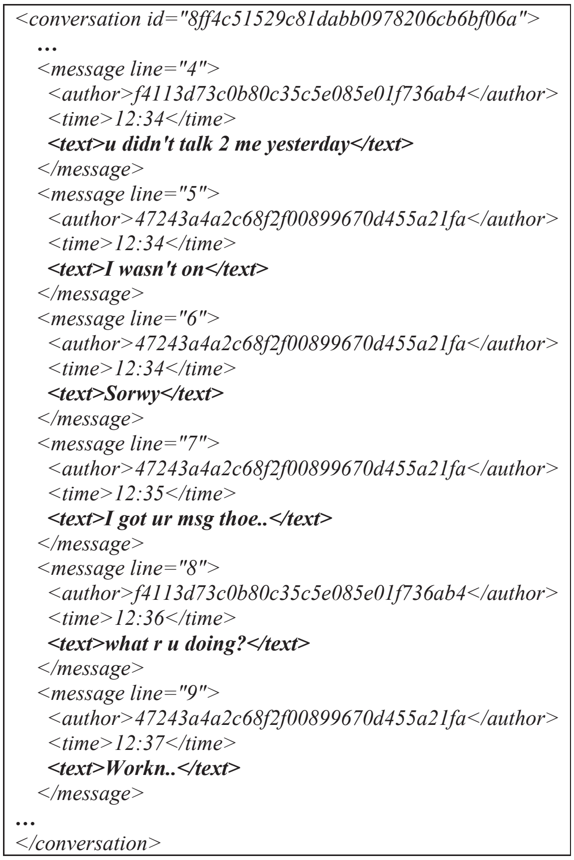
Given that the parallelism is implemented at the hardware level in Graphical Processing Units, they take advantage of the parallel nature of neural-network-based learning models by accelerating execution. Hence, we utilized Calcul Québec, a high-performance computing cluster in Canada (Calcul Quebec, 2016) which has several NVIDIA's Tesla K80 GPUs. We ran the processes on one K80 GPU with 24 GB of memory and 2496 processor cores. In fact, implementing the architecture discussed in Section Proposed CNN architecture cannot be done on a CPU.

Dataset

We conducted the experiments on a public data set in PAN-2012 competition (Inches and Crestani, 2012). The predatory instances in this dataset have been gathered by a non-profit organization called *Perverted Justice* (Perverted Justice, 2016). These conversations occurred between experts who posed as juveniles and convicted prolific online predators. The dataset has been used extensively in the literature (Kontostathis et al., 2010; Mcghee et al., 2011; Pendar, 2007). The extended dataset was used in PAN-2012 international competition as a reference dataset for the task of identifying either predatory messages or predators. Table 2 summarizes the characteristics of this dataset.

Table 2
Characteristics of the dataset.

PAN2012 Dataset	Training set	Sample size	66,927
		No. of predatory conversations	2016
		No. of unique users	97,695
		No. of unique predators	142
	Testing set	Sample size	155,128
		No. of predatory conversations	3737
		No. of unique users	218,716
		No. of unique predators	254



```
<conversation id="8ff4c51529c81dabb0978206cb6bf06a">
  ...
  <message line="4">
    <author>f4113d73c0b80c35c5e085e01f736ab4</author>
    <time>12:34</time>
    <text>u didn't talk 2 me yesterday</text>
  </message>
  <message line="5">
    <author>47243a4a2c68f2f00899670d455a21fa</author>
    <time>12:34</time>
    <text>I wasn't on</text>
  </message>
  <message line="6">
    <author>47243a4a2c68f2f00899670d455a21fa</author>
    <time>12:34</time>
    <text>Sorwy</text>
  </message>
  <message line="7">
    <author>47243a4a2c68f2f00899670d455a21fa</author>
    <time>12:35</time>
    <text>I got ur msg thoe..</text>
  </message>
  <message line="8">
    <author>f4113d73c0b80c35c5e085e01f736ab4</author>
    <time>12:36</time>
    <text>what r u doing?</text>
  </message>
  <message line="9">
    <author>47243a4a2c68f2f00899670d455a21fa</author>
    <time>12:37</time>
    <text>Workn..</text>
  </message>
...
</conversation>
```

Fig. 3. A sample snippet of a conversation.

The data set is formatted in a giant XML file for training set and another one for testing set. Since we are doing the analysis at the conversation level, the xml files need to be parsed in order to extract conversations. Each conversation contains the messages of each participant as shown in Fig. 3.

Originally, the dataset labels the users as predators or non-predators. Hence, in order to identify the predatory conversations, we had to re-label the data in a way that if at least one predator participates in a conversation, the conversation will be labeled as predatory. Since almost all of the predatory conversations have taken place between only two participants this is a reasonable way of labeling the data.

We used the open source sparse implementation of CNN provided by Rie Johnson available at riejohnson.com/cnn_download.html for our experiments.

Experiments' settings

Setting the optimal hyper parameters of a CNN is a challenging task that requires more research. Most researchers find sub-optimal choices of these parameters by

Table 3
Designated groups of experiments as well as their experiments' short name and description.

Group no.	Learning scheme	Description
1	SVM	Support Vector Machine implemented in (Chang and Lin, 2011)
	NN	Multi layer feed forward Neural Network with bag-of-words features
2	Pre-trained W2V-CNN	CNN with pre-trained word vectors obtained from Google News dataset available at (word2vec, 2016)
	W2V-CNN	CNN with word2vec word representation method (Mikolov et al., 2013)
	GloVe-CNN	CNN with Global Vector word representation method (Pennington et al., 2014)
3	Bow-CNN	CNN with bag-of-words feature representation described in (Johnson and Zhang, 2015a)
	One-hot CNN	CNN with concatenation of one-hot vectors described in (Johnson and Zhang, 2015a)

trying different combinations in their corresponding domain. We classify these parameters separately and describe our choice for each as follows.

- ***Regularization parameter and dropout rate:*** These two important parameters are mainly used to prevent overfitting. L2 regularization (Ng, 2004) is a common type of regularization used in neural network. It is worth mentioning that the regularizations in CNN are usually done in the top layer. Another important mechanism to prevent overfitting is the dropout that randomly deactivates a certain number of output units in the top-layer in the training process. We set dropout rate and the coefficient of L2 regularization to be 0.5 and 10^{-4}, respectively, that are the default settings in ConText2 and we found it efficient.
- ***Loss function:*** We used the square loss function since it outperformed other loss functions including logistic loss.
- ***Activation function:*** The rectifier function was used as the activation function of convolution layer since it led to better results compared to *tanh* or *sigmoid* function.

As a side note, we did not do any preprocessing other than converting the upper case letters into lower case. The reason is the informal and colloquial nature of the chat conversations. In fact, performing the simplest preprocessing step such as stop-word removal may damage the meaning of the message. To make it more concrete, this can be seen in the message "***i thought u wanted 2 come c me***". While a blind noise removal procedure may omit tokens such as ***c*** in this sentence, we know that it bears important meaning.

We conducted two series of experiments on the above-mentioned dataset. The former set of experiments aims to compare the classification performance of the proposed architecture to other common binary classification methods (see Section Investigating the effect of convolution) and the latter investigates the influence of adding extra convolution layer to a CNN architecture and compare it to the effect of adding an extra hidden layer in NN (see Section Adding extra convolution layers).

In order to compare the classification performance, we designated three groups of experiments as follows.

(1) *Non-deep learning methods*: This group encompasses two common binary classification method Support Vector Machine (SVM) and feed forward Neural Network (NN).

(2) *CNN models using word vector representations:* This group includes a CNN model that use pre-trained word2vec word vectors comprised of 3 million words. Each word is represented by a 300-dimensional vector

Table 4
Performance comparison for depth-1 CNNs with baselines (Support Vector Machines (SVM) and neural network (NN)). All the neural network models have the same number of neurons (i.e., 2000).

Learning scheme	Exp. No.	Settings	Precision (%)	Recall (%)	F$_1$-score (%)
SVM	1	*linear kernel*	78.13	50.06	61.02
NN	2	*nodes: 2000, encoding: frequency of unigram, vocab. size = 5000*	91.71	70.71	79.85
	3	*nodes: 2000, encoding: frequency of bigram, vocab. size = 7000*	92.14	68.54	78.60
Pre-trained W2V-CNN	4	*nodes:2000, skip-gram model with negative sampling, pooling type: max, word vector dimension: 3M × 100*	86.41	70.94	77.91
W2V-CNN	5	*skip-gram model with hierarchical softmax, nodes:2000, pooling type: max, word vector dimension:36,314 × 100*	89.24	70.71	78.90
	6	*skip-gram model with negative sampling, nodes:2000, pooling type: max, word vector dimension:36,314 × 100*	89.46	73.30	80.58
GloVe-CNN	7	*nodes:2000, pooling type: max, word vector dimension:36,350 × 100*	91.02	72.22	80.54
BOW-CNN	8	*nodes:2000, region size:(8 and 15), encoding: binary-encoded unigram, vocab. size = 5000, pooling type: max*	92.52	70.94	80.31
One-hot CNN	9	*nodes:2000, region size:(1,2 and 3), encoding: concatenation of one-hot vectors, vocab. size = 5000, pooling type: max*	91.57	72.41	**80.87**

The underlined numbers represent the best performance in each experiment group. The bold number is the maximum performance obtained through the whole experiments.

trained on part of Google News dataset and is available at (word2vec, 2016). Also this group includes CNN models that use word2vec and GloVe word vectors which were obtained by running the algorithm on the training corpus of PAN-2012.

(3) *CNN models that learn the word representations implicitly*: This group comprises models that do not use word vectors as their input and learn the representation directly from the training input (see Section Proposed CNN architecture). CNN models that use bag-of-words and one-hot encoding and were described in Section Proposed CNN architecture, belong to this group.

Table 3 summarizes these groups of experiments as well as the short name and the description of the architecture used in the experiment.

The results are described in the following sub-section.

Investigating the effect of convolution

In order to investigate the effect of convolution, we conducted a set of experiments in which we compare the performance of a traditional neural network with one hidden layer, to that of a CNN with one convolution layer. We used a fixed number of computation units (i.e., 2000) with rectifier activation functions in the hidden layer and convolution layer of NN and CNN respectively. In addition, all CNN models were given 100 epochs to run so that they have equal time to converge. The step size parameter of gradient descent algorithm was changed within {0.01, 0.05, 0.1, 0.15, 0.2, 0.25} for each experiment and the value with the best classification performance was chosen.

Table 4 shows the results of experiments. Experiments #2 and #3 use feed forward neural network with unigram and bigram features. Experiment #4 uses the pre-trained word vectors trained on a part of Google News documents by Google's researchers (word2vec, 2016). Experiments #5 and #6 use CNNs with word2vec word representation learned from the training corpus at hand using word2vec tool (word2vec, 2016). Similarly, experiment #7 shows the result for CNN with GloVe word representation learned from the training corpus. The GloVe's source code is available at (Pennington et al., 2016). Finally, CNNs in experiments #8 and #9 learn the word representation internally without using any pre-trained word vectors.

We tried out different parameter settings for each of the experiments and selected the best combination based on the classification performance. The top F_1-score in each group is underlined. In Table 4, '*Vocab. size*' is the maximum size of the vocabulary extracted from the corpus, pooling type refers to either max pooling (Max) or average pooling (Avg) described in Section Convolutional neural networks for text classification. Additionally, '*frequency of uni/bigram*' were explained as *bag-of-words feature encoding* in Section Proposed CNN architecture. Similarly, *concatenation of one-hot vectors* refers to *one-hot feature encoding* described in Section Proposed CNN architecture.

As seen in Table 4, one-hot CNN has the best classification performance among all methods and W2V-CNN and GloVe-CNN have the next best performances. As we expected, the pre-trained word2vec model (i.e., experiment #4) has a poor classification performance compared to other methods. This supports the argument mentioned in Section Proposed CNN architecture, about the inefficiency of general pre-trained word vectors in specific application domains.

An interesting point to note is about the relatively significant difference between precision and recall in the result of all experiments, which technically means that the number of false negatives is more than that of false positives. This implies that all models have more difficulty in identifying predatory conversations that apparently seem to be safe. These predatory conversations might start with normal greetings. They may continue with implicitly sexual messages that have been carefully designed by the predator. Finally, they may end with an attempt to set an appointment with the minor. This type of sophistication requires the analysis of psycholinguistic features, in addition to lexical features, that is beyond the scope of this work.

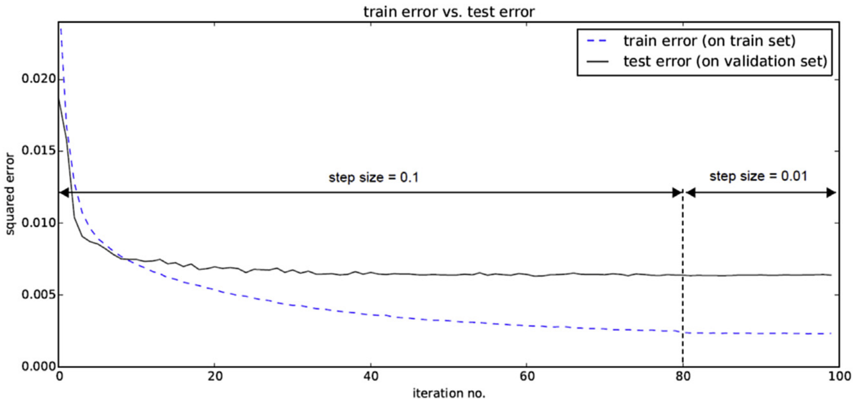


Fig. 4. Train and test errors for 100 iterations of CNN with the proposed architecture (experiment No. 9). The test error starts to converge after 80 iterations.

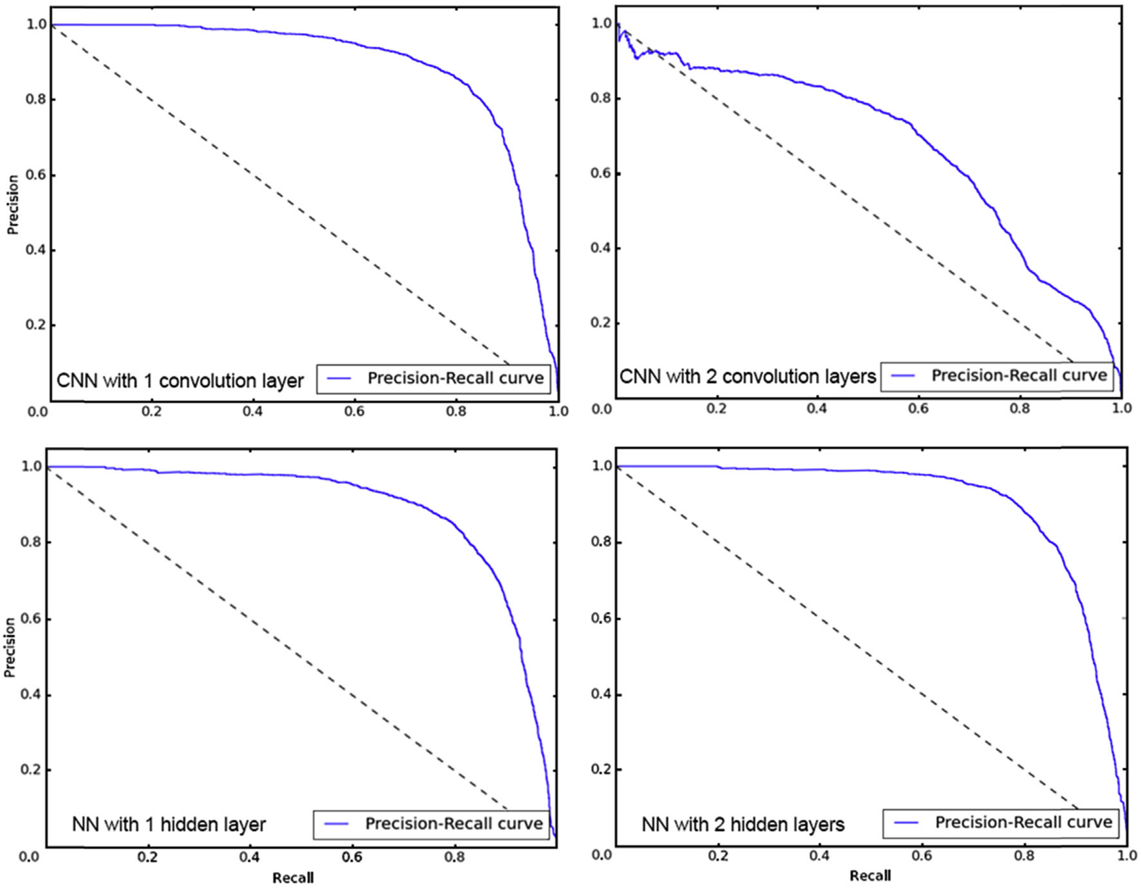


Fig. 5. Precision-Recall curves for showing the effect of extra convolution/hidden layers on CNN and regular NN. The performance of CNN with two convolution layers has decreased, while that of the normal NN has increased.

Fig. 4 shows the changes of training and testing errors for the outperforming approach (experiment No. 9).

As seen in Fig. 4, after iteration 80, the test error begins to shrink and almost converges to a constant value for the remaining 20 iterations. A common practice to obtain a smooth test error is to reduce the step size parameter of gradient descent algorithm after a specific number of iterations.

In stochastic gradient descent, the order of processing the data in each iteration of training affects the result of training (LeCun et al., 2012). As is typically done in neural network training, our training was done by mini-batch stochastic gradient descent (SGD), which randomly selects mini batches (e.g., of size 100) of data points and averages gradients over each mini batch. Therefore, changing the order of input data points merely has the effect of changing the seed of randomization for SGD and does not change the performance significantly, not more than natural fluctuations. We conducted experiments by changing the order of input data points (by setting different random seeds and shuffling the initial order) and confirmed that one-hot CNN still outperformed the others. The convergence of the algorithm can be observed in Fig. 4.

Adding extra convolution layers

To investigate the effect of adding extra convolution layers to a CNN architecture and also comparing the same effect on traditional NN, we conducted another set of comparative experiments. In these experiments, we measure the performance change in two CNN architectures (one with a single convolution layer and the other one with two convolution layers), and then, we compare this change to that of two original NNs (one with one hidden layer and the other one with two hidden layers). Fig. 5 compares the Precision-Recall curves for traditional NN and CNN with one and two hidden/convolution layers.

As seen in Fig. 5, the performance of CNN with two convolution layers is lower than that of a CNN with a single convolution layer. On the contrary, the performance of the traditional NN increases with adding an extra hidden layer.

This raises the question that whether having a deep CNN would perform better than a CNN with only one single convolution layer in text classification. Our experiments show that having only one convolution layer in CNN would be more efficient. This might happen due to the occurrence of overfitting. That is, when more than one convolution layer is used in the CNN architecture the model tends to overfit. We draw some conclusions in this regard in the next section.

Conclusion and discussion

We showed that using one layer of convolution has a positive effect on classification accuracy. Even though one might benefit from having several convolution layers (i.e., a deeper structure) in image processing tasks, according to our experiments, it might not be the case in natural language processing tasks. The reason is that as the number of layers in the hierarchy increases the CNN training algorithm tends to overfit to the extent that cannot be cured easily by regularization or dropout mechanism. In this case, in spite of the fact that we tested many combinations of architectures with two or more convolution layers and even allow them to run for more iterations, the best performance was obtained by having only one single convolution layer in the architecture. Another important point to consider is the fact that the massive parallelism of the GPU allowed us to utilize relatively large number of neurons (several thousands) in both NN and CNN experiments. Whereas the traditional neural networks that run on the CPU can technically utilize a much lower number of neurons and have poorer performance (close to that of SVM) than what we obtained for NN experiments in this research.

It is also worth to mention some of the best practices that we learned through our experiments. Although they are not claimed as being always true, they might be of help for other researchers in this specific field.

- Unlike traditional approaches, we did not perform any preprocessing procedure other than changing the uppercase letters to lower case. We experienced that doing procedures such as stop-word removal, and removing numbers or symbols (e.g., "*?*" and "*!*"), decreases the classification performance.
- It is important to note that keeping the punctuations should be done with caution. While capturing the predators writing style might be beneficial when dealing with roughly the same group of predators in the same police jurisdiction for a certain period of time, it might make the same model have a lower accuracy on completely different group of predators in another jurisdiction. Hence, if the designer aims to build a general model, which is applicable to different jurisdictions, it might not be a good idea to capture the writing style of the authors by keeping punctuations.
- Based on our results, general pre-trained word vectors are not suitable for being used in specific domains such as OPI.
- Rectifier activation function outperformed other activation functions including (*tanh* and *sigmoid*)
- Having only one convolution layer led to better results compared to deeper structures with more than one convolution layer.
- Using the concatenation of one-hot vectors led to better results than simply using binary representation of unigrams.
- Another interesting observation was that decreasing the step size after a certain number of iterations is usually helpful for the training convergence.

Future work

As discussed in Section Background, Recurrent Neural Networks and LSTMs have been shown to be efficient in text analysis and other NLP tasks such as speech recognition. The main challenge about their applicability is the difficulty of the training process that makes the training less practical in dealing with large documents. We anticipate that new approaches will emerge to tackle this challenge and these models are good candidates to be used in the field of OPI in the future and we plan to apply them to this domain.

Johnson and Zhang (2015b) have shown that a semi-supervised extension of one-hot CNN outperforms supervised one-hot CNN (which we experimented with). For further performance improvements, it is possible to apply the semi-supervised one-hot CNN to our task by using the training data as unlabeled data ignoring labels, analogous to training word2vec on the training data. The strength of the semi-supervised one-hot CNN derives from the fact that from unlabeled data it learns embeddings of word sequences or text regions such as "r u der", which can convey higher-level concepts than single words in isolation such as "r" (which are more primitive). However, for simplicity, we did not test it in this work.

Acknowledgment

This research has been supported by the Natural Sciences and Engineering Research Council of Canada (N00009). We would also like to thank sergeants D. Karneyeff-Bisson and M. Grenier in Sûreté du Quebec for having illuminating discussions on this topic. In addition, we give our special thanks to Dr. Rie Johnson, the main developer of ConText, for having several valuable technical discussions.

Appendix 1. Scripts of the model's configurations and parameters for experiments in Table 4.

The following table contains the Linux shell scripts of the models to reproduce the experiments of Table 4. Token files should be already generated as inputs of these scripts. They can be generated by using the sample scripts in the source code of ConText 2.0.

Experiment #6: CNN with word2vec vectors trained on the corpus at hand

```
##### Using given word vectors as input to CNN.
##### Word vectors are in word2vec-format binary file.
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/software-gpu/cuda/7.5.18/lib64
gpu=-1         # Change this to, e.g., "gpu=0" to use a specific GPU.
mem=12       # pre-allocate 12GB device memory
gpumem=${gpu}:${mem}

prep=../bin/prepText
cnet=../bin/conText

#-----  Pre-trained word vectors  -----
options=LowerCase
dim=100                                                # dimensionality of word vectors.
wv_bin_fn=data/PANTrainVectors-dim100.wvbin      # path to the binary word vector file.
#-----------------------------------------------

#---  write vocabulary (word mapping) of word vectors to a file.
wm_fn=data/PAN-wordvec.wmap.txt
$cnet $gpu write_word_vectors_mapping wordvec_bin_fn=$wv_bin_fn word_map_fn=$wm_fn

#---  Extract training data vocabulary.
trn_voc=data/PAN-trn.vocab
$prep gen_vocab input_fn=data/PAN-rand-train.tok vocab_fn=$trn_voc $options \
   WriteCount max_vocab_size=5000

#---  Merge the word vector vocabulary and training data vocabulary.
xvoc=data/PAN-wv-trn.vocab
$prep merge_vocab input_fns=${wm_fn}+${trn_voc} vocab_fn=$xvoc

#---  Convert text to one-hot vectors and generate target files.
for set in train test; do
  #---  region vectors of size 1 = one-hot vectors
  rnm=data/PAN-${set}-p1
  $prep gen_regions AllowZeroRegion region_fn_stem=$rnm \
   input_fn=data/PAN-rand-${set} vocab_fn=$xvoc \
   $options text_fn_ext=.tok label_fn_ext=.cat label_dic_fn=data/PAN-cat.dic \
   patch_size=1 patch_stride=1 padding=0
done

#---  Convert word vector files to the format that cnn trainer can read.
w_fn=data/PAN-wv-trn.pmat
$cnet $gpu adapt_word_vectors wordvec_bin_fn=$wv_bin_fn word_map_fn=$xvoc \
   weight_fn=$w_fn rand_param=0.01 random_seed=1

#---  Training.
#---    Layer-0: word embedding layer
#---    Layer-1: convolution layer of region size 3, stride 1, padding 2  followed by max-pooling with one
        pooling region.

opt=0NoIntercept  # Fine-tune word vectors without intercepts (standard).
wcoeff=1        # Use this to scale word vectors.
```

```
neurons=2000
$cnet $gpumem train layers=2 $opt \
   0activ_type=None 0nodes=$dim 0weight_fn=$w_fn 0weight_coeff=$wcoeff \
   1patch_size=3 1patch_stride=1 1padding=2 \
   1activ_type=Rect 1nodes=$neurons 1pooling_type=Max 1num_pooling=1 \
   datatype=sparse x_ext=.xsmatvar y_ext=.y data_dir=data \
   trnname=PAN-train-p1 tstname=PAN-test-p1 \
   reg_L2=1e-4 step_size=0.01 loss=Square mini_batch_size=100 momentum=0.9 random_seed=1 \
   num_iterations=100 step_size_scheduler=Few step_size_decay=0.1 step_size_decay_at=80 \
   test_interval=1 LessVerbose save_fn=output/PAN_wv_model > log_output/PAN_wv.log

../bin/conText -1 cnn_predict model_fn=output/PAN_wv_model.ite100 \
     prediction_fn=output/PAN_wv_nodes2000_prediction.txt \
     WriteText datatype=sparse tstname=PAN-test-p1 data_dir=data x_ext=.xsmatvar
```

Experiment #8: CNN with BOW feature encoding

```
####  Input: token file (one review per line)
####        label file (one label per line)

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/software-gpu/cuda/7.5.18/lib64

gpu=-1          # Change this to, e.g., "gpu=0" to use a specific GPU.
mem=12       # pre-allocate 12GB device memory
gpumem=${gpu}:${mem}

prep_exe=../bin/prepText
cnn_exe=../bin/conText

options="LowerCase UTF8"

#---  Step 1. Generating vocabulary

max_num=5000
vocab_fn=data/NEWPAN_86n4_trn-${max_num}.vocab

$prep_exe gen_vocab input_fn=data/PAN-rand-train.tok vocab_fn=$vocab_fn \
    max_vocab_size=$max_num $options WriteCount

  #---  Step 2. Generating Region files---
  for pch_sz in 8 15; do
    for set in train test; do
      rnm=data/NEWPAN_86n4_${set}-patch${pch_sz}
      $prep_exe gen_regions \
         Bow-convolution VariableStride \
         region_fn_stem=$rnm input_fn=data/PAN-rand-${set} vocab_fn=$vocab_fn \
         $options text_fn_ext=.tok label_fn_ext=.cat \
         label_dic_fn=data/PAN_cat.dic \
         patch_size=$pch_sz patch_stride=1 padding=$((pch_sz-1))
    done
  done
 #---------------------------------------------

  #---  Step 3. Training using GPU
  log_fn=log_output/NEWPAN_86n4.log
  perf_fn=perf/NEWPAN_86n4-perf.csv
  echo
  echo Training CNN and testing ...
  # total number of nodes should be 2000

$cnn_exe $gpumem cnn test_interval=1 random_seed=1 extension=multi conn0=0-top conn1=1-top \
    datatype=sparse_multi data_dir=data x_ext=.xsmatvar y_ext=.y \
    trnname=NEWPAN_86n4_train- tstname=NEWPAN_86n4_test- \
    0dataset_no=0 1dataset_no=1  data_ext0=patch8 data_ext1=patch15 \
    layers=2  0nodes=500 0pooling_type=Max 0num_pooling=1 0activ_type=Rect 0resnorm_width=500 \
    0resnorm_type=Cross 0resnorm_alpha=1 0resnorm_beta=0.5 \
    1nodes=1500 1pooling_type=Max 1num_pooling=1 1activ_type=Rect 1resnorm_width=1500 \
    1resnorm_type=Cross 1resnorm_alpha=1 1resnorm_beta=0.5 \
    loss=Square num_iterations=100 top_reg_L2=1e-4 step_size=0.2 top_dropout=0.5 \
    step_size_scheduler=Few step_size_decay=0.1 step_size_decay_at=80 mini_batch_size=100 \
momentum=0.9 evaluation_fn=$perf_fn save_fn=output/NEWPAN_86n4_model LessVerbose > ${log_fn}

 #---  Step 4. Testing using GPU
 ../bin/conText -1 cnn_predict model_fn=output/NEWPAN_86n4_model.ite100 \
    prediction_fn=output/NEWPAN_86n4_prediction.txt WriteText extension=multi \
    datatype=sparse_multi tstname=NEWPAN_86n4_test- data_ext0=patch8 data_ext1=patch15 \
    data_dir=data x_ext=.xsmatvar > output-test
```

Experiment #9: CNN with concatenation of one-hot vectors

```
#### Input: token file (one review per line)
####        label file (one label per line)

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/software-gpu/cuda/7.5.18/lib64

gpu=-1          # Change this to, e.g., "gpu=0" to use a specific GPU.
mem=12       # pre-allocate 12GB device memory
gpumem=${gpu}:${mem}

prep_exe=../bin/prepText
cnn_exe=../bin/conText

options="LowerCase UTF8"

#---  Step 1. Generate vocabulary
echo Generaing vocabulary from training data ...

max_num=5000
vocab_fn=data/NEWPAN_700n9_trn-${max_num}.vocab
#stopword_fn=data/stopwords
$prep_exe gen_vocab input_fn=data/PAN-rand-train.tok vocab_fn=$vocab_fn \
    max_vocab_size=$max_num $options WriteCount

#---  Step 2. Generating region files---
echo
echo Generating region files with region size 2 and 3 ...

for pch_sz in 1 2 3; do
  for set in train test; do
    rnm=data/NEWPAN_700n9_${set}-patch${pch_sz}
    $prep_exe gen_regions \
      region_fn_stem=$rnm input_fn=data/PAN-rand-${set} vocab_fn=$vocab_fn \
      $options text_fn_ext=.tok label_fn_ext=.cat \
      label_dic_fn=data/PAN_cat.dic \
      patch_size=$pch_sz patch_stride=1 padding=$((pch_sz-1))
  done
done
#------------------------------------------------
#---  Step 3. Training using GPU
log_fn=log_output/NEWPAN_700n9-seq.log
perf_fn=perf/NEWPAN_700n9-seq-perf.csv
echo
```

```
echo Training CNN and testing ...
# total number of nodes should be 2000

$cnn_exe $gpumem cnn extension=multi conn0=0-top conn1=1-top conn2=2-top \
   data_dir=data trnname=NEWPAN_700n9_train- tstname=NEWPAN_700n9_test- \
   reg_L2=0 top_reg_L2=1e-4 step_size=0.1 top_dropout=0.5 \
   0nodes=500 0resnorm_width=500 1nodes=500 1resnorm_width=500 2nodes=1000 \
   2resnorm_width=1000 \
   LessVerbose test_interval=1 evaluation_fn=$perf_fn save_fn=output/NEWPAN_700n9_model \
   loss=Square num_iterations=100 step_size_scheduler=Few \
   step_size_decay=0.1 step_size_decay_at=80 mini_batch_size=100 \
   0dataset_no=0 1dataset_no=1 2dataset_no=2 data_ext0=patch1 data_ext1=patch2 data_ext2=patch3 \
   layers=3 pooling_type=Max num_pooling=1 activ_type=Rect \
   random_seed=1 datatype=sparse_multi x_ext=.xsmatvar y_ext=.y \
   momentum=0.9 \
   resnorm_type=Cross resnorm_alpha=1 resnorm_beta=0.5 > ${log_fn}

#---  Step 4. Testing using GPU
../bin/conText -1 cnn_predict model_fn=output/NEWPAN_700n9_model.ite100 \
   prediction_fn=output/NEWPAN_700n9_prediction.txt WriteText extension=multi \
   datatype=sparse_multi tstname=NEWPAN_700n9_test- data_ext0=patch1 data_ext1=patch2 \
   data_ext2=patch3 data_dir=data x_ext=.xsmatvar > output-test
```

References

Bengio Y. Learning deep architectures for AI. Found Trends Mach Learn 2009;2(1):1–127.

Bogdanova D, Rosso P, Solorio T. Exploring high-level features for detecting cyberpedophilia. Comput Speech Lang 2014;28(1):108–20.

Cano A, Fernandez M, Alani H. Detecting child grooming behaviour patterns on social media. In: Aiello L, McFarland D, editors. Social informatics8851. Springer International Publishing; 2014. p. 412–27.

Chang C-C, Lin C-J. LIBSVM: a library for support vector machines. ACM Trans Intelligent Syst Technol 2011;2(3). 27:1–27:27.

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa PP. Natural language processing (almost) from scratch. JMLR 2011;12:2493–537.

Ebrahimi M, Suen CY, Ormandjieva O, Krzyzak A. Recognizing predatory chat documents using semi-supervised anomaly detection. In: Presented at the document recognition and Retrieval XXIII, San Francisco, CA, USA: electronics and imaging 2016; 2016. p. 1–9.

Eriksson G, Karlgren J. Features for modelling characteristics of conversations. In: Presented at the notebook for PAN at CLEF 2012, Rome, Italy: CLEF 2012; 2012.

Escalante HJ, Villatoro-Tello E, Juárez A, Montes-y-Gómez M, Villaseñor L. Sexual predator detection in chats with chained classifiers. In: Proceedings of the 4th workshop on computational approaches to subjectivity, sentiment and social media analysis. Atlanta, Georgia: Association for Computational Linguistics; 2013. p. 46–54.

Goodfellow I, Yoshua B, Courville A. Deep learning. 2016. Book in preparation for MIT Press.

Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9(8):1735–80.

Inches G, Crestani F. Overview of the international sexual predator identification competition at PAN-2012. 2012 [CLEF (working notes)].

İrsoy O, Cardie C. Opinion mining with deep recurrent neural networks. In: Proceedings of the conference on empirical methods in natural language processing; 2014. p. 720–8 [Doha, Qatar].

Johnson R. ConText v2. Retrieved May 22, 2016, from. 2016, May 15. http://riejohnson.com/cnn_download.html.

Johnson R, Zhang T. Effective use of word order for text categorization with convolutional neural networks. In: 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2015.

Johnson R, Zhang T. Semi-supervised convolutional neural networks for text categorization via region embedding. In: Presented at the NIPS, Montreal; 2015.

Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences. ACL 2014;2014.

Kang I-S, Kim C-K, Kang S-J, Na S-H. IR-based k-nearest neighbor approach for identifying abnormal chat users. In: Presented at the notebook for PAN at CLEF 2012, Rome, Italy: CLEF 2012; 2012.

Kim Y. Convolutional neural networks for sentence classification. In: Proceedings of the conference on empirical methods in natural language processing; 2014 [Doha, Qatar].

Kontostathis A. Toward the tracking and categorization of internet predators. In: Proceeding of text mining workshop 2009 held in conjunction with Ninth Siam International Conference Data Mining; 2009.

Kontostathis A, Edwards L, Leatherman A. Text mining and cybercrime. In: Text mining. John Wiley & Sons, Ltd; 2010. p. 149–64.

Le QV, Mikolov T. Distributed representations of sentences and documents. In: Proceedings of the 31th international conference on machine learning, ICML 2014, Beijing, China, 21–26 June 2014; 2014. p. 1188–96.

LeCun Y, Kavukcuoglu K, Farabet C. Convolutional networks and applications in vision. In: Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on; 2010. p. 253–6.

LeCun Y, Bottou L, Orr G, Müller K-R. Efficient backprop. In: Neural networks: tricks of the trade. Springer Berlin Heidelberg; 2012. p. 9–48.

Mcghee I, Bayzick J, Kontostathis A, Edwards L, Mcbride A, Jakubowski E. Learning to identify internet sexual predation. Int J Electron Commer 2011;15(3):103–22.

Mikolov T, Karafiát M, Burget L, Cernocký J, Khudanpur S. Recurrent neural network based language model. In: INTERSPEECH 2010, 11th annual conference of the international speech communication association, Makuhari, Chiba, Japan, September 26–30, 2010; 2010. p. 1045–8.

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In: Presented at the advances in neural information processing systems 26: 27th annual conference on neural information processing systems 2013; 2013. p. 3111–9.

Morris C. Identifying online sexual predators by SVM classification with lexical and behavioral features (Master of Science Thesis). Canada: University Of Toronto; 2013, January 30. Retrieved from, ftp.cs.toronto.edu/pub/gh/Morris,Colin-MSc-thesis-2013.pdf.

Ng AY. Feature selection, L1 vs. L2 regularization, and rotational invariance. In: Proceedings of the twenty-first international conference on machine learning. New York, NY, USA: ACM; 2004. p. 78.

Pendar N. Toward spotting the pedophile telling victim from predator in text chats. Washington, USA,: IEEE; 2007. p. 235–41.

Pennington J, Socher R, Manning CD. GloVe: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP 2014); 2014. p. 1532–43.

Pennington J, Socher R, Manning CD. GloVe. Retrieved May 20, 2016, from. 2016, May 10. http://nlp.stanford.edu/projects/glove/.

Perverted Justice. Retrieved from, http://www.perverted-justice.com; 2016, May 21.

Calcul Quebec. Retrieved January 20, 2016, from, http://www.calculquebec.ca/en/; 2016, May 24.

Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986;323(6088):533–6.

Schmidhuber J. Deep learning in neural networks: an overview. Neural Netw 2015;61:85–117.

Socher R, Lin CC-Y, Ng AY, Manning CD. Parsing natural scenes and natural language with recursive neural networks. In: Getoor L, Scheffer T, editors. ICML. Omnipress; 2011. p. 129–36.

Socher R, Perelygin A, Wu J, Chuang J, Manning CD, Ng A, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing. Seattle, Washington, USA: Association for Computational Linguistics; 2013. p. 1631–42.

Villatoro-Tello E, Juárez-González A, Escalante HJ, Montes-y-Gómez M, Villaseñor-Pineda L. A two-step approach for effective detection of misbehaving users in chats. In: Presented at the notebook for PAN at CLEF'12. Rome, Italy: CLEF'12; 2012.

word2vec. Retrieved May 20, 2016, from, https://code.google.com/archieve/p/word2vec; 2016, May 10.

Zhang Y, Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. CoRR, abs/1510.03820. Retrieved from. 2015. http://arxiv.org/abs/1510.03820.