Irony Detection in Twitter using different methods

Chen Zhou, Huan Wang, Rui Pan
chenzhou@umass.edu, huanwang@umass.edu, ruip@umass.edu

Abstract

We present a survey of methods for sentiment analysis, specifically irony detection in English Tweets, through this paper. The methods are naïve Bayes classifier, averaged and non-averaged perceptron, and neural network algorithm such as a combination of CNN, LSTM and DNN. We run these algorithms on a labeled dataset provided by SemEval-2018. At the end of the paper, we discuss the issues and problems we found and compare the performance of these methods.

1 Introduction

Sentiment analysis is a significant task in natural language processing and has many useful applications in real world. These years, social media has grown drastically, and it provides a place for creative and figurative language such as irony and humor. The task of irony is not only important from a theoretical point of view. It also has practical applications in text mining, author profiling, detecting online harassment and human-computer interactive system. According one of the definitions of irony, "it is often identified as a trope or figurative language use whose actual meaning differs from what is literally enunciated." Since irony can express both positive and negative statements, irony detection appears to be difficult. Moreover, people often use informal forms of communication, which contains creative spelling, misspellings, new words and new abbreviations. It also increases the difficulty of this task.

We determined Twitter as our target database, since there is a huge number of personal feelings and containing variety of topics and popular network language. We finally chose the dataset provided by SemEval-2018 which includes 3834 labeled tweets.

From what we have learned in this semester, we try to use several famous algorithms (especially CNN and LSTM) to build models and compare the performances. As an important deep learning technique, the convolutional neural network (CNN) has recently been well studied and widely employed for sentimental analysis. Due to its capability of deep representation learning and sentence-level information awareness, we think it will perform pretty good on sarcasm detection problem. Long short-term memory (LSTM)[i] is a recurrent neural network (RNN) architecture. The excellent memory of LSTM can also deal with the long-term dependencies of concepts in the sentence. Therefore, we construct a LSTM architecture to make prediction and compare it with other models.

In this paper, following this inspiration we tackle the ironic problem by proposing several different models based on the algorithms as we mentioned before. We also try to build a mode which contains CNN and LSTM to test the accuracy rate of predictions. Our results show LSTM model performs best, the accuracy rate of this one in 93.03%. CNN model also performs well with 89.76 accuracy rate%. However, the accuracy rate of CNN+LSTM model is only 80.2%, in our mind, overfitting is the main reason to cause this result. But Naïve Bayes performs pretty good on this dataset which has accuracy rate of 92.57% with 1 pseodocound. We conclude that the performance efficiency of models is based on type or complexity of datasets. We discuss the next steps for future work at the end of paper.

2 Related Work

Semantic modelling of sentence meaning is a well-researched and well-known topic in NLP. However, this information is informal and unstructured so that it is also a challenging topic for research in natural language processing. Since the 'bad language' in Twitter and a noticeable drop of accuracy for start of-the-art constituency parsers on tweets, the ironic detection modelling of tweets has captured the attention of researchers.

To build a semantic representation of a sentence with high accuracy of detecting sarcasm on social media, researchers have used multiple syntactic structure to compose a total representation as a model. (Mitchell and Lapata, 2010) defined the composition function[ii] of a sentence by algebraic operations over word meaning vectors to obtain sentence meaning vectors. (Nakagawa et al., 2010) describes a Tree-CRF classifier which uses a data-driven dependency parser which called "malt parser"[iii] , to obtain a parse tree for a sentence, and whose composition function uses the head-modifier relations of the parse tree. Recent techniques also include concept-level analysis for both knowledge-based[iv].

For ironic detection, due to the complexity of the task and the somewhat poorer accuracy of start-of-the-art constituency parsers on tweets, researchers have considered some unique model such as LSTM and build more complex models to solve this problem. In this situation, detection of irony is assumed to require a representation of the dialog context. (Joshi, A., Bhattacharyya, P. and Carman, M. J) propose two different computational frameworks to detect sarcasm that integrate the textual and visual modalities. The first approach exploits visual semantics trained on an external dataset, and concatenates the semantics features with state-of-the-art textual features. The second method adapts a visual neural network initialized with parameters trained on ImageNet to multimodal sarcastic posts[v]. Results show the positive effect of combining modalities for the detection of sarcasm across platforms and methods. (Liu B) also gives in-depth introduction to sentiment analysis and opinion mining and introduces some of the existing techniques for solving its sub-problems such as document sentiment classification, aspect-based sentiment classification and opinion spam detection.[vi] (Kalchbrenner et al., 2014) proposed a convolution neural network (CNN) with dynamic k-max pooling[vii], considering max pooling as function of input length. (Q. Huang, R. Chen, X. Zheng and Z. Dong) built a model uses the pre-trained word vectors as input and employs CNN to gain significant local features of the text[viii], then features are fed to two-layer LSTMs, which can extract context-dependent features and generate sentence representation for sentiment classification. (Yazhi Gao, W. Rong, Y. Shen and Z. Xiong) was trying to figure out the relationship between the length of convolutional filters and the performance of CNN classifier[ix], the study on commonly used datasets has shown its potential in identifying the different roles of specific N-grams in a sentence respectively and merging their contribution in a weighted classifier. (Lukin and Walker, 2013) proposed a potential bootstrapping method[x] for sarcasm classification in social dialogue to expand lexical N-gram cues related to sarcasm as well as lexicon-syntactic patterns.

More generally, (Aniruddha Ghosh and Tony Veale) proposes a neural network semantic model for the task of sarcasm detection which is composed of Convoluted Neural Network(CNN) and followed by a Long or short-term memory (LSTM) network and finally a Deep neural network(DNN)[xi]. This model yields an F-score of 0.92 for sarcasm detection. Nevertheless, the accuracy score depends on the complexity and size of the datasets, this complex model may not perform well enough.

Other work that is closely related is the detection of cyberbullying or trolling. Cyberbullying is a form of bullying or harassment using electronic means or attack another individual in an online community and it is recognized as a serious social problem in common, especially for adolescents. (E. Cambria and P. Chandra)[xii] designed a troll detector is trained by first identifying the concepts most commonly used by trolls and use the related concepts to expand the experiment results. This work uses a new opinion mining and sentiment analysis paradigm-Sentic Computing to analyze the texts.

3 Data

We used the dataset provided by SemEval 2018 task 3- Irony detection in English tweets. The dataset has tweets with binary labels. Label 0 indicates the tweet is non-ironic and label 1 indicates the tweet is ironic.

Examples:

- 1 I love hearing people's explanations about the dating world in America in the library while I am studying

- 0 Senate Just Confirmed a New Cabinet Member That Has 2nd Amendment Supporters Furious - The Senate confirme... http://t.co/3TvNFNw8tn

In the dataset, the total amount of data to 4792 tweets, among which 2396 instances are ironic and 2396 instances are not.

The group of semEval 2018 task 3 constructed this dataset by searching Twitter for the hashtags #irony, #sarcasm and #not. All tweets were collected between 01/12/2014 and 04/01/2015 and represent 2,676 unique users. According to the description of this dataset provided by semEval 2018, all tweets were manually labeled using a fine-grained annotation scheme for irony (Van Hee et al., 2016c) to minimize the noise introduced by groundless irony-related hashtags. Prior to data annotation, the entire corpus was cleaned by removing retweets, duplicates and non-English tweets, and replacement of XML-escaped characters (e.g., &).

The dataset has been annotated by the group of semEval 2018 task 3. They annotated 3000 tweets and introduced the methods used for annotation. The following is description provided by them.

The entire corpus was annotated by three students in linguistics and second-language speakers of English, which each annotated one third of the corpus. Brat (Stenetorp et al., 2012) was used as the annotation tool. To assess the reliability of the annotations, an annotation agreement study was carried out on a subset of the corpus (100 instances), resulting in Fleiss Kappa scores of 0.72 for recognizing (the different forms of) ironic instances. Annotation of the corpus resulted in the following distribution of the different classes:

- Verbal irony by means of a polarity contrast: 1,728 instances
- Other types of verbal irony: 267 instances
- Situational irony: 401 instances
- Non-ironic: 604 instances

Based on the annotations, 2,396 instances are ironic (1,728 + 267 + 401) while 604 are not. To balance class representation in the corpus, 1,792 non-ironic tweets were added from a background corpus. The tweets were manually checked to ascertain that they are non-ironic and are devoid of irony-related hashtags. This brings the total amount of data to 4,792 tweets (2,396 ironic + 2,396 non-ironic).

For the SemEval-2018 competition, this corpus was randomly split into a training (80% or 3,833 instances) and test (20%, or 958 instances) set. We have a total of 3833 labeled English Tweets as our training set.

We define 0.8/0.2 split on training set by using *sklearn.model_selection* package. We used 80% of training set or 3067 instances for training and 20% of training set or 767 instances for estimating accuracy. We plan to first use cross validation for training set to choose the model with lowest estimated out of sample error. Then train it with the full training set and predict the target outputs for the samples in the test set. Next, we will upload submission to the SemEval-2018 competition website and get the report accuracy rate.

4 Method

We conduct the task of irony detection as a classification problem (ironic or not). We chose naïve Bayes as the baseline algorithm and implemented averaged perceptron, non-averaged perceptron, combinations of deep neural network algorithms (e.g. CNN, DNN, and LSTM).

For Neural network, we first converted the input tweet into a matrix. Then,we tested our model with different settings of the hyperparameters for CNN (number of filter, filter size), LSTM (hidden memory dimension dropout ratio) and DNN (number of hidden memory units) both individually and by combination. For each model, we will do k-fold cross validation first among the training set, then choose the model with the best result and apply the model to the full training dataset.

4.1 Naïve Bayes

Naïve Bayes classifier is one of simple probability classifiers base on applying Bayes' theorem with strong independence between the features. Naïve Bayes classifiers make two simplifying but significant assumptions. First one is the bag of words assumption, we assume position doesn't matter and we assume that the features f_1, f_2,..., f_n only encode word identity and not position. Second one is Naïve Bayes assumption as follow:

$$P(f_1, f_2, ..., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c)$$

The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

For the text problem, we have to consider word positions and also do it in log space to avoid underflow and increase speed. So, it can be expressed as:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Pseudocode of Naïve Bayes:

```
function BOOTSTRAP($x, b$) returns $p\text{-}value(x)$
  Calculate $\delta(x)$
  for $i = 1$ to $b$ do
    for $j = 1$ to $n$ do    # Draw a bootstrap sample $x^{*(i)}$ of size n
      Select a member of $x$ at random and add it to $x^{*(i)}$
    Calculate $\delta(x^{*(i)})$
  for each $x^{*(i)}$
    $s \leftarrow s + 1$ if $\delta(x^{*(i)}) > 2\delta(x)$
  p-value($x$) $\approx \frac{s}{b}$
  return p-value($x$)
```

4.2 Classifier Perceptron

The perceptron is a classic learning algorithm invented in 1957 by Frank Rosenblatt. It can used for classification or structured prediction. For the task of irony detection, we use binary perceptron. For the training part, the perceptron algorithm iterates through the dataset, and predict for each training data. If the predicted label is correct, do nothing and predict the next training data. If the predicted label is incorrect, the perceptron updates weights to fix prediction error.

Generally, perceptron does not converge, in order to improve its performance, we also used the averaged perceptron. It is the same as vanilla perceptron except it maintains a collection of weight vectors and use the averaged weight vector at test time.

Pseudocode of averaged perceptron:

```
Initialize t = 1, $\theta_0 = 0$, $S_0 = 0$
For 10 iterations:
    For each training data (x,y):
        Predict $y^* = \text{argmax}_{y'} \theta^T f(x,y')$
        If y = y*, do nothing
        Else:
            $g = f(x,y) - f(x,y^*)$
            $\theta_t = \theta_{t-1} + rg$
            $S_t = S_{t-1} + (t-1)rg$
            t++
Calculate averaged $\theta$ based on S
```

The codes are adopted from hw2 and has been adjusted for this task.

4.3 Neural Network

Semantic modeling of sentence meaning using neural networks has been a target of attention in the social media community. Neural network architectures, such as CNN, DNN, RNN, and Recursive Neural Networks (RecNN) have shown excellent capabilities for modeling complex word composition in a sentence. A sarcastic text can be considered elementally as a sequence of text signals or word combinations. RNN is a perfect fit for modeling temporal text signals as it includes a temporal memory component, which allows the model to store the temporal contextual information directly in the model. It can aggregate the entire sequence into a temporal context that is free of explicit size constraints. Among the many implementations of RNNs, LSTMs are easy to train and do not suffer from vanishing or exploding gradients while per-forming back propagation through time. LSTM has the capability to remember long distance temporal dependencies. Moreover, as they performs temporal text modeling over input features, higher level modeling can distinguish factors of linguistic variation within the input. CNNs can also capture temporal text sequence through convolutional filters. CNNs reduce frequency variation and convolutional filters connect a subset of the feature space that is shared across the entire input (Chan and Lane, 2015). (Dos Santos et al., 2015) have shown that CNNs can directly capture temporal text patterns for shorter texts, yet in longer texts, where temporal text patterns may span across 15 to 20 words, CNNs must rely on higher-level fully connected layers to model long distance dependencies as the maximum convolutional filter width for a text is 5 (Figure 1).

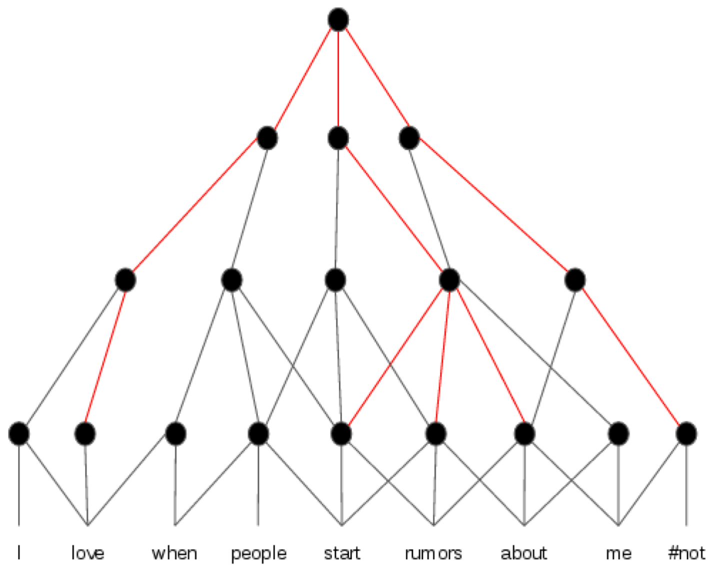

Figure 1. Sentence modeling with CNN

Another major limitation of CNNs is the fixed convolutional filter width, which is not suitable for different lengths of temporal text patterns and cannot always resolve dependencies properly. Obtaining the optimal filter size is expensive and corpus dependent, while LSTM operates without a fixed context window size. LSTM's performance can be improved by providing better features. Following the proposal of (Vincent et al., 2008), it can be beneficial to exploit a CNN's ability to reduce frequency variation and map input features into composite robust features and using it as an input to a LSTM network. DNNs are appropriate for mapping features into a more separable space. A fully connected DNN, added on top of an LSTM network, can provide better classification by mapping between output and hidden variables by transforming features into an output space. In the following section we define our proposed network in detail.

4.3.1 CNN Model

The CNN model we build in the project looks roughly as follows:

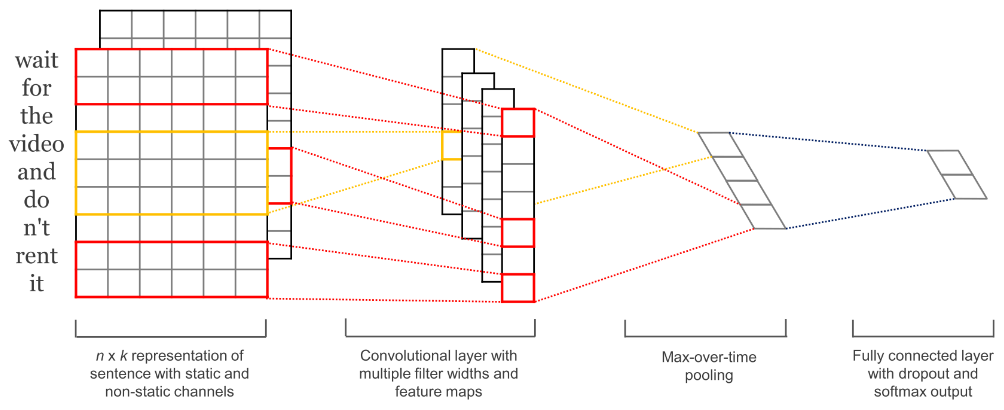
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 2. CNN followed by Fully Connected

The first layer is embedding layer, which convert the tweet into a vector by replacing each word with its dictionary index $s \in R^{l \times n}$. We know that tweet may have different lengths of input, to this problem, the tweet vector is padded and the tweet is converted into matrix $s \in R^{1 \times l}$, where l is the maximum length of tweets in the input corpus. The input vector is fed to the embedding layer which converts each word into a distributional vector of dimension D. Thus the input tweet matrix is converted to $s \in R^{l \times D}$. For each position p in the input matrix s, there is a window w_p of k consecutive words, denoted as:

$$w_p = [I_p, I_{p+1}, \dots, I_{p+k-1}]$$

The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time. A filter f convolves with the window vectors (k-grams) at each position in a valid way to generate a feature map $c \in R^{|S|-k+1}$ each element c_p of the feature map for window vector ω_p is produced as follows:

$$c_p = \text{func}(\omega_p \circ f + b)$$

where $\circ$ is element-wise multiplication, $b \in R$. is a bias term and func is a nonlinear transformation function that can be sigmoid, hyperbolic tangent, etc. Max-over-time pooling is then applied over the obtained feature map. We use multiple filters of different sizes and output from each filter is concatenated to get the final overall feature vector. This feature vector act as input for the fully-connected layer. We train the entire model by minimizing the binary cross-entropy loss over a mini-batch of training examples of size e.

$$E(y, \hat{y}) = \sum_{i=1}^{e} y_i \log(\hat{y}_i)$$

4.3.2 LSTM Model

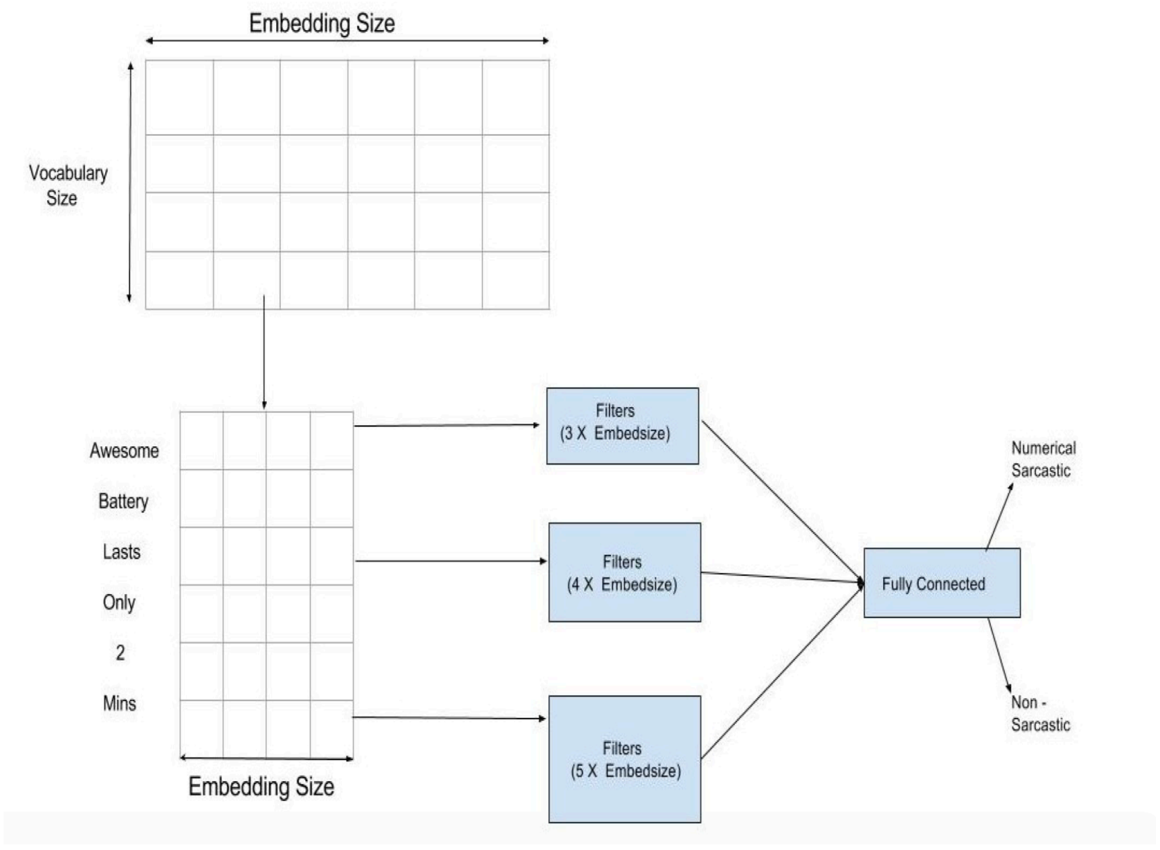

Figure 3. LSTM architecture for Sarcasm

RNN have demonstrated the power to capture sequential information in a chain-like neural network. RNN networks include a temporal memory component, which allows the model to store the temporal contextual information directly in the model. At each time step, it considers the current input x_t and hidden state h_{t-1}. Thus, Standard RNNs becomes unable to learn long-term dependencies as the gap between two time steps becomes too large. We adopted the standard architecture of LSTM proposed by (Hochreiter and Schmidhuber, 1997).

The LSTM architecture has a range of repeated modules for each time step as in a standard RNN, which is able to plot long term dependencies by defining each memory cell with a set of gates R^d, where d is the memory dimension of hidden state of LSTM, and it does not suffer from vanishing or exploding gradient while performing back propagation through time. LSTM contains three gates, which are functions of x_t and h_{t-1}: input gate i_t, forget gate f_t, and output gate o_t. The gates jointly decide on the memory update mechanism. Equation (3) and (2) denote the amount of information to be discarded or to be stored from and to store in memory. Equation (5) denotes the output of the cell c_t.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \qquad (2)$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \qquad (3)$$

$$q_t = tanh(W_q[h_{t-1}, x_t] + b_q) \qquad (4)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \qquad (5)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot q_t \qquad (6)$$

$$h_t = o_t \odot tanh(c_t) \qquad (7)$$

4.3.3 CNN-LSTM Model

This architecture is shown in Figure 3. The input matrix representation for the Tweet T is obtained from the Embedding matrix E as described above. Filters of size w*d where w is filter wide, as the maximum convolutional filter width for a text is 5 and d is the tweet word embedding dimension, slides over the input matrix I of the tweet in order to extract the features. We passed the output of the convolutional network through a pooling layer and max-pooling is used with size 4. All the filters are of same dimension and after performing pooling operation over their outputs, we obtained a concatenated feature matrix denoted as:

$$C = [c_1; c_2; \dots c_n]$$

where n in c_n denotes the total number of filters used in the architecture. Feature matrix $C \in R^{l \times n}$, each $c_i \in R^l$, where *l* is the dimension obtained after pooling operation. Let $x_j \in R^{1 \times n}$ is vector obtained from matrix *C*. Vector x_j is the input for the LSTM cell at *jth* time step and the LSTM cell runs for l time steps taking different input obtained from matrix C, at each timestep. At the end of lth timestep, output from the LSTM cell act as input for the fully connected DNN layer, which produces a higher order feature set based on the LSTM output, which is easily separable for the desired number of classes. Finally a softmax layer is added on top of the DNN layer. Training of network is performed by minimizing the binary cross-entropy error.

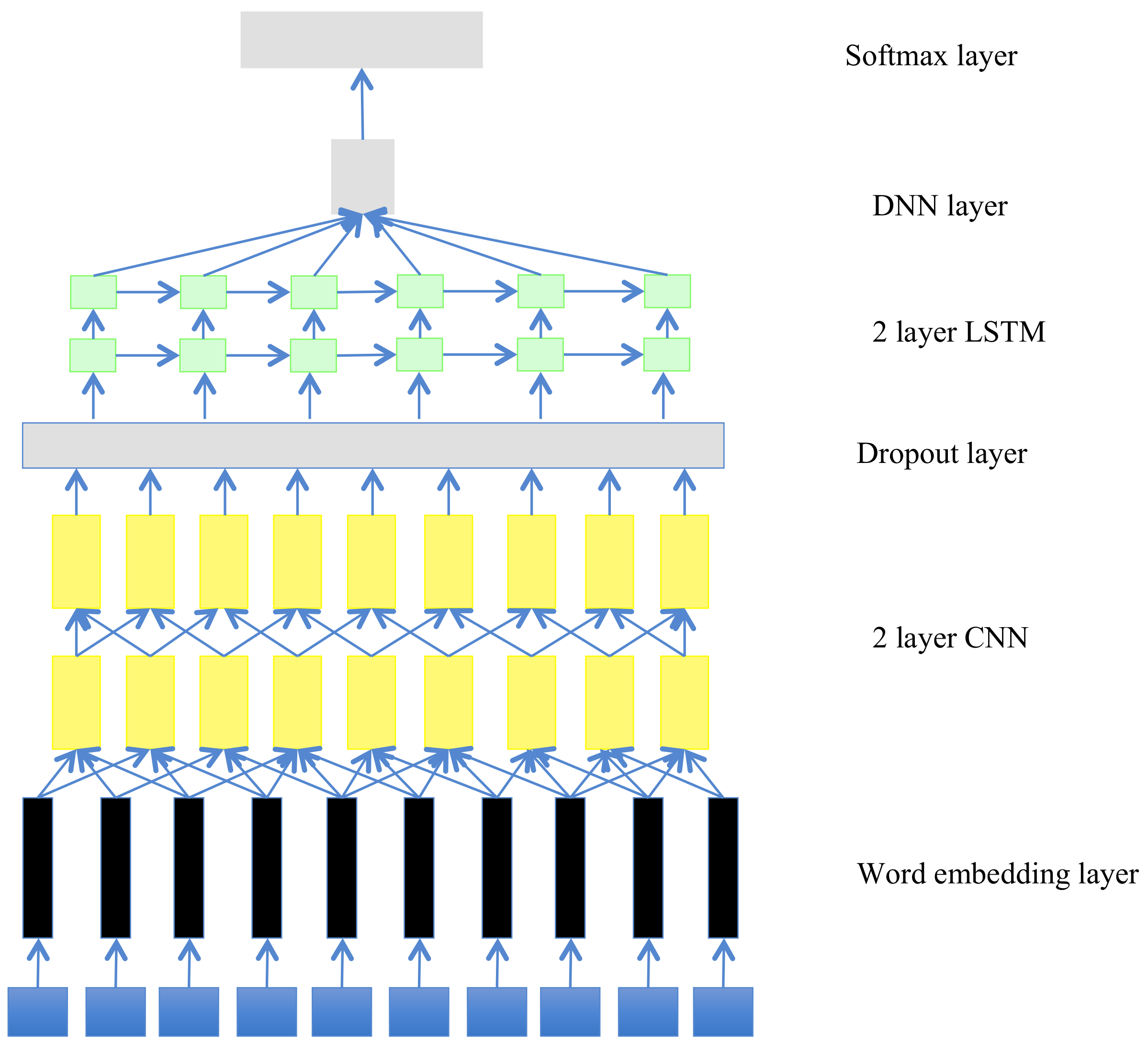

Figure 4. CNN Followed by LSTM Network

5 Results

5.1 Naïve Bayes

We evaluate the model with different pseudocount parameter. The accuracy for different pseudocount parameter are shown in the table. From the table we can see the accuracy goes down when the pseudocount parameter increase.

pseudocount parameter	1	10	100	100
accuracy(%)	92.57	84.75	76.14	79.14

Table 1. Naïve Bayes accuracy with different pseudocount parameter

5.2 Classifier Perceptron

We used 4 different step sizes {0.01, 0.1, 1, 10} for both vanilla perceptron and averaged perceptron, and ran 10 iterations for each model. The following is the "accuracy vs iterations" graph for the 8 models. Averaged perceptron models are likely to achieve an accuracy around 0.62 at the last iteration, and 0.60 for vanilla perceptron models. Averaged perceptron models have better accuracy scores than vanilla perceptron models. According to the graph, step size does not affect the results very much, and a larger step size are likely to result a better accuracy score, which is not expected.

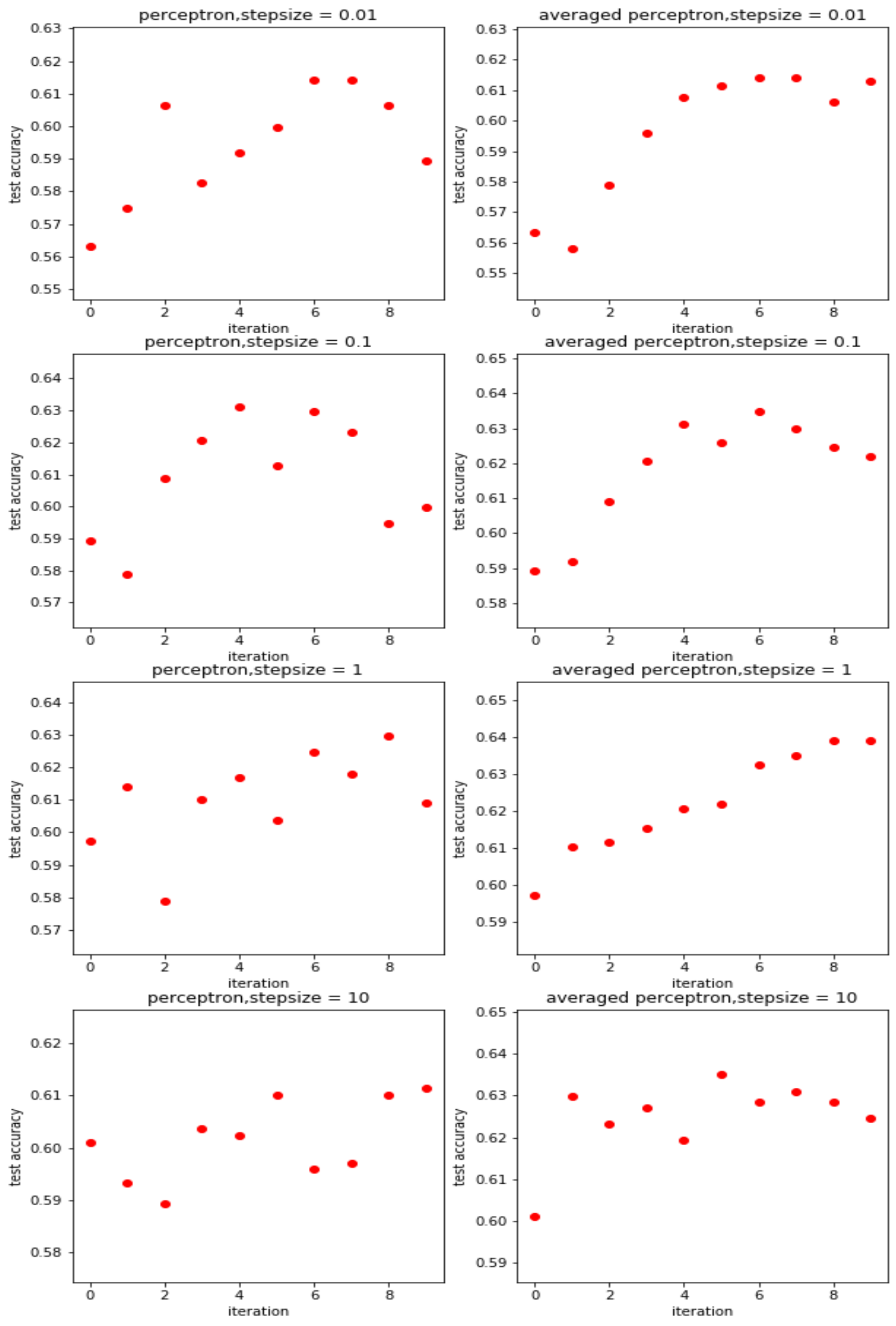

Figure 5. Classifier perceptron accuracy with different stepsize

5.3 Neural Network

Model	Parameter	Accuracy
CNN	filter size = 64, filter width = 2	0.8727
	filter size = 128, filter width = 2	0.8895
	filter size = 64, filter width = 3	0.8830
	filter size = 128, filter width = 3	0.8976
LSTM	hidden memory unit = 64	0.9198
	hidden memory unit = 128	0.9303
CNN+LSTM	filter size = 64, filter width = 3, HMU = 64	0.8020
	filter size = 128, filter width = 3, HMU = 128	0.8210

Table 2. Neural Network accuracy results

For CNN Model, we use 64 and 128 number of filters each of size 2 and 3. In the pooling layer, we perform max-over time pooling over the output from each filter. For LSTM model, we investigated with different hidden unit dimension as 64 and 128. For CNN-LSTM Model, number of filters is 64 and 128, each with the same dimension. All the deep learning experiments were implemented using tensorflow and kares.

The best results we can get using Neural Network is LSTM model with 128 hidden memory unit. The corresponding accuracy is 0.9303.

6. Discussion and Future Work

In this paper, we conclude the LSTM model has the best accuracy rate, Naïve Bayes model and CNN model also perform well enough. Even though some of our models perform very well on this dataset, one is not as good as we expected. The most complex model which is based on CNN+LSTM performs not as good as others. We think the main reason is overfitting, since the capacity of the model decrease the loss and increase variance simultaneously. The type and complexity of data is also a reason to affect the final results. Our dataset is not very difficult to detect the property, therefore, that complex model was caused to be overfitting easily.

Today's world is becoming more digitalized by each second passing. With many number of people preferring to spend more time online, huge amount of data is generated. Mostly, the data from social media is largely about recent events or comments of friends' behavior. Many creative ironic words and expressions are generated at the same time, it must be more difficult for the future ironic detection task. For future work, we can also perform a trend analysis for more difficult data to separate the semantic space of sarcasm and non-sarcasm more efficiently.

References

i S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

ii Jeff Mitchell, Mirella Lapata Composition in Distributional Models of Semantics, ISSN: 0364-0213 print / 1551-6709 online

iii Penelope Brown and Stephen C Levinson. 1978. Universals in language usage: Politeness phenomena. In Questions and politeness: Strategies in social interaction

iv A. Charng-Rurng Tsai, C.E. Wu, R. Tzong-Han Tsai, J. Yung-jen Hsu Building a concept-level sentiment dictionary based on commonsense knowledge IEEE Intell. Syst., 28 (2) (2013), pp. 22-30

v Joshi, A., Bhattacharyya, P. and Carman, M. J.: 2016, Automatic Sarcasm Detection: A Survey, CoRR abs/1602.03426.

vi Liu, B.: 2012, Sentiment Analysis and Opinion Mining, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers.

vii Kalchbrenner N, Grefenstette E, Blunsom P. A Convolutional Neural Network for Modelling Sentences arXiv:1404.2188 [cs.CL]

viii Q. Huang, R. Chen, X. Zheng and Z. Dong, "Deep Sentiment Representation Based on CNN and LSTM," 2017 International Conference on Green Informatics (ICGI), Fuzhou, China, 2017, pp. 30-33.
doi: 10.1109/ICGI.2017.45

ix Yazhi Gao, W. Rong, Y. Shen and Z. Xiong, "Convolutional Neural Network based sentiment analysis using Adaboost combination," 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, 2016, pp. 1333-1338.

x JOUR Extracting relevant knowledge for the detection of sarcasm and nastiness in the social web Raque Justoa, Thomas Corcoranb, Stephanie M. Lukinb Marilyn Walker bM, InésTorres

xi Ghosh, A., & Veale, T. (2016). Fracking Sarcasm using Neural Network. WASSA@NAACL-HLT.

xii E. Cambria, P. Chandra, A. Sharma, A. Hussain, Do not feel the trolls, in: Proceedings of the Third International Workshop on Social Data on the Web (SDoW2010), vol. I, 2010..