CS 224D Final Project Report - Entity Level Sentiment Analysis for Amazon Web Reviews

Y. Ahres, N. Volk
Stanford University
Stanford, California
yahres@stanford.edu,nvolk@stanford.edu

Abstract

Aspect specific sentiment analysis for reviews is a subtask of ordinary sentiment analysis with increasing popularity. In this work, the goal is to characterize the sentiment of specific aspects in camera web reviews with various recurrent neural networks that are tailored to this purpose to predict a vector of aspect sentiments at review level: The baseline of a simple RNN is extended to a bidirectional RNN (BRNN), a LSTM and a bidirectional LSTM (BLSTM). Furthermore, different methods are presented and evaluated how to deal with highly biased/skewed data which is a common problem with reviews. In particular, a novel approach using aspect information content increasing mini-batch sampling is presented. Models are evaluated using F1 scores.

1 Introduction

Well-known tasks for Natural Language Processing (NLP) involve general sentiment analysis for various types of texts or speech. Such models are capable to output the general sentiment of a sentence or piece of text. However, a special subset of tasks such as mining product reviews requires more than high level sentiment analysis, rather it requires entity level sentiment analysis (or aspect specific sentiment analysis) on the level of review-specific features (for instance the sentiment/score of ”display” in a product review of a smart phone). Other applications are public opinion predictions, opinion mining or emotion detection. In many applications, the goal is to perform this sentiment analysis over time. This analysis can naturally be used for other tasks such as recommender systems or summarizing tasks. [War+11] [Ser+15]

2 Problem statement

In order to extract entity level sentiments, Amazon reviews should be analyzed in the following way: 1- extract most important features of a product and 2- assign an overall score for each of them. This will allow us to structure the information from thousands of reviews and add a value to the end-customer by summarizing the reviews into a comprehensive yet concise table. The specific problem formulation: Given a review as form of a sentence l_i, identify the sentiments/scores $y_{a,i}$ of relevant features/aspects a.

3 Related Work

Recent work ([PSM14]) introduced a new model that allows neural nets (NN) to evaluate contextual sentiment: The proposed Global Belief-Recursive Neural Network (RecNN) represents the state-of-the-art for granular sentiment analysis. In order to correctly capture contextual sentiment, a backward step from upper tree nodes is introduced here. A different approach is obtained by considering aspect specific sentiment analysis using hierarchical deep learning according to [LSM]. Here, separate aspect sentiment models (SAS) or Joint Multi-Aspect Sentiment models (JMAS) train root-node-level softmax classifiers with aspect and sentiment as classificiation outputs. However, a prelabeling of the aspects/product features is required for these models. This work should explore different recurrent neural networks (RNN) including bi-directional recurrent NN and Long-Shert-Term Memeory (LSTM) RNNs that try to capture aspect specific sentiment through context.

4 Dataset

First, we used the public available data sets used in [HL04] and [DLY08]. The data contains reviews about cameras on Amazon. The reviews are annotated with product feature names and the respective sentiment. Clearly, each of these sentences has the entity reviewed and the level of sentiment about it. Therefore the tasks we perform come down to a NER combined with sentiment analysis. After conducting a variety of experiments we concluded that the amount of reviews is not sufficient for our purpose. Therefore, more than 5000 reviews were labelled manually. The total data set contains 7149 labelled reviews with the following aspects: Picture, Prize, Battery, Portability, Features. Aspect sentiments are labelled on a scale from $-5...5$, however, for our purpose we only consider aspect sentiments on a scale from -1 to 1: $s_i = -1, 0, 1$. We also preprocessed the publicly available data to fit the aspect mentioned into our aspect buckets. For instance, by indicating, size[+1], we detect that they refer to portability with a positive sentiment. This involved some manual word and sentiment translation. Please note that we are also making our datasets publicly available for future research and projects.

Dataset split	Product	No. of reviews/sentences
Hu and Liu, KDD-2004	DVD Player	836
	Camera	380
	Camera	642
Ding, Liu and Yu, WSDM-2008	Canon	349
	Canon	229
Manually labelled	SonyA3000	597
	Sony W800	230
	Polaroid Z2300	570
	Panasonic Lumix	1297
	Nikon S8100	1491
	Nikon COOLPIX L830	986
	Canon EOS Rebel	118
	Canon PowerShot SX700	389

Figure 1: Product review dataset structure: each aspect is labeled with sentiments from $-5...0...5$.

5 Mathematical formulation

5.1 Simple Recurrent Neural Network (RNN)

The model that should be evaluated and deployed first and serve as a baseline is a RNN that is tailored to our purpose: In contrast to providing an output (e.g. via softmax) for each node/word, the model should only output a final prediction at the end of the sentence. A sketch of the model is shown in Figure 2. In order to capture the entire context the backpropagation-through-time (bptt) parameter was selected to always exceeds the sentence length, or in other words: the model always considers the entire sentence content.

As explained above, for each of the 5 aspects, we have $-1, 0, 1$, so a one-hot vector of 3 elements for each one; -1 being the most negative, 1 the most positive and 0 neutral. Note that if a review does not mention one aspect, it is assumed neutral towards it. Therefore, for 5 different aspects, the prediction $\hat{y} \in R^{15}$.

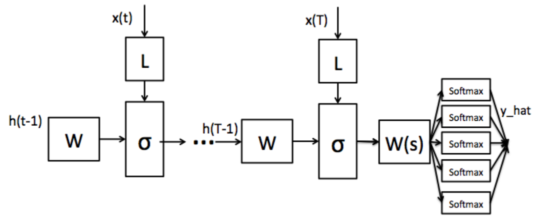

Figure 2: Simple RNN for aspect level sentiment analysis

For $x^{(1)}, x^{(2)}$, ... the forward propagation of this model is defined as follows:

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(t)}) \tag{1}$$

The final output for each aspect results in:

$$\hat{y} = softmax(W_s h^{t=T}) \tag{2}$$

where $\hat{y}$ is the concatenation of the single predictions for each aspect of the camera:

$$\hat{y} = \begin{pmatrix} \hat{y_1} \\ \hat{y_2} \\ \hat{y_3} \\ \hat{y_4} \\ \hat{y_5} \end{pmatrix} \tag{3}$$

T is the last t. Clearly, we compute our sentiment only at the end and we need to learn the matrices W_x, W_h, W_s and L (word vectors).

The idea behind such a structure is that RNNs accumulate the sentiment over the whole sentence and output them as shown above.

5.2 Bidirectional Recurrent Neural Network (BRNN)

A huge drawback of the standard RNN is that post-word context is not considered sufficiently as the sentence is observed only in one direction. However, in order to determine aspect sentiment, the direction should not matter and therefore, a bidirectional recurrent neural network is implemented: Accumulating in two directions rather than one doubles the number of parameters and allows for more flexibility. Here, the model runs through the sequence in reverse order with a different set of parameters that have to be updated. A simplified sketch of the model is shown in Figure 3. Note that, to specify the backward channel, we just need to invert the sequence of words and perform the same RNN as we did before, on the other direction. The final softmax output is calculated concatenating h_f and h_b from both directions:

$$h_f^{(t)} = \sigma(W_{h_f} h_f^{(t-1)} + W_x x^{(t)}) \tag{4}$$

$$h_b^{(t)} = \sigma(W_{h_b} h_b^{(t-1)} + W_x x_{inverted}^{(t)}) \tag{5}$$

$$\hat{y} = softmax(W_{s,brnn} \begin{pmatrix} h_f \\ h_b \end{pmatrix} + b_s) \tag{6}$$

5.3 Long-term Short-term Memory (LSTM-) RNN

In order to capture the context of aspects in a more granular way, a LSTM-RNN should be deployed: Here, rather than just scanning a word sequence in order, the model stores information in gated units,

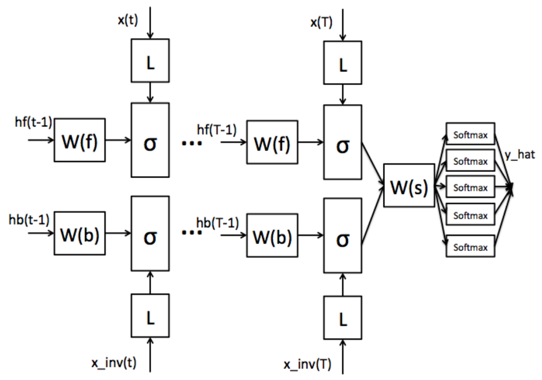

Figure 3: BRNN for aspect level sentiment analysis

in particular an input gate $i^{(t)}$ with weight on the current cell, a forget gate $f^{(t)}$ to forget non-relevant information, an output gate $o^{(}t)$ to specify, how relevant the current cell content is and a gate $\tilde{}c^{(}t)$ for the new memory cell. For time series tasks of unknown length LSTMs are often capable of storing and forgetting relevant/non-relevant information better than common RNNs. $c^{(t)}$ and $h^{(t)}$ are final states and hidden vectors, respectively.

$$i^{(t)} = \sigma(W_i x^t + U_h h^{(t-1)}) \tag{7}$$

$$f^{(t)} = \sigma(W_f x^{(t)} + U_f h^{(t-1)}) \tag{8}$$

$$o^{(t)} = \sigma(W_o x^{(t)} + U_o h^{(t-1)}) \tag{9}$$

$$\tilde{c^{(t)}} = tanh(W_c x^{(t)} + U_c h^{(t-1)}) \tag{10}$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c^{(t)}} \tag{11}$$

$$h^{(t)} = o^{(t)} \circ tanh(c^{(t)}) \tag{12}$$

The final prediction results in:

$$\hat{y} = softmax(W_y h + b_y) \tag{13}$$

In order to avoid errors in the backpropagation and to streamline implemenation, the model was implemented using the Python-Theano Library [Ber+10].

5.4 Bidirectional Long-term Short-term Memory (LSTM-) RNN / BLSTM-RNN

Similar to the BRNN as an improvement of a simple RNN, a bidirectional LSTM-RNN is scanning the sequence of words in reverse order using a second set of parameters. The final output prediction is calculated as the concatenation of the final hidden vectors from the original sequence and the reversed sequence:

$$\hat{y} = softmax(W_y \begin{pmatrix} h_f^T \\ h_b^T \end{pmatrix} + b_y) \tag{14}$$

Figure 4 displays the structure of the BLSTM.

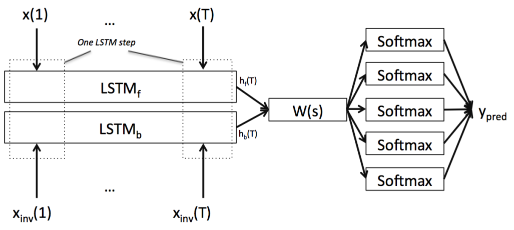

Figure 4: B-LSTM-RNN (BLSTM) for aspect level sentiment analysis

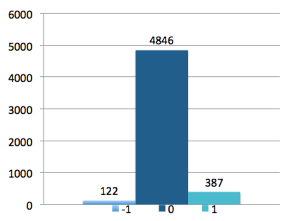

Figure 5: Distribution of aspect level sentiment in the datasets

6 Implementation

The first implementation with a standard RNN performs fairly poorly. One main reason is the following: Most reviews do not contain detectable aspects with positive/negative sentiment, and if so, they mostly contain one or two non-neutral aspects only. Therefore, the prior distribution of our data is very biased towards the 0-class (neutral class) as you can observe in Figure 5.

The model tends to always predict 0 and is not capable to predict -1 or 1. Different approaches to deal with our biased prior distribution should be implemented and evaluated:

6.1 Method 1: Preprocessing highly biased data

A straightforward method to deal with highly biased data is sub-sample the majority class or duplicate minority classes. Due to our limited data, duplication is deployed in the way that each non-trivial review is duplicated i-times where i is the number of non-trivial aspects contained in this review. This method directly affects the training data and can be used as a preprocessing for other method as well. We shall test it on its own and with combinations with the other proposed methods.

6.2 Method 2: Weighted cost RNN

Another approach is to modify the cost function directly in the model: In order to overcome the problem that the model does not predict a positive/negative sentiment, the cost in this case is multiplied by a weighting term $w_i > 1$ for the non-trivial classes, and $w_i < 1$ respectively for the 0 class. The cross entropy cost function for one aspect therefore results in:

$$E_a = \sum_i w_i y_i log(\hat{y_i}) \tag{15}$$

with the following constraints:

$$w_1 = w_3, \sum_i w_i = 1 \tag{16}$$

Intuitively, this means that we penalize the cost function more when we miss existing sentiments. It forces the model to look for sentiments. If the weights assigned to w_1 and w_3 are smaller then w_2,

```
Data: $Y$,$n_{candidate}$,$size$,$n_{batches}$,$replacement$
Result: $batches$
$batches \leftarrow EmptyList()$;
for $i = 1..n_{batches}$ do
    $current\_batch \leftarrow EmptyList()$
    for $j = 1..size$ do
        $candidates \leftarrow sample\_candidates(Y, n_{candidate})$
        $best \leftarrow choose\_best(Y, candidates, current\_batch)$
        $current\_batch \leftarrow current\_batch + best$
        if $replacement = False$ then
            $discard(Y, best)$
        end
    end
    $batches \leftarrow batches + current\_batch$
end
```

Algorithm 1: Create Information augmented mini batches

```
Data: $Y$,$candidates$,$current\_batch$
Result: $best$
forall the $candidate \in candidates$ do
    if $defined\_entropy(current\_batch + candidate)$ then
        $entropies[candidate] \leftarrow computeEntropy(current\_batch + candidate)$
    end
    else
        $entropies[candidate] \leftarrow 0$
    end
end
$best \leftarrow argmax(entropies)$
```

Algorithm 2: *choose_best* procedure

this results in a model that predicts only zeros. On the other extreme, if w_1 and w_3 are too large compared to w_2, the neural net will always predict something and have even lower accuracy. It is up to us to find optimal weights using cross validation and F1 scores on the dev set. The optimal weights were found on our fastest model and kept for the rest of the evaluation, it was: $w = [1.1, 0.8, 1.1]$

6.3 Method 3: Aspect information content increasing mini-batch sampling

In this paper, we propose a novel approach to solve this problem. This heuristic approach is based on a mini-batch gradient descent augmented with information gain. One can think about it as a way to create rich mini batches in terms of entropy. Each minibatch is created by randomly sampling data from the training set and use it to build a minibatch with the highest possible entropy. Entropy is the expected value of information contained in a set. Therefore, maximizing it is likely to maximize the information capability of each batch. The Algorithm 1 shows how we create these batches based on the data.

As shown in Algorithm 1, we selected an arbitrary number of candidates to add the mini batch, generally between 3 and 5. And we find the best one by choosing the one that maximizes the entropy of the built batch. We've developed this algorithm, with and without replacement of the candidates added to mini batches. However, the one with replacement seems to work better. We will keep it from now on.

Among the challenges of the implementation we encountered, one fundamental limitation of the entropy definition and computation. The entropy is defined as follows:

$$H(X) = \sum_{i \in C} P(x_i)I(x_i) = -\sum_{i \in C} P(x_i)log_2(P(x_i)) \tag{17}$$

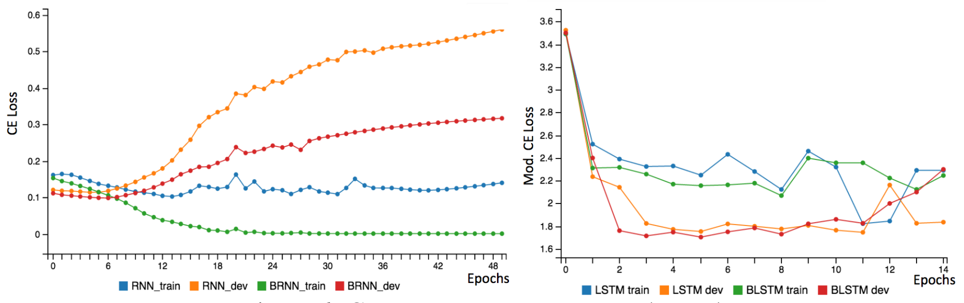

Figure 6: Cross Entropy Loss vs Epoch number

Where C defines the set of classes, here -1, 0 and 1 we have. And the probabilities are computed using the following:

$$P(x_i) = \frac{n_i}{\sum_j n_j} \tag{18}$$

Where n_i is the number of times the class (sentiment) i appeared in the data, independently of the aspect. However, when building the mini batches, we start with an empty set. Therefore, initially all probabilities are 0 and the entropy is not defined unless all the classes are represented at least once. To deal with this issue, we set 0 entropy when either one of the probabilities is zero. This will immediately give a non-trivial advantage any candidate that would contain a sentiment that is not yet in the batch. If none of the candidates contains the missing class, we select one randomly.

These three approaches were added and combined to our previously presented models. In the next section, we shall compare them along with the models implemented.

7 Tweaking and Optimization of Parameters

In order to tweak and find the right parameters for our model, we divided our data sets into three sets: a training set, a development set for cross validation and optimization and a test set that will be used in the next section to give our final scores. In this section, we describe how we performed our optimizations and show how the models behave depending on it. For each of the models described, many parameters had to be tweaked. However, due to time and hardware constraint, we weren't able to perform a joint optimization that would have resulted in optimal setting. Instead, we chose to fix some parameters, like the learning rate, to reasonable value and tweak the others (word vector dimension dim_{wv}, number of epochs n_{epochs}) jointly or marginally depending on the complexity of the model and how much time it takes. Figure 6 shows how the BRNN and the baseline RNN behave through the different epoch on the left and how the LSTM and BLSTM behave on the right.

Clearly, we see that from epoch 8, the models start overfitting. Therefore we stick to an epoch 8 for the rest of the evaluation. We also wanted to tweak the parameters of our novel approach but mini-batch GD can be incredibly time consuming.

Following implementations in the course, dim_{wv} was selected to 100. For the LSTM, the hidden layer dimensions dim_{dl} were selected to be 30.

8 Results and Discussion

To evaluate the performance we use the F1-score, which is equivalent to a harmonic mean of recall and precision and has a higher significance than one of these metrics on its own. For our multiclass classification problem we select the macro-averaged F1-score as it gives equal weight to all classes and therefore puts emphasis on the rare classes, too, which is our aim:

$$F_i = \frac{2\pi_i\rho_i}{\pi_i + \rho_i} \tag{19}$$

$$F_macro = \frac{\sum_i F_i}{n_{classes}} \tag{20}$$

where π is the precision and ρ the recall. It is calculated from the local categories and then averaged without considering the data distribution/weights. The micro-averaged F1 score is computed globally:

$$\pi_{global} = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}, \rho_{global} = \frac{sum_i TP_i}{\sum_i (TP_i + FN_i)} \tag{21}$$

$$F_m icro = \frac{2\pi_{global}\rho_{global}}{\pi_{global} + \rho_{global}} \tag{22}$$

Table 2 shows the performance metrics of our models. Please note that in our case, the accuracy does not reflect the overall performance. The baseline performs poorly, predicting only zeros. However, since most of the data is composed of zeros, the accuracy and recall weighted averages are pretty high.

Model+Extension	Precision	Recall	F1(macro)	F1(micro)
RNN (baseline)	81.2%	89.1%	31.0%	81.0%
RNN+Duplicate	84.3%	89.4%	31.2%	85.5%
RNN_Weighted + Duplicate	86.5%	89.9%	32.4%	85.6%
BRNN	85.3%	90.1%	34.9%	86.3%
BRNN+Duplicate	85.7%	89.5%	41.2%	87.3%
BRNN_Weighted + Duplicate	89.9%	90.5%	42.2%	87.5%
BRNN+mini-batches	**90.8%**	**90.8%**	**46%**	**90.87%**
BRNN_Weighted+mini-batches	86.1%	89.9%	36.7%	87.1%
LSTM+Duplicate	81.2%	89.9%	32.2%	90.4%
LSTM_Weighted+Duplicate	83.3%	89.3%	34.4%	89.7%
LSTM+mini-batches	84.3%	82.4%	39.0%	83.1%
LSTM_Weighted+mini-batches	84.1%	85.5%	38.2%	86.2%
BLSTM+Duplicate	81.2%	89.9%	32.2%	90.4%
BLSTM_Weighted+Duplicate	81.9%	88.6%	32.2%	89.2%
BLSTM+mini-batches	83.7%	87.6%	36.5%	88.3%
BLSTM_Weighted+mini-batches	83.9%	83.7%	38.3%	84,4%

Figure 7: Results of different models using the strategies described in Section 8

The basic LSTM and BLSTM perform very similarly poor as the baseline RNN. As we can see from the table, the BRNN combined with our novel approach based on augmented mini-batches performs the best in all metrics. Intuitively, one can think of it as the best way to overcome biased distribution in our case given the lack of flexibility and the very high bias we suffered from.

9 Conclusion

In this paper, we compared various approaches to tackle entity-level sentiment analysis. Among the challenges we encountered: the lack of high quality-labeled data online forced us to label our own based on Amazon reviews; we implemented various extensions to overcome the highly biased distribution including a novel method based on information gain. This method, combined with a bidirectional recurrent neural network outperformed all the other models we implemented. This is very promising for this entropy based method and extensive experiments will be made to evaluate its efficiency for broader applications. Note that an interactive portfolio has been implemented to explore our work and results. It can be found at:

https://frozen-caverns-5581.herokuapp.com

And the code can be found at:

https://github.com/youssefahres/EntitySentiment

Please note that the data labeled will be made available online for the Stanford community and upon request for non-Stanford affiliates.

References

[HL04] Minqing Hu and Bing Liu. "Mining and summarizing customer reviews". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 168–177.

[DLY08] Xiaowen Ding, Bing Liu, and Philip S Yu. "A holistic lexicon-based approach to opinion mining". In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM. 2008, pp. 231–240.

[Ber+10] James Bergstra et al. "Theano: A CPU and GPU math compiler in Python". In: *Proc. 9th Python in Science Conf*. 2010, pp. 1–7.

[War+11] Charles B Ward et al. "Empath: A framework for evaluating entity-level sentiment analysis". In: *Emerging Technologies for a Smarter World (CEWIT), 2011 8th International Conference & Expo on*. IEEE. 2011, pp. 1–6.

[PSM14] Romain Paulus, Richard Socher, and Christopher D Manning. "Global Belief Recursive Neural Networks". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2888–2896.

[Ser+15] Jesus Serrano-Guerrero et al. "Sentiment analysis: A review and comparative analysis of web services". In: *Information Sciences* 311 (2015), pp. 18–38.

[LSM] Himabindu Lakkaraju, Richard Socher, and Chris Manning. "Aspect Specific Sentiment Analysis using Hierarchical Deep Learning". In: ().