Milestone 2: Leveraging Large Amounts of Weakly Supervised Data for Multi-Language Sentiment Classification (WWW 2017)

Chun-Sheng CHUNG [*1] **Jun-Lin WU** [*1]

1. Readme

1.1. Dataset link

1.1.1. Word Embeddings

The initial Word Embeddings model we used is released by the author of the paper, which was trained with Word2Vec on 590 million English Tweets using 52 dimensions. https://spinningbytes.com/resources/embeddings/

1.1.2. Tweets

For distant-supervised learning phase, we collect tweets data where each tweet contains at least one emoticon (positive or negative). We use smile and sad emoticons as weak-supervised positive and negative labels. These emoticons are then removed from the tweets. https://drive.google.com/file/d/19JF0XuHHneNPUNWVE

1.1.3. Annotated Tweets

For supervised learning phase, we use annotated tweets from the Github repository provided by the authors of the original paper. https://github.com/spinningbytes/deep-mlsa

1.2. How to import data

Word embeddings and its word-index dictionary are npy and pickle files, numpy and pickle python library can be utilized to import them. For tweets file, we use python build-in input function. For annotated tweets, we use pandas library.

1.3. Additional library

In original paper, the authors used Keras to implement multi-layer CNN model. In this project, we rewrite it with Google's Tensorflow Python library.

1.4. Other environment setting

We use Python3 and Tensorflow version 1.3 to implement baseline method.

2. Training process

2.1. Goal

The goal of the project is to implement the methods described in the original paper, which is to develop a multilingual sentiment classifier of tweets with limited amount of supervised data.

Due to the fact that it is very expensive to acquire human-labeled sentiment data for such training and the lack of data in other languages, the authors of the original paper came up with the idea of exploiting emoticons in tweets as distant-supervised data. This kind of data is relatively easy to gather with Twitter API, and less preprocessing is needed by human. with the proposed method, the authors were able to win the first place in Task 4 of SemEval-2016.

2.2. Network Struture

The structure of the network is shown in Figure 1 and explained in the following paragraphs. The selections of activation functions and optimizers are the same as the original paper: ReLU for activation functions of convolution layers and the hidden layer, and AdaDelta optimizer is applied.

2.2.1. Preprocessing: Tweet Word Encoding

Preprocessing of the tweets is required before feeding them into the neural network model. We first lowercase the tweets, then replace the account mentions and URLs with specific tokens. Then, the NLTK Tweet tokenizer is applied to segment the tweets into tokens. Finally, we use a word dictionary, which is paired with the Word2Vec embedding matrix, to encode the tokens into indices of rows in the embedding matrix.

Tokens unknown by the embedding matrix are assigned to a zero vector at the end of the matrix, but the vector is

[*]Equal contribution [1]National Tsing Hua University, Taiwan. Correspondence to: Chun-Sheng CHUNG <f0923206757@gmail.com>, Jun-Lin WU <phejimlin@gmail.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

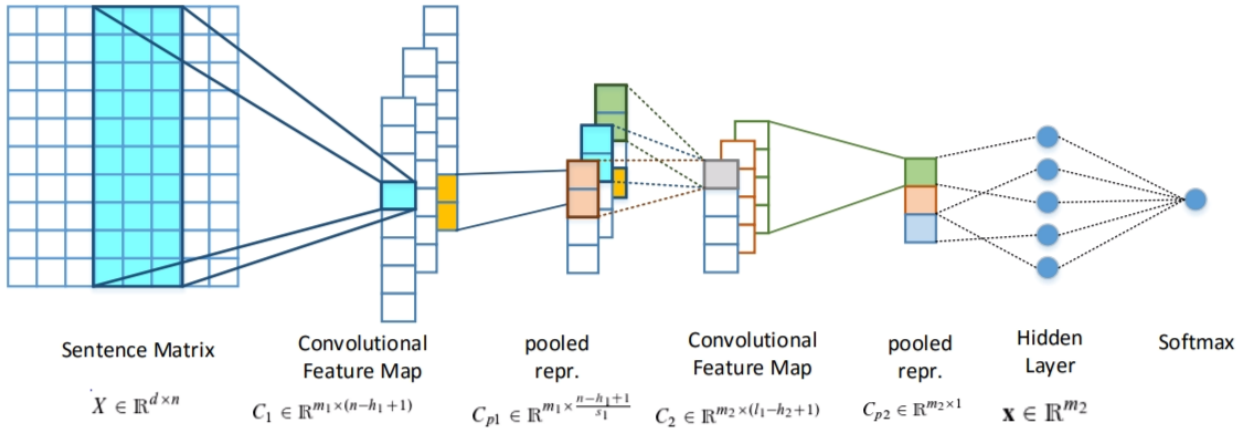

Figure 1. The network structure from original paper

still trainable to capture the information of unknown tokens. All of the tweets are padded to the maximum length of the tweets with another untrainable zero vector.

2.2.2. Word Embedding Layer

The encoded tweets are converted into Word2Vec vectors in this layer using simple lookup function.

2.2.3. Convolution Layers

After the tweets are converted into word embedding vectors, they will go through 2 sets of convolution and pooling layers, each contains 200 filters. The weights of the filters are initialized with truncated normal distribution with standard deviation of 0.1. The padding method is set to *padding*.

The window size of the first convolution layer is set to 4, which means applying 4-gram to the tweets. the window size of the first max-pooling layer is set to 4 and stride is 2.

The second convolution layer has a window size of 3, and the max-pooling is performed to extract the maximum value of each filter. After these operations, we will have a vector of 200 features.

2.2.4. Hidden layer and Softmax Layer

The features are then sent into a fully-connected hidden layer of 200 nodes to perform the transformation of said 200 features. Finally, a softmax layer gives the prediction: *positive*, *negative*, or *neutral*. The weights in the hidden layer are initialized with Xavier initializer.

2.3. Training phases

Following the original paper, we performed 2 phases of training. The training process is illustrated in Figure 2.

Dataset	Total	Neutral	Neg.	Pos.
Word embeddings	300M	-	-	-
Pre-training	1.1M	-	0.5M	0.6M
Training	18044	7544	2748	7752
Test	20632	10342	3231	7059

Table 1. Dataset details

(The first word-embedding training phase is omitted because we have the provided word embedding file.)

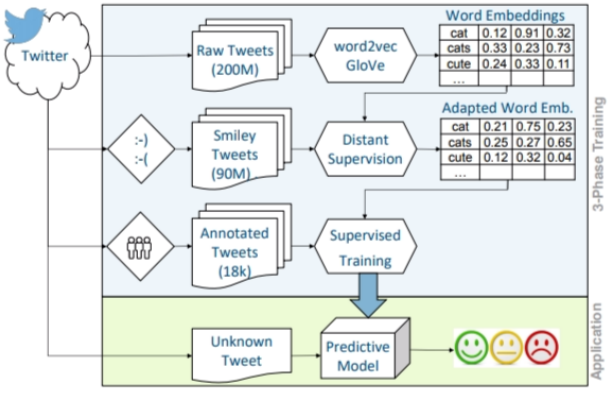

Figure 2. The training process from original paper

2.3.1. Distant Supervision Phase

In distant supervision phase, we use the collected tweets with emoticons as input. The emoticons are used as positive/negative labels, and are removed from the tweets. The details of data is shown in Table 1. The network is trained on this dataset for 50 epochs.

The main reason for this phase is that in the original Word2Vec embedding space, some keywords for sentiment classification are very close to one another due to the similar position in a sentence. For example, the words "great" and "awful" can have very similar neighbor words in a sentence, such as "It feels great/awful", but the sentiment of this two words are totally different. By training the network with an amount of distance-supervised data, the embedding matrix can be updated to include the concept of sentiment, and split the words mentioned above in the embedding space. The changes of word embeddings are illustrated in Figure3 and Figure 3. Although the differences in our embeddings are not very obvious in the figures, it is still beneficial to the total accuracy in the next phase.

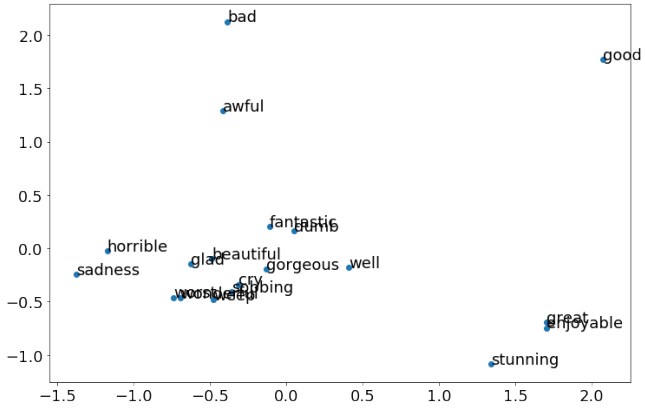

Figure 3. Word2Vec Embedding Before Distant-Supervised Training

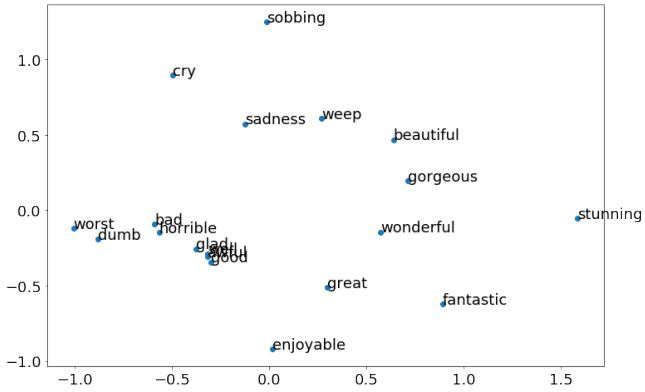

Figure 4. Word2Vec Embedding After Distant-Supervised Training

2.4. Supervised Training Phase

In this phase, the model, which has an updated Word2Vec embedding matrix, is trained with about 18K supervised data for 50 epochs, and the performance of the model is evaluated with another 20K annotated tweets after each epoch.

3. Result

The following graphs and tables show the results of our experiments.

Method	Acc. @ k (%)		F1 @ k (%)	
	20	25	20	25
Supervised-Only w/o Trainable Embedding	0.505	0.562	0.484	0.546
Supervised-Only w/ Trainable Embedding	0.595	0.595	0.598	0.596
Distant-Supervised	**0.616**	**0.628**	**0.602**	**0.618**

Table 2. Accuracy and Results for Different Models

Figure5 show that the testing loss is the lowest when the model is trained for about 21 epochs (which is epoch no. 20 because the training starts at epoch no. 0).We can also see that after 30 epochs, the loss value of testing data raises back up again, which is a sign of over-fitting.

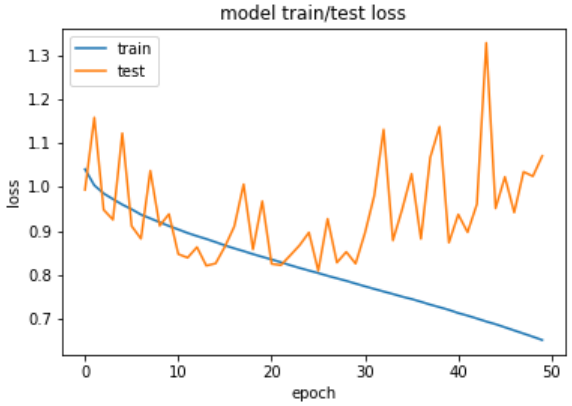

Figure 5. Loss vs. Epoch in Supervised Phase

The following figures are screenshots of the model outputs at epoch no. 20 and 25.

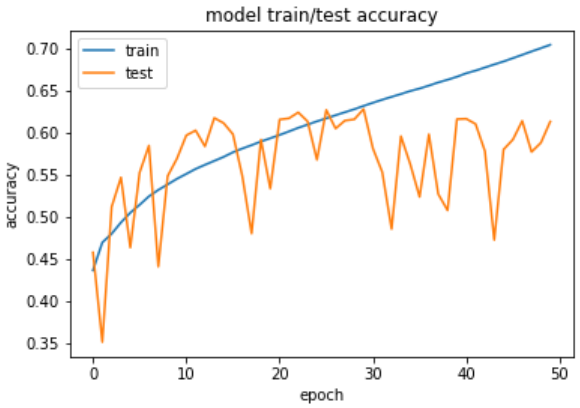

Figure 6. Accuracy vs. Epoch in Supervised Phase

```
Current epoch:  20
2017-11-24T17:12:40.103938: step 725, loss 0.838224, acc 0.623047, f1 0.581599
2017-11-24T17:12:41.299577: step 730, loss 0.664928, acc 0.720703, f1 0.719695
2017-11-24T17:12:42.509960: step 735, loss 0.673733, acc 0.720703, f1 0.715281
2017-11-24T17:12:43.721469: step 740, loss 0.623567, acc 0.751953, f1 0.746797
2017-11-24T17:12:44.924998: step 745, loss 0.787749, acc 0.595703, f1 0.543553
2017-11-24T17:12:46.151285: step 750, loss 0.727245, acc 0.65625, f1 0.640395
2017-11-24T17:12:47.347063: step 755, loss 0.828857, acc 0.664062, f1 0.609912

Evaluation:
2017-11-24T17:12:47.905245: step 756, loss 0.824516, acc 0.616372, f1 0.602908
```

Figure 7. Program Output at Epoch No. 20

```
Current epoch:  25
2017-11-24T17:13:25.228396: step 905, loss 0.699867, acc 0.664062, f1 0.651615
2017-11-24T17:13:26.397433: step 910, loss 0.594641, acc 0.755859, f1 0.756566
2017-11-24T17:13:27.596779: step 915, loss 0.726976, acc 0.626953, f1 0.593541
2017-11-24T17:13:28.808586: step 920, loss 0.640307, acc 0.755859, f1 0.759462
2017-11-24T17:13:30.011660: step 925, loss 0.675514, acc 0.695312, f1 0.662511
2017-11-24T17:13:31.229505: step 930, loss 0.580996, acc 0.753906, f1 0.751009
2017-11-24T17:13:32.452140: step 935, loss 0.788758, acc 0.675781, f1 0.6253

Evaluation:
2017-11-24T17:13:33.014883: step 936, loss 0.808649, acc 0.627762, f1 0.617908
```

Figure 8. Program Output at Epoch No. 25