Final Project Report

Wenxian Zhang, Maosen Zhang

May 6, 2021

1 Introduction

1.1 Problem statement

Out project is to extend a project of classifying texts to different types. The topic is identification of political framing based on tweet texts, tweet issues and tweet authors. We choose a baseline and do some extensions based on it. The baseline we choose is a simple neural network model and we will discuss possible extensions and our choice of extensions in the following sections.

1.2 Baseline

Our baseline is a simple neural network with the following structures:

1. Feature representation: Bag of Words for all features. For each sample $x^{(i)} \in X$,
   - Feature mapping that maps each sample into a vector with size of all existing words. Each element in sample vector represents frequency of this word in the matching sample.
   $$\phi(x^{(i)}) = \{(\phi(x^{(i)})_{tweet\_text}), \phi(x^{(i)})_{tweet\_issue}), \phi(x^{(i)})_{tweet\_author}))\} \tag{1}$$
   - For representation of labels, we just use natural number order. If there are K different types of labels, for each training sample with a label k:
   $$k \in \{0, 1, ..., K\} \tag{2}$$
2. Neural network:
   - Input layer
   - Hidden layer 1
   $$Z = W_1 * x + b_1 \tag{3}$$
   $$H = sigmoid(Z) \tag{4}$$
   - Hidden layer 2
   $$U = W_2 * H + b_2 \tag{5}$$
   - Output layer from softmax function.
   $$f(x; w) = F_{softmax}(U) \tag{6}$$
3. Hyper-parameter:
   - Splitting ratio of training data = 0.7,splitting ratio of validation data = 0.2,splitting ratio of test data = 0.1
   - learning rate = 0.001,weight decay rate = 0.001
   - Number of epochs = 50
   - Loss function: NLLLoss
   - Optimizer: Adam
4. Evaluation: training accuracy 99.77%, validation accuracy 44%, test accuracy 46%.

1.3 Possible extensions

For a NLP problem, it always consists of these steps: data pre-processing, building a suitable model and inference. We can do extensions based on them:

1. Data pre-processing: Bad-of-Words, word embedding, sentence embedding, pre-trained word2Vec, get more data, etc.
2. model: more complicated models based on neural network-CNN,RNN,etc.
3. inference: simple prediction, voted prediction, average prediction, etc.
4. Real world knowledge: we can have some basic knowledge about this topic and corresponding data corpus, which are useful for us to choose appropriate extensions.

2 Feature representation

2.1 Motivation

To convert texts into numerical representations, there are many ways to handle raw data. Different ways of representing words will emphasize different aspects of data. For example, Bag-of-Word will emphasize importance of frequency of each word in sentences, and word embedding will include level of similarity among words. This is part of data pre-processing extensions we state in the previous section. Also, based on our real world knowledge, we can easily think that similarity among words will also help identify political framing because similar words are enough to show the similar information. We do not need words to be totally the same. Therefore, trying to use word embedding may be a good way.

2.2 Algorithm

Instead of BoW, we use word embedding to represent texts, issues and authors in data corpus. The feature concatenation is still the same:

$$\phi(x^{(i)}) = \{(\phi(x^{(i)})_{tweet\_text}), \phi(x^{(i)})_{tweet\_issue}), \phi(x^{(i)})_{tweet\_author}))\} \tag{7}$$

But the feature function changed. For issues and authors, there is only one word in each data type. So, we can just match the word to corresponding embedding vector. However, for data type texts, there are different lengths of different sentences. Each data will have a sentence including number of words. If we just convert each word to matching embedding vector and concatenate them together, the dimension for each data will be different.

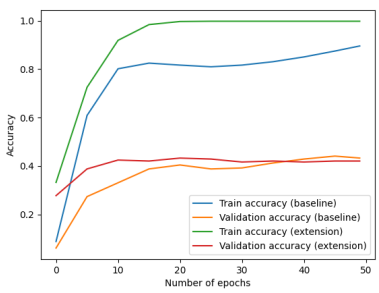
There are several ways to map list of words from a sentence to a vector and we implement all of them in our code [1].

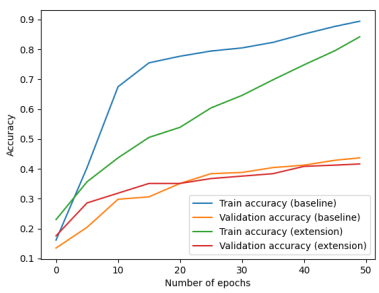
1. Sum: Convert all words into their matching embedding vectors, and add them together
2. Average: Convert all words into their matching embedding vectors, add them together and average them
3. Min: Convert all words into their matching embedding vectors, and take the (component-wise) minimum of them
4. Max: Convert all words into their matching embedding vectors, and take the (component-wise) maximum of them
5. Padding: Convert all words into their matching embedding vectors, and pad to get same dimension
6. Pre-trained word2vec: this comes from w2v.bin provided by homework2.

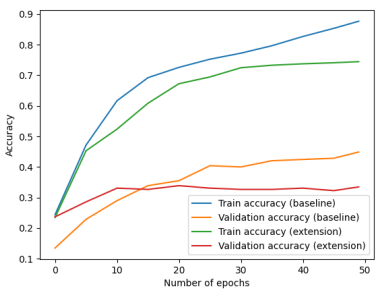
2.3 Experiment

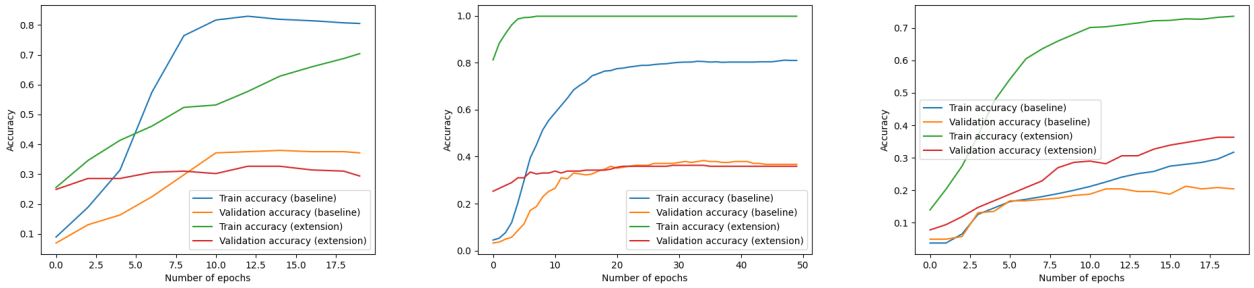
In order to compare performance of different word representations, we keep other variables unchanged and just modify pre-processing step. We run the same model and use the same inference method to BoW and word embedding with above 5 feature mapping ways, and plot accuracy of BoW versus accuracy of word embedding with different ways of feature mapping. What's more, we also directly compare test accuracy for these two methods of word representations.

Below are plots for comparison. The plots are comparisons for accuracy of using Bag-of-Word versus accuracy of using word embedding with feature by mapping sum/average/min/max/padding/pre-trained word2vec(By order). In order to save training time, we do not run many epochs for them to show the comparison because the plots have already showed the results.









The green and blue lines represent accuracy of training set and the red and orange lines represent accuracy of validation set. Performance of word embedding with feature mapping by average/min/max is not better than our baseline. But the left plots (sum,padding,pre-trained word2vec) show a better result: using word embedding during data pre-processing will significantly improve the performance and prediction accuracy will increase a lot. What's more, using word embedding representation will also decrease number of epochs needed.

For test accuracy, the highest accuracy comes from word embedding with feature mapping by summing all vectors. The accuracy value is 60.3%, which is also higher than test accuracy of our baseline.

3 Model optimization

3.1 Motivation

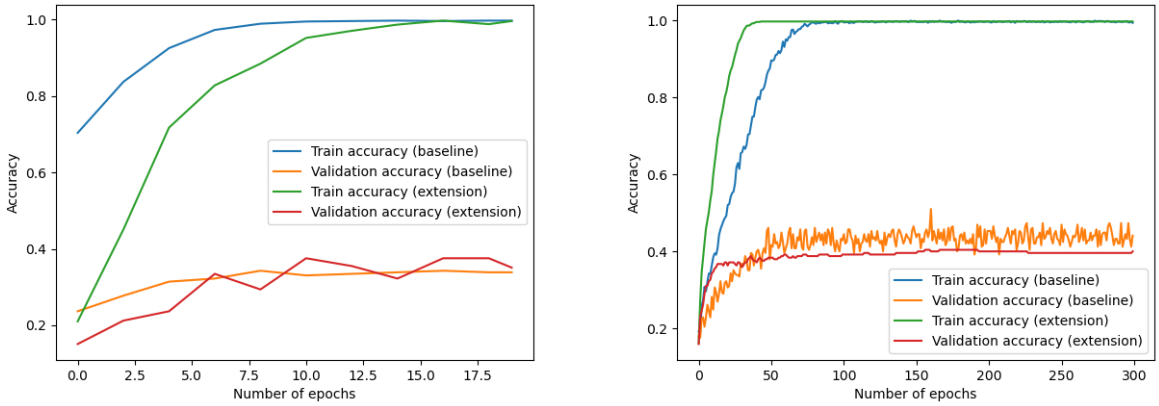
Model selection is an important part in ML and NLP problem. This aspect is also mentioned in the previous section as one option to extend. A simple model may omit some information in the data corpus and result in bad performance, or a improper model will learn in wrong way. Based on our real word knowledge, we know that if we want to classify a group of texts, meaning of words and the contexts are important. Some models will highlight importance of word frequency or word contexts. If we can catch this kind of features, performance may also be improved.

3.2 Algorithm

We choose convolutional neural network to replace original neural network structure because CNN can learn something from sequence order and context information. We add convolutional layer with kernel size list [3,4,5] and also add max-pooling and dropout layer to construct a new neural network structure.

3.3 Experiment

Here, we need to compare difference between our baseline (simple neural network structure with two hidden layer) and the new CNN model. Therefore, we keep other values of hyper-parameters unchanged and just use these two models to train same data to check their training, validation and test accuracy. Below is plot to show the training process:



We can see that CNN will perform better than a simple neural network with two hidden layers. The validation accuracy becomes higher, and it has less influence of over-fitting.

4 Leveraging unlabeled data

4.1 Motivation

We are provided with around 1200 labeled data and 30,000 unlabeled data from Github. The amount of labeled data is small but the amount of unlabeled data is large. What's more, we also split those labeled data into training set, validation set and test set with 70%, 20% and 10% of data corpus. Therefore, the amount of training samples is just about 800.

As we know, small amount of training data will lead to over-fitting and trained parameters will not be so generalized to other samples. If we can increase the amount of available labeled data, our model can learn more things from the data corpus.

4.2 Algorithm

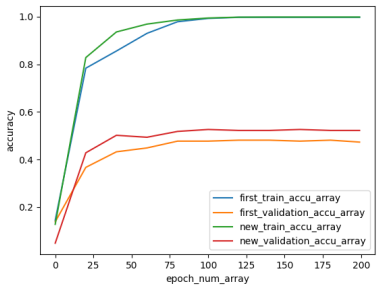
There are many ways suggested by different papers. Generally, after we train a model using original training data and use this model with trained weights to infer labels for those unlabeled data, we can use those inferred samples by two ways [2]:

1. Merge all unlabeled data into original labeled data with their inferred labels.
2. Pick part of unlabeled data with their inferred labels by some criteria and merge those selected samples into original labeled data-sets.

The first one will also result in serious effects of noise. We select the second way by picking top-k confident samples for different labels with adjustable k and confidence criteria to measure their priority [2, 3]. We firstly train the model with original labeled training samples, and then use this model with original trained weights to label all unlabeled data. Finally, we use top-k confidence criteria to pick some labels from larger corpus and merge them together to form a new data corpus. Based on this new data-set, we retrain the model and use this new model with new trained weights to infer labels of test set.

4.3 Experiment

To show performance, the basic concept is to compare difference of original and new trained weights. Without changing other hyper-parameters, we use the same model to train original labeled data and merged data, comparing training accuracy, validation accuracy by plot and comparing test accuracy by predicting results. Below is the plot with word embedding representation and neural network model. The figure is comparison.



As we can see, the accuracy of training set and validation set from new trained model both perform better than than from model with original trained weights.

5 Summary

We extend our baseline which is a simple neural network model with Bag-of-Word feature representation by three aspects: feature representation, more data and model selection. Specifically, for feature representation, we tried about six different ways to combine word embedding vectors. Also, we tried CNN model and semi-supervised learning. The highest accuracy is about 63.4%.

References

[1] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, 80:150–156, 2016.

[2] Zijun Sun, Chun Fan, Xiaofei Sun, Yuxian Meng, Fei Wu, and Jiwei Li. Neural semi-supervised learning for text classification under large-scale pretraining. *arXiv preprint arXiv:2011.08626*, 2020.

[3] I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.