CS287 HW1: Classification

Shayne O'Brien shayneob@mit.edu
David McClure dclure@mit.edu

February 1, 2018

1 Introduction

The objective of this problem set was to construct four models of text classifiers in Pytorch to classify the sentiment of input sentences from the Stanford Sentiment Treebank (SST-2) [5] dataset as positive or negative. In particular, we implement and successfully train a naive Bayes unigram classifier as described by Wang and Manning [6], a logistic regression model over word types, a continuous bag-of-word (CBOW) neural network with embeddings similar to that of Mikolov et al. [3], and a variant of a simple convolutional neural network (CNN) from Kim [2]. After replicating the models as described in the literature, we then experiment with modifications to the CBOW and CNN architectures that improve performance, and also propose a variation of the CNN architecture that significantly outperforms any of the other models.

2 Problem Description

In the Stanford Sentiment Treebank sentiment classification task, we are provided with a corpus of sentences taken from movie reviews. Each sentence has been tagged as either positive, negative, or neutral; we follow Kim [2] in removing the neutral examples and formulating the task as a binary decision between positive and negative sentences.

3 Model and Algorithms

For each model variant, we formalize prediction y for test case k as

$$y^{(k)} = \sigma(\mathbf{W} \cdot \mathbf{x}^{(k)} + b) \tag{1}$$

where σ is an activation function, $\mathbf{W}$ are our learned weights, $\mathbf{x}^{(k)}$ is our feature vector for input k, and b is a bias vector. Note that in this problem set, $y^{(k)} \in \{1, 2\}$ for all k since we only consider positive and negative sentiment inputs in the SST-2 dataset. We use Pytorch for all model implementations, and all models are trained for 10 epochs each using batches of size 10, a learning rate of 1e-4, the Adam optimizer, and the negative log likelihood loss function. The only exception to this setup was for multinomial naive Bayes, which was fit in one epoch with learning parameter $\alpha = 1.0$.

3.1 Multinomial Naive Bayes

Let $\mathbf{f}^{(i)} \in \mathbb{R}^{|V|}$ be the feature count vector for training case i with classification label $y^{(i)}$. V is the set of features, and $\mathbf{f}_j^{(i)}$ represents the number of occurrences of feature V_j in training case i. Define the count vectors of $\mathbf{f}^{(i)}$ as $\mathbf{p} = \alpha + \sum_{i:y^{(i)}=-1} \mathbf{f}^{(\mathbf{i})}$, for the smoothing parameter α. We follow Wang and Manning in binarizing the counts; $\mathbf{x}^{(\mathbf{k})} = \mathbf{1}\left\{\mathbf{f}^{(k)} > 0\right\}$. With regard to Equation (1), $\mathbf{W}$ is the log-count ratio between the number of positive and negative examples, $b = \log(N_+/N_-)$ where N_+ and N_- are the number of positive and negative training cases in the training dataset, $\mathbf{f}^{(k)}$ is the number of occurrences of input x_i, and $\mathbf{1}$ is a binary indicator function that maps $\mathbf{f}^{(k)}$ to 1 if greater than 0 and 0 otherwise. We consider only unigrams as features.

3.2 Logistic Regression

We learn weight and bias matrices $\mathbf{W}$ and b such to optimize

$$y = \sigma(\sum_i \mathbf{W} \cdot \mathbf{x}_i + b) \tag{2}$$

where x_i is $|V|$-dimensional vector representing bag-of-words unigram counts for each training sample. In our implementation, we represent $\mathbf{W}$ and b with a single fully-connected layer, which maps directly to a two unit output layer under sigmoid activation.

3.3 Continuous Bag-of-Words

In the CBOW architecture, each word in a sentence input of word-length n is mapped to a k-dimensional embedding vector. The embedding vectors for all words are averaged to produce a single feature vector $\mathbf{e}$ that represents the entire input. In particular,

$$\mathbf{e} = \frac{1}{|n|} \sum_{i=1}^{|n|} \mathbf{e}_i \tag{3}$$

where $\mathbf{e} \in \mathbf{R}^{k_i}$ and $\mathbf{e}_i \in \mathbf{R}^k$, and $k_i = k$ for all i are the dimensions of the sentence embedding and word embeddings, respectively. This encoding $\mathbf{e}$ is then passed into a single fully-connected layer that maps directly to two output units, representing output classes, under softmax activation.

3.4 Convolutional Neural Network

Let $\mathbf{x}_i \in \mathbb{R}^k$ be the k-dimensional word vector correspond to the i-th word in the input sentence. After padding all sentences in an input batch to the same length n, where n is the maximum length sentence of all sentences in the batch, each sentence is then represented as

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus ... \oplus \mathbf{x}_n \tag{4}$$

where $\oplus$ is the concatenation operation. Let $\mathbf{x}_{i:i+j}$ represent the concatenation of words $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, ..., $\mathbf{x}_{i+j}$. In Convolutional neural networks, we apply convolution operations $\mathbf{w} \in \mathbb{R}^{h,k}$ with filter size h to produce features, where the filter size is effectively the window size of words to convolve over. Let c_i be a feature generated by this operation. Then

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b) \tag{5}$$

where b is a bias term and f is the rectified linear unit (ReLU) function. Applying filter length size w over all possible windows of the words in our input sentence produces the feature map

$$\mathbf{c}_w = [c_1, c_2, ..., c_{n-h+1}] \tag{6}$$

In our implementation, we convolve over filter sizes $h \in \{3, 4, 5\}$ and then concatenate the features of each $\mathbf{c}_h$ into a single vector. We apply a max-over-time pooling operation (Collobert et al, 2011) to this vector of concatenated feature maps, denoted $\mathbf{c}$, and get $\hat{c}$ = max($\mathbf{c}$). We then apply dropout with p = 0.50 to $\hat{c}$ as regularization measure against overfitting, pass this into a fully-connected layer and compute the softmax over the output.

3.5 Modified CNN

Finally, we also implemented with a series of modifications to the CNN architecture to give a slight performance improvement on the SST-2 dataset. In this implementation, we utilize Stanford's GloVe pre-trained vectors [4], we make these changes:

- Following Kim [2], we use two copies of word embedding table during the convolution and max-pooling steps – one that is non-static, or updated during training as a regular module in the model, and another that is omitted from the optimizer and preserved as static throughout the training run. In the forward pass of the model, these two sets of embeddings are concatenated together along the "channel" dimension, and then passed into the three convolutional layers as a single tensor, with two values for each of the 300 dimensions in GloVe model.
- After producing the combined feature vector representing the max-pooled features from the three convolutional kernels, we simply add the non-padded word count of the input as a single extra dimension, producing a 301-dimension tensor which then gets mapped to the 2-unit output. From an engineering standpoint, we find that this marginally improves performance on the SST-2 dataset, where, on average, positive sentences are slightly longer than negative ones – 19.41 words versus 19.17. It's not clear whether this would hold across different data sets, or if it's specific to SST-2. (Though it's also not entirely clear that wouldn't, and seems to imply an interesting corpus-linguistic question – are "positive" sentences generally longer than "negative" ones?)

4 Experiments

In addition to the two changes described above, we also experimented with a wide range of other modifications to the CNN architecture, including:

- Combining the CBOW model with the CNN architecture by concatenating the maxpooled CNN vectors with the averaged CBOW vector before mapping to the final output units.
- Adding more linear layers between the convolutions and the final output units.

- Using larger feature maps in the convolutional layers ([200-500], instead of 100.)
- A series of hand-engineered features that tried to capture character-level, syntactical, or corpus-linguistic information that might get missed by the embeddings – the average length of words in the sentence; the average frequency of the words (queried via the `wordfreq` Python package); the relative offset inside the sentence of the root verb in the dependency tree; whether or not the dependency tree includes a negated root verb.
- Static and non-static channels of pre-trained as well as randomly initialized word embeddings, and all combinations of these varieties (e.g. static randomized, non-static randomized, static pre-trained) via multichannel word embeddings.

Although many of these affected the speed at which the model fit the training data, none of them improved the classification accuracy on the validation set – the best performance came from just the two changes described in section 3.5. Below, we report the classification accuracies achieved by each of the models outlaid in Section 3 on the test set:

Model	Acc.
LOGISTIC REGRESSION	78.03
UNIGRAM NAIVE BAYES	82.48
CBOW	83.75
CNN	85.34
MODIFIED CNN	85.94

Table 1: Classification accuracy on the SST-2 dataset.

Though the modified CNN was the best-performing model, it only beats the vanilla CNN by about half a percent, even though the multi-channel embedding architectures takes about twice as long to train. It's also notable how competitive the unigram naive Bayes baseline is, especially given that it fits in about two seconds on a 2017 MacBook Pro, compared to 20-30 minutes for the CNN architectures. We suspect that adding bigrams and trigrams as features to the Naive Bayes would make it an even more competitive baseline than simply using unigrams as features, as per Wang and Manning (2012).

5 Conclusion

We trained 5 classes of models – logistic regression and naive Bayes baselines, a simple CBOW architecture, a replication of Kim's CNN architecture, and a modified CNN that makes a handful of small tweaks that improve performance on the SST-2 data. Though our results are competitive with reported scores in the literature, we were unable to match the performance reported by Kim (88.1%), using the architecture that we followed for the CNN.

References

[1] R. Collobert, J. Weston, L. Bottou, M. Karlen, P. Kavukcuoglu. "Natural Language Processing (almost) from Scratch." Journal of Machine Learning Research 12 (2011), pages 2493-2537.

[2] Y. Kim. "Convolutional Neural Networks for Sentence Classification." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 17461751, October 25-29, 2014, Doha, Qatar.

[3] T. Mikolov, K. Chen, G. Corrado, J. Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781, January 1, 2013.

[4] J. Pennington, R. Socher, C. Manning. 2014. "GloVe: Global Vectors for Word Representation."

[5] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, C. Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Proceedings of EMNLP 2013.

[6] S. Wang, C. Manning. "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 9094, Jeju, Republic of Korea, 8-14 July 2012.