CS388 Mini 2: Sentiment Analysis with Neural Networks

Kelsey Ball
The University of Texas at Austin
kelseytaylorball@gmail.com

1 Introduction

Sentiment analysis is the task of labeling a text as either *positive* or *negative* in terms of emotional valence. In this project, we implement two neural networks for sentiment analysis: a feed-forward neural network and a recurrent neural network. We discuss trade-offs in speed, model complexity, and performance.

1.1 Data

We evaluate on a dataset of movie reviews from Rotten Tomatoes. Below are some example reviews from the dataset with their labels:

Positive:

Good fun, good action, good acting, good dialogue, good pace, good cinematography.

Negative:

This isn't a new idea.

2 Feed-Forward Neural Network

A Deep Averaging Network (DAN) a la Iyyer et. al (2015) is a multi-layer, feed-forward neural network that represents a sentence as the average of its word embeddings, then learns a deep neural network for classification of the sentence embedding.

2.1 Hyperparameters

Our basic DAN has two hidden layers of size 100 with ReLU activation and 50-dim fixed gloVe embeddings. We train our network for 10 epochs using Adam with a step size of $1e-3$ and a batch size of 1. In Table 1, we show the performance of this model using two methods of aggregating word embeddings into a sentence embedding: `FFNN-avg` is a DAN which averages the word embeddings in a sentence, while `FFNN-sum` sums the embeddings. We also compare these two implementations against a trivial, non-neural baseline which predicts the label *positive* for all examples.

Model	Accuracy	Train time	Inf. time
Baseline	50.92	0.00s	0.01s
FFNN-avg	73.62	44.13s	1.31s
FFNN-sum	74.54	44.48s	1.30s

Table 1: Baseline vs. FFNN.

Both averaging and summing seem to work similarly well; the difference in accuracy between the two methods is smaller than the variance in accuracy in both methods across several trials.

2.2 Embeddings

In Table 2, we compare different approaches to learning embeddings in the DAN model: `frozen` refers to pre-trained gloVe vectors whose values are not updated during training; `fine-tuned` refers to pre-trained gloVe vectors which *a*re updated during training; and `from scratch` refers to randomly initializing and learning embeddings for each word in the training vocabulary. Our best model uses 300-dim, fine-tuned gloVe vectors.

Model	Accuracy	Train time
50-dim, frozen	73.62	53.54s
300-dim, frozen	78.78	53.39s
50-dim, fine-tuned	80.05	829.32s
300-dim, fine-tuned	82.00	1928.45s
50-dim, from scratch	78.21	825.27s
300-dim, from scratch	77.06	2010.96s

Table 2: Different embedding approaches.

3 Recurrent Neural Network

We also implement a sentiment classifier using a recurrent neural network. Our model is a bidirectional, 1-layer LSTM with hidden dimension

100, where the input to the LSTM at each timestep is the embedding for each word in the sequence. Our best model uses fine-tuned, 300-dim gloVe vectors; however, we also experiment with freezing the embeddings as well as learning them from scratch. Our model is trained with Adam with a step size of $1e-3$ and and batch size of 1 for 10 epochs. We also implemented batched gradient descent, but the implementation likely has a bug given that the accuracy dropped to that of a random baseline for many different batch sizes.

3.1 Embeddings

Here, we again compare different approaches to learning embeddings. For all the experiments described in Table 3, we used a bidirectional, 1-layer LSTM.

Emb dim/update	Accuracy	Train time
50-dim, frozen	76.72	703.45s
300-dim, frozen	80.96	777.95s
50-dim, fine-tuned	81.54	1004.76s
300-dim, fine-tuned	82.34	3132.66s
50-dim, from scratch	76.69	993.39s
300-dim, from scratch	78.82	3243.28s

Table 3: Embedding dim and learning strategy vs. accuracy, train time.

Fine-tuning the 50-dim embeddings resulted in a 6.28% relative increase in accuracy, but a 42.67% relative increase in training time. This tradeoff is even steeper for the 300-dim embeddings, where the relative increase in accuracy is smaller than for the 50-dim embeddings, yet the training time more than triples.

3.2 Layers

In Table 4, we compare unidirectional vs. bidirectional LSTM's with 1 vs. 2 layers. For these experiments, we used fixed, 50-dim gloVe embeddings.

Num layers/dirs	Accuracy	Train time
1-layer, uni-dir	79.13	389.40s
2-layer, uni-dir	79.35	869.12s
1-layer, bi-dir	78.67	721.14s
2-layer, bi-dir	76.95	1470.34

Table 4: Num layers, directionality vs. accuracy, train time.

Surprisingly, accuracy does not increase significantly when adding directions/layers; however, the training time more than doubles.

4 Discussion

Despite having far fewer parameters, our DAN models achieve the same accuracy as LSTM's with the same hidden size and embeddings. Furthermore, it takes more than 10 times longer to train the smallest LSTM relative to the equivalent DAN. As such, DAN's or other simple feed-forward networks ought to be strongly considered for sentence-level classification before using larger and more complex RNN's.

References

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daume III. 2015. Deep Unordered Composition ´ Rivals Syntactic Methods for Text Classification. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL).