Bag-of-words sentiment classification

(DS-GA 1011, assignment 1)

Melanie Tosik (`mt3685`)

October 2018

1 Overview

For the PyTorch implementation, please see: `https://github.com/melanietosik/bow-sentiment-classifier`.

In the ablation study, we evaluate the following hyperparameters, mostly in order:

- Tokenization scheme
  0. Baseline, `string.split()`
  1. Tokenization using `spaCy`
  2. Tokenization using `spaCy`, filtering of stop words and punctuation
  3. Tokenization using `spaCy`, filtering of stop words and punctuation, lemmatization
- Learning rate (`torch.optim.adam`): `1e-2`, `1e-3`, `1e-4`
- N-gram and vocabulary size
  - N-gram size (inclusive): `1, 2, 3, 4`
  - Vocabulary size: `10,000, 50,000, 100,000`
- Embedding size: `50`, `100`, `200`
- Optimizer: `torch.optim.adam`, `torch.optim.sgd`
- Linear annealing of learning rate: `true`, `false`
- Number of epochs: `2, 5, 10`

Overall, the best model is trained using the following set of hyperparameters:

⋆ Tokenization using `spaCy`, filtering of stop words and punctuation

⋆ Learning rate (`torch.optim.adam`): `1e-3`

⋆ N-gram and vocabulary size: `1` and `50,000`

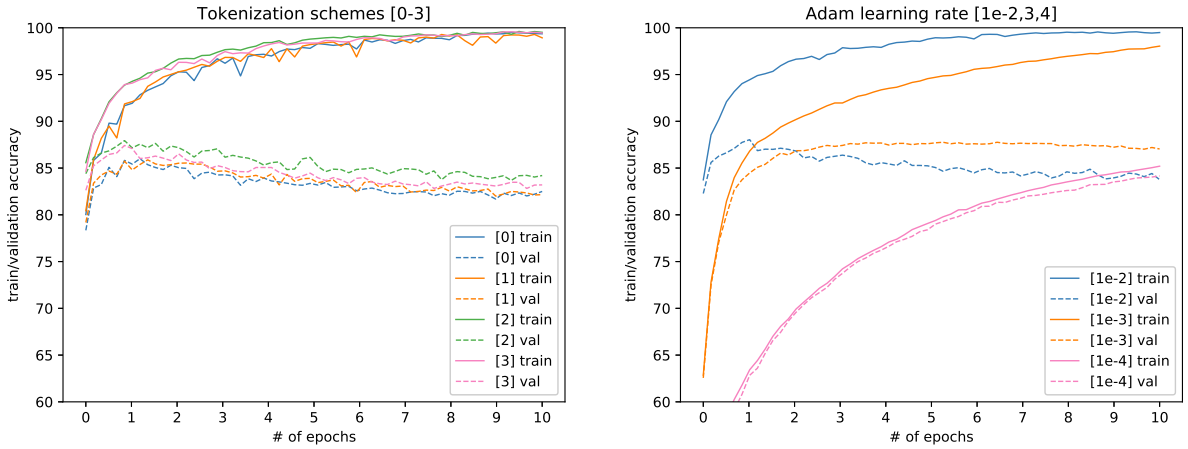
⋆ Embedding size: `200`

⋆ Optimizer: `torch.optim.adam`

⋆ Linear annealing of learning rate: `false`

⋆ Number of epochs: `2`

The accuracy on the testing for the final trial is `86.212` (cf. `trial.log`).



(a) Tokenization scheme

(b) Learning rate

Figure 1: Tokenization schemes and learning rate

2 Results

Given the page limit, we can only briefly discuss the most relevant results. Please refer to the `results/` and `plots/` folders in the GitHub repository for the full set of experiments.

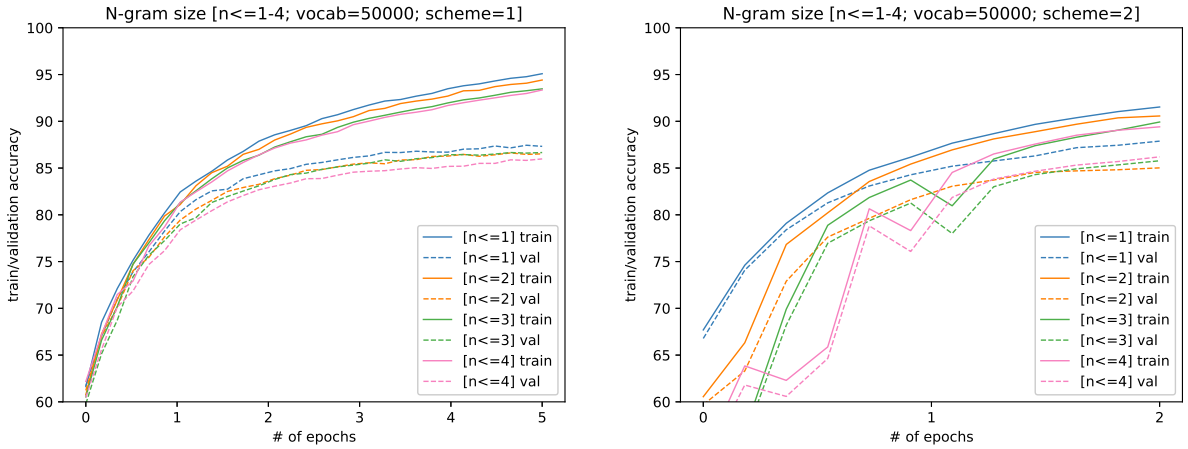
We start by evaluating four increasingly rigorous **tokenization schemes**. The remaining hyperparameters are fixed to the hyperparameter set from the previous lab. Figure 1 (a) illustrates the results. Tokenization using `spaCy` and filtering of stop words and punctuation performs best (for unigrams). The plot also indicates that our model is overfitting to the training data.

Since the model is overfitting, we check the **learning rate** next. The current optimizer is `torch.optim.adam`, with a learning rate of `1e-2`. Figure 1 (b) shows the results for three different experimental learning rates. The PyTorch default learning rate of `1e-3` results in a nice learning curve, whereas a rate of `1e-2` indicates overfitting, and `1e-4` indicates that the learning rate is too small. The optimal learning rate probably lies somewhere between `1e-3` and `1e-4`.

Based on the plots in Figure 1, it seems that we can also reduce the **number of epochs** to prevent overfitting and speed up model training. For the upcoming experiments we reduce the number of epochs from `10` to `2`.

Next, we start experimenting with the **n-gram and vocabulary size**. Previously, we used unigrams and a maximum vocabulary size of `10,000` tokens. The first experiment (not listed) indicates that unigrams outperform larger n-grams for a vocabulary size of `10,000`. Note that n-grams of size n also include all possible n-grams of size $n-1, n-2, \ldots$; e.g. trigrams also include bigrams and unigrams. Since n-grams larger than unigrams are unlikely to perform well on such a small vocabulary, we increase the vocabulary size to `50,000`. We also re-evaluate a less rigorous tokenization scheme, because *including* stop words might be beneficial for n-gram document classification. The results for n-grams of size `1-4`, a vocabulary size of `50,000`, and tokenization schemes `1` and `2` are illustrated in Figure 2. The plots show that even when increasing the vocabulary size and relaxing the preprocessing scheme, the unigram model still performs best on the validation set. While the learning curves in Figure 2 (a) are little more smooth, the highest validation accuracy is achieved by the unigram model on the second tokenization scheme, as shown in Figure 2 (b). Increasing the vocabulary size to `100,000` did not yield significant improvements for the validation accuracy. For the exact scores and additional plots, please refer to the output of the `plot.py` script.

To recap, our current best model is using unigram data processed with tokenization scheme `2`, a maximum vocabulary size of `50,000`, the Adam optimizer with a learning rate of `1e-3`, and is trained for `2` epochs.



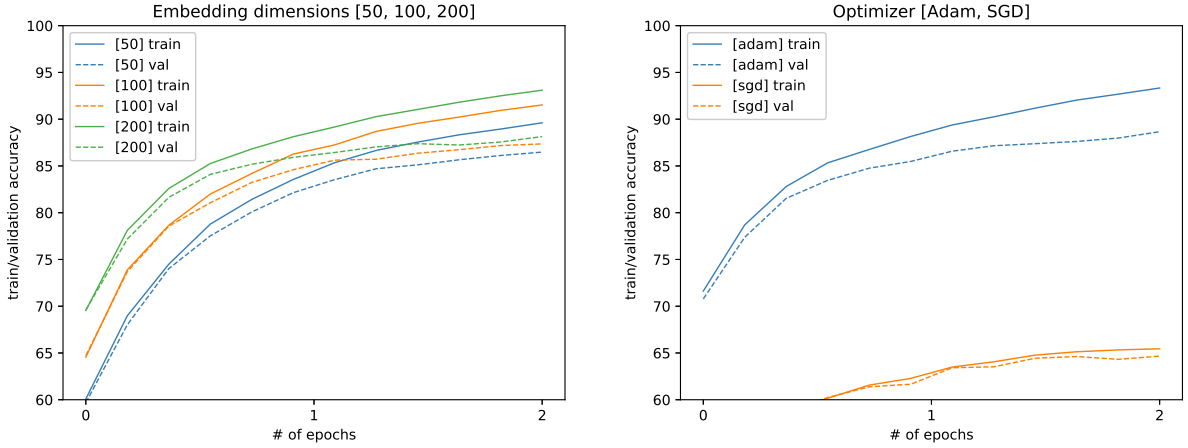
(a) N-gram size and relaxed tokenization scheme

(b) N-gram size and rigorous tokenization scheme

Figure 2: N-gram size and vocabulary size

Next, we can optimize the **size of the word embeddings**. We test embedding sizes of `50, 100, 200`. Figure 3 (a) shows that setting the size of the embeddings to `200` dimensions yields best results. Since the latest learning curves have not quite converged yet, we will run the experiment again with a larger number of epochs at the end. Figure 3 (b) shows the training and validation accuracies for two different **optimizers**, namely `torch.optim.adam` and `torch.optim.sgd`. Both optimizers are initiated with their corresponding PyTorch default parameters. Adam clearly outperforms the SGD optimizer in the early training and validation stages. In general, SGD would probably require a much higher number of epochs in order to converge. In the interest of time, we will continue our study with the Adam optimizer and a learning rate of `1e-3`.

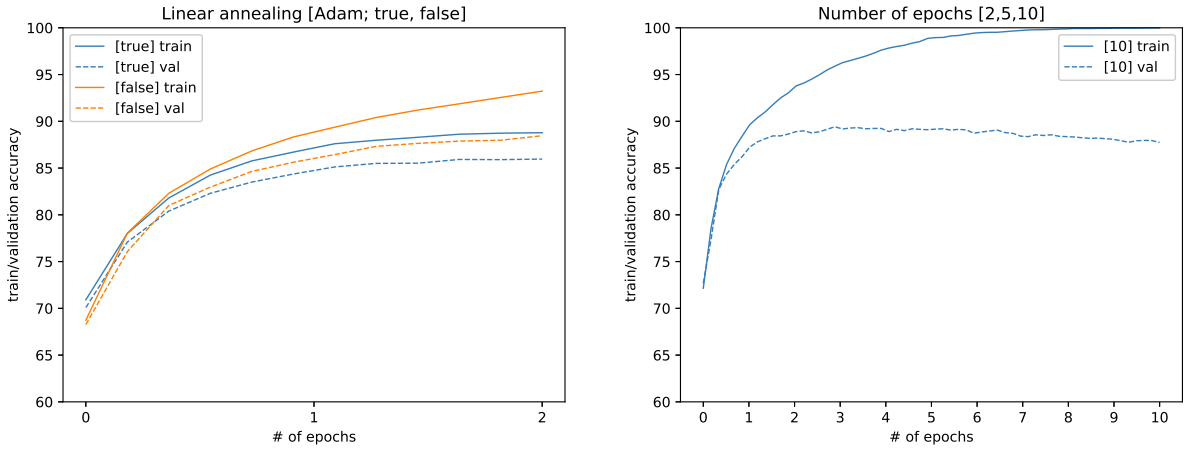
For completeness, we also tested **linear annealing of the learning rate**, i.e. linearly reducing the learning rate over the course of the network training phase. The results are shown in Figure 4 (a). Linear annealing of the learning rate did not improve validation accuracy. Finally, we run one last trial with the **number of epochs** set to `10`. Figure 4 (b) shows the results. From the plot, we conclude that a smaller number of epochs is preferable to prevent the model from overfitting to the training data. In the future, we could properly implement early stopping based on the prediction accuracy on the validation set.



(a) Embedding size

(b) Optimizer

Figure 3: Embedding size and optimizer



(a) Linear annealing of learning rate

(b) Number of epochs

Figure 4: Linear annealing of learning rate and number of epochs

Since there is no room for additional tables, for the exact/best accurracies for each experiment, please refer to the outputs (commented inline) of `plot.py`. The output of the `main.py` script for the final trial (incl. the evaluation on the testing set) is provided in `trial.log`. The final accuracy on the testing set is `86.212`.

Finally, we can use the `id2token` mapping to reconstruct validation samples for which our best model makes correct or incorrect predictions. Below are listed one correct and one incorrect example. Please see `trial.log` for the full set of (preprocessed) examples.

> `[true: pos, predicted: pos]`
> This is the Neil Simon piece of work that got a lot of praises! "The Odd Couple" is a one of a kind gem that lingers within. You got Felix Ungar(Jack Lemmon); a hypochondriac, fussy neat-freak, and a big thorn in the side of his roommate, Oscar Madison(Walter Matthau); a total slob. These men have great jobs though. Felix is a news writer, and Oscar is a sports writer. Both of these men are divorced, Felix's wife is nearby, while Oscar's is on the other side of the U.S. (The West Coast). Well, what can you say? Two men living in one roof together without driving each other crazy, is impossible as well as improbable. It's a whole lot of laughs and a whole lot of fun. I liked the part where when those two British neighbors that speak to both gentlemen, and after Oscar kicked out Felix, he gets lucky and lives with them when he refused to have dinner with them the night earlier. It's about time that Felix needed to lighten up. I guess all neat-freaks neat to lighten up. They can be fussy, yet they should be patient as well. A very fun movie, and a nuevo classic. Neil Simon's "The Odd Couple" is a must see classic movie. 5 STARS!

> `[true: neg, predicted: pos]`
> This is not a good movie. Too preachy in parts and the story line was sub par. The 3D was OK, but not superb. I almost fell asleep in this movie. The story is about 3 young flies that want to have adventure and follow up on it. The characters are lacking, I truly do not care about these characters and feel that there was nothing to keep an adult interested. Pixar this is not. I would have liked to see more special 3D effects. Also I wold like to see more fly jokes than the mom constantly saying "Lord of the flies" Pretty sexist in showing the women as house wives and fainting.

The first (correct) sample actually seems relatively difficult to classify, given how many negative words the story line contains (e.g. *fussy*, *thorn*, *slob*, *crazy*, *impossible*, *improbable*, *refused*). The second (incorrect) sample on the other hand is a great example of where n-grams could be helpful, e.g. in classifying *not a good movie*, *see more jokes/special effects*, *not superb*. It would be interesting to analyze actual data samples in more depth to see if there are similar patters that could help motivate the overall model architecture.