Your Name : Arnav Singla

Your Email address : arnavs21@iitk.ac.in

GitHub repo for all the code and plots : github

1. Training Set Construction (5 pts)

Construct the training set for the amazon review dataset as instructed and report the following statistics.

Statistics	
the total number of unique words in T	21701
the total number of training examples in T	1985
the ratio of positive examples to negative examples in T	999/986
the average length of document in T	73
the max length of document in T	1554

2. Performance of deep neural network for classification (20 pts)

Suggested hyperparameters:

1. Data processing:
    a. Word embedding dimension: 100
    b. Word Index: keep the most frequent 10k words
2. CNN
    a. Network: Word embedding lookup layer -> 1D CNN layer -> fully connected layer -> output prediction
    b. Number of filters: 100
    c. Filter length: 3
    d. CNN Activation: Relu
    e. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)
3. RNN:
    a. Network: Word embedding lookup layer -> LSTM layer -> fully connected layer(on the hidden state of the last LSTM cell) -> output prediction
    b. Hidden dimension for LSTM cell: 100
    c. Activation for LSTM cell: tanh
    d. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)

	Accuracy	Training time(in seconds)
RNN w/o pretrained embedding	0.62	13.31
RNN w/ pretrained embedding	0.81	10.02
CNN w/o pretrained embedding	0.68	8.18
CNN w/ pretrained embedding	0.82	37.19

3. Training behavior (10 pts)

Plot the training/testing objective, training/testing accuracy over time for the 4 model combinations (correspond to 4 rows in the above table). In other word, there should be 2*4=8 graphs in total, each of which contains two curves (training and testing).

RNN w/o pretrained embedding

- training/testing objective over time
- training/testing accuracy over time

RNN w/ pretrained embedding

- training/testing objective over time
- training/testing accuracy over time

CNN w/o pretrained embedding

- training/testing objective over time
- training/testing accuracy over time

CNN w/ pretrained embedding

- training/testing objective over time
- training/testing accuracy over time

4. Analysis of results (10 pts)

Discuss the complete set of experimental results, comparing the algorithms to each other. Discuss your observations about the various algorithms, i.e., differences in how they performed, different parameters, what worked well and didn't, patterns/trends you observed across the set of experiments, etc. Try to explain why certain algorithms or approaches behaved the way they did.

Four experiments were conducted involving two distinct models: CNN and LSTM, both with and without the utilization of pre-trained embeddings. Notably, the models demonstrated commendable performance when pre-trained embeddings were employed. Conversely, their efficacy diminished considerably in the absence of pre-trained embeddings.

Multiple regularization techniques, including weight decay, various optimizers, and dropout, were rigorously explored. However, the outcomes remained relatively consistent across different experimental settings.

The absence of pre-trained embeddings rendered both models highly susceptible to overfitting, primarily due to the limited size of the training dataset. Consequently, the model performance exhibited considerable fluctuations, hindered by rapid memorization of noise in the data, thus impeding generalization.

When pre-trained embeddings were employed, both CNN and LSTM models exhibited comparable performance. CNNs excelled in scenarios characterized by local document structures, whereas LSTMs showcased superior performance in instances with extensive long range dependencies. Notably, CNNs necessitated more training epochs compared to LSTM models.

Systematic parameter tuning encompassing batch size, learning rate, embedding dimension, dropout rates, and output dimensions of linear layers revealed that both models performed optimally with nearly identical parameter configurations. In the absence of pre-trained embeddings, LSTM models exhibited subpar performance. Their complexity demanded a substantial volume of data for effective generalization, which was unattainable given the dataset's modest size. Even regularization techniques like weight decay and dropout failed to significantly enhance test accuracy.

Conversely, CNNs, under analogous experimental conditions, delivered relatively modest but respectable test accuracy. CNNs exhibited an ability to generalize effectively without an extensive training dataset.

A consistent observation across all experiments was the rapid convergence of training accuracy to unity. Consequently, a reduced learning rate was necessitated to ensure smoother convergence and to prevent overfitting, with careful consideration of an optimal stopping point in the training process.

5. The software implementation (5 pts)

Add detailed descriptions about software implementation & data preprocessing, including:

1. **A description of what you did to preprocess the dataset to make your implementations easier or more efficient.**

The initial preprocessing phase involved several essential steps for sentiment classification:

- Conversion to lowercase: All text was converted to lowercase.
- Stopword and non-alphabetical character removal: Stopwords and non-alphabetical characters were removed.
- SOS Token Introduction: The Start-of-Sequence (SOS) token was added at the beginning of each sentence.
- Padding: Sentences were uniformly padded to a specified maximum length, facilitating efficient model training and testing. Out-of-vocabulary words were omitted during tokenization.

To enhance training efficiency, the raw text was then transformed into the necessary input and output NumPy arrays. These arrays were saved in advance of model input, streamlining and expediting the training process.

2. **A description of major data structures (if any); any programming tools or libraries that you used;**

Utilized Libraries: PyTorch (for models), NumPy, pickle, JSON, Matplotlib, spaCy

All the models were trained on mps (m1 metal device) using PyTorch

The initial step involved the storage of the training dataset in a .json format, ensuring seamless accessibility.

When employing pre-trained embeddings:

The pre-processed text underwent tokenization using spaCy's tokenizer. Subsequently, pre-trained embeddings were loaded into a dictionary, establishing a mapping from words to their corresponding embedding vectors (in list form). These embeddings were utilized to transform the lists of tokens for each sentence into two-dimensional lists. This process was independently applied to both the training and testing data, resulting in the creation of a three-dimensional NumPy array. The resulting array was then saved using the pickle library.

In the absence of pre-trained embeddings:

A vocabulary was constructed from the entire training dataset. From this vocabulary, the top 10,000 most frequently occurring words (excluding stop words) were selected to form a dictionary. Two dictionaries were generated: one mapping words to indices and the other mapping indices back to words. These dictionaries facilitated the conversion of token lists

into lists of corresponding indices for both the training and testing data. The resulting data was stored in two-dimensional NumPy arrays and subsequently saved using the pickle library.

3. Strengths and weaknesses of your design, and any problems that your system encountered;

Algorithm Drawback:
One limitation of the algorithm is the omission of out-of-vocabulary words, potentially resulting in the loss of crucial information essential for sentiment classification.
Batch Training:
Training was conducted in batches, enhancing model resilience against random noise and promoting smoother convergence.
CNN (Convolutional Neural Networks):

- CNNs excel at capturing local patterns and features in input data, making them effective, particularly with limited training data.
- They may struggle with long-range dependencies and sequential information in text data.

LSTM (Long Short-Term Memory):

- LSTMs are adept at capturing long-range dependencies and sequential context in text data.
- Training LSTMs is quite challenging due to vanishing and exploding gradient issues; gradient clipping was applied to mitigate noise.
- LSTMs require substantial training data for optimal generalization, which can be limiting with a smaller dataset.
- LSTMs, when utilized without pre-trained embeddings, exhibit a pronounced tendency towards overfitting the training data, primarily attributable to the limited dataset size. Consequently, their capacity to generalize effectively to the test data is notably constrained