# Recurrent Convolutional Neural Networks for Semantic Classification

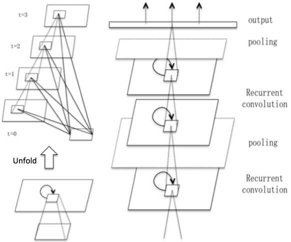
Liu, Chen

Department of Computer Science and Technology
Tsinghua University

Mentor： Hu, Xiaolin

# Content

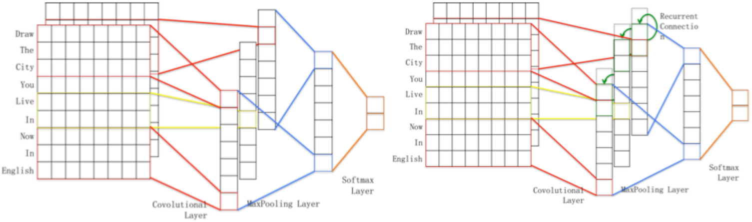
- Convolutional neural network has achieved remarkable results in tasks such as image recognition.
- Add recurrent connection, we can extract more hierarchical features.
  - Intuition: A recurrent convolutional layer is equivalent to a mini deep CNN with shared parameter.
  - Formula: input: $\mathbf{X}$, input filters: $\mathbf{F}_{in}$, recurrent state at time $t$: $\mathbf{S}_t$, recurrent filters: $\mathbf{F}_r$; at time $t$: $\mathbf{S}_t = \sigma(\mathbf{X} \otimes \mathbf{F}_{in} + \mathbf{S}_{t-1} \otimes \mathbf{F}_r)$
  - Loop times=iterations in recurrent states before outputing the result to next layer.



# Model

## Baseline & Proposed Model

- Our problem is to do semantic classification of a given sentence.
- Thanks to word2vec method, we can embed each word into a vector space. Therefore, we can use a matrix to denote a sentence and do convolutional operation on that.
- Each dimension of word vector is independent (no local features), so the convolution operators should align with the input in column.
- Baseline CNN v.s Our Proposed RCNN



- Movie Review
  - 2-class problem (positive/negative). 5331 sentences each(blanced).
  - Traing:Validation:Test sets=8:1:1
  - Cross validation and take the average result.
- Stanford Sentiment Treebank(SST)
  - 5-class problem (very positive/positive/neutral/negative/very negative). 11855 sentences in total.
  - Predefined training:validation:test sets=7:1:2.
- Implemented mainly by theano package
  - CNN module: single-layer CNN
  - DCNN module: multi-layer CNN
  - RCNN module: single-layer RCNN
  - DRCNN module: multi-layer RCNN
  - Other modules: save models, reload, visualization.
- Source code is maintained at Github （https://github.com/liuchen11/RCNNSentence）

## 1-layer CNN/RCNN's performance on Movie Review

| Name | Type | Dropout | Loops | Precision |
|---|---|---|---|---|
| BaselineCNN | CNN | / | / | 80.80% |
| DropoutCNN | CNN | 0.5 | / | 80.46% |
| RCNN1 | RCNN | / | 1 | 75.07% |
| RCNN2 | RCNN | / | 2 | 77.02% |
| RCNN3 | RCNN | / | 3 | 79.91% |
| RCNN5 | RCNN | / | 5 | 79.89% |
| DropoutRCNN1 | RCNN | 0.5 | 1 | 77.83% |
| DropoutRCNN2 | RCNN | 0.5 | 2 | 77.01% |
| DropoutRCNN3 | RCNN | 0.5 | 3 | 79.36% |
| DropoutRCNN5 | RCNN | 0.5 | 5 | 79.61% |

Table: Single-layer CNN and RCNN's performance on Movie Reiew dataset. We have 3 different widths of filters=[3,4,5]. Number of feature maps is 100 for CNN and 80 for RCNN such that they have approximately the same number of parameters (about 450000). Each hyper-parameter setting is run for 5 times and take the average result.(Following experiments are using the same method.) RCNN performances slightly poorer than CNN.

## 1-layer CNN/RCNN's performance on SST

| Name | Type | Dropout | Loops | Precision |
|---|---|---|---|---|
| BaselineCNN | CNN | / | / | 46.64% |
| DropoutCNN | CNN | 0.5 | / | 47.37% |
| RCNN1 | RCNN | / | 1 | 44.52% |
| RCNN2 | RCNN | / | 2 | 45.61% |
| RCNN3 | RCNN | / | 3 | 45.88% |
| RCNN5 | RCNN | / | 5 | 47.87% |
| DropoutRCNN1 | RCNN | 0.5 | 1 | 46.29% |
| DropoutRCNN2 | RCNN | 0.5 | 2 | 46.87% |
| DropoutRCNN3 | RCNN | 0.5 | 3 | 46.42% |
| DropoutRCNN5 | RCNN | 0.5 | 5 | 47.46% |

Table: Same hyper-parameter settings as models on Movie Review. CNN and RCNN have almost the same performance on SST.

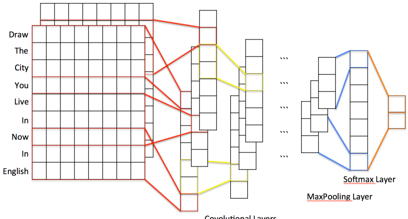
| Name | 1LCNN | 2LCNN | 3LCNN | 4LCNN |
|---|---|---|---|---|
| Layer0 | 3*300*1*300 | 3*300*1*200 | 3*300*1*250 | 3*300*1*128 |
| Pool0-1 | / | 2*1 | 1*1 | 1*1 |
| Layer1 | / | 5*1*200*250 | 3*1*250*150 | 3*1*128*128 |
| Pool1-2 | / | / | 1*1 | 1*1 |
| Layer2 | / | / | 5*1*150*150 | 3*1*128*128 |
| Pool2-3 | / | / | / | 2*1 |
| Layer3 | / | / | / | 3*1*128*256 |
| Global Pooling | | | | |
| Dropout | / | / | / | / |
| #Parameters | 270k | 430k | 450k | 312k |
| Precision | 80.2% | 79.22% | 78.24% | 79.23% |

Table: For each convolutional layer, A*B*C*D means filter's height=A, filter's width=B, #input feature maps=C and #output feature maps=D. We can see that deep models do not work in this context.

# Results&Analysis
## Multi-layer Models on SST

| Name | 1LCNN | 2LCNN | 3LCNN |
|---|---|---|---|
| Layer0 | 3*300*1*512 | 3*300*1*256 | 3*300*1*256 |
| Pool0-1 | / | 2*1 | 2*1 |
| Layer1 | / | 5*1*256*128 | 3*1*256*128 |
| Pool1-2 | / | / | 1*1 |
| Layer2 | / | / | 5*1*128*128 |
| Dropout | / | / | / |
| #Parameters | 460k | 400k | 480k |
| Precision | 47.01% | 45.15% | 43.61% |

Table: Like on the dataset 'Movie Review', deep models even perform worse on SST.



- Until now, we use the 300-dim word vectors trained from 'GoogleNews' corpus(>10 Billion), to generate the input sentence matrix. For vectors appearing in our corpus but not in 'GoogleNews', we use radom vectors. In this experiment, we will look into the impact of input word vectors. We use word vectors generated from the following methods respectively.
  - 'GoogleNews': same method above.
  - 'Vec': We pick up about 100k sentences randomly from the internet (much smaller than GoogleNews) to train a sets of word vectors. It has 50-dim(Vec50), 100-dim(Vec100) and 300-dim(Vec300) versions.
  - 'Random': Randomly assign a vector to a word. It also has 50-dim(Random50), 100-dim(Random100) and 300-dim(Random300) versions.
- From the results in next slide, we can see that unsupervised corpus used to train word vectors is very essential to improve model's performance(higher precision and lower variance).

| Vectors | Words don't appear | Precision | Variance($10^{-5}$) |
|---|---|---|---|
| GoogleNews | 13% | 47.26% | 2.42 |
| Vec300 | 37% | 42.08% | 18.83 |
| Vec100 | 37% | 43.07% | 3.54 |
| Vec50 | 37% | 42.77% | 0.98 |
| Random300 | 100% | 41.26% | 13.12 |
| Random100 | 100% | 41.60% | 10.46 |
| Random50 | 100% | 41.64% | 8.08 |

Table: Model's hyper-parameter settings are the same as 'BaselineCNN' except the filter's width of first layer which should be consistent with word vector's dimension. Each input vector setting is run for 10 times. The test dataset is SST.

| Fact\Predict | Very Negative | Negative | Neutral | Positive | Very Positive |
|---|---|---|---|---|---|
| Very Negative | 98 | 137 | 6 | 36 | 2 |
| Negative | 90 | 352 | 36 | 145 | 9 |
| Neutral | 27 | 130 | 30 | 193 | 10 |
| Positive | 4 | 43 | 13 | 386 | 62 |
| Very Positive | 5 | 7 | 3 | 223 | 161 |

Table: Classification result of 'BaselineCNN' on SST

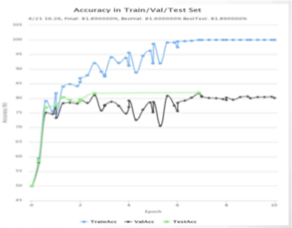
| Fact\Predict | Very Negative | Negative | Neutral | Positive | Very Positive |
|---|---|---|---|---|---|
| Very Negative | 49 | 203 | 13 | 14 | 0 |
| Negative | 30 | 491 | 37 | 68 | 6 |
| Neutral | 2 | 200 | 54 | 126 | 6 |
| Positive | 0 | 111 | 17 | 326 | 8 |
| Very Positive | 1 | 26 | 7 | 213 | 152 |

Table: Classification result of 'RCNN5' on SST

- Compared to CNN, RCNN make fewer 'completely-opposite' mistakes.

- After analyzing mistakenly classified sentences, we can find that most of them contains a word of the opposite emotion. It is typically a indirect expression or irony.(e.g. *It feels like a community theater production of a great broadway play: even at its best, it will never hold a candle to the original.*)
- On the contrary, most sentences that are correctly classified have explicit adjectives indicating emotions.
- ‘BaselineCNN’ is more likely to put sentence with negative words like ‘never’, ‘not’ into the opposite categories. This indicates that it fails to capture some structures, especially high-order N-gram structures.
- ‘RCNN5’ is more powerful to capture N-gram features, while it makes more ‘tiny mistakes’(like confusion between very negative/negative).

- Below are the learning curves of ‘BaselineCNN’(left) and ‘RCNN5’(right) on training(blue)/validation(black)/test(green) sets.
- It is obvious that ‘BaselineCNN’ converges faster than ‘RCNN5’ and is more stable.
- If run for many times, we can find that RCNN’s performance has higher variance.

# Conclusion

- Drawbacks of shadow networks.
  - Only can extract statistical features
  - Unable to extract structural features
- Failure of recurrent structure in semantic classification
  - Big gap between the number of neurons in each layer. It arises from independence among each dimension of word vectors.
  - Difficulty to fine-tune the model. (higher variance and slower convergence)
- Possible improvement.
  - Do transformation of input sentence matrix to ensure local features of both dimensions.
  - Train word vectors via CNN. (Most Popular word embedding algorithms are base on full-connected networks instead of CNN)

# References


- Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]. Advances in neural information processing systems. 2013: 3111-3119.
- Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.
- Liang M, Hu X. Recurrent Convolutional Neural Network for Object Recognition[C]. Proceedings of IEEE conference on Computer Vision and Pattern Recognition. 2015:3367-3375
- Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.
- Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences[J]. arXiv preprint arXiv:1404.2188, 2014.

# References


- Pang B, Lee L. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales[C]. Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2005: 115-124.
- Socher R, Perelygin A, Wu J Y, et al. Recursive deep models for semantic compositionality over a sentiment treebank[C]. Proceedings of the conference on empirical methods in natural language processing (EMNLP). 2013, 1631: 1642.