Convolutional Neural Networks for Sentence Classification

Yoon Kim
New York University

Agenda

Deep learning in Natural Language Processing

- Deep learning has achieved state-of-the-art results in computer vision (Krizhevsky et al., 2012) and speech (Graves et al., 2013).
- NLP: fast becoming (already is) a hot area of research.
- Much of the work involves learning word embeddings and performing composition over the learned embeddings for NLP tasks.

Word Embeddings (or Word Vectors)

- Traditional NLP: Words are treated as indices (or "one-hot" vectors in $\mathbf{R}^V$)
  - Every word is orthogonal to one another.
  - $\mathbf{w}_{mother} \cdot \mathbf{w}_{father} = 0$
- Can we embed words in $\mathbf{R}^D$ with $D \leq V$ such that semantically close words are likewise 'close' in $\mathbf{R}^D$? (i.e. $\mathbf{w}_{mother} \cdot \mathbf{w}_{father} > 0$)
  - Yes!
  - Don't (necessarily) need deep learning for this: Latent Semantic Analysis, Latent Dirichlet Allocation, or simple context counts all give dense representations.

Neural Language Models (NLM)

- Another way to obtain word embeddings.
- Words are projected from $\mathbf{R}^V$ to $\mathbf{R}^D$ via a hidden layer.
- D is a hyperparameter to be tuned.
- Various architectures exist. Simple ones are popular these days (right).
- Very fast—can train on billions of tokens in one day with a single machine.

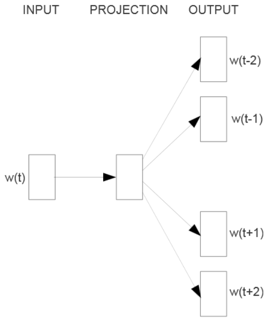


Figure 1: Skipgram architecture of Mikolov et al. (2013)

Linguistic regularities in the obtained embeddings

- The learned embeddings encode semantic and syntactic regularities:
  - $\mathbf{w}_{big} - \mathbf{w}_{bigger} \approx \mathbf{w}_{slow} - \mathbf{w}_{slower}$
  - $\mathbf{w}_{france} - \mathbf{w}_{paris} \approx \mathbf{w}_{korea} - \mathbf{w}_{seoul}$
- These are cool, but not necessarily unique to neural language models.

"[...] the neural embedding process is not discovering novel patterns, but rather is doing a remarkable job at preserving the patterns inherent in the word-context co-occurrence matrix."

Levy and Goldberg, "Linguistic Regularities in Sparse and Explicit Representations", CoNLL 2014

But the embeddings from NLMs are still good!

"We set out to conduct this study [on context-counting vs. context-predicting] because we were annoyed by the triumphalist overtones often surrounding predict models, despite the almost complete lack of a proper comparison to count vectors. Our secret wish was to discover that it is all hype, and count vectors are far superior to their predictive counterparts. [...] Instead we found that the predict models are so good that, while the triumphalist overtones still sound excessive, there are very good reasons to switch to the new architecture."

Baroni et al., "Don't count, predict! A systematic comparision of context-counting vs. context-predicting semantic vectors", ACL 2014

Using word embeddings as features in classification

- The embeddings can be used as features (along with other traditional NLP features) in a classifier.
- For multi-word composition (e.g. sentences and phrases), one could (for example) take the average.
- This is obviously a bit crude... can we do composition in a more sophisticated way?

Recursive Neural Tensor Networks (RNTN)

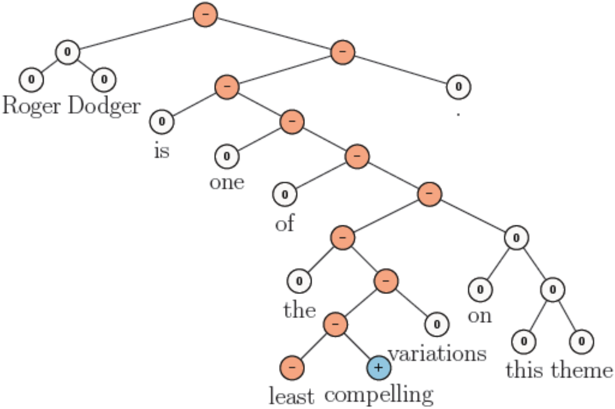


Figure 2: Socher et al., "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank", EMNLP 2013

RNTN

- Extended the previous state-of-the-art in sentiment analysis by a large margin.
- Best performing out of a family of recursive networks (Recursive Autoencoders, Socher et al., 2011; Matrix-Vector Recursive Neural Networks, Socher et al., 2012).
- Composition function is expressed as a tensor—each slice of the tensor encodes different composition.
- Can discern negation at different scopes.

RNTN

- Need parse trees to be computed beforehand.
- Phrase-level classification is expensive to obtain.
- Hard to adopt to other domains (e.g. Twitter).

Convolutional Neural Networks (CNN)

- Originally invented for computer vision (Lecun et al, 1989).
- Pretty much all modern vision systems use CNNs.

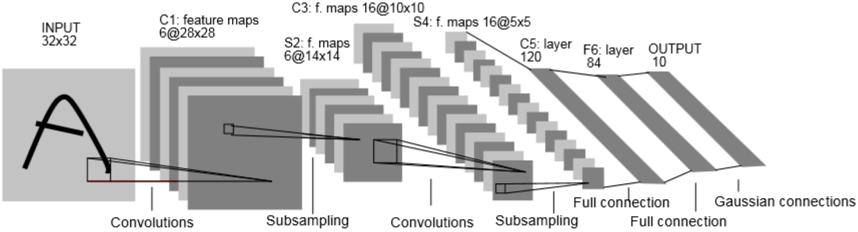


Figure 3: LeCun et al., "Gradient-based learning applied to document recognition", IEEE 1998

Brief tutorial on CNNs

- Key idea 1: Weight sharing via convolutional layers
- Key idea 2: Pooling layers
- Key idea 3: Multiple feature maps

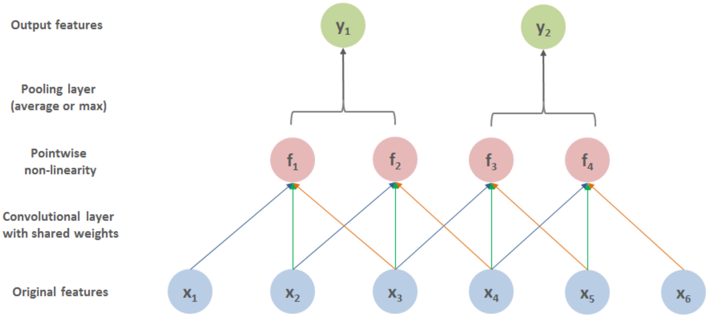


Figure 4: 1-dimensional convolution plus pooling

CNN: 2-dimensional case

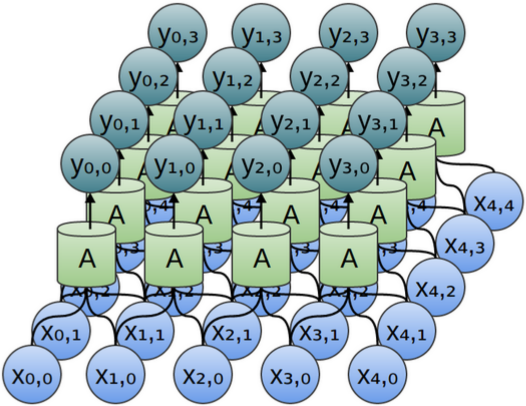


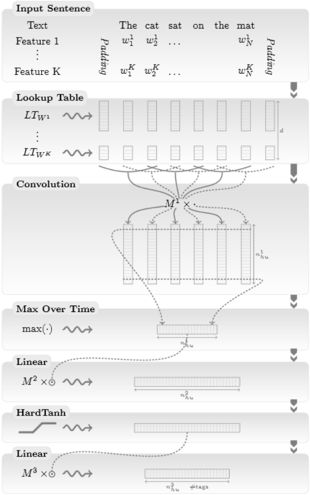
Figure 5: 2-dimensional convolution. From `http://colah.github.io/`

CNN details

- Shared weights means less parameters (than would be the case if fully connected).
- Pooling layers allow for local invariance.
- Multiple feature maps allow different kernels to act as specialized feature extractors.
- Training done through backpropagation.
- Errors are backpropagated through pooling modules.

CNNs in NLP

- Collobert and Weston used CNNs to achieve (near) state-of-the-art results on many traditional NLP tasks, such as POS tagging, SRL, etc.
- CNN at the bottom + CRF on top.
- Collobert et al., “Natural Language Processing (almost) from scratch”, JLMR 2011.



CNNs in NLP

- Becoming more popular in NLP
  - Semantic parsing (Yih et al., "Semantic Parsing for Single-Relation Question Answering", ACL 2014)
  - Search query retrieval (Shen et al., "Learning Semantic Representations Using Convolutional Neural Networks for Web Search", WWW 2014)
  - Sentiment analysis (Kalchbrenner et al., "A Convolutional Neural Network for Modelling Sentences", ACL 2014; dos Santos and Gatti, "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts", COLING 2014)
- Most of these networks are quite complex, with multiple convolutional layers.

Dynamic Convolutional Neural Network

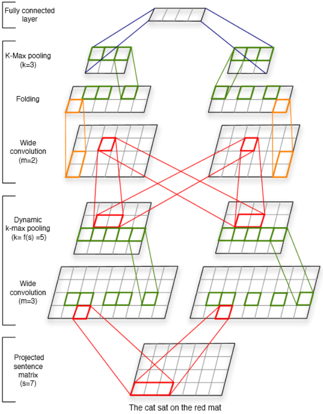
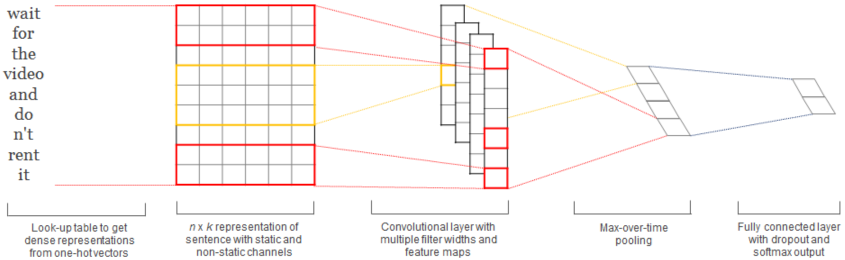


Figure 6: Kalchbrenner et al., "A Convolutional Neural Network for Modelling Sentences", ACL 2014

How well can we do with a simple CNN?

Collobert-Weston style CNN with pre-trained embeddings from `word2vec`



CNN architecture

- One layer of convolution with ReLU ($f(x) = x_+$) non-linearity.
- Multiple feature maps and multiple filter widths.
- Filter widths of 3, 4, 5 with 100 feature maps each, so 300 units in the penultimate layer.
- Words not in `word2vec` are initialized randomly from $U[-a, a]$ where a is chosen such that the unknown words have the same variance as words already in `word2vec`.
- Regularization: Dropout on the penultimate layer with a constraint on L_2-norms of the weight vectors.
- These hyperparameters were chosen via some light tuning on one of the datasets.

Dropout

- Proposed by Hinton et al. (2012) to prevent co-adaptation of hidden units.
- During forward propagation, randomly "mask" (set to zero) each unit with probability p. Backpropagate only through unmasked units.
- At test time, do not use dropout, but scale the weights by p.
- Like taking the geometric average of different models.
- Rescale weights to have L_2-norm $= s$ whenever L_2-norm $> s$ after a gradient step.

Note on SGD: Adagrad vs. Adadelta

- Adagrad (Duchi et al., 2011)

  $$w_{t+1} = w_t - \frac{\eta}{\epsilon + \sqrt{\sum_{i=1}^{t} g_i^2}} g_t$$

- Adadelta (Zeiler, 2012)

  $w_{t+1} = w_t - \sqrt{\frac{\epsilon + s_t}{\epsilon + q_t}} g_t$, where s_t and q_t recursively defined as,
  $s_t = \rho s_{t-1} + (1-\rho)(w_t - w_{t-1})^2$
  $q_t = \rho q_{t-1} + (1-\rho) g_t^2$

- Adadelta generally required fewer epochs to reach the (local) minima, even with a higher η on Adagrad.
- But both eventually give similar results (Adagrad slightly more stable).
- Use Adadelta to quickly search the hyperparameter space and then build final model with Adagrad.

Datasets

Sentence/phrase-level classification tasks

Data	c	l	N	$\lvert V \rvert$	$\lvert V_{pre} \rvert$	Prev SotA
MR	2	20	10662	18765	16448	79.5
SST-1	5	18	11855	17836	16262	48.7
SST-2	2	19	9613	16185	14838	87.8
Subj	2	23	10000	21323	17913	93.6
TREC	6	10	5952	9592	9125	95.0
CR	2	19	3775	5340	5046	82.7
MPQA	2	3	10606	6246	6083	87.2

- c: number of labels
- l: average sentence length
- N: number of sentences
- $|V|$: vocab size ($|V_{pre}|$ is words already in `word2vec`)

Baseline: Randomly initialize all words (CNN-rand)

Data	Prev SotA	CNN-rand
MR	79.5	76.1
SST-1	48.7	45.0
SST-2	87.8	82.7
Subj	93.6	89.6
TREC	95.0	91.2
CR	82.7	79.8
MPQA	87.2	83.4

- Baseline model doesn't do too well...

Model 1: Keep the embeddings fixed (CNN-static)

Data	Prev SotA	CNN-rand	CNN-static
MR	79.5	76.1	81.0
SST-1	48.7	45.0	45.5
SST-2	87.8	82.7	86.8
Subj	93.6	89.6	93.0
TREC	95.0	91.2	92.8
CR	82.7	79.8	84.7
MPQA	87.2	83.4	89.6

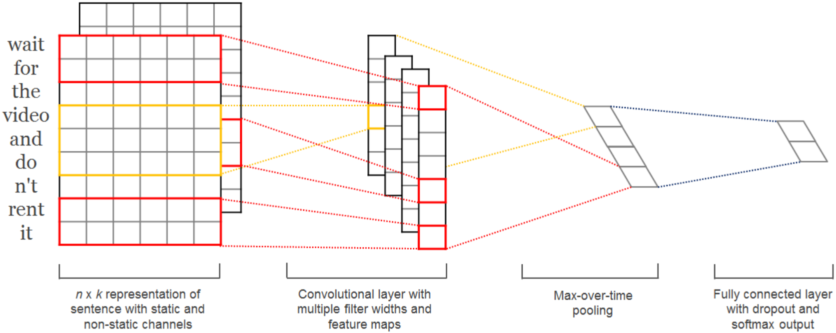
- Even a simple model does very well!
- `word2vec` embeddings are "universal" enough that they can be used for different tasks without having to learn task-specific embeddings.
- Same hyperparameters for all datasets.

Model 2: Fine-tune embeddings for each task (CNN-nonstatic)

Data	Prev SotA	CNN-rand	CNN-static	CNN-nonstatic
MR	79.5	76.1	81.0	81.5
SST-1	48.7	45.0	45.5	48.0
SST-2	87.8	82.7	86.8	87.2
Subj	93.6	89.6	93.0	93.4
TREC	95.0	91.2	92.8	93.6
CR	82.7	79.8	84.7	84.3
MPQA	87.2	83.4	89.6	89.5

- Fine-tuning vectors helps, though not that much.
- Perhaps our embeddings are overfitting (given the relatively small training sample)?

Model 3: Multi-channel CNN



- Two "channels" of embeddings (i.e. look-up tables).
- One is allowed to change, while one is kept fixed.
- Both initialized with `word2vec`.

Model 3 performance is mixed

Data	Prev SotA	CNN-nonstatic	CNN-multichannel
MR	79.5	81.5	81.1
SST-1	48.7	48.0	47.4
SST-2	87.8	87.2	88.1
Subj	93.6	93.4	93.2
TREC	95.0	93.6	92.2
CR	82.7	84.3	85.0
MPQA	87.2	89.5	89.4

- Performance is not statistically different from CNN-nonstatic.

Fine-tuned embeddings (on SST)

	Most Similar Words for	
	Static	Non-static
bad	*good* *terrible* *horrible* *lousy*	*terrible* *horrible* *lousy* *stupid*
good	*great* *bad* *terrific* *decent*	*nice* *decent* *solid* *terrific*

- `good` and `bad` are similar to each other in original `word2vec` because interchanging them will still result in a grammatically correct sentence.
- The model learns to discriminate adjectival scales.
- sim(`good`, `nice`) $>$ sim(`good`, `great`)

Fine-tuned embeddings (on SST)

	Static	Non-static
n't	*os*	*not*
	ca	*never*
	ireland	*nothing*
	wo	*neither*
!	*2,500*	*2,500*
	entire	*lush*
	jez	*beautiful*
	changer	*terrific*
,	*decasia*	*but*
	abysmally	*dragon*
	demise	*a*
	valiant	*and*

- `n't` was already in `word2vec` but had meaningless embeddings.
- `!` and `,` were not in `word2vec`.
- The network learns that `!` is associated with effusive words and that `,` is conjunctive (though not very well).
- Not sure if the multichannel architecture is the right way to regularize embeddings.

Further Observations

- Width/multiple feature maps are important up to a point.

Width/Feature Maps	10	25	50	100
2	75.8	78.4	78.1	78.5
3	78.9	80.0	79.6	79.2
4	78.1	81.6	80.1	79.9
5	80.0	79.6	81.0	80.5
6	79.0	80.5	82.1	81.9
7	80.8	81.1	81.1	82.3

- Performance on one fold of the MR dataset

Further Observations

- ReLU, Tanh, Hard Tanh all gave similar results (contrary to vision). Might be different if we have deeper architectures (ReLU is robust to gradient saturation).
- L_2-norm constraint on the penultimate layer is important.
- When using pre-trained vectors, initializing unknown words to have similar variance as the pre-trained ones helps.
- Existing software makes it easy to train neural nets (`Theano`, `Torch`).
- Briefly experimented with Collobert-Weston (SENNA) embeddings trained on Wikipedia—`word2vec` was much better.

Future work

- Regularizing the fine-tuning process:
  - Keep `word2vec` embeddings fixed, fine-tune only unknown words.
  - Have extra-dimensions which are allowed to change.
  - Be smarter about initializing unknown words.
- Recurrent architectures, though difficult to train, seem promising for sentence composition/classification
  - Sutsekever et al., Sequence to Sequence Learning with Neural Networks, arXiv 2014.
  - Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, arXiv 2014.
  - Kalchbrenner and Blunsom, Recurrent Convolutional Neural Networks for Discourse Compositionality, ACL Workshop 2013.
- Document-level classification.

Paper/slides/code available at `http://www.yoon.io`