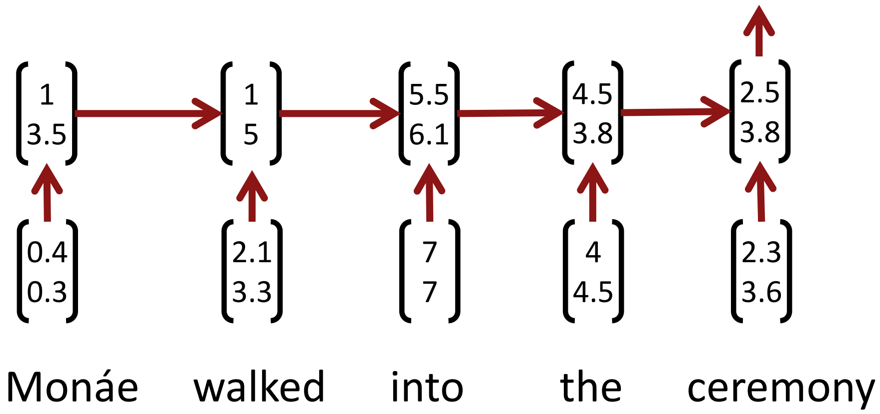
1. From RNNs to Convolutional Neural Nets

- Recurrent neural nets cannot capture phrases without prefix context
- Often capture too much of last words in final vector



- E.g., softmax for word prediction is usually calculated based on the last step

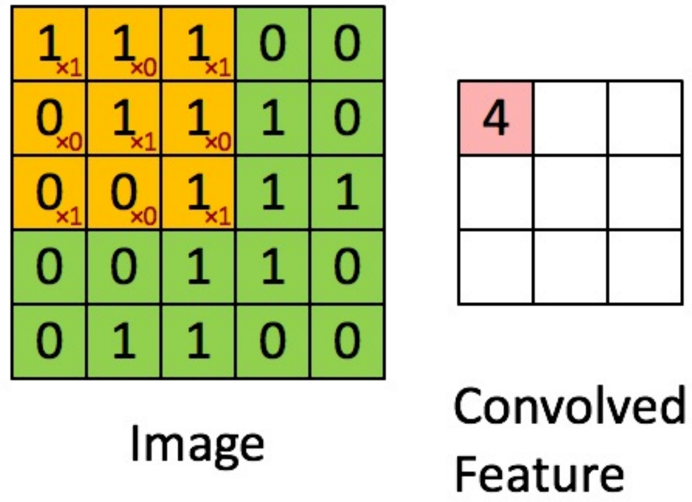
From RNNs to Convolutional Neural Nets

- Main Convolutional Neural Net (CNN/ConvNet) idea:
  - What if we compute vectors for every possible word subsequence of a certain length?
- Example: “tentative deal reached to keep government open” computes vectors for:
  - tentative deal reached, deal reached to, reached to keep, to keep government, keep government open
- Regardless of whether phrase is grammatical
  - Not very linguistically or cognitively plausible
- Then group them afterwards (more soon)

What is a convolution anyway?

- 1d discrete convolution generally: $(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$
- Convolution is classically used to extract features from images
  - Models position-invariant identification
  - Go to cs231n!

- 2d example →
- Yellow color and red numbers show filter (=kernel) weights
- Green shows input
- Pink shows output



From Stanford UFLDL wiki

A 1D convolution for text

tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3

Apply a **filter** (or **kernel**) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3

0.0	0.50
0.5	0.38
-2.6	0.93
0.8	0.31
1.3	0.21

+ bias

➔ non-linearity

1D convolution for text with padding

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

∅,t,d	−0.6
t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3
g,o,∅	−0.5

Apply a **filter** (or **kernel**) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

3 channel 1D convolution with padding = 1 and 3 filters

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

Could also use (zero)
padding = 2
Also called “wide convolution”

conv1d, padded with max pooling over time

Ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	0.0	0.0	0.0	0.0

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

max p	0.3	1.6	1.4

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

conv1d, padded with ave pooling over time

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

∅,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,∅	−0.5	−0.9	0.1

ave p	−0.87	0.26	0.53

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

In PyTorch

```
batch_size = 16
word_embed_size = 4
seq_len = 7
input = torch.randn(batch_size, word_embed_size, seq_len)
conv1 = Conv1d(in_channels=word_embed_size, out_channels=3,
               kernel_size=3)  # can add: padding=1
hidden1 = conv1(input)
hidden2 = torch.max(hidden1, dim=2)  # max pool
```

Other (maybe less useful) notions: stride = 2

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

∅,t,d	−0.6	0.2	1.4
d,r,t	−0.5	−0.1	0.8
t,k,g	−0.2	0.1	1.2
g,o,∅	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Local max pool, stride = 2

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

∅,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,∅	−0.5	−0.9	0.1
∅	−Inf	−Inf	−Inf

∅,t,d,r	−0.6	1.6	1.4
d,r,t,k	−0.5	0.3	0.8
t,k,g,o	0.3	0.6	1.2
g,o,∅,∅	−0.5	−0.9	0.1

conv1d, *k*-max pooling over time, *k* = 2

∅	**0.0**	**0.0**	**0.0**	**0.0**
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	**0.0**	**0.0**	**0.0**	**0.0**

∅,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,∅	−0.5	−0.9	0.1

2-max p	0.3	1.6	1.4
	−0.2	0.6	1.2

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Other somewhat useful notions: dilation = 2

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

∅,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,∅	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

1,3,5	0.3	0.0
2,4,6		
3,5,7		

2	3	1
1	−1	−1
3	1	0

1	3	1
1	−1	−1
3	1	−1

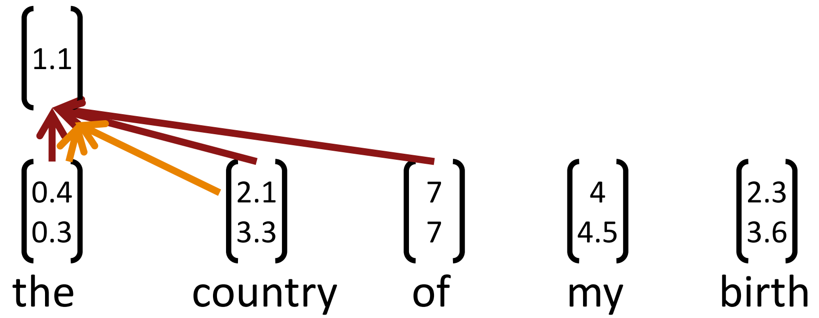
2. Single Layer CNN for Sentence Classification

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014. https://arxiv.org/pdf/1408.5882.pdf

- Goal: Sentence classification:
  - Mainly positive or negative sentiment of a sentence
  - Other tasks like:
    - Subjective or objective language sentence
    - Question classification: about person, location, number, …

Single Layer CNN for Sentence Classification

- A simple use of one convolutional layer and **pooling**
- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus x_2 \oplus \cdots \oplus \mathbf{x}_n$ (vectors concatenated)
- Concatenation of words in range: $\mathbf{x}_{i:i+j}$ (symmetric more common)
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (over window of *h* words)
- Note, filter is a vector
- Filter could be of size 2, 3, or 4 words:

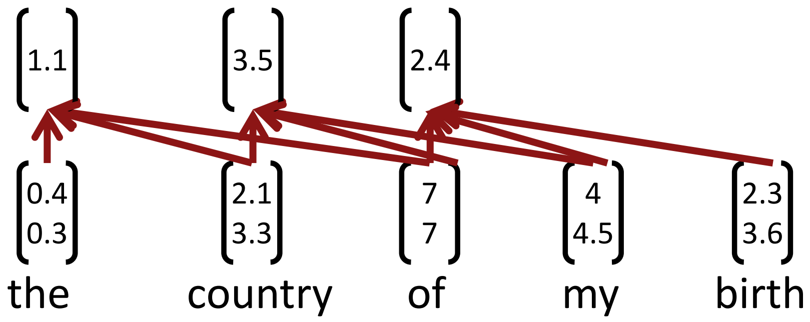


Single layer CNN

- Filter **w** is applied to all possible windows (concatenated vectors)
- To compute feature (one *channel*) for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length *h*: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Pooling and channels

- Pooling: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)
- From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$

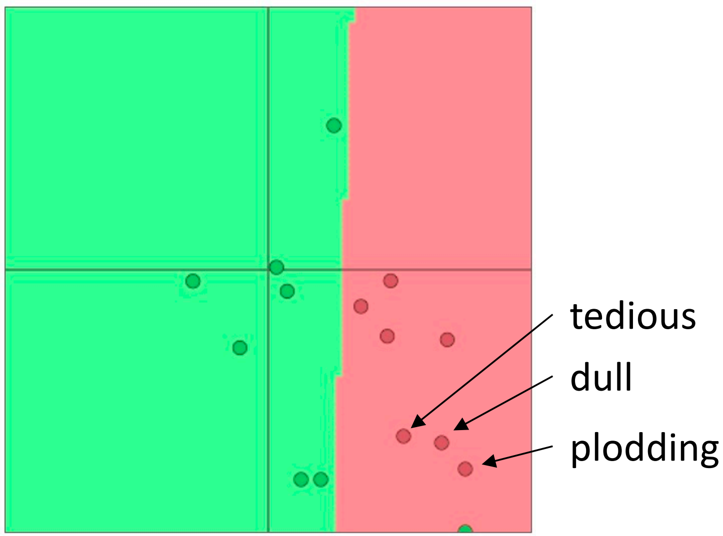
- Use multiple filter weights **w** (i.e., multiple channels)
- Useful to have different window sizes *h*
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** can be variable

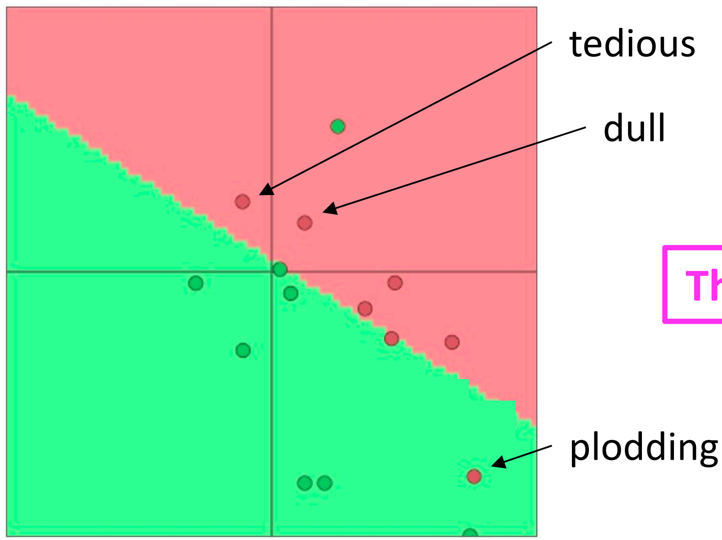
$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- So, we can have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.
  - Even without padding

A pitfall when fine-tuning word vectors

- **Setting:** We are training a model for movie review sentiment building on word vectors
- In the training data we have “tedious”, “dull”; in the testing data we have “plodding”
- The pre-trained word vectors have all three similar:
- **Question: What happens when we update the word vectors?**
- **Answer:** Words in the training data move around; other words stay where they were





This can be bad!

Channel doubling multi-channel input idea

- Initialize model with pre-trained word vectors (e.g., word2vec or Glove)

- Start with two copies

- Backprop into only one set, keep other "static"
  - Fine-tuning should be useful for improving word vectors for task
  - But there is a problem that words in pre-training (and maybe runtime data) but not in training data will not move. So, it also makes sense to leave all word vectors where they are and to only update the parameters above the word vectors
  - Having two copies is an attempt to get the best of both worlds

- Both channel sets are added to c_i before max-pooling

Classification after one CNN layer

- First one convolution, followed by one max-pooling
  - To obtain final feature vector: $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$
    (assuming *m* filters **w**)
  - Used 100 feature maps each of sizes 3, 4, 5

- Simple final softmax layer

$$y = softmax\left(W^{(S)}z + b\right)$$

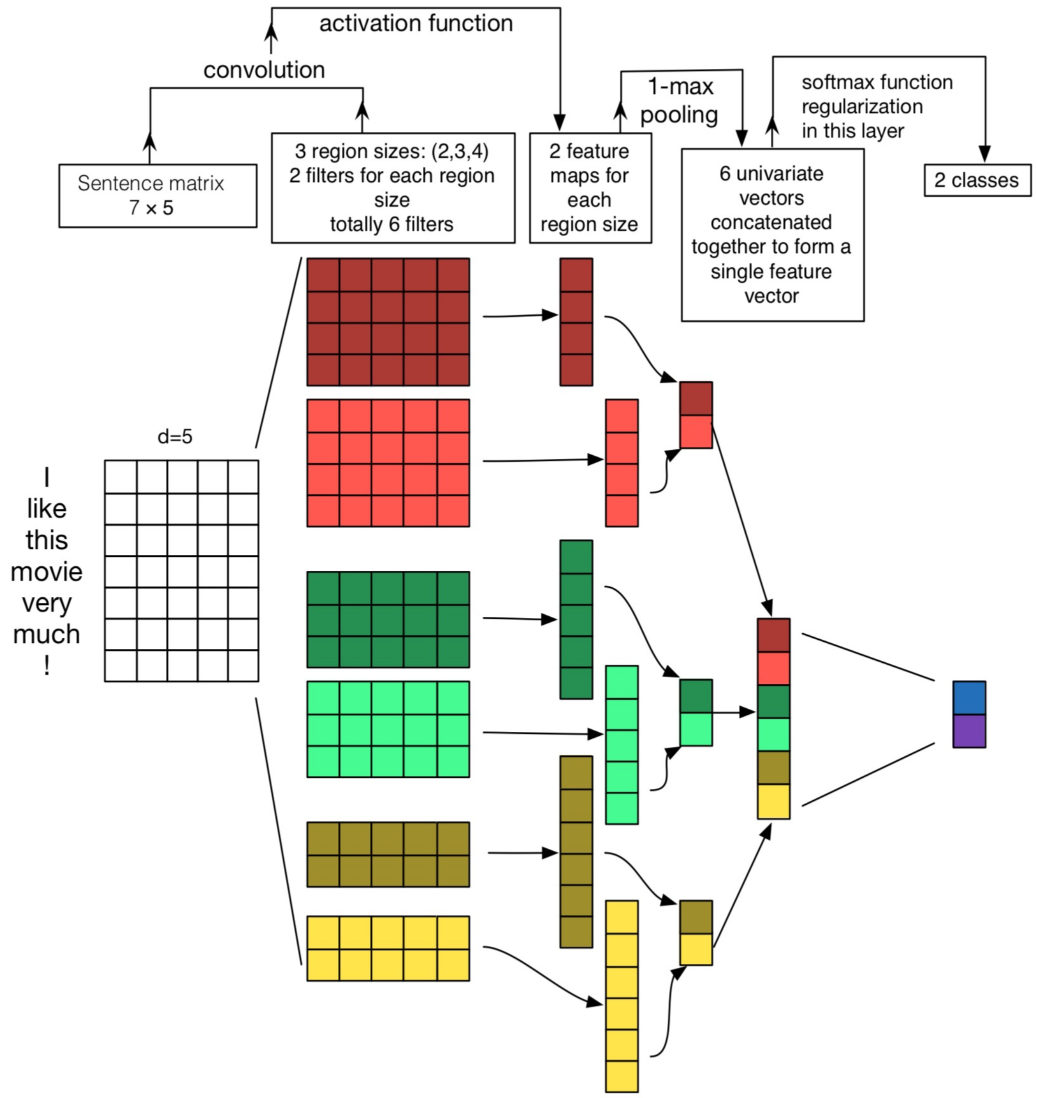
Kim (2014)

From:

Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

https://arxiv.org/pdf/1510.03820.pdf

(follow on paper, not famous, but a nice picture)



All hyperparameters in Kim (2014)

- Find hyperparameters based on dev set
- Nonlinearity: ReLU
- Window filter sizes h = 3, 4, 5
- Each filter size has 100 feature maps
- Dropout p = 0.5
  - Kim (2014) reports **2–4%** accuracy improvement from dropout
- L_2 constraint *s* for rows of softmax, *s* = 3
- Mini batch size for SGD training: 50
- Word vectors: pre-trained with word2vec, *k* = 300

- During training, keep checking performance on dev set and pick highest accuracy weights for final evaluation

Experiments on text classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Problem with comparison?

- Dropout gives 2–4 % accuracy improvement
- But several compared-to systems didn't use dropout and would possibly gain equally from it

- Still seen as remarkable results from a simple architecture!

- Differences from window architecture we described in an early lecture:
  - Many filters and pooling