STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 584 Natural Language Processing

Convolutional Neural Networks for NLP

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **24 hours**.
- **40%** penalty for late submissions within **24-48 hours**.
- After 48 hours, you get **NO** points on the assignment.

Convolutional Neural Networks

- Originally, convolutional neural networks are specialized networks for application in **computer vision**
  - they accept images as raw input (preserving spatial information),
  - and build up (learn) a hierarchy of features (no hand-crafted features necessary).

Convolutional Neural Networks

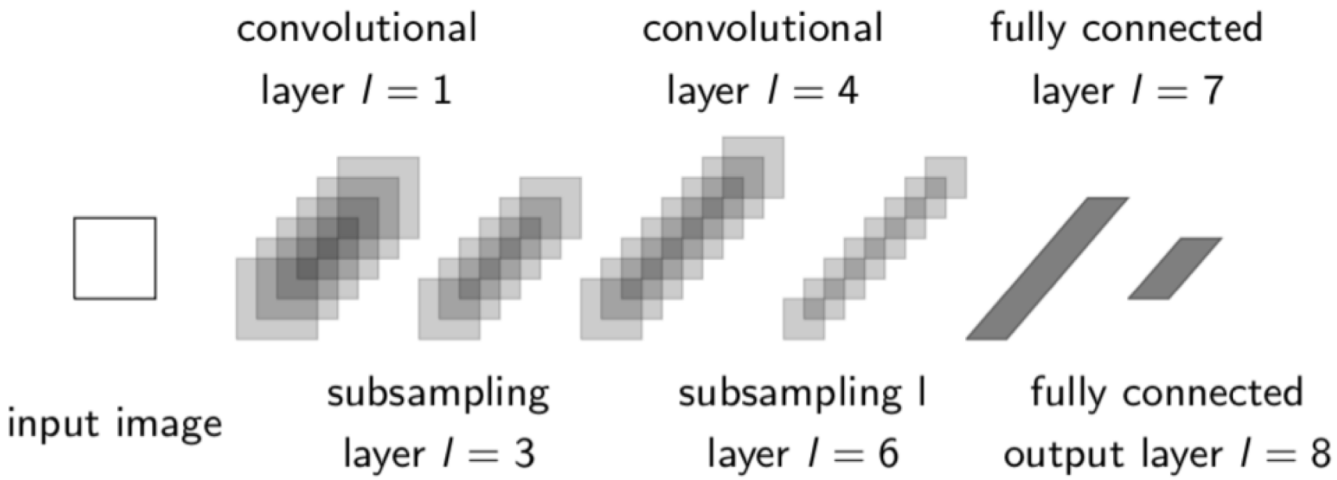
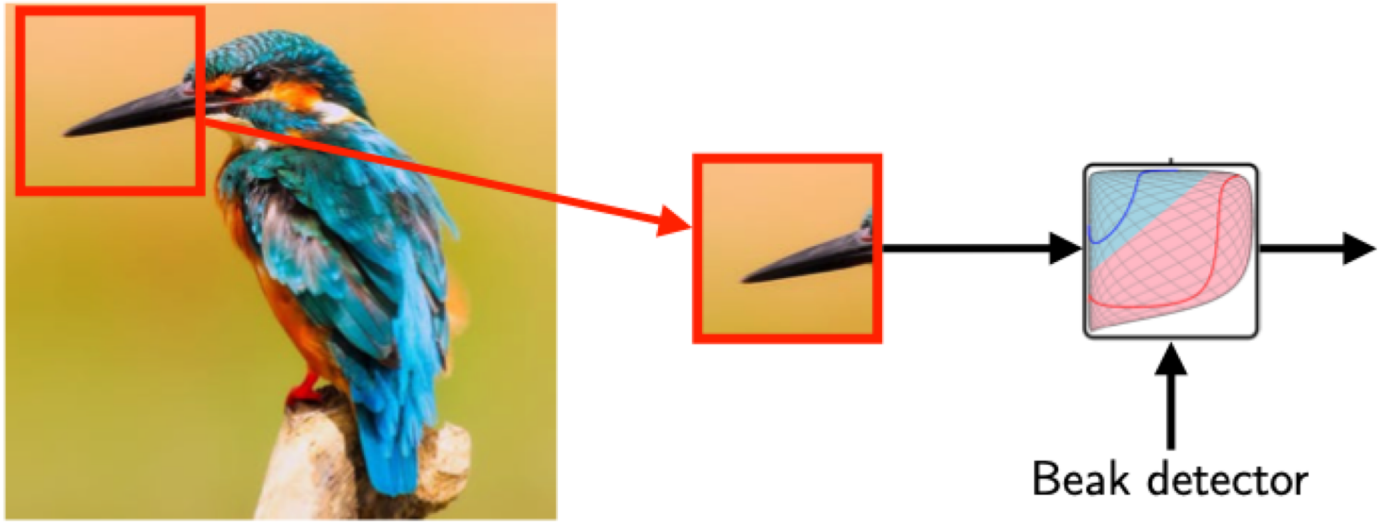


Figure: The architecture of the original convolutional neural network, as introduced by LeCun et al. (1989)

- Original Convolutional Neural Network [5], [6] aims to build up a feature hierarchy by alternating
  - convolutional layer − non-linearity layer − subsampling layer
- followed by a multilayer perceptron for classification

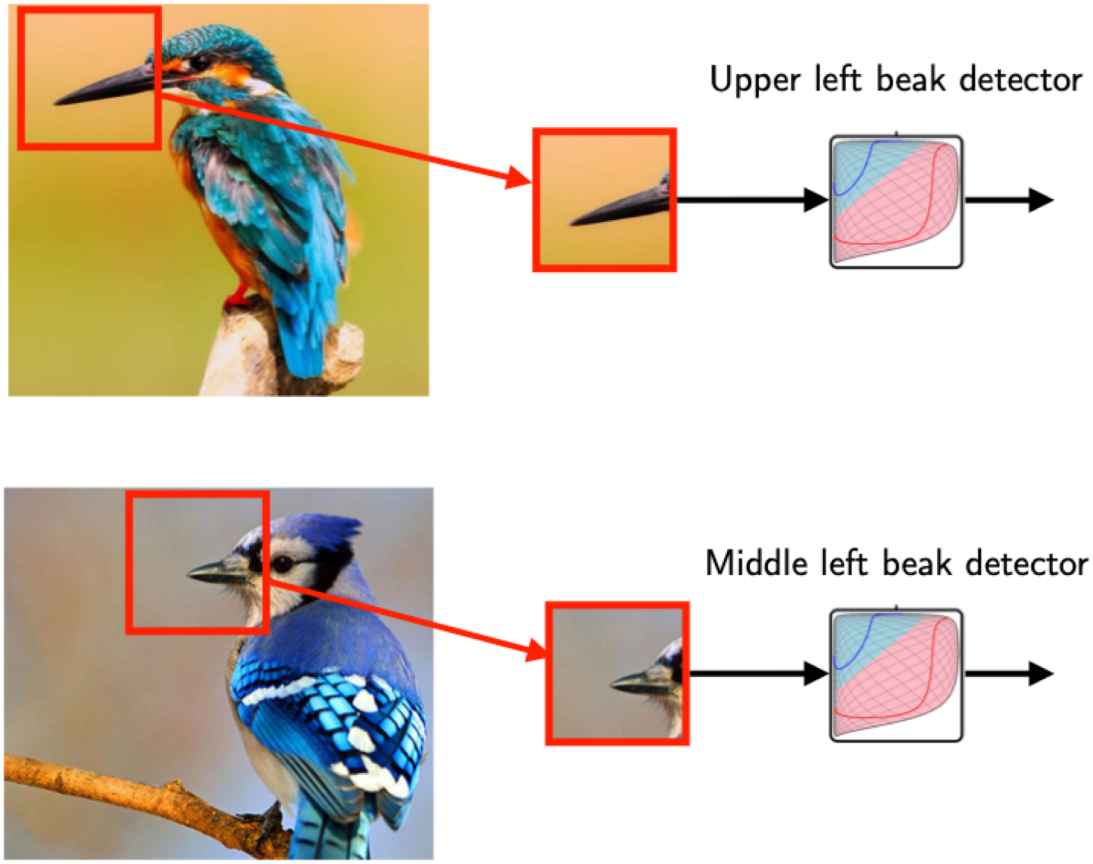
Why convolutional layer?



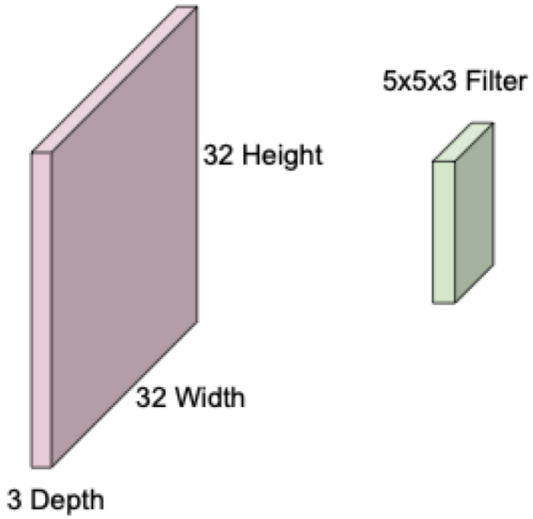
- Some patterns are much smaller than the whole image
- A neuron does not have to see the whole image to discover the pattern.
- Connecting to small region with less parameters.

Why convolutional layer?

- The sample pattern appear in different regions.



Convolutional operation



- Convolve the image with the filter, “slide over the image spatially, computing dot products”
- Filters always extend the full depth of the input volume.
- Taking a dot product between the filter and a small 5x5x3 chunk of the image becomes a Scalar
- It has 5x5x3=75 dot product + bias

Convolutional layer - example

- Slide over the image spatially with the filter, computing dot products

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6x6 input

1	-1	-1
-1	1	-1
-1	-1	1

filter 1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

feature map 1

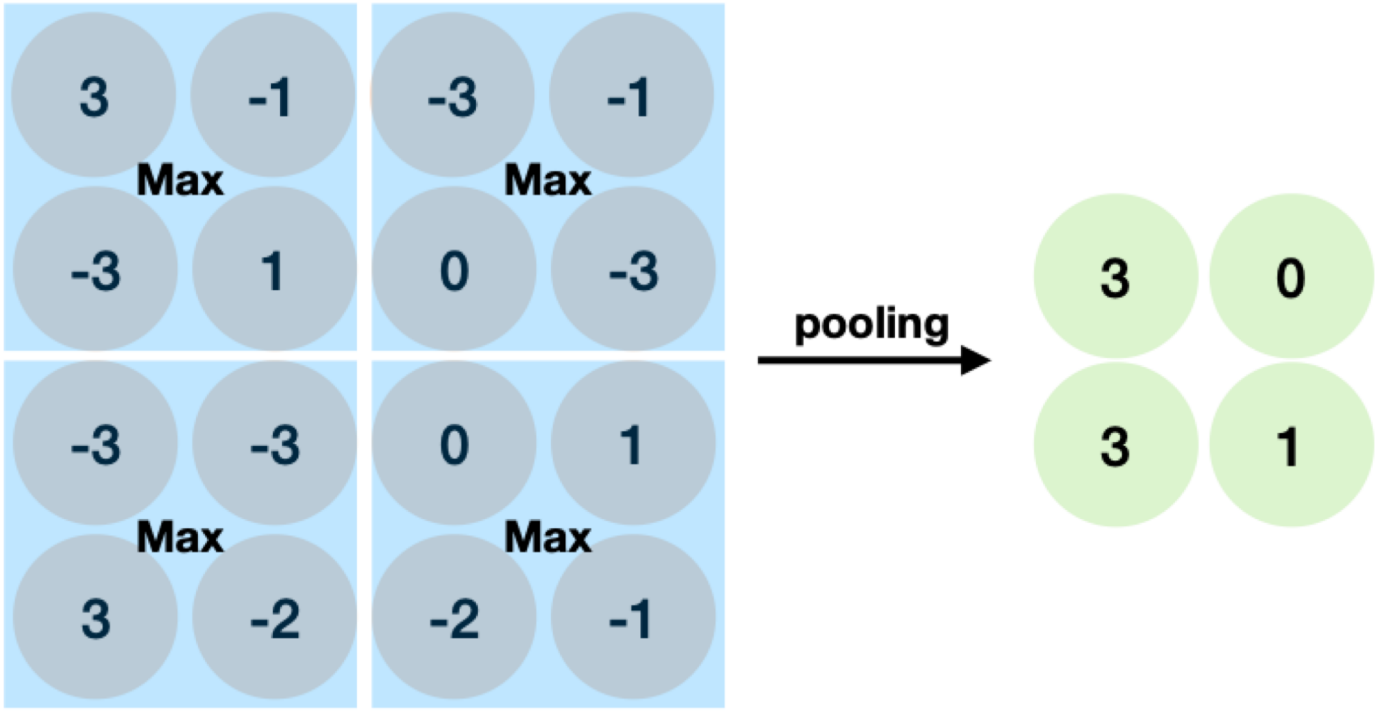
Non-linearity layer

- Given feature maps, a non-linear layer applies an activation function to all these features maps:

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right)$$

- where f operates point-wise
- usually f is the hyperbolic tangent

Max pooling/subsampling layer



Max-pooling layer: 2 × 2 filter, stride 2

Putting it all together

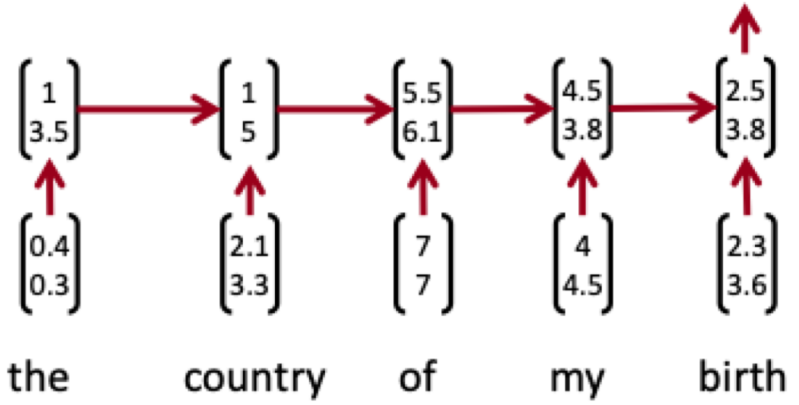
- Remember: A convolutional network alternates
  convolutional layer - non-linearity layer - subsampling layer

- to build up a hierarchy of feature maps and uses a **multilayer perceptron** for classification.

Y. LeCun, et al. “Backpropagation applied to handwritten zip code recognition,”http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf

From RNNs to CNNs

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



- E.g. softmax is often only calculated at the last step

From RNNs to CNNs

- Main CNN idea:
  - what if we compute vectors for every possible word subsequences 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)

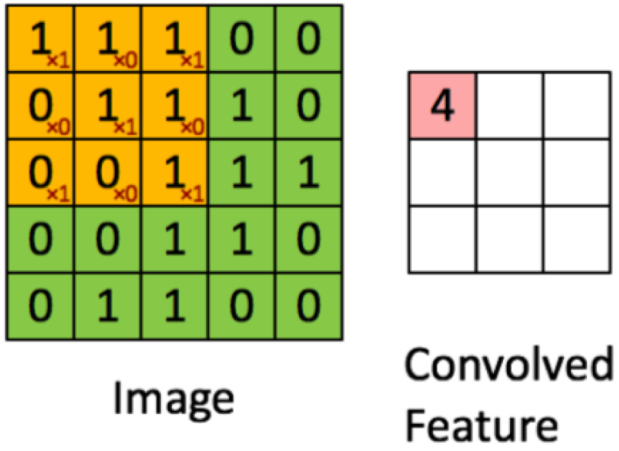
Convolution definition

- 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

- 2d example ->
- Yellow color and red number show filter(kernel) weights
- Green shows input
- Pink shows output



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

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

Apply a filter (kernel) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

A 1D convolution for text with padding

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
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,Ø	−0.5

Apply a filter (kernel) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

3 channel 1D convolution with padding

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

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

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

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
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 less useful notions: stride = 2

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
d,r,t	−0.5	−0.1	0.8
t,k,g	−0.2	0.1	1.2
g,o,Ø	−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

Less useful: local max pool, stride=2

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
Ø	−Inf	−Inf	−Inf

Ø,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,Ø,Ø	−0.5	−0.9	0.1

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

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

2-max p	−0.2	1.6	1.4
	0.3	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

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

1,3,5	-0.9	-1.5
2,4,6		
3,5,7		

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

2	3	1
1	−1	−1
3	1	0

1	3	1
1	−1	−1
3	1	−1

Single layer CNN for sentence classification

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014.
  - Paper: https://arxiv.org/pdf/1408.5882.pdf
  - Code: https://github.com/yoonkim/CNN_sentence [Theano!]
- A variant of convolutional NNs of Collobert, Weston et al. (2011)
- 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: $x_i \in \mathbb{R}^k$
- Sentence $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (vectors concatenated)
- Concatenation of words in range: $x_{i:i+j}$ (symmetric more common)
- Convolutional filter: $w \in \mathbb{R}^{hk}$ (over window of h words)
- Note that filter is a vector!
- Filter could be size of 2,3, or 4

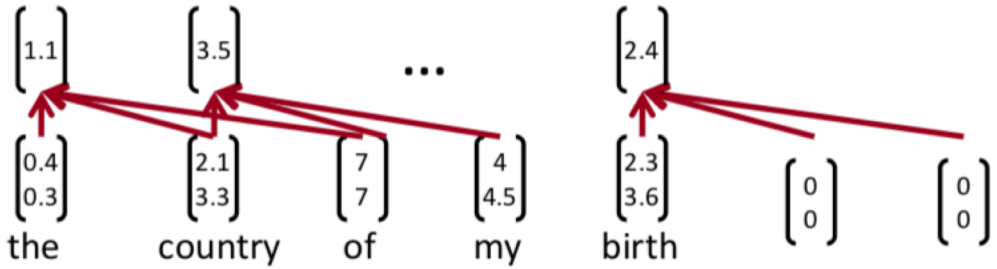
Single layer CNN

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

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

- Sentence $x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n$
- All possible windows of length h: $\{x_{1:h}, x_{2:h+1}, ..., x_{n-h+1:n}\}$

- Result is a feature map: $c=[c_1, c_2, ..., c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Pooling and channels

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

- Pooled single number: $\hat{c} = \max \mathbf{c}$

- Use multiple filter weights w
- Useful to have different window sizes h
- Because of max pooling $\hat{c} = \max \mathbf{c}$ length of c is irrelevant
- So we could have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc

Multi-channel input data

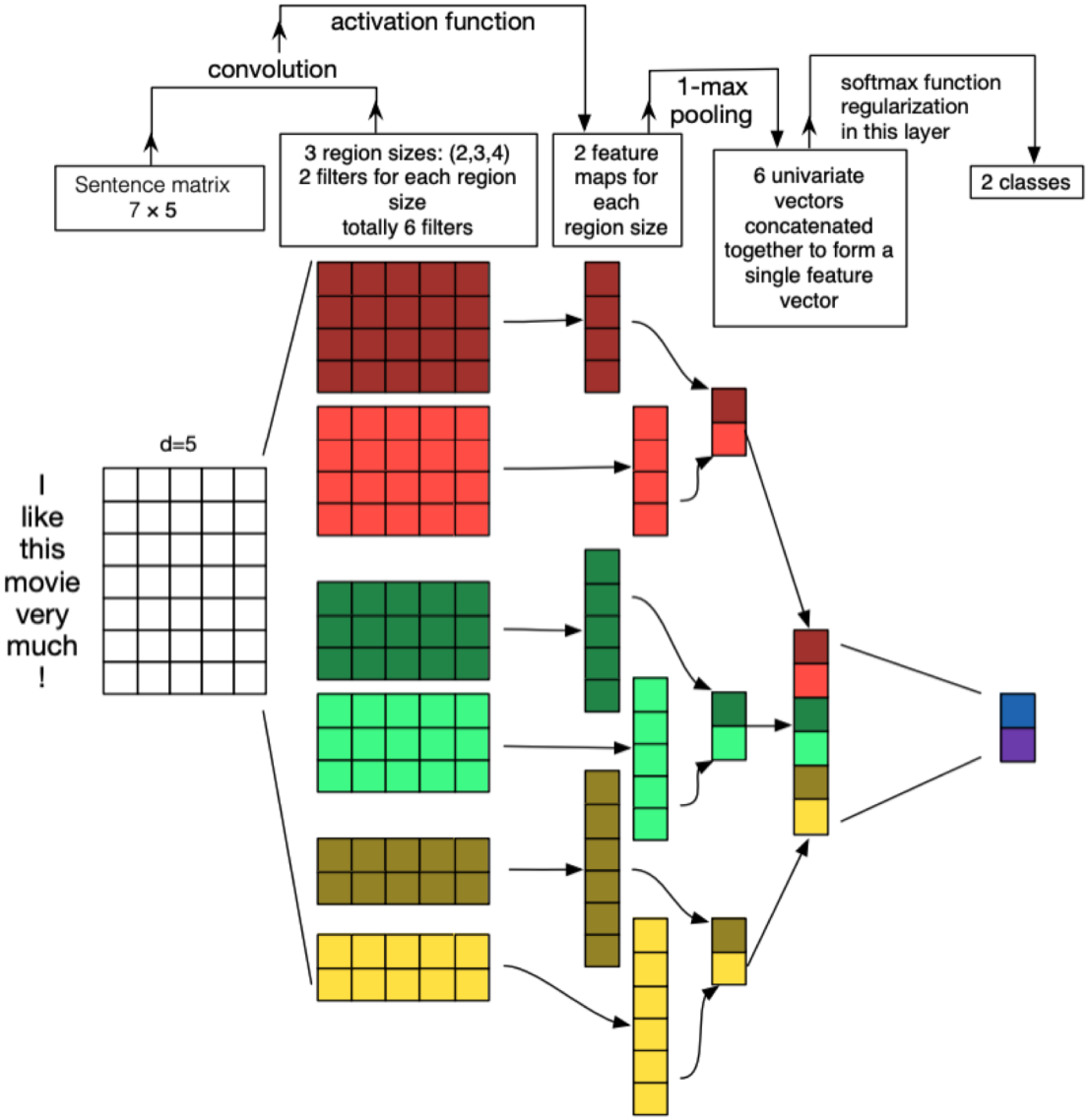
- Initialize with pre-trained word vectors (word2vec, glove)
- Start with two copies
- Backprop into only one set, keep other “static”
- Both channel sets are added to ci before max-pooling

Classification after one CNN layer

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

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

One example



Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutio Neural Networks for Sentence Classification
https://arxiv.org/pdf/1510.03820.pdf
(follow on paper, not famous, but a nice picture)

Regularization

- Use Dropout: Create masking vector r of Bernoulli random variables with probability p (a hyperparameter) of being 1
- Delete features during training:

$$y = \text{softmax}(W^{(S)}(r \circ z) + b)$$

- Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations) (Srivastava, Hinton, et al. 2014)
- At test time, no dropout, scale final vector by probability p

$$\hat{W}^{(S)} = pW^{(S)}$$

- Also: Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number s (also a hyperparameter)
- If $||W_c^{(S)}|| > s$, then rescale it so that $||W_c^{(S)}|| = s$:
- Not very common

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
- L2 constraints 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

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?

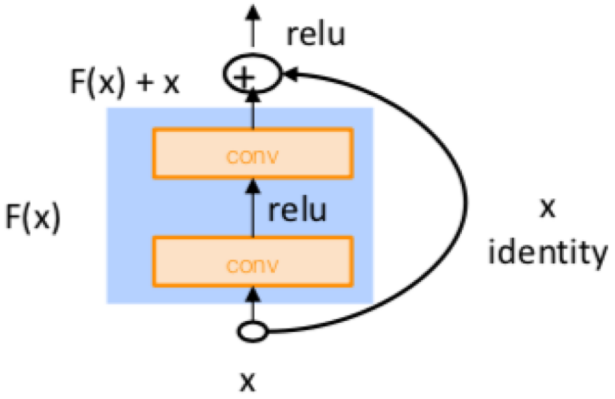
- Dropout gives 2-4% accuracy improvement, but several compared-to systems didn't use dropout and would possibly gain equally from it
- Still seen as remarkable results from a simple architecture!
- Difference to window and RNN architectures we described in previous lectures: pooling, many filters, and dropout
- Some of these ideas can be used in RNNs too

Model comparison: Our growing toolkit

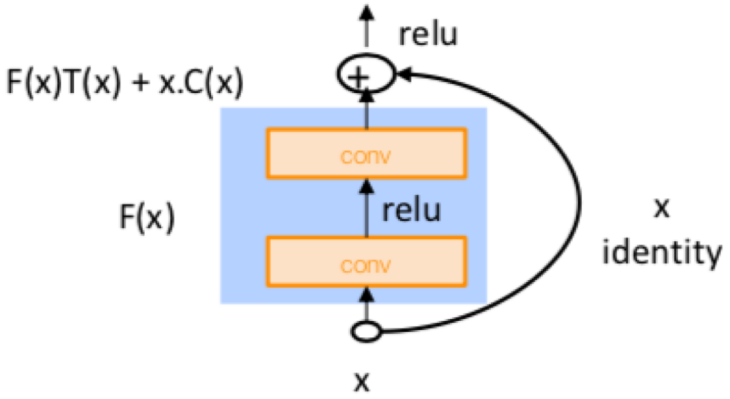
- **Bag of Vectors**: Surprisingly good baseline for simple classification problems. Especially if followed by a few ReLU layers! (See paper: Deep Averaging Networks)
- **Window Model**: Good for single word classification for problems that do not need wide context. E.g., POS, NER.
- **CNNs**: good for classification, need zero padding for shorter phrases, hard to interpret, easy to parallelize on GPUs. Efficient and versatile
- **Recurrent Neural Networks**: Cognitively plausible (reading from left to right), not best for classification (if just use last state), much slower than CNNs, good for sequence tagging and classification, great for language models, can be amazing with attention mechanisms

Gated units used vertically

- The gating/skipping that we saw in LSTMs and GRUs is a general idea, which is now used in a whole bunch of places
- You can also gate vertically
- Indeed the key idea – summing candidate update with shortcut connection – is needed for very deep networks to work



Residual block
(He et al. ECCV 2016)



Highway block
(Srivistava et al. NeurIPS 2015)

Note: pad x for conv so same size when add them

Batch Normalization (BatchNorm)

- Often used in CNNs
- Transform the convolution output of a batch by scaling the activations to have zero mean and unit variance
  - This is the familiar Z-transform of statistics
  - But updated per batch so fluctuation don't affect things much
- Use of BatchNorm makes models much less sensitive to parameter initialization, since outputs are automatically rescaled
  - It also tends to make tuning of learning rates simpler
- PyTorch: nn.BatchNorm1d

Ioffe and Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. http://proceedings.mlr.press/v37/ioffe15.pdf

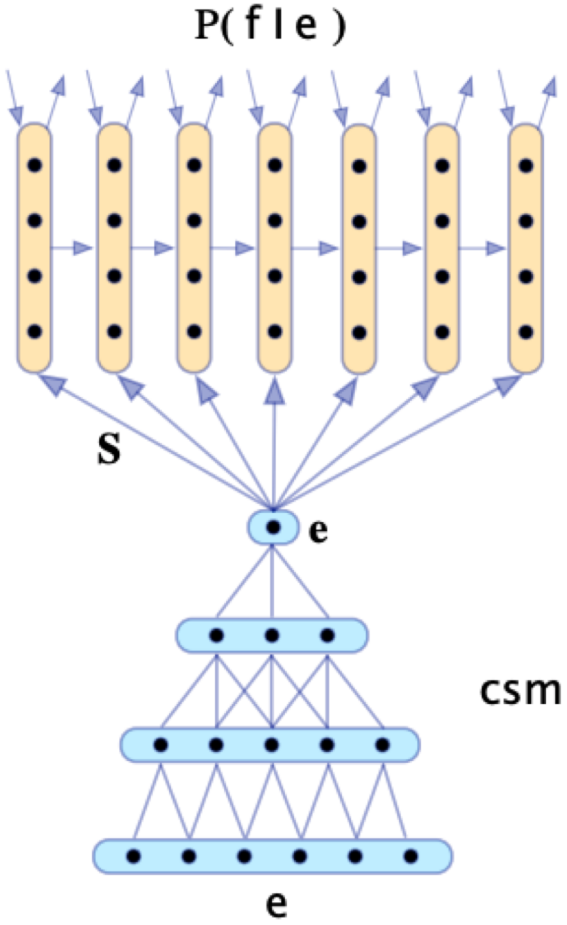
1x1 Convolutions

- Does this concept make sense?!? Yes.
- 1x1 convolutions, a.k.a. Network-in-network (NiN) connections, are convolutional kernels with kernel_size=1
- A 1x1 convolution gives you a fully connected linear layer across channels!
- It can be used to map from many channels to fewer channels
- 1x1 convolutions add additional neural network layers with very few additional parameters
  - Unlike Fully Connected (FC) layers which add a lot of parameters

Lin, Chen, and Yan. 2013. Network in network. https://arxiv.org/abs/1312.4400

CNN application: Translation

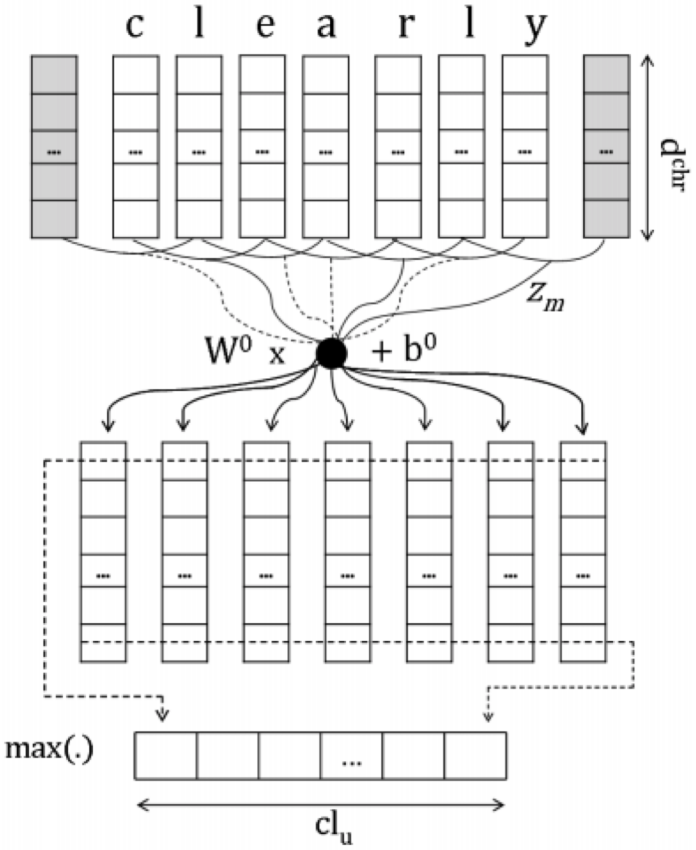
- One of the first successful neural machine translation efforts
- Uses CNN for encoding and RNN for decoding



Kalchbrenner and Blunsom (2013). "Recurrent Continuous Translation Models". https://www.aclweb.org/anthology/D13-1176

Learning character-level representations
for part-of-speech tagging

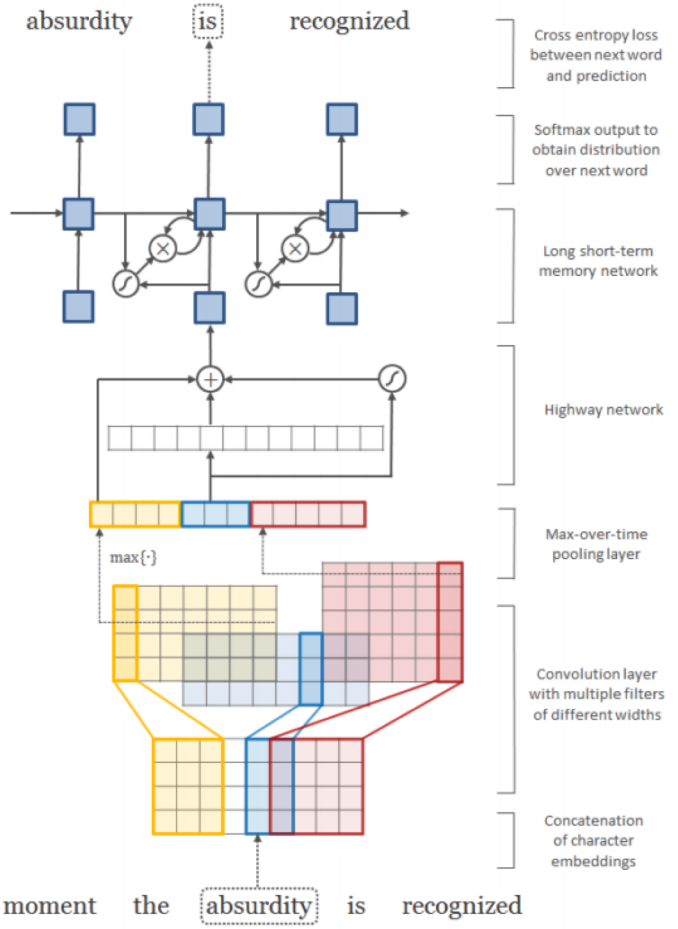
- Convolution over characters to generate word embeddings
- Fixed window of word embeddings used for PoS tagging



Dos Santos and Zadrozny (2014). http://proceedings.mlr.press/v32/santos14.html

Character-aware neural language models

- Character-based word embedding
- Utilizes convolution, highway network, and LSTM



Kim, Jernite, Sontag, and Rush 2015. https://arxiv.org/pdf/1508.06615.pdf

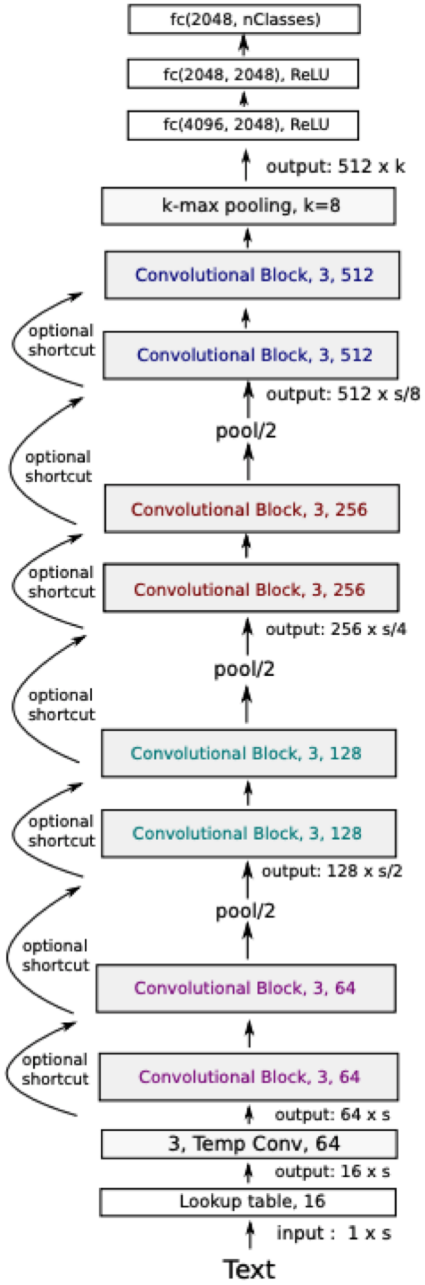
Very deep convolutional network for text classification

- Starting point: sequence models (LSTMs) have been very dominant in NLP; also CNNs, Attention, etc., but all the models are basically not very deep -- not like the deep models in Vision
- What happens when we build a vision-like system for NLP
- Works from the character level

Conneau, Schwenk, Lecun, Barrault. EACL 2017. https://www.aclweb.org/anthology/E17-1104

Very deep CNN architecture

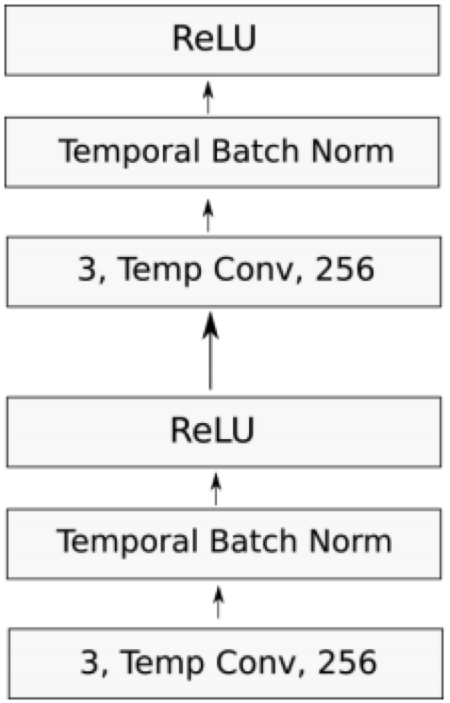
- The system very much looks like a vision system in its design, similar to VGGnet or ResNet.
- It looks very unlike a typical Deep Learning NLP system.
- Result is constant size, since text is truncated or padded
- Local pooling at each stage halves temporal resolution and doubles number of features



Conneau, Schwenk, Lecun, Barrault. EACL 2017. https://www.aclweb.org/anthology/E17-1104

Convolutional block in VD-CNN

- Each convolutional block is two convolutional layers, each followed by batch norm and a ReLU nonlinearity
- Convolutions of size 3
- Pad to preserve (or halve when local pooling) dimension



Conneau, Schwenk, Lecun, Barrault. EACL 2017. https://www.aclweb.org/anthology/E17-1104

Experiments

- Use large text classification datasets
  - Much bigger than the small datasets quite often used in NLP, such as in the Yoon Kim (2014) paper.

Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

Conneau, Schwenk, Lecun, Barrault. EACL 2017. https://www.aclweb.org/anthology/E17-1104

Experiments

Corpus:	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
Method	n-TFIDF	n-TFIDF	n-TFIDF	ngrams	Conv	Conv+RNN	Conv	Conv
Author	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Xiao]	[Zhang]	[Zhang]
Error	7.64	2.81	1.31	4.36	37.95*	28.26	40.43*	4.93*
[Yang]	-	-	-	-	-	24.2	36.4	-

Table 4: Best published results from previous work. Zhang et al. (2015) best results use a Thesaurus data augmentation technique (marked with an *). Yang et al. (2016)'s hierarchical methods is particularly adapted to datasets whose samples contain multiple sentences.

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

Table 5: Testing error of our models on the 8 data sets. No data preprocessing or augmentation is used.

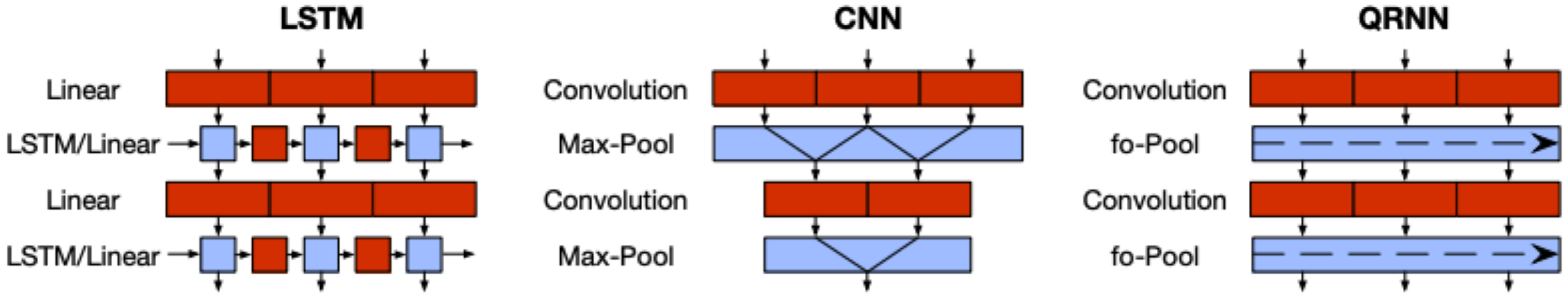
Conneau, Schwenk, Lecun, Barrault. EACL 2017. https://www.aclweb.org/anthology/E17-1104

RNNS are slow

- RNNs are a very standard building block for deep NLP
- But they parallelize badly and so are slow
- Idea: Take the best and parallelizable parts of RNNs and CNNs
- Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017

Quasi-Recurrent Neural Network

- Tries to combine the best of both model families



- Convolutions for parallelism across time:

$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

$$\longrightarrow$$

$$\mathbf{Z} = \tanh(\mathbf{W}_z * \mathbf{X})$$
$$\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$$
$$\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X}),$$

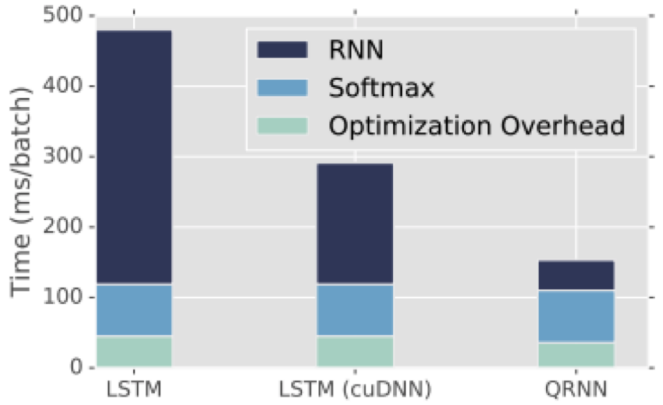
- Element-wise gated pseudo-recurrence for parallelism across channels is done in pooling layer: $\mathbf{h}_t = \mathbf{f}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \circ \mathbf{z}_t$

Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017.
https://arxiv.org/abs/1611.01576

Q-RNN experiments: language modeling

Model	Parameters	Validation	Test
LSTM (medium) (Zaremba et al., 2014)	20M	86.2	82.7
Variational LSTM (medium, MC) (Gal & Ghahramani, 2016)	20M	81.9	79.7
LSTM with CharCNN embeddings (Kim et al., 2016)	19M	–	78.9
Zoneout + Variational LSTM (medium) (Merity et al., 2016)	20M	84.4	80.6
Our models			
LSTM (medium)	20M	85.7	82.0
QRNN (medium)	18M	82.9	79.9
QRNN + zoneout ($p = 0.1$) (medium)	18M	82.1	78.3

Perplexity on validation and test sets.



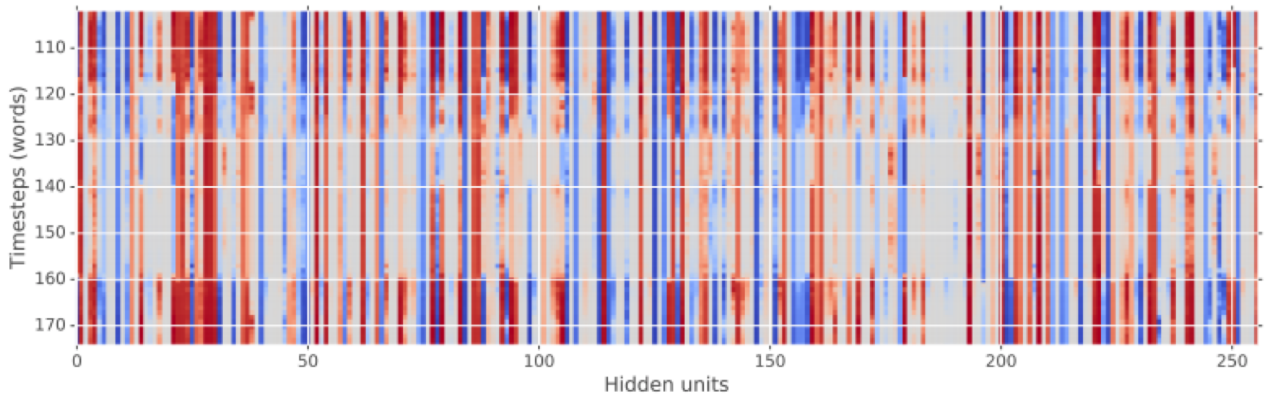
		Sequence length				
		32	64	128	256	512
Batch size	8	5.5x	8.8x	11.0x	12.4x	16.9x
	16	5.5x	6.7x	7.8x	8.3x	10.8x
	32	4.2x	4.5x	4.9x	4.9x	6.4x
	64	3.0x	3.0x	3.0x	3.0x	3.7x
	128	2.1x	1.9x	2.0x	2.0x	2.4x
	256	1.4x	1.4x	1.3x	1.3x	1.3x

Faster

Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017.
https://arxiv.org/abs/1611.01576

Q-RNN for sentiment analysis

- Often better and faster than LSTMs
- More interpretable
- Example: Initial positive review, review starts out positive
  - At 117: “not exactly a bad story”
  - at 158: “I recommend this movie to everyone, even if you’ve never played the game”



Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017.
https://arxiv.org/abs/1611.01576

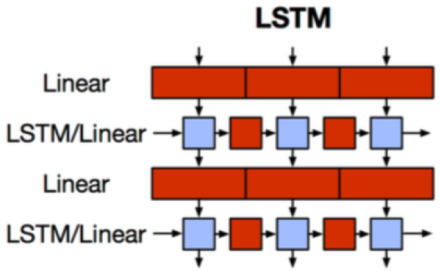
Q-RNN limitations

- Didn't work for character-level LMs as well as LSTMs
  - Trouble modeling much longer dependencies?
- Often need deeper network to get as good performance as LSTM
  - They're still faster when deeper
  - Effectively they use depth as a substitute for true recurrence

Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017.
https://arxiv.org/abs/1611.01576

Problems with RNNs & Motivation for Transformers

- We want parallelization but RNNs are inherently sequential



- Despite GRUs and LSTMs, RNNs still **gain from attention mechanism** to deal with long range dependencies – **path length** between states grows with sequence otherwise
- But if attention gives us access to any state ... maybe we don't need the RNN?

Quasi-Recurrent Neural Networks by James Bradbury, Stephen Merity, Caiming Xiong & Richard Socher. ICLR 2017.
https://arxiv.org/abs/1611.01576

STEVENS

INSTITUTE *of* TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu

Thank You