Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 11: ConvNets for NLP

Lecture Plan

Lecture 11: ConvNets for NLP

1. Announcements (5 mins)
2. Intro to CNNs (20 mins)
3. Simple CNN for Sentence Classification: Yoon (2014) (20 mins)
4. CNN potpourri (5 mins)
5. Deep CNN for Sentence Classification: Conneau et al. (2017) (10 mins)
6. If I have extra time the stuff I didn’t do last week …

1. Announcements

- Complete mid-quarter feedback survey by tonight (11:59pm PST) to receive 0.5% participation credit!

- Project proposals (from ***every*** team) due this Thursday 4:30pm
  - A dumb way to use late days!
  - We aim to return feedback next Thursday

- Final project poster session: Mon Mar 16 evening, Alumni Center
  - Groundbreaking research!
  - Prizes!
  - Food!
  - Company visitors!

Welcome to the second half of the course!

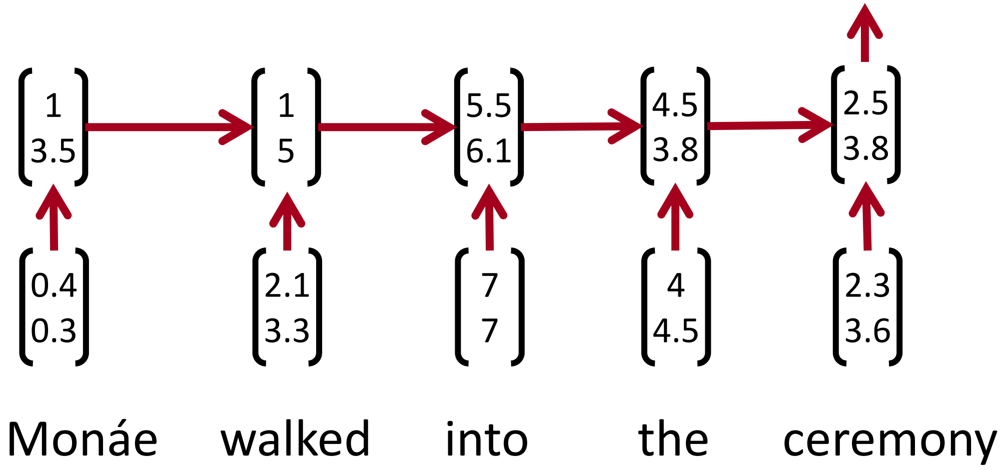
- Now we’re preparing you to be **real** DL+NLP researchers/practitioners!

- Lectures won’t always have all the details
  - It's up to you to search online / do some reading to find out more
  - This is an active research field! Sometimes there’s no clear-cut answer
  - Staff are happy to discuss things with you, but you need to think for yourself

- Assignments are designed to ramp up to the real difficulty of project
  - Each assignment deliberately has less scaffolding than the last
  - In projects, there’s no provided autograder or sanity checks
  - → DL debugging is hard but you need to learn how to do it!

2. 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 is often only calculated at the last step

From RNNs to Convolutional Neural Nets

- Main 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)

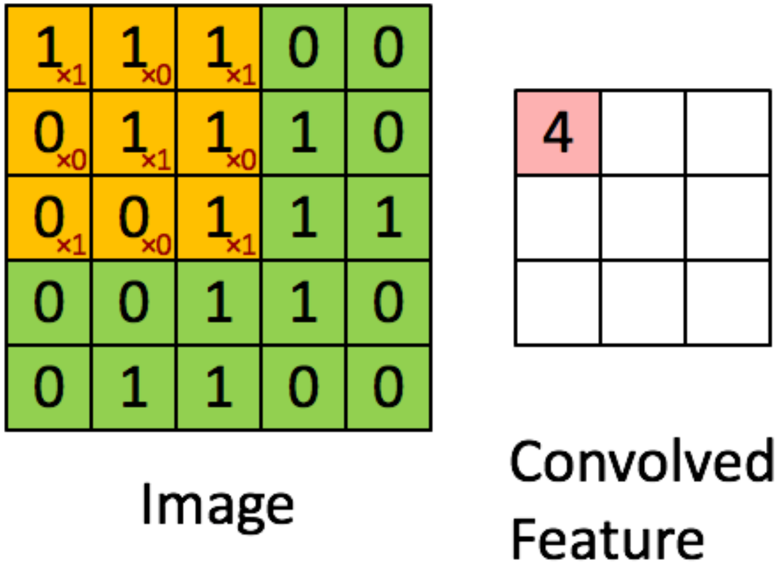
CNNs

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



$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

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

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

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

3	1	2	−3
−1	2	1	−3
1	1	−1	1

+ 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

∅	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

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

∅	**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

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

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

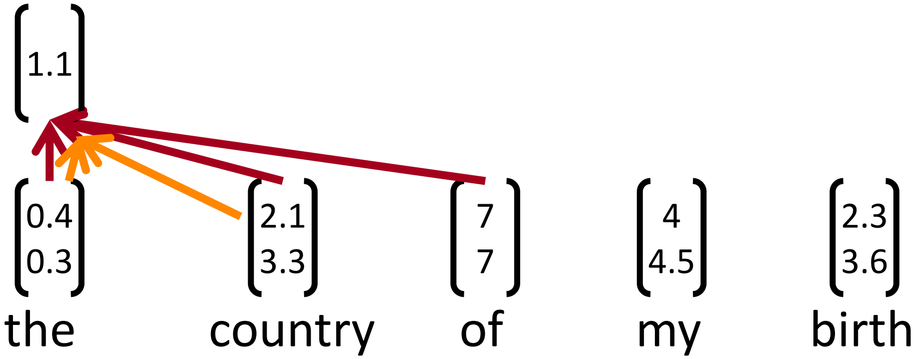
3. Single Layer CNN for Sentence Classification

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014. https://arxiv.org/pdf/1408.5882.pdf
  Code: https://arxiv.org/pdf/1408.5882.pdf [Theano!, etc.]
- A variant of convolutional NNs of Collobert, Weston et al. (2011) Natural Language Processing (almost) from Scratch.

- 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:

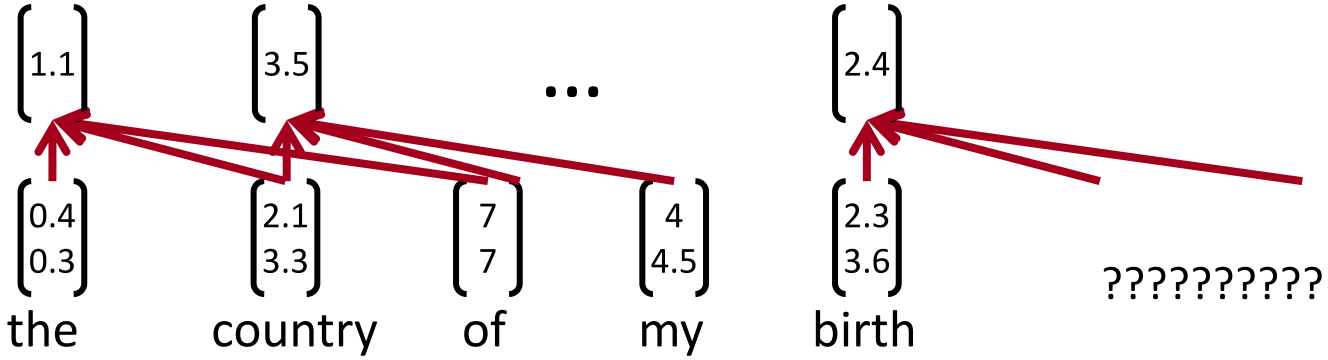


Single layer CNN

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

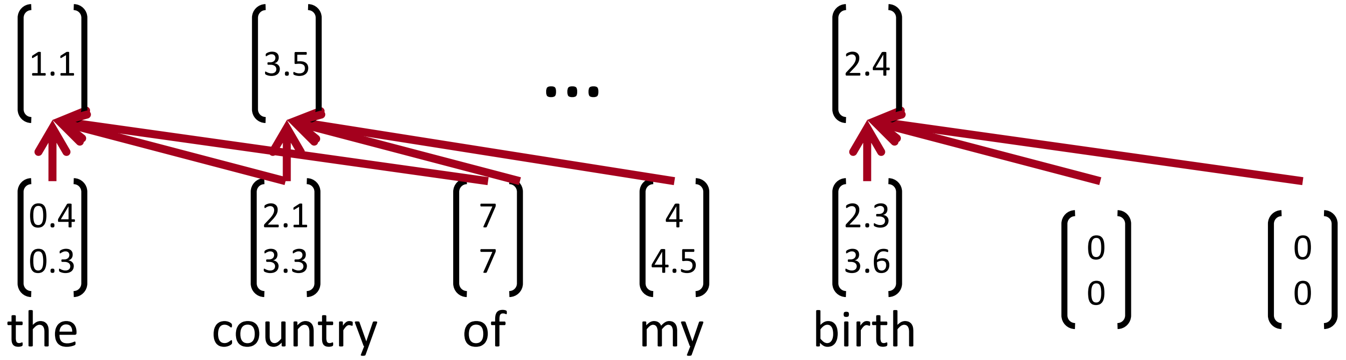


Single layer CNN

- 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: $\hat{c} = \max\{\mathbf{c}\}$

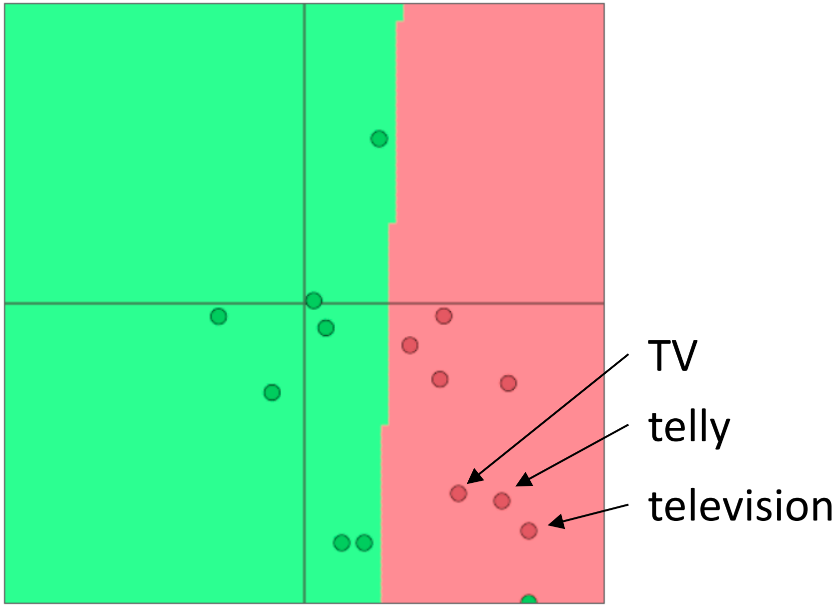
- Use multiple filter weights **w** (i.e. multiple channels)
- Useful to have different window sizes *h*
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** irrelevant

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

- So we could have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

A pitfall when fine-tuning word vectors

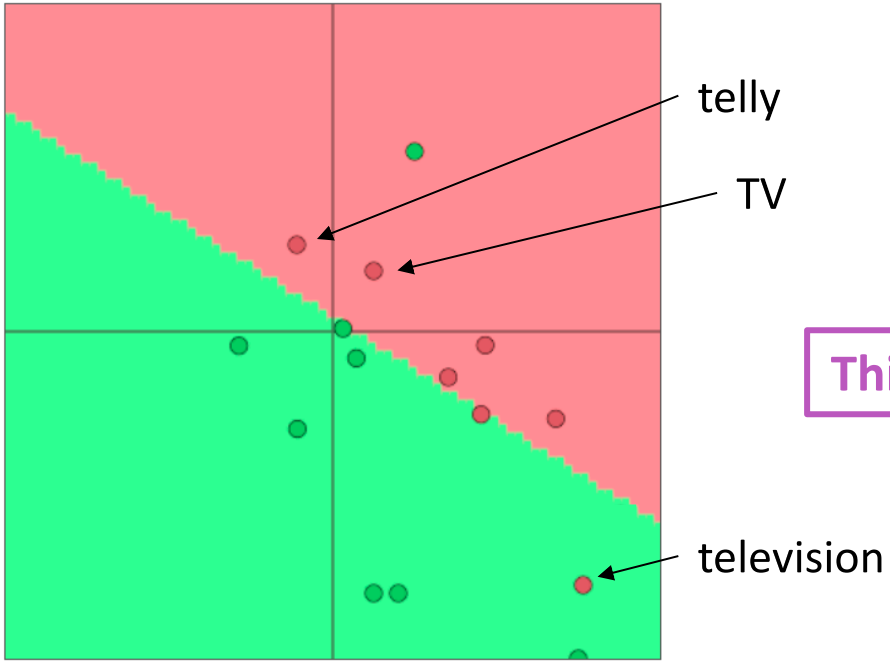
- **Setting:** We are training a logistic regression classification model for movie review sentiment using single words.
- In the training data we have “TV” and “telly”
- In the testing data we have “television”
- The pre-trained word vectors have all three similar:



- **Question: What happens when we update the word vectors?**

A pitfall when fine-tuning word vectors

- **Question:** What happens when we update the word vectors?
- **Answer:**
  - Those words that are in the training data move around
    - "TV" and "telly"
  - Words not in the training data stay where they were
    - "television"



This can be bad!

So what should I do?

- **Question:** Should I use available "pre-trained" word vectors
  Answer:
  - Almost always, yes!
  - They are trained on a huge amount of data, and so they will know about words not in your training data and will know more about words that are in your training data
  - Have 100s of millions of words of data? Okay to start random
- **Question:** Should I update ("fine tune") my own word vectors?
- **Answer:**
  - If you only have a small training data set, don't train the word vectors
  - If you have have a large dataset, it probably will work better to train = update = fine-tune word vectors to the task

Multi-channel input idea

- Initialize with pre-trained word vectors (word2vec or Glove)

- Start with two copies

- Backprop into only one set, keep other "static"

- 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

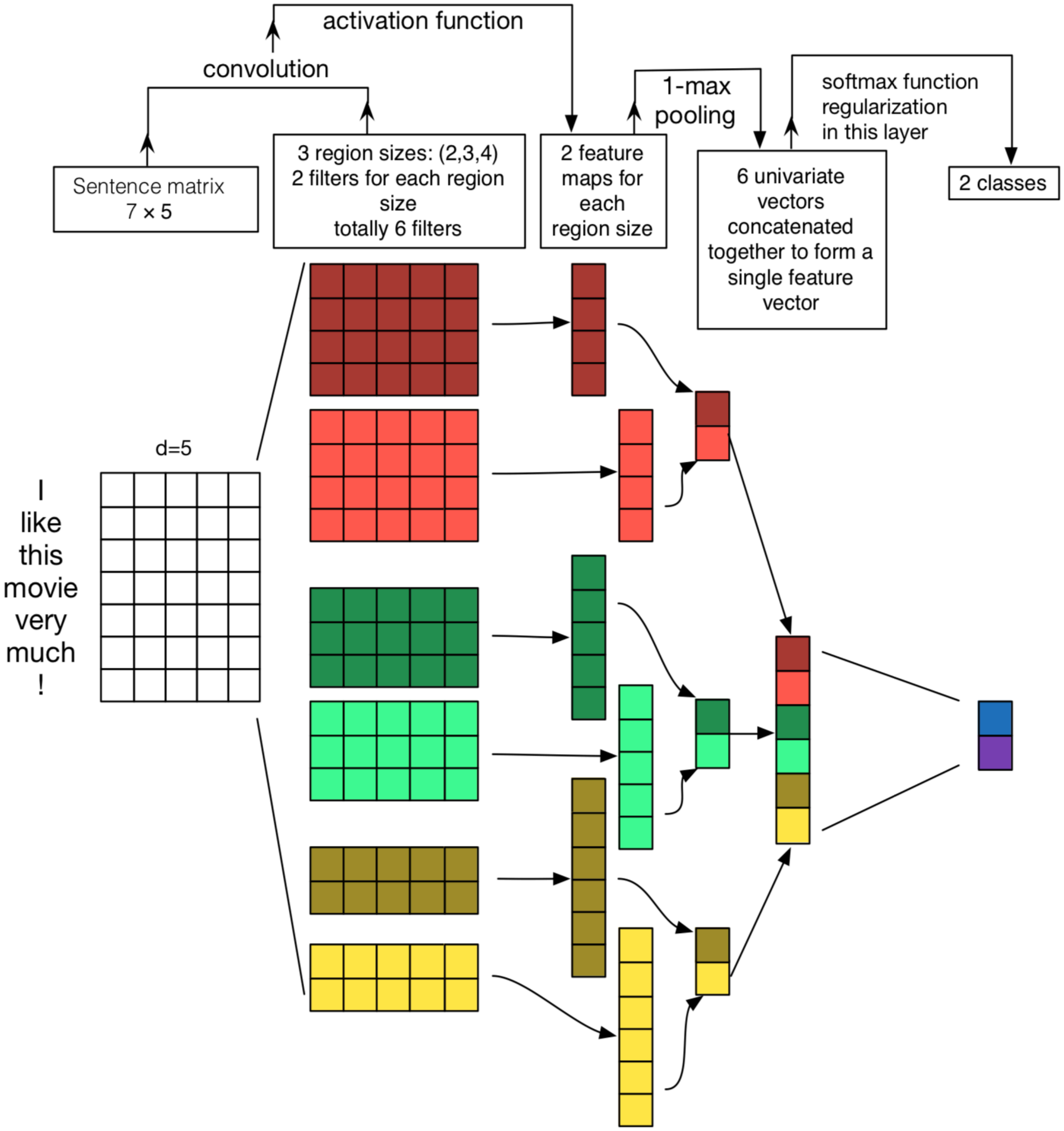
$$y = softmax\left(W^{(S)}z + b\right)$$

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)



activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Regularization

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

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

- 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*

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

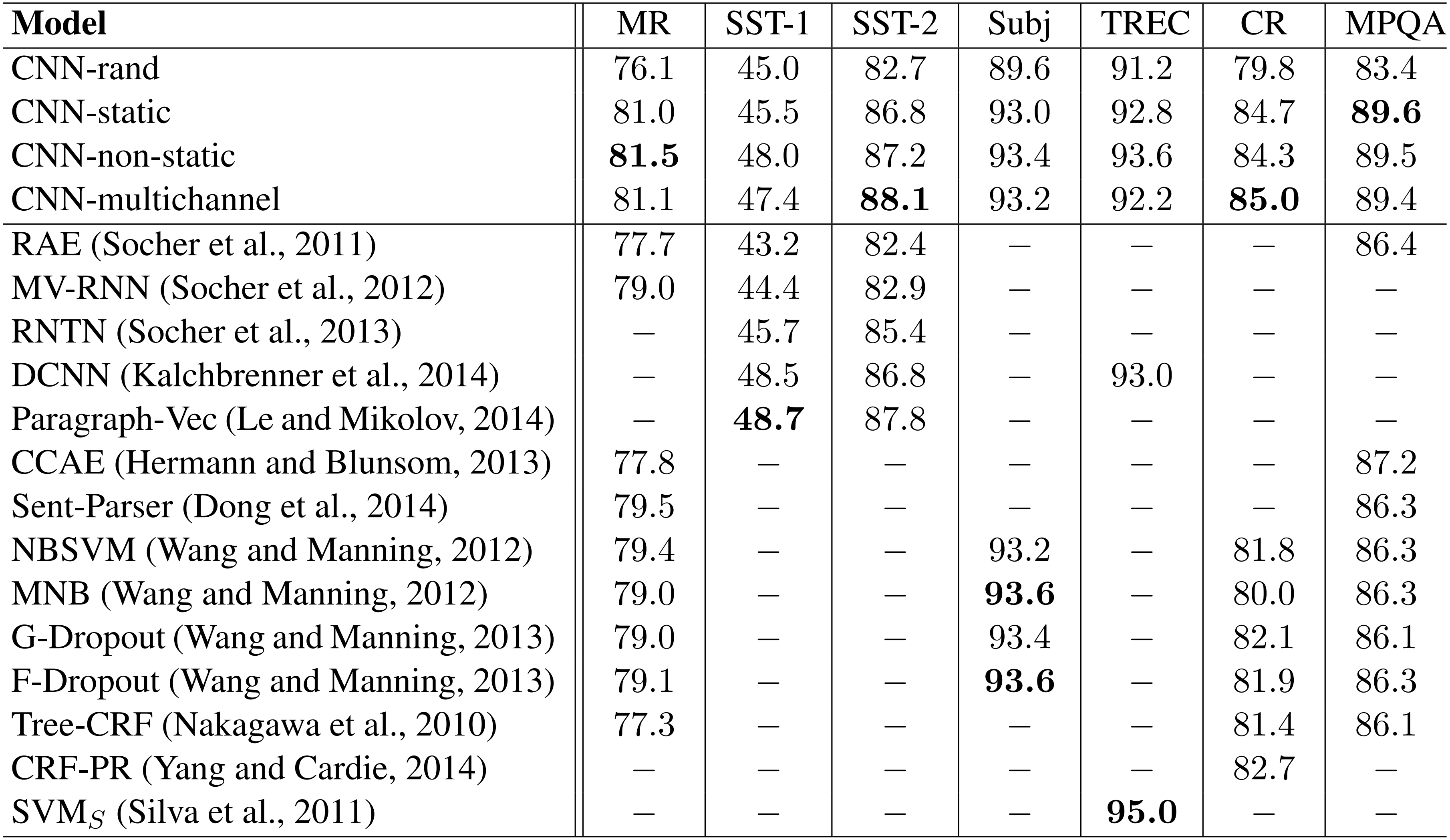
- **Also:** Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number *s* (also a hyperparameter)
- If $\|W_{c\cdot}^{(S)}\| > s$, then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$
- Not very common

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 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!

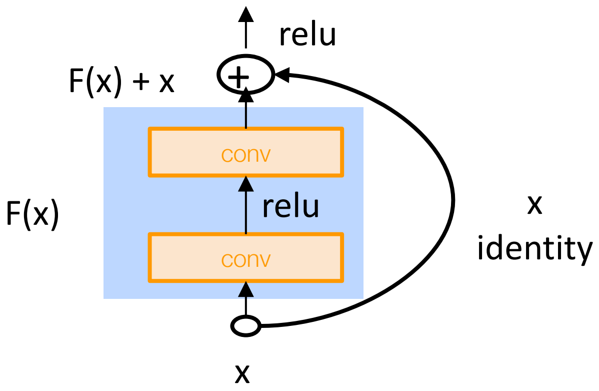
- Differences to window and RNN architectures we described in previous lectures: pooling, many filters, and dropout
- Some of these ideas can be used in RNNs too

4. 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, somewhat implausible/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

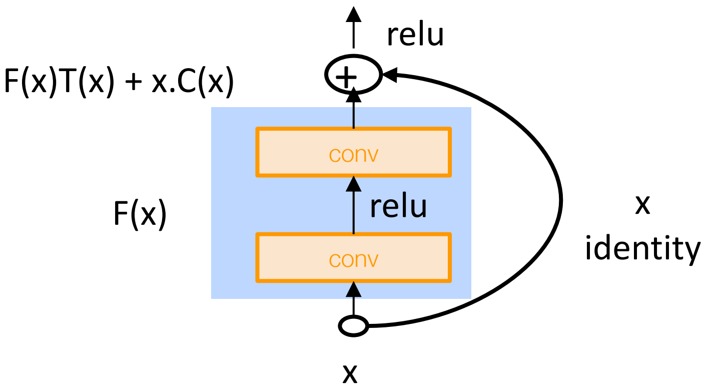
Gated units used vertically

- 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)

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



Highway block
(Srivistava et al. NeurIPS 2015)

Note: can set C(x) = (1 – T(x)) more like GRU

Batch Normalization (BatchNorm)

[Ioffe and Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.]

- 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 fluctuations 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

- Related but different: LayerNorm, standard in Transformers

Size 1 Convolutions

[Lin, Chen, and Yan. 2013. Network in network. arXiv:1312.4400.]

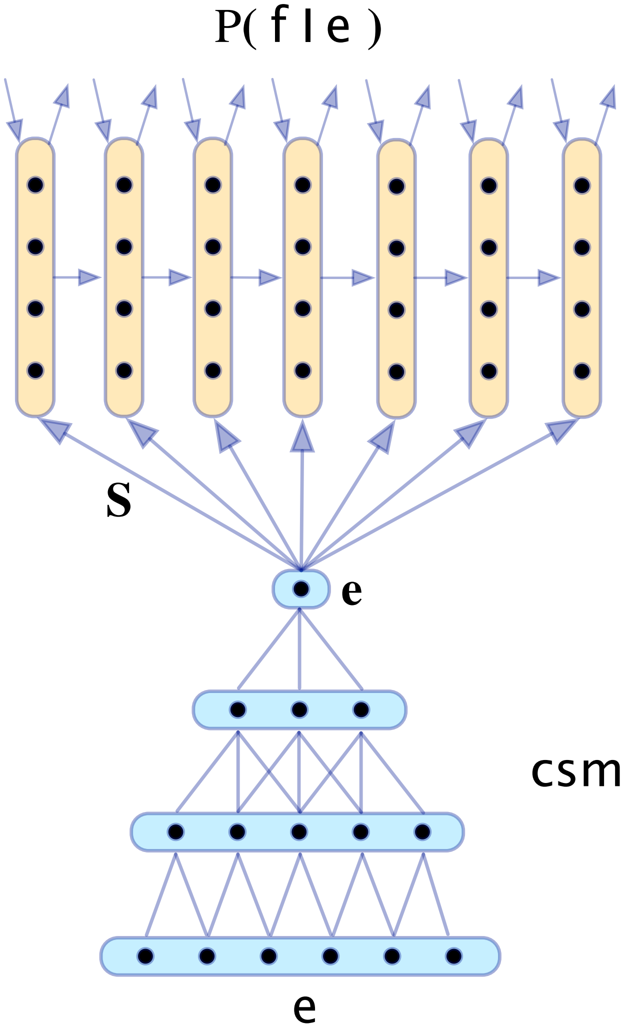
- **Does this concept make sense?!? Yes.**
- Size 1 convolutions ("1x1"), a.k.a. Network-in-network (NiN) connections, are convolutional kernels with kernel_size=1
- A size 1 convolution gives you a fully connected linear layer across channels!
- It can be used to map from many channels to fewer channels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

CNN application: Translation

- One of the first successful neural machine translation efforts

- Uses CNN for encoding and RNN for decoding

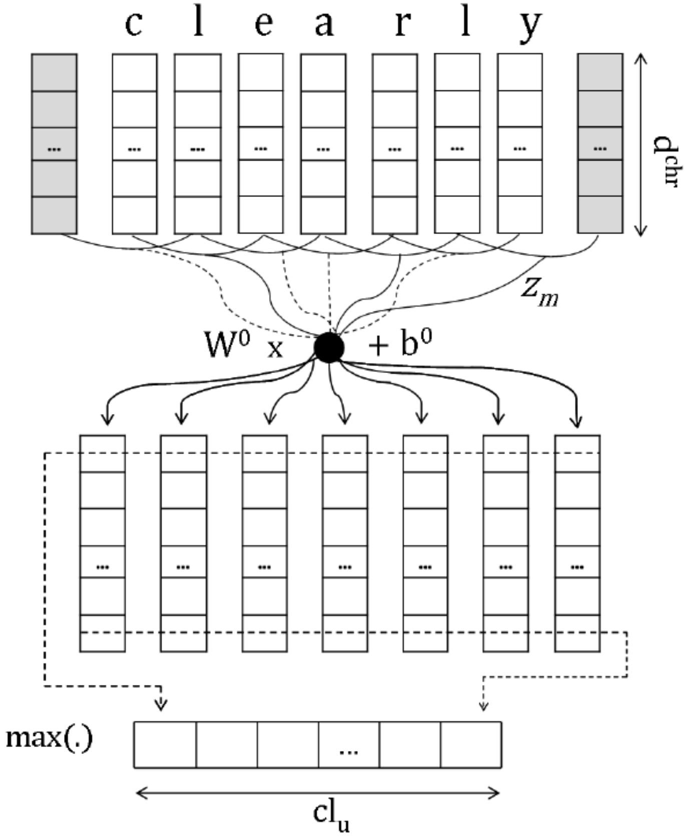
- Kalchbrenner and Blunsom (2013)
  “Recurrent Continuous Translation Models”



Learning Character-level Representations for Part-of-Speech Tagging

Dos Santos and Zadrozny (2014)

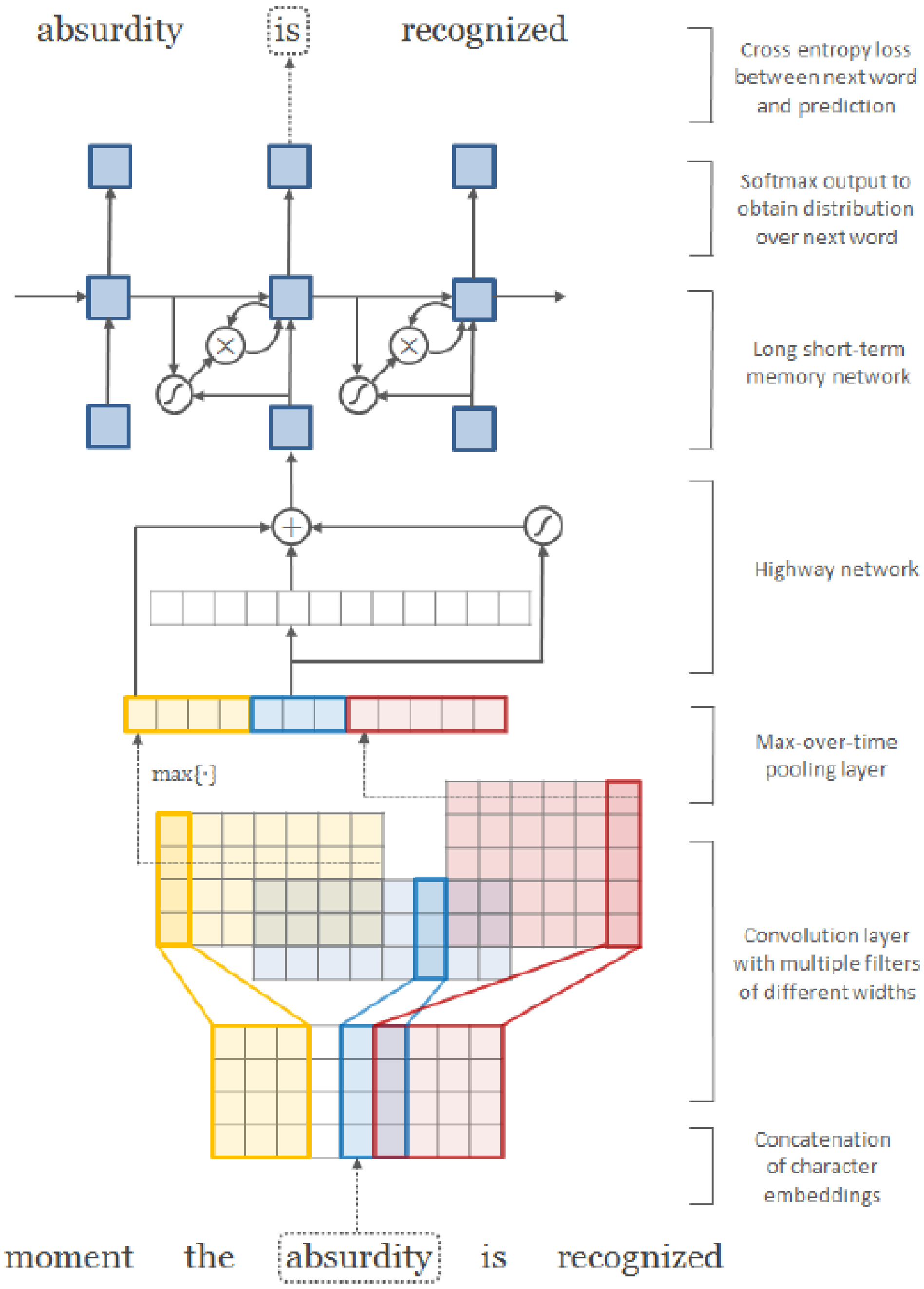
- Convolution over characters to generate word embeddings
- Fixed window of word embeddings used for PoS tagging



Character-Aware Neural Language Models

(Kim, Jernite, Sontag, and Rush 2015)

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


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

5. Very Deep Convolutional Networks for Text Classification

- Conneau, Schwenk, Lecun, Barrault. EACL 2017.
- 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

VD-CNN architecture

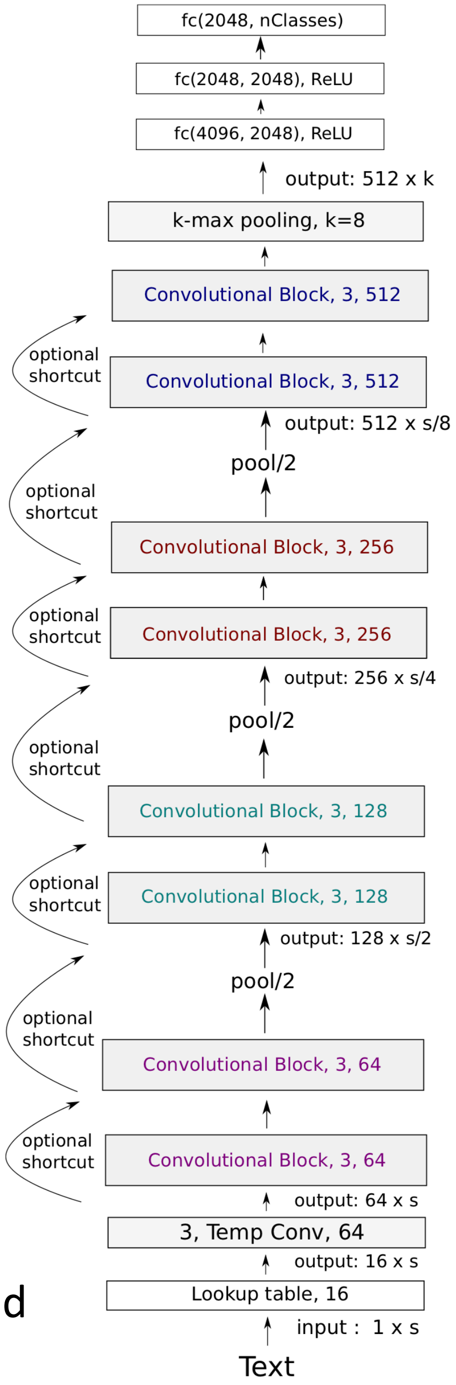
The system very much looks like a vision system in its design, similar to VGGnet or ResNet

It looks unlike most typical Deep Learning NLP systems

Result is constant size, since text is truncated or padded

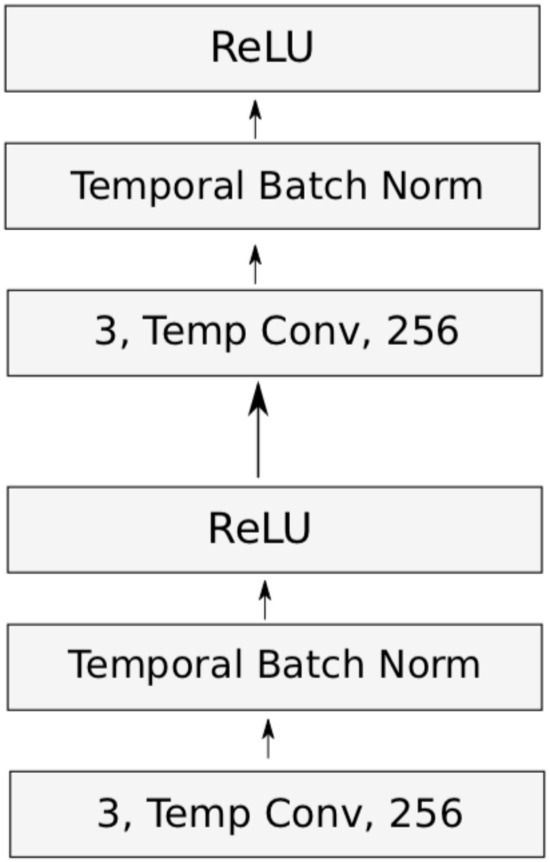
Local pooling at each stage halves temporal resolution and doubles number of features

s = 1024 chars; 16d embed



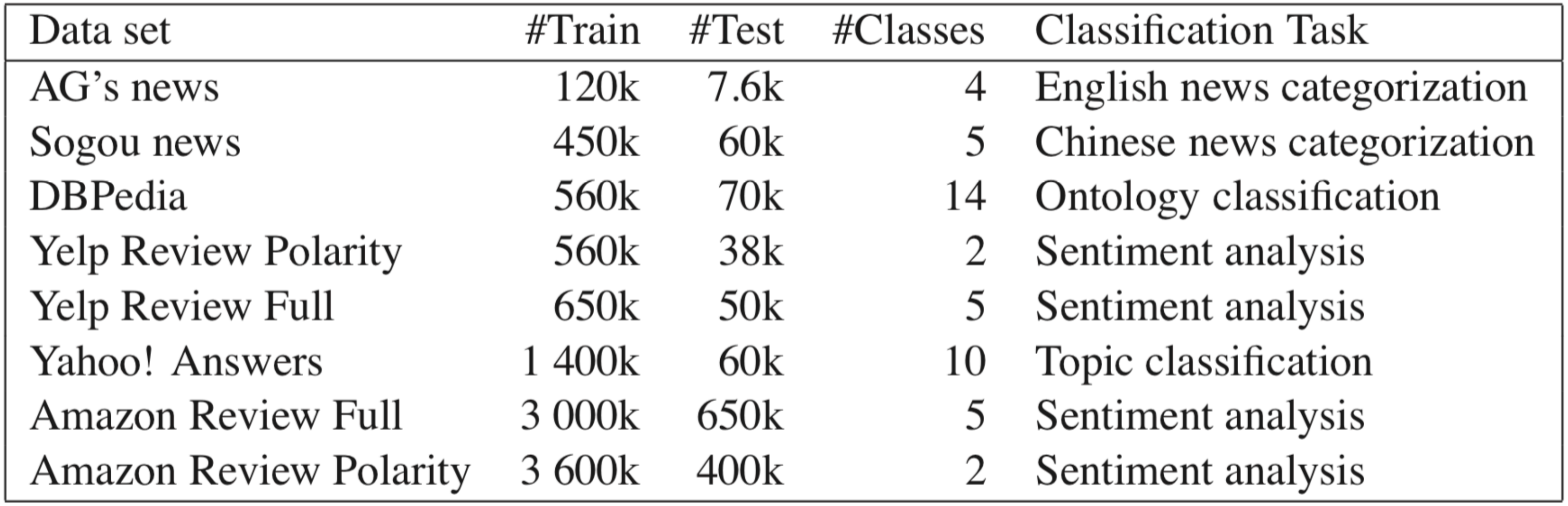
Convolutional block in VD-CNN

- 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



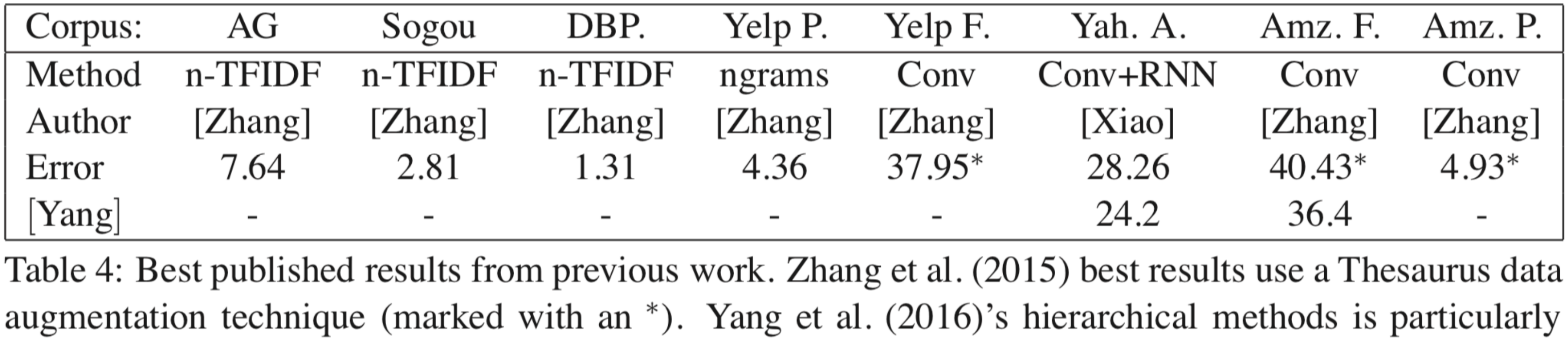
Experiments

- Use large text classification datasets
  - Much bigger than the small datasets used in the Yoon Kim (2014) paper



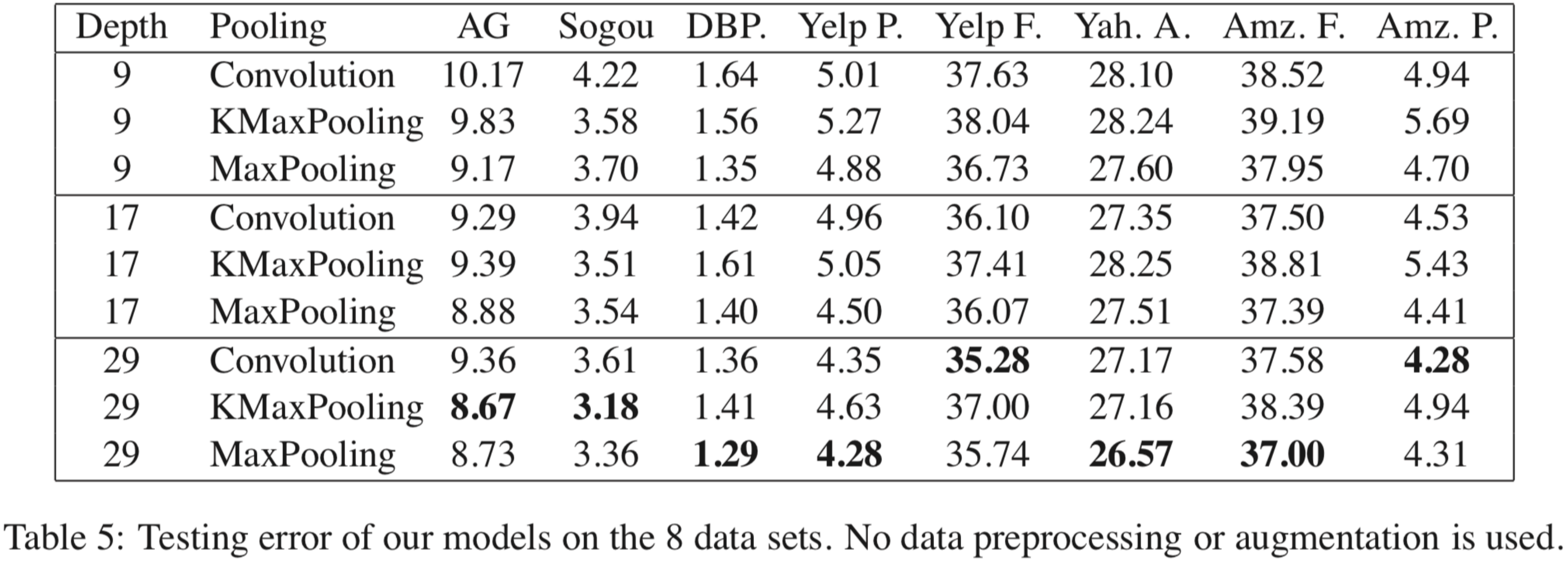
Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers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

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



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

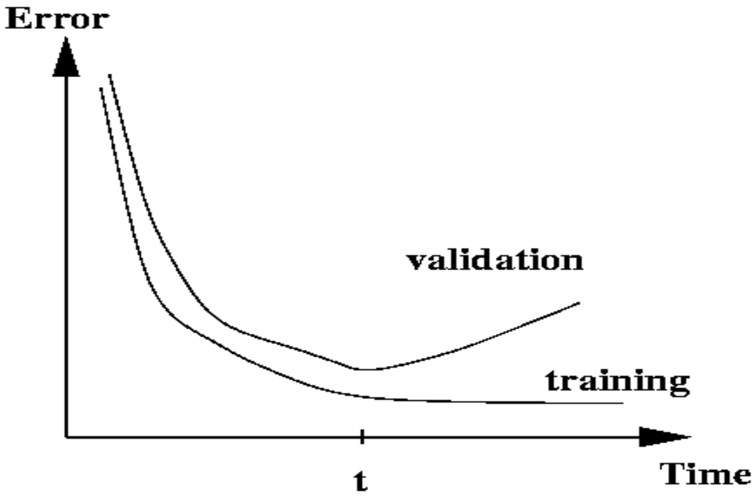
Table 5: Testing error of our models on the 8 data sets. No data preprocessing or augmentation is used.

7. Pots of data

- Many publicly available datasets are released with a **train/dev/test** structure. **We're all on the honor system to do test-set runs only when development is complete.**
- Splits like this presuppose a fairly large dataset.
- If there is no dev set or you want a separate tune set, then you create one by splitting the training data, though you have to weigh its size/usefulness against the reduction in train-set size.
- Having a fixed test set ensures that all systems are assessed against the same gold data. This is generally good, but:
  - It is problematic where the test set turns out to have unusual properties that distort progress on the task.
  - It doesn't give any measure of variance.
  - It's only an unbiased estimate of the mean if only used once.

Training models and pots of data

- When training, models **overfit** to what you are training on
  - The model correctly describes what happened to occur in particular data you trained on, but the patterns are not general enough patterns to be likely to apply to new data
- The way to avoid problematic overfitting (lack of generalization) is using **independent** validation and test sets …



Training models and pots of data

- You build (estimate/train) a model on a **training set.**
- Often, you then set further hyperparameters on another, independent set of data, the **tuning set**
  - The tuning set is the training set for the hyperparameters!
- You measure progress as you go on a **dev set** (development test set or validation set)
  - If you do that a lot you overfit to the dev set so it can be good to have a second dev set, the **dev2** set
- **Only at the end**, you evaluate and present final numbers on a **test set**
  - Use the final test set **extremely** few times … ideally only once

Training models and pots of data

- The **train**, **tune**, **dev**, and **test** sets need to be completely distinct
- It is invalid to test on material you have trained on
  - You will get a falsely good performance. We usually overfit on train
- You need an independent tuning set
  - The hyperparameters won't be set right if tune is same as train
- If you keep running on the same evaluation set, you begin to overfit to that evaluation set
  - Effectively you are "training" on the evaluation set … you are learning things that do and don't work on that particular eval set and using the info
- To get a valid measure of system performance you need another untrained on, **independent** test set … hence dev2 and final test

8. Getting your neural network to train

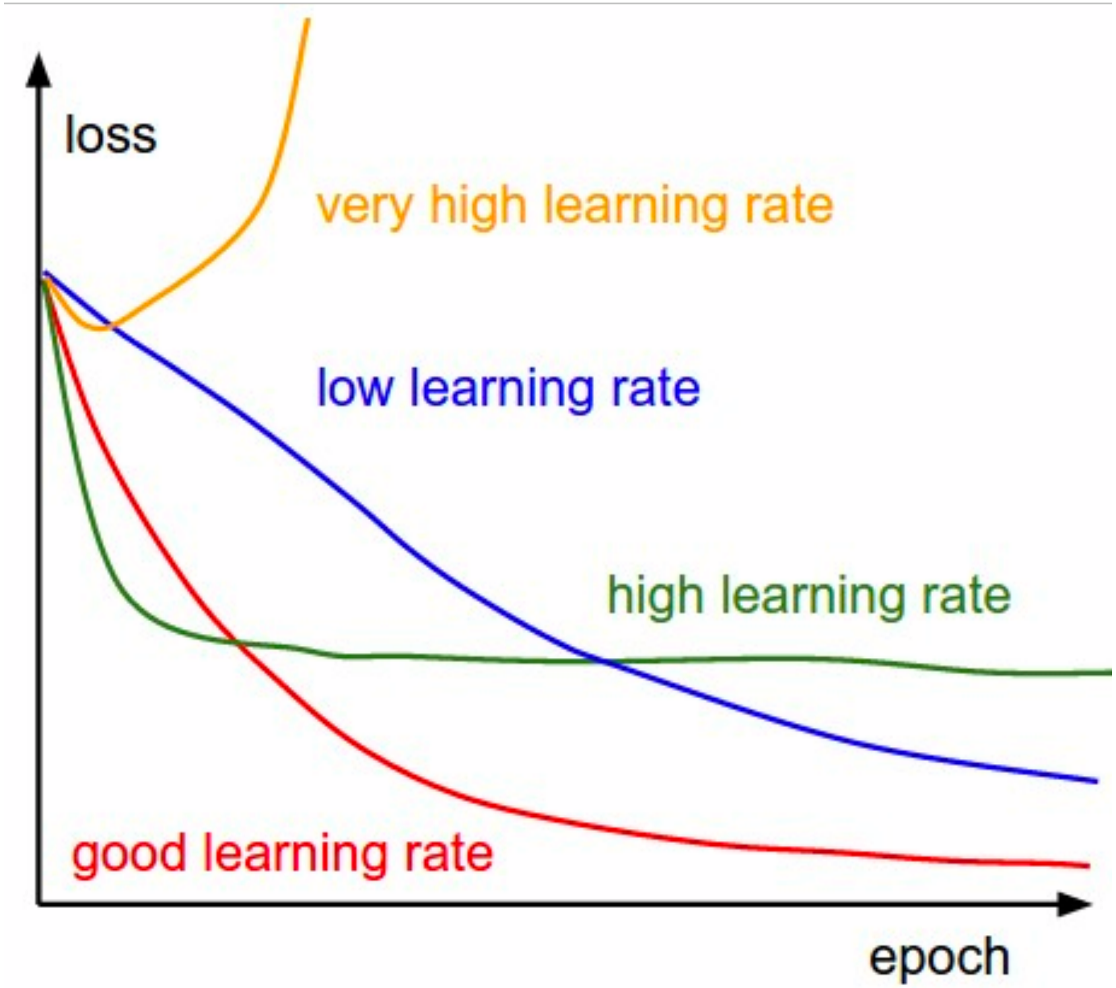
- Start with a positive attitude!
  - **Neural networks want to learn!**
    - If the network isn’t learning, you’re doing something to prevent it from learning successfully

- Realize the grim reality:
  - **There are lots of things that can cause neural nets to not learn at all or to not learn very well**
    - Finding and fixing them (“debugging and tuning”) can often take more time than implementing your model

- It’s hard to work out what these things are
  - But experience, experimental care, and rules of thumb help!

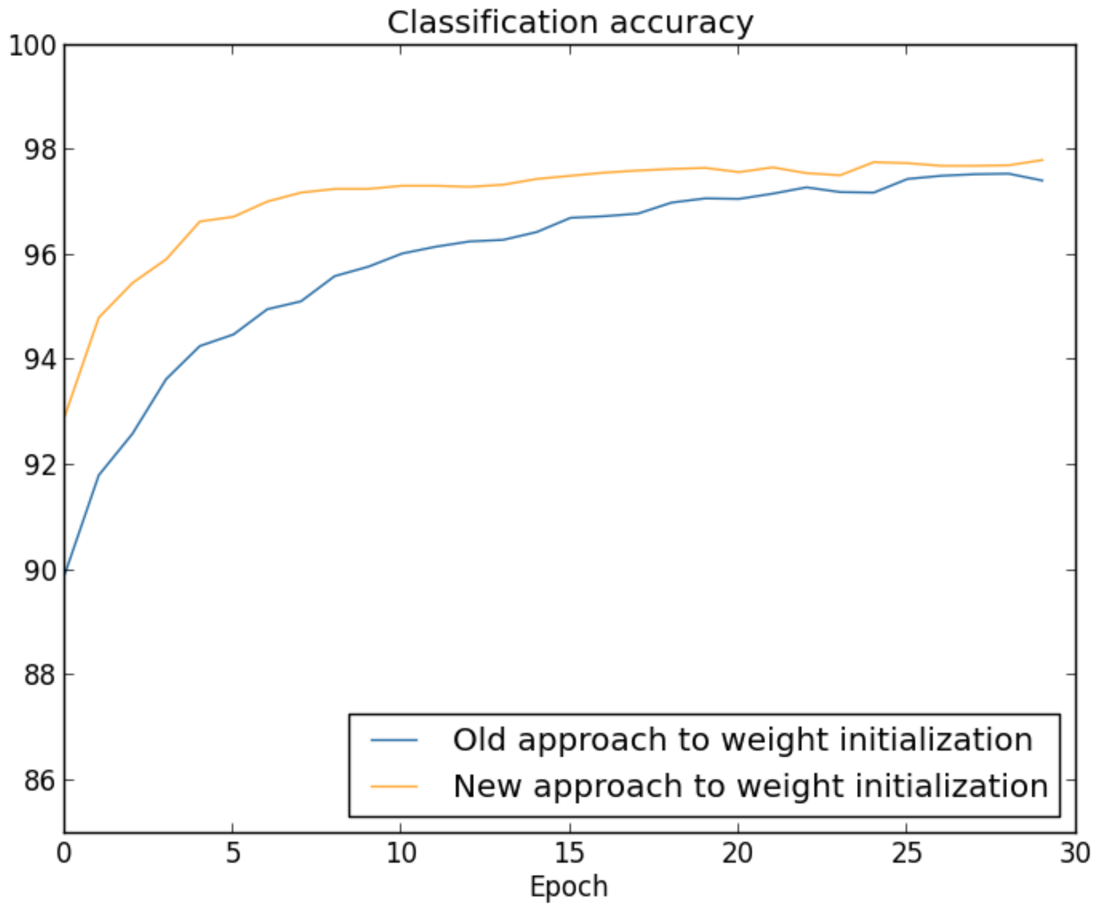
Models are sensitive to learning rates

- From Andrej Karpathy, CS231n course notes



Models are sensitive to initialization

- From Michael Nielsen http://neuralnetworksanddeeplearning.com/chap3.html



Training a gated RNN

1. Use an LSTM or GRU: *it makes your life so much simpler!*
2. Initialize recurrent matrices to be orthogonal
3. Initialize other matrices with a sensible (**small!**) scale
4. Initialize forget gate bias to 1: *default to remembering*
5. Use adaptive learning rate algorithms: *Adam, AdaDelta, …*
6. Clip the norm of the gradient: *1–5 seems to be a reasonable threshold when used together with Adam or AdaDelta.*
7. Either only dropout vertically or look into using Bayesian Dropout (Gal & Gahramani – can do but not natively in PyTorch)
8. *Be patient! Optimization takes time*

[Saxe et al., ICLR2014;
Ba, Kingma, ICLR2015;
Zeiler, arXiv2012;
Pascanu et al., ICML2013]

Experimental strategy

- Work incrementally!
- Start with a very simple model and get it to work!
  - It’s hard to fix a complex but broken model
- Add bells and whistles one-by-one and get the model working with each of them (or abandon them)

- Initially run on a tiny amount of data
  - You will see bugs much more easily on a tiny dataset
  - Something like 4–8 examples is good
  - Often synthetic data is useful for this
  - Make sure you can get 100% on this data
    - Otherwise your model is definitely either not powerful enough or it is broken

Experimental strategy

- Run your model on a large dataset
  - It should still score close to 100% on the training data after optimization
    - Otherwise, you probably want to consider a more powerful model
    - Overfitting to training data is **not** something to be scared of when doing deep learning
      - These models are usually good at generalizing because of the way distributed representations share statistical strength regardless of overfitting to training data
- But, still, you now want good generalization performance:
  - Regularize your model until it doesn’t overfit on dev data
    - Strategies like L2 regularization can be useful
    - But normally **generous dropout** is the secret to success

Details matter!

- Be very familiar with your (train and dev) data, don’t treat it as arbitrary bytes in a file!
  - Look at your data, collect summary statistics
  - Look at your model’s outputs, do error analysis
- Tuning hyperparameters is **really** important to almost all of the successes of NNets

Good luck with your projects!