# Modelling Sentence Pair Similarity with Multi-Perspective Convolutional Neural Networks

ZHUCHENG TU | CS 898 SPRING 2017

JULY 17, 2017

# Outline

Motivation
- Why do we want to model sentence similarity?
- Challenges

Existing Work on Sentence Modeling

Multi-Perspective CNN

Modifications and Results

Future Work

# Motivation

Modeling the similarity of a pair of sentences is critical to many NLP tasks:

- Paraphrase identification, ex. plagiarism detection or detecting duplicate questions
- Question answering, ex. answer selection
- Query ranking

# What makes sentence modelling hard?

- Different ways of saying the same thing
- Little annotated training data
  - “Difficult to use sparse, hand-crafted features as in conventional approaches in NLP”(He et al., 2015)

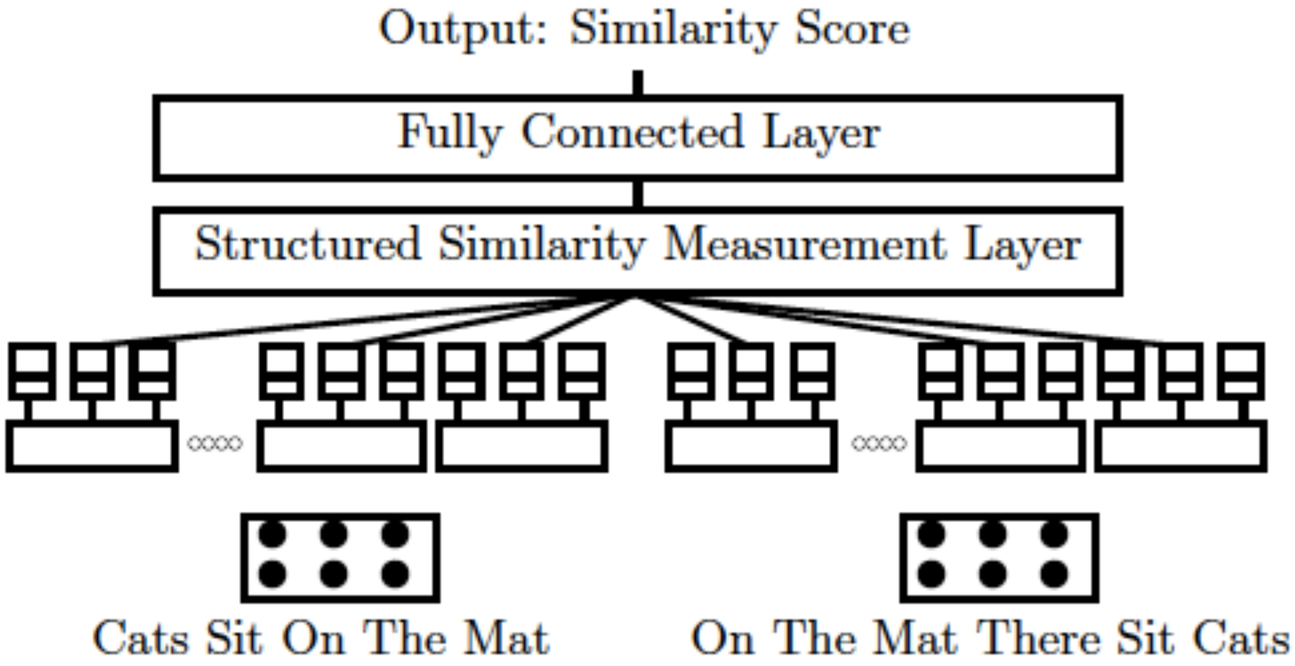
# Existing Work

- Before deep learning methods, methods included
  - N-gram overlap on word and characters
  - Knowledge-based, e.g. using WordNet
  - …
  - Combinations of these methods and multi-task learning
- Deep learning methods:
  - Collobert and Weston (2008) trained CNN in multitask setting
  - Kalchbrenner et al. (2014) used dynamic k-max pooling to handle variable sized input
  - Kim (2014) used fixed & learned word vectors and varying window sizes & convolution filters
  - more CNNs…
  - Tai et al. (2015) and Zhu et al. (2015) used tree-based LSTM

# Multi-Perspective CNN

- Based on: Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-Perspective sentence similarity modeling with convolutional neural networks. *In Proceedings of EMNLP*, pages 1576–1586.
- Compare sentence pairs using a “multiplicity of perspectives”
- Two components: **sentence model** and **similarity measurement layer**
- Advantages:
  - Do not use syntax parsers
  - Do not need unsupervised pre-training step

# Multi-Perspective CNN Architecture




Sentence Model

# Preparing Input

- Use **GloVe** (840B tokens, 2.2M vocab, 300d vectors) to create sentence embedding
- Use values from **Normal(0, 1)** for words not found in vocab
- Pad sentence embedding to create uniformly-sized batches for faster GPU training

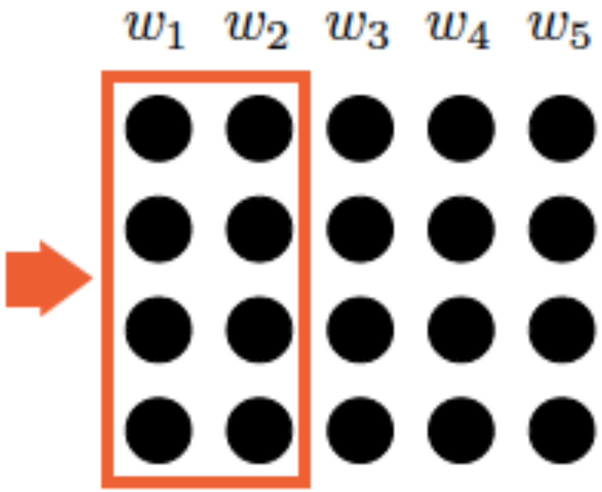
A group of kids is playing in a yard and an old man is standing in the background

A group of boys in a yard is playing and a man is standing in the background

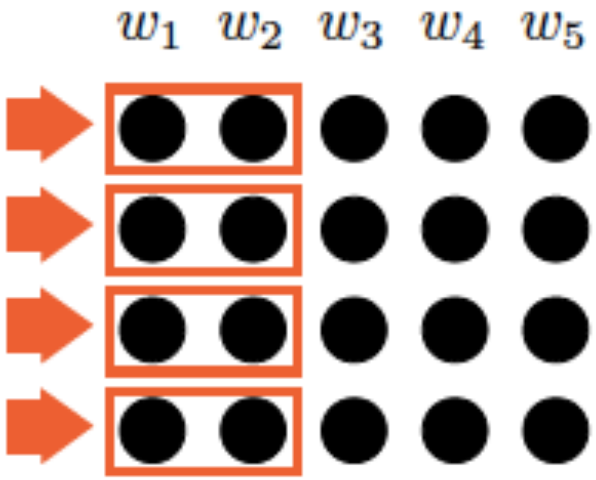
# Sentence Modelling: Multi-Perspective Convolution

Two types of convolution for each sentence

**Holistic filters**

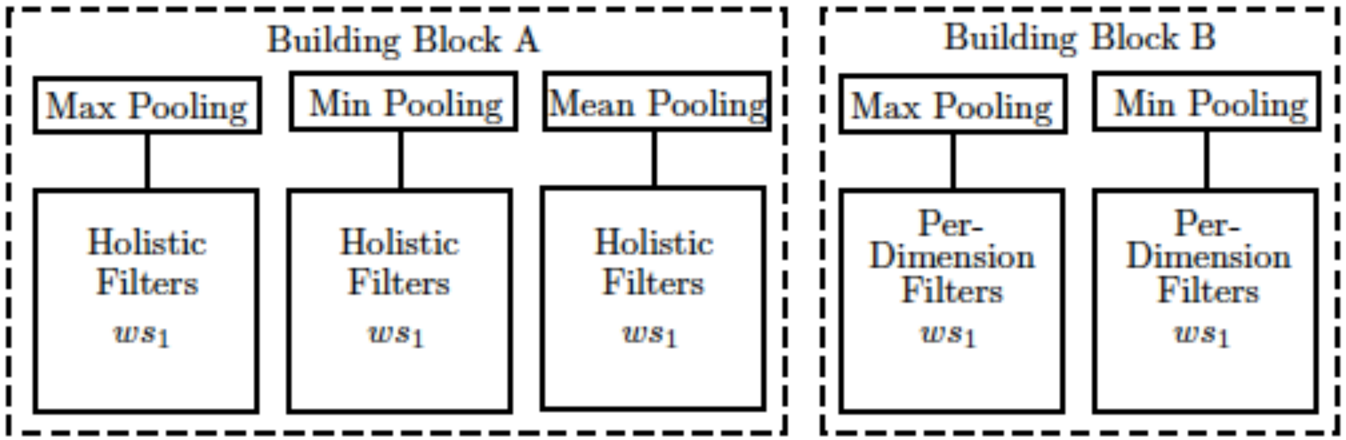



**Per-dimensional filters**



# Sentence Modeling: Multiple Pooling

Multiple types of pooling for type of convolution, we call the group of filters for a particular convolution type a “Block”



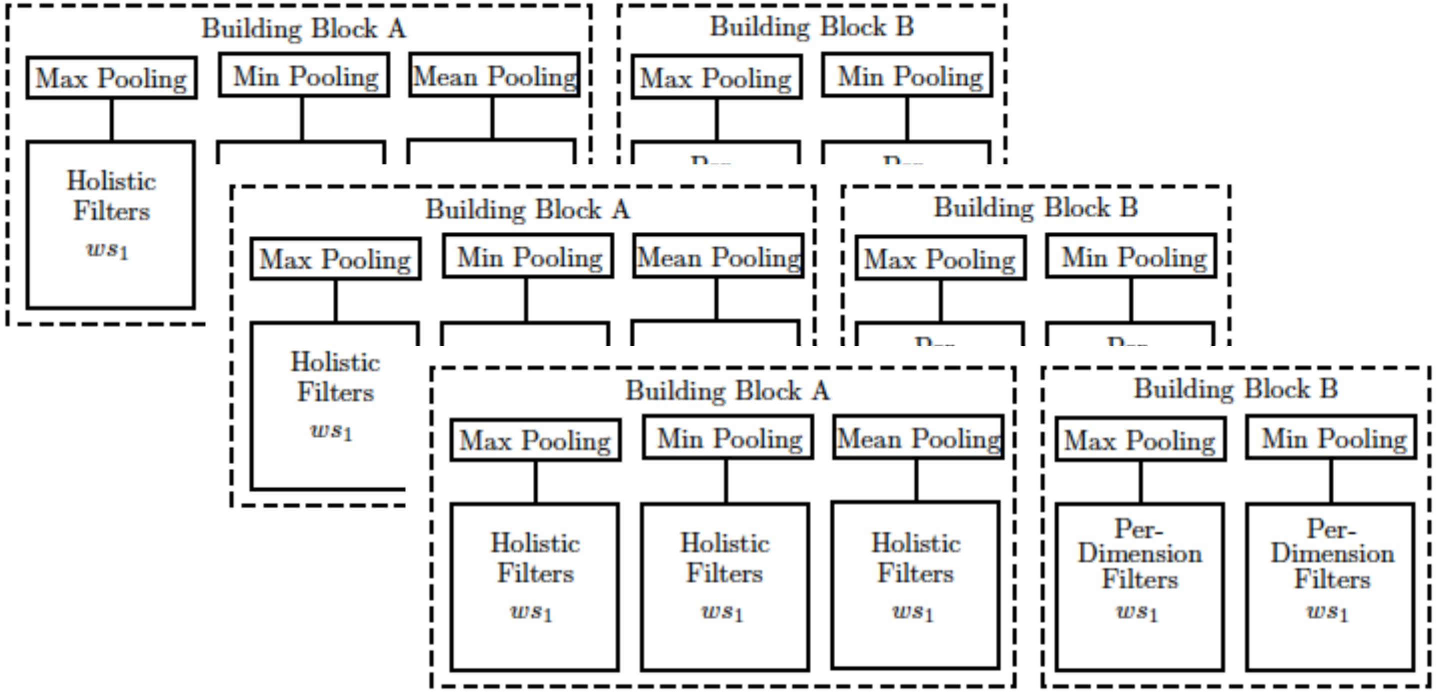
# Sentence Modeling: Multiple Window Sizes

Multiple blocks, each corresponding to a particular width

ws = 1

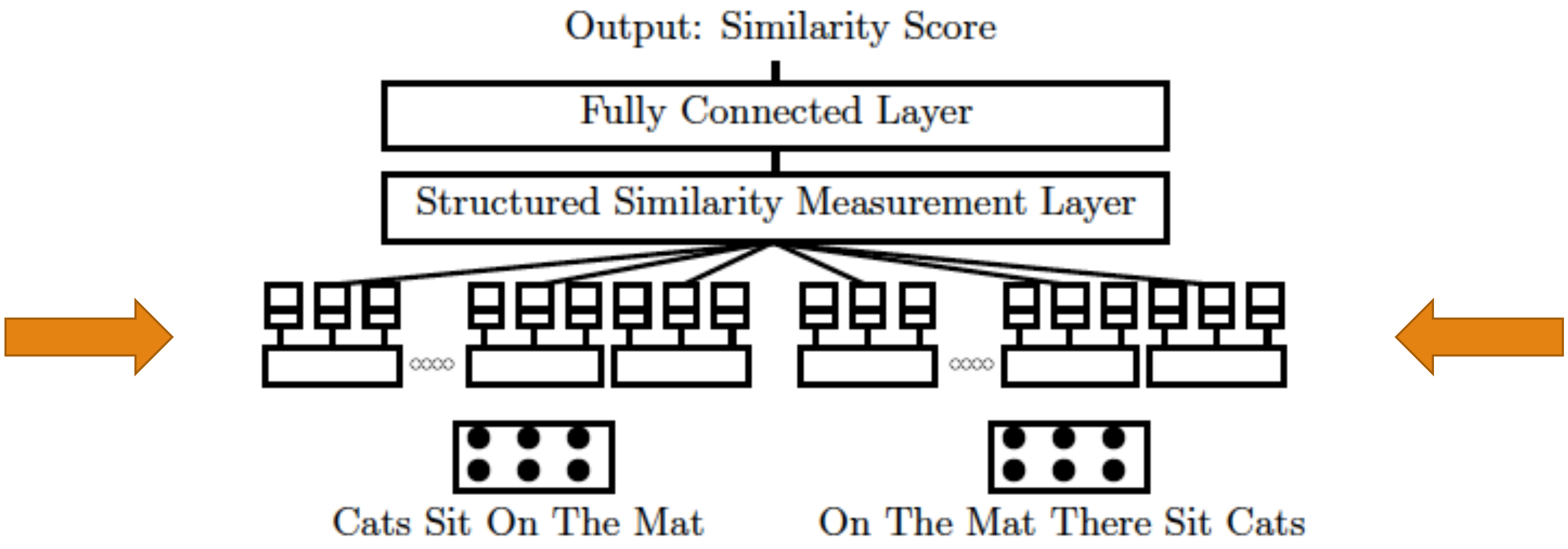
ws = 2

ws = 3




A special ws = ∞ corresponds with the entire sentence

# Sentence Modelling: Putting it together



# Sentence Modelling: Putting it together

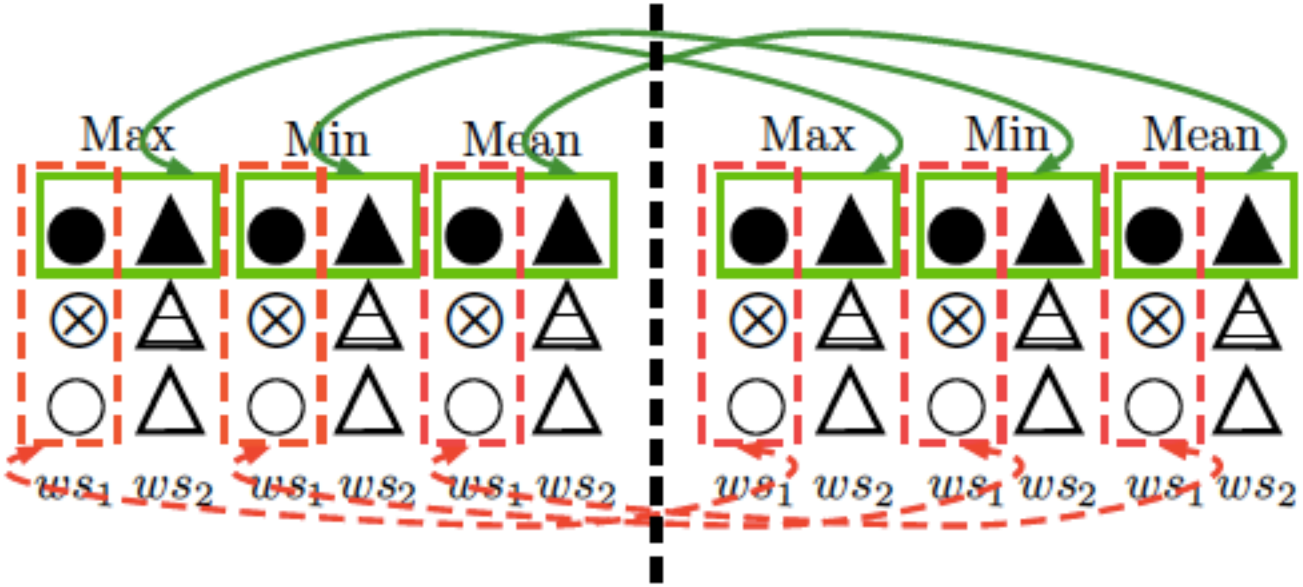
```
for ws in filter_widths:
    if np.isinf(ws):
        continue

    holistic_conv_layers.append(nn.Sequential(
        nn.Conv1d(n_word_dim, n_holistic_filters, ws),
        nn.Tanh()
    ))

    per_dim_conv_layers.append(nn.Sequential(
        nn.Conv1d(n_word_dim, n_word_dim * n_per_dim_filters, ws, groups=n_word_dim),
        nn.Tanh()
    ))

self.holistic_conv_layers = nn.ModuleList(holistic_conv_layers)
self.per_dim_conv_layers = nn.ModuleList(per_dim_conv_layers)
```

```
def _get_blocks_for_sentence(self, sent):
    block_a = {}
    block_b = {}
    for ws in self.filter_widths:
        holistic_conv_out = self.holistic_conv_layers[ws - 1](sent) if not np.isinf(ws) else sent
        block_a[ws] = {
            'max': F.max_pool1d(holistic_conv_out, holistic_conv_out.size()[2]).view(-1, self.n_word_dim),
            'min': F.max_pool1d(-1 * holistic_conv_out, holistic_conv_out.size()[2]).view(-1, self.n_word_dim),
            'mean': F.avg_pool1d(holistic_conv_out, holistic_conv_out.size()[2]).view(-1, self.n_word_dim)
        }

        # only compute per-dimension convolution for non-infinity widths
        if np.isinf(ws):
            continue

        per_dim_conv_out = self.per_dim_conv_layers[ws - 1](sent)
        block_b[ws] = {
            'max': F.max_pool1d(per_dim_conv_out, per_dim_conv_out.size()[2]).view(-1, self.n_word_dim, self.n_per_dim_filters),
            'min': F.max_pool1d(-1 * per_dim_conv_out, per_dim_conv_out.size()[2]).view(-1, self.n_word_dim, self.n_per_dim_filters)
        }
    return block_a, block_b
```

# Similarity Measurement Layer

- We can flatten the outputs from the different blocks into a 1D vector and compare the result
- Problem: different parts of the flattened vector represent different results, so comparing flattened vector might capture less information
- Instead, we can compare over non-flattened local regions

# Local Region Comparisons




Horizontal comparison:
comparing local regions of the two sentences based on matching pooling method and window size for holistic filters only. Compare using **cosine distance and Euclidean distance**.

Vertical comparison:
Similar, but in vertical direction for both holistic and per-dimension filters. Compare using **cosine distance, Euclidean distance, and element-wise absolute value**.

# Other Model Details

- Fully-Connected Layer: After similarity measurement, add two linear layers with tanh activation layer in between
- Final layer is log-softmax layer

```
(holistic_conv_layers): ModuleList (
  (0): Sequential (
    (0): Conv1d(300, 300, kernel_size=(1,), stride=(1,))
    (1): Tanh ()
  )
  (1): Sequential (
    (0): Conv1d(300, 300, kernel_size=(2,), stride=(1,))
    (1): Tanh ()
  )
  (2): Sequential (
    (0): Conv1d(300, 300, kernel_size=(3,), stride=(1,))
    (1): Tanh ()
  )
)
```

```
(per_dim_conv_layers): ModuleList (
  (0): Sequential (
    (0): Conv1d(300, 6000, kernel_size=(1,), stride=(1,), groups=300)
    (1): Tanh ()
  )
  (1): Sequential (
    (0): Conv1d(300, 6000, kernel_size=(2,), stride=(1,), groups=300)
    (1): Tanh ()
  )
  (2): Sequential (
    (0): Conv1d(300, 6000, kernel_size=(3,), stride=(1,), groups=300)
    (1): Tanh ()
  )
)
```

```
(final_layers): Sequential (
  (0): Linear (50760 -> 150)
  (1): Tanh ()
  (2): Linear (150 -> 5)
  (3): LogSoftmax ()
)
```

# Re-Implementation

- Model used in the paper was written in Torch
- Re-implement model in PyTorch as a part of wider efforts in research group
- Make some changes to the network and compare performance

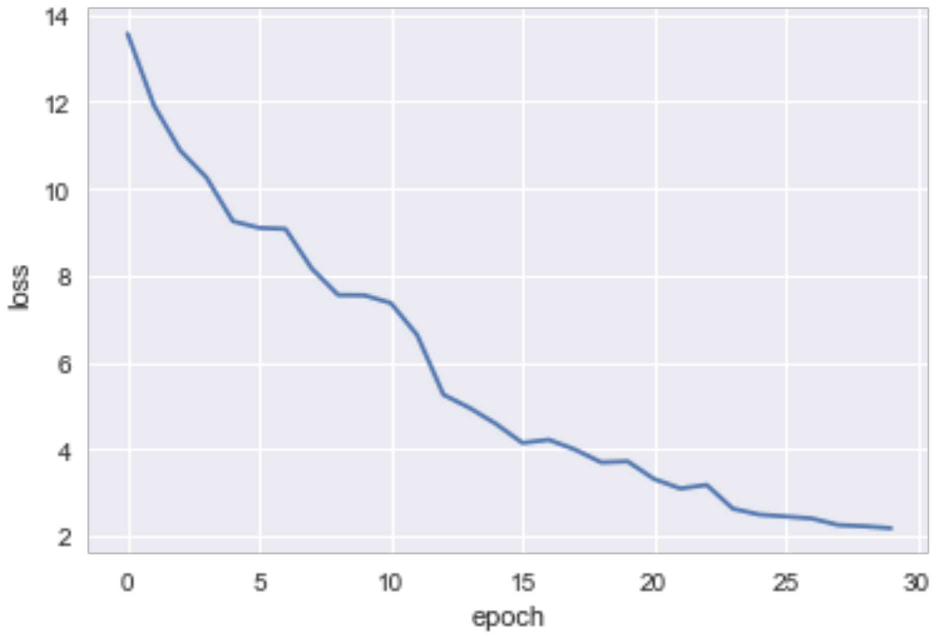
# Datasets for experiments

- SICK
  - Sentences Involving Compositional Knowledge
  - 9927 sentence pairs – 4500 training, 500 dev, 4927 testing
  - Scores are in range [1, 5]
- MSRVID
  - Microsoft Video Paraphrase Corpus
  - 1500 sentence pairs – 750 training, 750 testing
  - Since no dev set is provided, ~20% of the training data is held out for validation in each epoch
  - Scores are in range [0, 5]

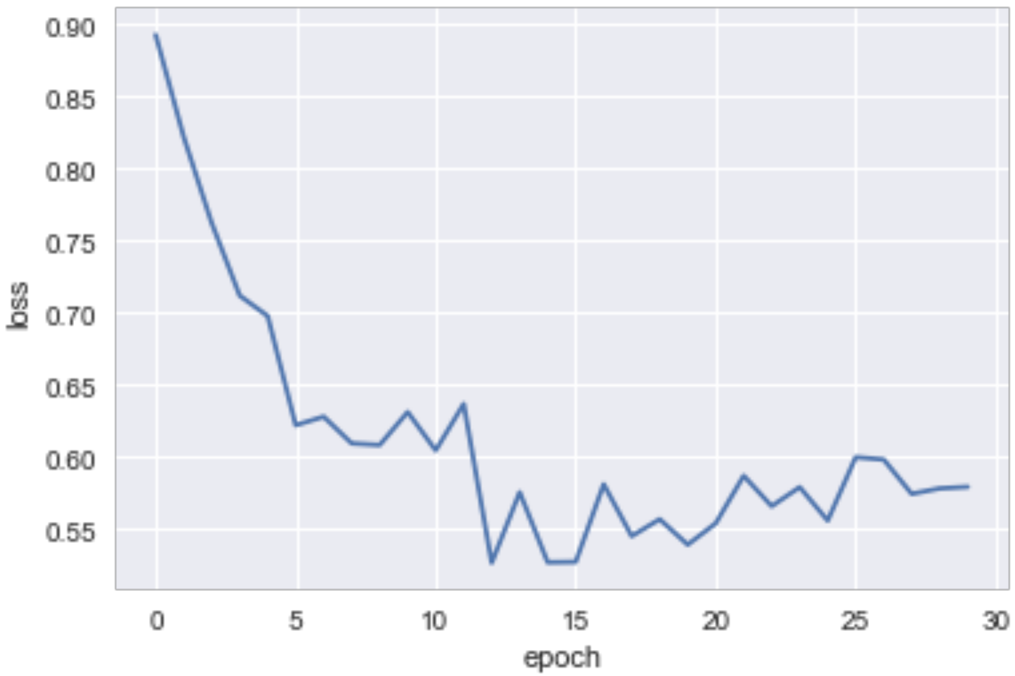
# Training

- Use 300 spatial filters and 20 per-dimension filters
- Both datasets are trained using Adam, using KL-divergence loss with L2 regularization penalty of 0.001
- Use batch size of 64 for SICK, 16 for MSRVID
- Learning rate: initially, 0.1, but decreases by a factor of ~3 if validation performance do not improve after 2 epochs (reduce learning rate on plateau)
- Shuffle training data after every epoch

# Learning Curve



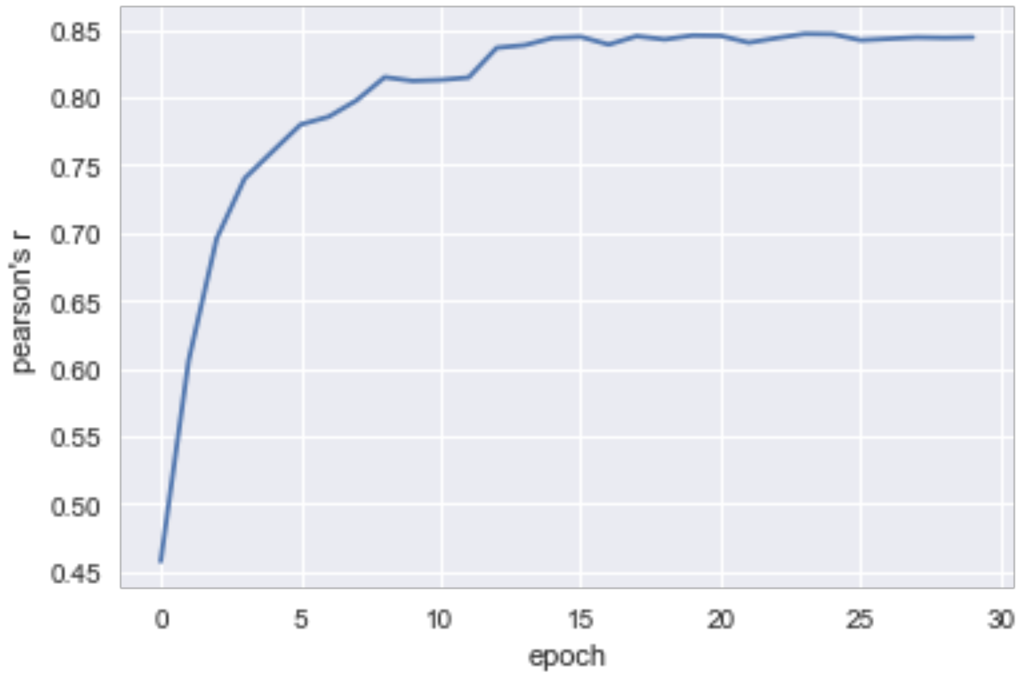

Training set loss for SICK dataset




Dev set loss for SICK dataset

*Note: training set loss is showing summed loss over batches, dev set loss is showing average loss per batch. Due to oversight. I did not have time before the presentation to make them consistent.

# Evaluation metric curve




Pearson's r on dev set

# Benchmark of Re-Implementation

SICK Dataset

| | r | $\rho$ |
|---|---|---|
| 2-layer Bidirectional LSTM | 0.8488 | 0.7926 |
| Tai et al (2015) Const. LSTM | 0.8491 | 0.7873 |
| Tai et al (2015) Dep. LSTM | 0.8676 | 0.8083 |
| Paper | 0.8686 | 0.8047 |
| **Re-impl.** | **0.8553** | **0.7905** |

MSRVID Dataset

| | r |
|---|---|
| Beltagy et al. (2014) | 0.8300 |
| Bär et al. (2012) | 0.8730 |
| Šarić et al. (2012) | 0.8803 |
| Paper | 0.9090 |
| **Re-impl.** | **0.8668** |

r refers to Pearson's r
$\rho$ refers to Spearman's $\rho$

# Modification 1: Dropout

SICK Dataset

| | r | $\rho$ |
|---|---|---|
| 2-layer Bidirectional LSTM | 0.8488 | 0.7926 |
| Tai et al (2015) Const. LSTM | 0.8491 | 0.7873 |
| Tai et al (2015) Dep. LSTM | 0.8676 | 0.8083 |
| Paper | 0.8686 | 0.8047 |
| **Re-impl. w/ modif.** | **0.8590** | **0.7917** |
| | +0.0037 | +0.0012 |

MSRVID Dataset

| | r |
|---|---|
| Beltagy et al. (2014) | 0.8300 |
| Bär et al. (2012) | 0.8730 |
| Šarić et al. (2012) | 0.8803 |
| Paper | 0.9090 |
| **Re-impl. w/ modif.** | **0.8788** |
| | +0.012 |

Using dropout probability = 0.5

# Modification 2: Batch Renormalization

SICK Dataset

| r | $\rho$ |
|---|---|
| 0.8016 | 0.7415 |

MSRVID Dataset

| r |
|---|
| 0.8604 |

Unfortunately batch normalization did not improve the performance with the default parameters

# Modification 3: Symmetric Compare Unit

SICK Dataset

| | r | ρ |
|---|---|---|
| 2-layer Bidirectional LSTM | 0.8488 | 0.7926 |
| Tai et al (2015) Const. LSTM | 0.8491 | 0.7873 |
| Tai et al (2015) Dep. LSTM | 0.8676 | 0.8083 |
| Paper | 0.8686 | 0.8047 |
| **Re-impl. w/ modif.** | **0.8565** | **0.7883** |
| | -0.0035 | -0.0034 |

MSRVID Dataset

| | r |
|---|---|
| Beltagy et al. (2014) | 0.8300 |
| Bär et al. (2012) | 0.8730 |
| Šarić et al. (2012) | 0.8803 |
| Paper | 0.9090 |
| **Re-impl. w/ modif.** | **0.8741** |
| | -0.0047 |

Compared with adding dropout as baseline, this did not improve performance

# Randomized Grid Search

```
sick_scores_df.sort_values('val', ascending=False)
```

| | file | test | val |
|---|---|---|---|
| 8 | grid_sick_lr_0.0016_eps_0.0038_reg_0.0011.txt | 0.838606 | 0.826844 |
| 9 | grid_sick_lr_0.0087_eps_0.0001_reg_0.0051.txt | 0.828084 | 0.824307 |
| 4 | grid_sick_lr_0.0008_eps_0.002_reg_0.0013.txt | 0.838737 | 0.823897 |
| 5 | grid_sick_lr_0.0009_eps_0.0011_reg_0.0218.txt | 0.785866 | 0.763221 |
| 6 | grid_sick_lr_0.0009_eps_0.0072_reg_0.0148.txt | 0.776737 | 0.756730 |
| 3 | grid_sick_lr_0.0004_eps_0.0025_reg_0.0384.txt | 0.585399 | 0.544012 |
| 1 | grid_sick_lr_0.0001_eps_0.0601_reg_0.0007.txt | 0.538176 | 0.530472 |
| 7 | grid_sick_lr_0.0009_eps_0.0757_reg_0.0008.txt | 0.538565 | 0.527721 |
| 0 | grid_sick_lr_0.0001_eps_0.0024_reg_0.0002.txt | 0.527839 | 0.498647 |
| 2 | grid_sick_lr_0.0002_eps_0.0008_reg_0.079.txt | 0.495018 | 0.453034 |

```
msrvid_scores_df.sort_values('val', ascending=False)
```

| | file | test | val |
|---|---|---|---|
| 19 | grid_msrvid_lr_0.0066_eps_0.0017_reg_0.0005.txt | 0.879973 | 0.999691 |
| 4 | grid_msrvid_lr_0.0003_eps_0.0002_reg_0.0007.txt | 0.863920 | 0.999338 |
| 11 | grid_msrvid_lr_0.0016_eps_0.0001_reg_0.0015.txt | 0.889749 | 0.999146 |
| 12 | grid_msrvid_lr_0.0018_eps_0.004_reg_0.0001.txt | 0.863612 | 0.998825 |
| 1 | grid_msrvid_lr_0.0002_eps_0.0001_reg_0.0019.txt | 0.863329 | 0.996565 |
| 24 | grid_msrvid_lr_0.0089_eps_0.0016_reg_0.0039.txt | 0.883729 | 0.995516 |
| 17 | grid_msrvid_lr_0.0048_eps_0.004_reg_0.0057.txt | 0.877002 | 0.992095 |
| 22 | grid_msrvid_lr_0.0078_eps_0.0006_reg_0.0002.txt | 0.888155 | 0.990571 |
| 2 | grid_msrvid_lr_0.0002_eps_0.0008_reg_0.0002.txt | 0.859497 | 0.990201 |
| 16 | grid_msrvid_lr_0.002_eps_0.0117_reg_0.0001.txt | 0.861618 | 0.989668 |
| 14 | grid_msrvid_lr_0.0027_eps_0.0169_reg_0.0003.txt | 0.861385 | 0.986688 |
| 5 | grid_msrvid_lr_0.0003_eps_0.0018_reg_0.0013.txt | 0.855682 | 0.980119 |

+0.001

test and val metrics show Pearson's r. Found better performance for MSRVID dataset. As an improvement, can try picking from a random set of reasonable discrete parameters instead. Thanks to Salman Mohammed for randomized hyperparameter search script.

# Work in Progress

- Adding attention module in parallel with convolution layers (Yin et al., 2016)
- Adding sparse features (e.g. idf) to first linear layer
- Evaluate performance on other tasks
  - TrecQA for question answering
  - SNLI for inference (contradiction, entailment, neutral)

# References


- Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-Perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of EMNLP*, pages 1576–1586.
- Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*, pages 160–167.
- Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.
- Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods for Natural Language Processing*.
- Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

# References Cont’ed


- Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1604–1612.
- Daniel Bar, Chris Biemann, Iryna Gurevych, and ¨ Torsten Zesch. 2012. UKP: computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 435–440.
- Frane Sari ˇ c, Goran Glava ´ s, Mladen Karan, Jan ˇ Snajder, ˇ and Bojana Dalbelo Basiˇ c. 2012. TakeLab: systems ´ for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 441–448.
- Islam Beltagy, Katrin Erk, and Raymond Mooney. 2014. Probabilistic soft logic for semantic textual similarity. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1210–1219.
- Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. *Abcnn: Attention-based convolutional neural network for modeling sentence pairs*. In ACL, 2016.