Outputs are discrete and correspond to positions in the input. Thus, the output "dictionary" varies per example.

Q: Can we think of cases where we need such dynamic size dictionary?
(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $C^\mathcal{P} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.

(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $C^\mathcal{P} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.
Pointer Networks: Handling Variable Size Output Dictionary

(a) Sequence-to-Sequence

(b) Ptr-Net
Pointer Networks: Handling Variable Size Output Dictionary

• Fixed-Size Dictionary

\[ u_j^i = v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, \ldots, n) \]
\[ a_j^i = \text{softmax}(u_j^i) \quad j \in (1, \ldots, n) \]

\[ d_i' = \sum_{j=1}^{n} a_j^i e_j \]

the updated decoder hidden state!, d_i,d'_i are concatenated and feed into a softmax over the fixed size dictionary

• Dynamic Dictionary

\[ u_j^i = v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, \ldots, n) \]

\[ p(C_i|C_1, \ldots, C_{i-1}, \mathcal{P}) = \text{softmax}(u_i^i) \]

the decoder hidden state is used to selected the location of the input via interaction with the encoder hidden states e_j
Pointer Networks: Handling Variable Size Output Dictionary

(a) LSTM, $m=50$, $n=50$
(b) Truth, $n=50$
(c) Truth, $n=20$
(d) Ptr-Net, $m=5-50$, $n=500$
(e) Ptr-Net, $m=50$, $n=50$
(f) Ptr-Net, $m=5-20$, $n=20$

Ground Truth

Predictions

Predictions: tour length is 3.523

Ground Truth: tour length is 3.518
## Pointer Networks: Handling Variable Size Output Dictionary

<table>
<thead>
<tr>
<th>Method</th>
<th>Trained ( n )</th>
<th>( n )</th>
<th>Accuracy</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM [1]</td>
<td>50</td>
<td>50</td>
<td>1.9%</td>
<td>FAIL</td>
</tr>
<tr>
<td>+ATTENTION [5]</td>
<td>50</td>
<td>50</td>
<td>38.9%</td>
<td>99.7%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>50</td>
<td>50</td>
<td>72.6%</td>
<td>99.9%</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
<td>--------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>LSTM [1]</td>
<td>5</td>
<td>5</td>
<td>87.7%</td>
<td>99.6%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>5</td>
<td>92.0%</td>
<td>99.6%</td>
</tr>
<tr>
<td>LSTM [1]</td>
<td>10</td>
<td>10</td>
<td>29.9%</td>
<td>FAIL</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>10</td>
<td>87.0%</td>
<td>99.8%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>50</td>
<td>69.6%</td>
<td>99.9%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>100</td>
<td>50.3%</td>
<td>99.9%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>200</td>
<td>22.1%</td>
<td>99.9%</td>
</tr>
<tr>
<td>PTR-NET</td>
<td>5-50</td>
<td>500</td>
<td>1.3%</td>
<td>99.2%</td>
</tr>
</tbody>
</table>
Table 2: Tour length of the Ptr-Net and a collection of algorithms on a small scale TSP problem.

<table>
<thead>
<tr>
<th>n</th>
<th>Optimal</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Ptr-Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
</tr>
<tr>
<td>10</td>
<td>2.87</td>
<td>3.07</td>
<td>2.87</td>
<td>2.87</td>
<td>2.88</td>
</tr>
<tr>
<td>50 (A1 TRAINED)</td>
<td>N/A</td>
<td>6.46</td>
<td>5.84</td>
<td>5.79</td>
<td>6.42</td>
</tr>
<tr>
<td>50 (A3 TRAINED)</td>
<td>N/A</td>
<td>6.46</td>
<td>5.84</td>
<td>5.79</td>
<td>6.09</td>
</tr>
<tr>
<td>5 (5-20 TRAINED)</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
<td>2.12</td>
</tr>
<tr>
<td>10 (5-20 TRAINED)</td>
<td>2.87</td>
<td>3.07</td>
<td>2.87</td>
<td>2.87</td>
<td>2.87</td>
</tr>
<tr>
<td>20 (5-20 TRAINED)</td>
<td>3.83</td>
<td>4.24</td>
<td>3.86</td>
<td>3.85</td>
<td>3.88</td>
</tr>
<tr>
<td>25 (5-20 TRAINED)</td>
<td>N/A</td>
<td>4.71</td>
<td>4.27</td>
<td>4.24</td>
<td>4.30</td>
</tr>
<tr>
<td>30 (5-20 TRAINED)</td>
<td>N/A</td>
<td>5.11</td>
<td>4.63</td>
<td>4.60</td>
<td>4.72</td>
</tr>
<tr>
<td>40 (5-20 TRAINED)</td>
<td>N/A</td>
<td>5.82</td>
<td>5.27</td>
<td>5.23</td>
<td>5.91</td>
</tr>
<tr>
<td>50 (5-20 TRAINED)</td>
<td>N/A</td>
<td>6.46</td>
<td>5.84</td>
<td>5.79</td>
<td>7.66</td>
</tr>
</tbody>
</table>
Key-variable memory

We use similar indexing mechanism to index location in the key variable memory, during decoding, when we know we need to pick an argument, as opposed to function name. All arguments are stored in such memory.
Recursive/tree structured networks

Katerina Fragkiadaki
We have already discussed word vector representations that "capture the meaning" of word by embedding them into a low-dimensional space where semantic similarity is preserved.

But what about longer phrases? For this lecture, understanding of the meaning of a sentence is representing the phrase as a vector in a structured semantic space, where similar sentences are nearby, and unrelated sentences are far away.
How can we represent the meaning of longer phrases? By mapping them into the same vector space as words!
We have already discussed word vector representations that "capture the meaning" of word by embedding them into a low-dimensional space where semantic similarity is preserved.

But what about longer phrases? For this lecture, understanding of the meaning of a sentence is representing the phrase as a vector in a structured semantic semantic space, where similar sentences are nearby, and unrelated sentences are far away.

Sentence modeling is at the core of many language comprehension tasks sentiment analysis, paraphrase detection, entailment recognition, summarization, discourse analysis, machine translation, grounded language learning and image retrieval.
From Words to Phrases

• How can we know when larger units of a sentence are similar in meaning?
  • The snowboarders is leaping over a mogul.
  • A person on a snowboard jumps into the air.

• People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

  "A small crowd quietly enters the historical church". 
How can we know when larger units of a sentence are similar in meaning?

- The snowboarders is leaping over a mogul.
- A person on a snowboard jumps into the air.

People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by semantic composition of smaller elements.

"A small crowd quietly enters the historical church".
From Words to Phrases

• How can we know when larger units of a sentence are similar in meaning?
  • The snowboarders is leaping over a mogul.
  • A person on a snowboard jumps into the air.

• People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

”A small crowd quietly enters the historical church”.
From Words to Phrases

• How can we know when larger units of a sentence are similar in meaning?
  • The snowboarders is leaping over a mogul.
  • A person on a snowboard jumps into the air.

• People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by semantic composition of smaller elements.

"A small crowd quietly enters the historical church".
From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

"A small crowd quietly enters the historical church".
• How can we know when larger units of a sentence are similar in meaning?
  • The snowboarders is leaping over a mogul.
  • A person on a snowboard jumps into the air.

• People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by semantic composition of smaller elements.

"A small crowd quietly enters the historical church".
From Words to Phrases: 4 models

- Bag of words: Ignores word order, simple averaging of word vectors in a sub-phrase. Can’t capture differences in meaning as a result of differences in word order, e.g., "cats climb trees" and "trees climb cats" will have the same representation.

- Sequence (recurrent) models, e.g., LSTMs: The hidden vector of the last word is the representation of the phrase.

- Tree-structured (recursive) models: compose each phrase from its constituent sub-phrases, according to a given syntactic structure over the sentence

- Convolutional neural networks

Q: Does semantic understanding improve with grammatical understanding so that recursive models are justified?
From Words to Phrases: 4 models

• Bag of words: Ignores word order, simple averaging of word vectors in a sub-phrase. Can’t capture differences in meaning as a result of differences in word order, e.g., "cats climb trees" and "trees climb cats" will have the same representation.

• Sequence models, e.g., LSTMs: The hidden vector of the last word is the representation of the phrase.

• **Tree-structured (recursive) models**: compose each phrase from its constituent sub-phrases, according to a given syntactic structure over the sentence

• Convolutional neural networks

Q: Does semantic understanding improve with grammatical understanding so that recursive models are justified?
Recursive Neural Networks

Given a tree and vectors for the leaves, compute bottom-up vectors for the intermediate nodes, all the way to the root, via compositional function $g$. 
How should we map phrases into a vector space?

Use principle of compositionality

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.

Jointly learn parse trees and compositional vector representations

Parsing with compositional vector grammars, Socher et al.
The cat sat on the mat.
Learn Structure and Representation

The cat sat on the mat.
Recursive vs. Recurrent Neural Networks

Q: what is the difference in the intermediate concepts they build?

Slide adapted from Manning-Socher
Recursive neural nets require a parser to get tree structure.

Recurrent neural nets cannot capture phrases without prefix context and often capture too much of last words in final vector. However, they do not need a parser, and they are much preferred in current literature at least.
Recursive Neural Networks for Structure Prediction

- **Inputs:** Two candidate children's representations
- **Outputs:**
  1. The semantic representation if the two nodes are merged.
  2. Score of how plausible the new node would be.

Slide adapted from Manning-Socher
Recursive Neural Network (Version 1)

\[
\text{score} = 1.3 \quad \begin{bmatrix} 8 \\ 3 \\ 5 \\ 3 \end{bmatrix} = \text{parent}
\]

\[
\text{score} = U^T p
\]

\[
p = \tanh(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b),
\]

Same \( W \) parameters at all nodes of the tree

Slide adapted from Manning-Socher
The cat sat on the mat.
The cat sat on the mat.

Parsing a Sentence

Bottom-up beam search

Slide adapted from Manning-Socher
The cat sat on the mat.

Bottom-up beam search

Slide adapted from Manning-Socher
Cost function

- The score of a tree is computed by the sum of the parsing decision scores at each node:

\[ s(x, y) = \sum_{n \in \text{nodes}(y)} s_n \]

- \( x \) is sentence; \( y \) is parse tree
Cost function

• Max-margin objective:

\[ J = \sum_i s(x_i, y_i) - \max_{y \in A(x_i)} (s(x_i, y) + \Delta(y, y_i)) \]

parse trees resulting from beam search

• The loss \( \Delta(y, y_i) \) penalized all incorrect decisions
We update parameters, and sample new trees for every example periodically.

In practice, first we compute the top best trees from a PCFG (probabilistic context free grammar), and then we use those trees to learn the parameters of the recursive net, using backdrop through structure (similar to backdrop through time).

This means the trees for each example are not updated during parameter learning.

It is like a cascade.
RecursiveNN Version 1: Discussion

Single weight matrix RecursiveNN could capture some phenomena, but not adequate for more complex, higher order composition and parsing long sentences.

• There is no real interaction between the input words.

• The composition function is the same for all syntactic categories, punctuation, etc.
Version 2: Syntactically-Untied RNN

- We use the discrete syntactic categories of the children to choose the composition matrix.

- A TreeRNN can do better with different composition matrix for different syntactic environments.

- This gives better results

A, B, C are part of speech tags

Slide adapted from Manning-Socher
Version 2: Syntactically-Untied RNN

- **Problem:** Speed. Every candidate score in beam search needs a matrix-vector product.
- **Solution:** Compute score only for a subset of trees coming from a simpler, faster model (PCFG)
  - Prunes very unlikely candidates for speed
  - Provides coarse syntactic categories of the children for each beam candidate.
- **Compositional Vector Grammar = PCFG + TreeRNN**

Slide adapted from Manning-Socher
Version 2: Syntactically-Untied RNN

- Scores at each node computed by combination of PCFG and SU-RNN:

\[
s\left(p^{(1)}\right) = (v^{(B,C)})^T p^{(1)} + \log P(P_1 \rightarrow B \ C)
\]

- Interpretation: Factoring discrete and continuous parsing in one model:

\[
P((P_1, p_1) \rightarrow (B, b)(C, c)) = P(p_1 \rightarrow b \ c | P_1 \rightarrow B \ C) P(P_1 \rightarrow B \ C)
\]
Experiments

- Standard WSJ split, labeled F1
- Based on simple PCFG with fewer states
- Fast pruning of search space, few matrix-vector products
- 3.8% higher F1, 20% faster than Stanford factored parser

<table>
<thead>
<tr>
<th>Parser</th>
<th>Test, All Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford PCFG, (Klein and Manning, 2003a)</td>
<td>85.5</td>
</tr>
<tr>
<td>Stanford Factored (Klein and Manning, 2003b)</td>
<td>86.6</td>
</tr>
<tr>
<td>Factored PCFGs (Hall and Klein, 2012)</td>
<td>89.4</td>
</tr>
<tr>
<td>Collins (Collins, 1997)</td>
<td>87.7</td>
</tr>
<tr>
<td>SSN (Henderson, 2004)</td>
<td>89.4</td>
</tr>
<tr>
<td>Berkeley Parser (Petrov and Klein, 2007)</td>
<td>90.1</td>
</tr>
<tr>
<td>CVG (RNN) (Socher et al., ACL 2013)</td>
<td>85.0</td>
</tr>
<tr>
<td>CVG (SU-RNN) (Socher et al., ACL 2013)</td>
<td>90.4</td>
</tr>
<tr>
<td>Charniak - Self Trained (McClosky et al. 2006)</td>
<td>91.0</td>
</tr>
<tr>
<td>Charniak - Self Trained-ReRanked (McClosky et al. 2006)</td>
<td>92.1</td>
</tr>
</tbody>
</table>
• Learns soft notion of head words
• Initialization: $W^{(\cdots)} = 0.5[I_{n \times n} I_{n \times n} 0_{n \times 1}] + \epsilon$

Learning relative weighting is the best you can do with such linear interactions, $W_1c_1 + W_2c_2$

SU-RNN/CVG

Phrase similarity in Resulting Vector Representation

- All the figures are adjusted for seasonal variations
  - All the numbers are adjusted for seasonal fluctuations
  - All the figures are adjusted to remove usual seasonal patterns

- Knight-Ridder wouldn't comment on the offer
  - Harsco declined to say what country placed the order
  - Coastal wouldn't disclose the terms

- Sales grew almost 7% to $UNK m. from $UNK m.
  - Sales rose more than 7% to $94.9 m. from $88.3 m.
  - Sales surged 40% to UNK b. yen from UNK b.
SU-RNN Analysis

• Can transfer semantic information from single related example

• Train sentences:
  • He eats spaghetti with a fork.
  • She eats spaghetti with pork.

• Test sentences:
  • He eats spaghetti with a spoon.
  • He eats spaghetti with meat.
SU-RNN Analysis

(a) Stanford factored parser

(b) Compositional Vector Grammar

Slide adapted from Manning-Socher
Labeling

- We can use each node's representation as features for a softmax classifier:

\[ p(c|p) = \text{softmax}(Sp) \]

- Training similar to model in part 1 with standard cross-entropy error + scores of composition

Slide adapted from Manning-Socher
We just saw one way to make the composition function more powerful was by untying the weights $W$.

But what if words act mostly as an operator, e.g. "very" in very good, thus I do not want to take a weighted sum of the word vectors, I instead want to amplify "good"'s vector.
Version 3: Matrix-Vector RNNs

\[ p = f \left( W \begin{bmatrix} a \\ b \end{bmatrix} \right) \]

\[ p = f \left( W \begin{bmatrix} Ba \\ Ab \end{bmatrix} \right) \]
Matrix-Vector RNNs

Each word is represented by both a matrix and a vector

\[ p = \tanh(W \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + b) \]

- vector

- matrix

Recursive Matrix-Vector Model

... f(Ba, Ab) = ... 

Ba = very 

(a, A) 

good 

(b, B) 

movie 

(c, C)
Matrix-Vector RNNs

\[ p = f \left( W \begin{bmatrix} B_a \\ A_b \end{bmatrix} \right) \]

\[ P = g(A, B) = W_M \begin{bmatrix} A \\ B \end{bmatrix} \]

\[ W_M \in \mathbb{R}^{n \times 2n} \]
Predicting Sentiment Distributions

Good example for non-linearity in language

Slide adapted from Manning-Socher
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Features</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>POS, stemming, syntactic patterns</td>
<td>60.1</td>
</tr>
<tr>
<td>MaxEnt</td>
<td>POS, WordNet, morphological features, noun compound system, thesauri, Google n-grams</td>
<td>77.6</td>
</tr>
<tr>
<td>SVM</td>
<td>POS, WordNet, prefixes, morphological features, dependency parse features, Levin classes, PropBank, FrameNet, NomLex-Plus, Google n-grams, paraphrases, TextRunner</td>
<td>82.2</td>
</tr>
<tr>
<td>RNN</td>
<td>–</td>
<td>74.8</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>–</td>
<td>79.1</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>POS, WordNet, NER</td>
<td>82.4</td>
</tr>
</tbody>
</table>
Problems with MV-RNNs

- Parameters of the model grow quadratically with the size of the vocabulary (due to matrices)
- Can we find a more economical way to have multiplicative interactions in recursive networks?
- Recursive tensor networks
Compositional Function

- standard linear function + non-linearity, captures additive interactions:
  \[ p_1 = f \left( W \begin{bmatrix} b \\ c \end{bmatrix} \right), \quad p_2 = f \left( W \begin{bmatrix} a \\ p_1 \end{bmatrix} \right) \]

- matrix/vector compositions (Socher 2011): represent each word and phrase by both a vector and a matrix. The number of parameters grows with vocabulary.
  \[ p_1 = f \left( W \begin{bmatrix} Cb \\ Bc \end{bmatrix} \right), \quad P_1 = f \left( W_M \begin{bmatrix} B \\ C \end{bmatrix} \right) \]

- Recursive neural tensor networks. Parameters are both the word vectors as well as then composition tensor \( V \), shared across all node compositions. **Q:** what is the dimensionality of \( V \)?

\[ h = \left[ \begin{array}{c} b \\ c \end{array} \right]^T V^{[1:d]} \left[ \begin{array}{c} b \\ c \end{array} \right]; \quad h_i = \left[ \begin{array}{c} b \\ c \end{array} \right]^T V^{[i]} \left[ \begin{array}{c} b \\ c \end{array} \right] \]
Version 4: Recursive Neural Tensor Networks

Slide adapted from Manning-Socher
We train the parameters of the model so that we minimize classification error at the root node of a sentence (e.g., sentiment prediction, does this sentence feel positive or negative?) or, at many intermediate nodes if such annotations are available:

$$E(\theta) = \sum_i \sum_j t_j^i \log y_j^i + \lambda \|\theta\|^2$$
Evaluation

Plus + and minus - indicate sentiment prediction in the different places of the sentence.
Evaluation

- Using a dataset with fine grain sentiment labels for all (intermediate) phrases

<table>
<thead>
<tr>
<th>Model</th>
<th>Fine-grained All</th>
<th>Fine-grained Root</th>
<th>Positive/Negative All</th>
<th>Positive/Negative Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>67.2</td>
<td>41.0</td>
<td>82.6</td>
<td>81.8</td>
</tr>
<tr>
<td>SVM</td>
<td>64.3</td>
<td>40.7</td>
<td>84.6</td>
<td>79.4</td>
</tr>
<tr>
<td>BiNB</td>
<td>71.0</td>
<td>41.9</td>
<td>82.7</td>
<td>83.1</td>
</tr>
<tr>
<td>VecAvg</td>
<td>73.3</td>
<td>32.7</td>
<td>85.1</td>
<td>80.1</td>
</tr>
<tr>
<td>RNN</td>
<td>79.0</td>
<td>43.2</td>
<td>86.1</td>
<td>82.4</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>78.7</td>
<td>44.4</td>
<td>86.8</td>
<td>82.9</td>
</tr>
<tr>
<td>RNTN</td>
<td><strong>80.7</strong></td>
<td><strong>45.7</strong></td>
<td><strong>87.6</strong></td>
<td><strong>85.4</strong></td>
</tr>
</tbody>
</table>

Table 1: Accuracy for fine grained (5-class) and binary predictions at the sentence level (root) and for all nodes.
Correctly capturing compositionality of meaning is important for sentiment analysis due to **negations** that reverse the sentiment, e.g., "I didn’t like a single minute of this film", "the movie was not terrible" etc.

<table>
<thead>
<tr>
<th>Model</th>
<th>Negated Positive</th>
<th>Negated Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>biNB</td>
<td>19.0</td>
<td>27.3</td>
</tr>
<tr>
<td>RNN</td>
<td>33.3</td>
<td>45.5</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>52.4</td>
<td>54.6</td>
</tr>
<tr>
<td>RNTN</td>
<td><strong>71.4</strong></td>
<td><strong>81.8</strong></td>
</tr>
</tbody>
</table>

Table 2: Accuracy of negation detection. Negated positive is measured as correct sentiment inversions. Negated negative is measured as increases in positive activations.

Figure 8: Change in activations for negations. Only the RNTN correctly captures both types. It decreases positive sentiment more when it is negated and learns that negating negative phrases (such as *not terrible*) should increase neutral and positive activations.
Let’s go back to vanilla trees and use LSTMs instead of RNNs.

- Creates intermediate vectors for prefixes.
- Creates intermediate vectors for sub-phrases that are grammatically correct.
$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

$$i_t = \sigma \left( W^{(i)} x_t + U^{(i)} h_{t-1} + b^{(i)} \right),$$

$$f_t = \sigma \left( W^{(f)} x_t + U^{(f)} h_{t-1} + b^{(f)} \right),$$

$$o_t = \sigma \left( W^{(o)} x_t + U^{(o)} h_{t-1} + b^{(o)} \right),$$

$$u_t = \tanh \left( W^{(u)} x_t + U^{(u)} h_{t-1} + b^{(u)} \right),$$

$$c_t = i_t \odot u_t + f_t \odot c_{t-1},$$

$$h_t = o_t \odot \tanh(c_t),$$
LSTMS vs Tree-LSTMS

What if we use LSTM updates not in a chain but on trees produced by SoA dependency or constituency parsers?

We use a different forget gate for every child

\[
\begin{align*}
\tilde{h}_j &= \sum_{k \in C(j)} h_k, \\
i_j &= \sigma \left( W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)} \right), \\
f_{jk} &= \sigma \left( W^{(f)} x_j + U^{(f)} h_k + b^{(f)} \right), \\
o_j &= \sigma \left( W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)} \right), \\
u_j &= \tanh \left( W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)} \right), \\
c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \\
h_j &= o_j \odot \tanh(c_j),
\end{align*}
\]
Does children order matter?

- We use Child-sum tree-LSTMs for dependency trees
- We use N-ary (in particular binary) tree LSTMs on constituency trees
Experiments

- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences

\[ \hat{p}_\theta(y \mid \{x\}_j) = \text{softmax} \left( W^{(s)} h_j + b^{(s)} \right), \]
\[ \hat{y}_j = \arg \max_y \hat{p}_\theta(y \mid \{x\}_j). \]
Experiments

- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences

<table>
<thead>
<tr>
<th>Method</th>
<th>Fine-grained</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAE (Socher et al., 2013)</td>
<td>43.2</td>
<td>82.4</td>
</tr>
<tr>
<td>MV-RNN (Socher et al., 2013)</td>
<td>44.4</td>
<td>82.9</td>
</tr>
<tr>
<td>RNTN (Socher et al., 2013)</td>
<td>45.7</td>
<td>85.4</td>
</tr>
<tr>
<td>DCNN (Blunsom et al., 2014)</td>
<td>48.5</td>
<td>86.8</td>
</tr>
<tr>
<td>Paragraph-Vec (Le and Mikolov, 2014)</td>
<td>48.7</td>
<td>87.8</td>
</tr>
<tr>
<td>CNN-non-static (Kim, 2014)</td>
<td>48.0</td>
<td>87.2</td>
</tr>
<tr>
<td>CNN-multichannel (Kim, 2014)</td>
<td>47.4</td>
<td>88.1</td>
</tr>
<tr>
<td>DRNN (Irsoy and Cardie, 2014)</td>
<td>49.8</td>
<td>86.6</td>
</tr>
<tr>
<td>LSTM</td>
<td>46.4 (1.1)</td>
<td>84.9 (0.6)</td>
</tr>
<tr>
<td>Bidirectional LSTM</td>
<td>49.1 (1.0)</td>
<td>87.5 (0.5)</td>
</tr>
<tr>
<td>2-layer LSTM</td>
<td>46.0 (1.3)</td>
<td>86.3 (0.6)</td>
</tr>
<tr>
<td>2-layer Bidirectional LSTM</td>
<td>48.5 (1.0)</td>
<td>87.2 (1.0)</td>
</tr>
<tr>
<td>Dependency Tree-LSTM</td>
<td>48.4 (0.4)</td>
<td>85.7 (0.4)</td>
</tr>
<tr>
<td>Constituency Tree-LSTM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– randomly initialized vectors</td>
<td>43.9 (0.6)</td>
<td>82.0 (0.5)</td>
</tr>
<tr>
<td>– Glove vectors, fixed</td>
<td>49.7 (0.4)</td>
<td>87.5 (0.8)</td>
</tr>
<tr>
<td>– Glove vectors, tuned</td>
<td><strong>51.0 (0.5)</strong></td>
<td><strong>88.0 (0.3)</strong></td>
</tr>
</tbody>
</table>
Experiments

- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences

<table>
<thead>
<tr>
<th>Method</th>
<th>Pearson’s $r$</th>
<th>Spearman’s $\rho$</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois-LH (Lai and Hockenmaier, 2014)</td>
<td>0.7993</td>
<td>0.7538</td>
<td>0.3692</td>
</tr>
<tr>
<td>UNAL-NLP (Jimenez et al., 2014)</td>
<td>0.8070</td>
<td>0.7489</td>
<td>0.3550</td>
</tr>
<tr>
<td>Meaning Factory (Bjerva et al., 2014)</td>
<td>0.8268</td>
<td>0.7721</td>
<td>0.3224</td>
</tr>
<tr>
<td>ECNU (Zhao et al., 2014)</td>
<td>0.8414</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mean vectors</td>
<td>0.7577 (0.0013)</td>
<td>0.6738 (0.0027)</td>
<td>0.4557 (0.0090)</td>
</tr>
<tr>
<td>DT-RNN (Socher et al., 2014)</td>
<td>0.7923 (0.0070)</td>
<td>0.7319 (0.0071)</td>
<td>0.3822 (0.0137)</td>
</tr>
<tr>
<td>SDT-RNN (Socher et al., 2014)</td>
<td>0.7900 (0.0042)</td>
<td>0.7304 (0.0076)</td>
<td>0.3848 (0.0074)</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.8528 (0.0031)</td>
<td>0.7911 (0.0059)</td>
<td>0.2831 (0.0092)</td>
</tr>
<tr>
<td>Bidirectional LSTM</td>
<td>0.8567 (0.0028)</td>
<td>0.7966 (0.0053)</td>
<td>0.2736 (0.0063)</td>
</tr>
<tr>
<td>2-layer LSTM</td>
<td>0.8515 (0.0066)</td>
<td>0.7896 (0.0088)</td>
<td>0.2838 (0.0150)</td>
</tr>
<tr>
<td>2-layer Bidirectional LSTM</td>
<td>0.8558 (0.0014)</td>
<td>0.7965 (0.0018)</td>
<td>0.2762 (0.0020)</td>
</tr>
<tr>
<td>Constituency Tree-LSTM</td>
<td>0.8582 (0.0038)</td>
<td>0.7966 (0.0053)</td>
<td>0.2734 (0.0108)</td>
</tr>
<tr>
<td>Dependency Tree-LSTM</td>
<td><strong>0.8676 (0.0030)</strong></td>
<td><strong>0.8083 (0.0042)</strong></td>
<td><strong>0.2532 (0.0052)</strong></td>
</tr>
</tbody>
</table>
From RNNs to CNNs

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

\[
\begin{align*}
\begin{pmatrix} 1 \\ 3.5 \end{pmatrix} & \rightarrow \begin{pmatrix} 1 \\ 5 \end{pmatrix} & \rightarrow \begin{pmatrix} 5.5 \\ 6.1 \end{pmatrix} & \rightarrow \begin{pmatrix} 4.5 \\ 3.8 \end{pmatrix} & \rightarrow \begin{pmatrix} 2.5 \\ 3.8 \end{pmatrix} \\
\begin{pmatrix} 0.4 \\ 0.3 \end{pmatrix} & \rightarrow \begin{pmatrix} 2.1 \\ 3.3 \end{pmatrix} & \rightarrow \begin{pmatrix} 7 \\ 7 \end{pmatrix} & \rightarrow \begin{pmatrix} 4 \\ 4.5 \end{pmatrix} & \rightarrow \begin{pmatrix} 2.3 \\ 3.6 \end{pmatrix}
\end{align*}
\]

the country of my birth

- Softmax is often only at the last step.
From RNNs to CNNs

- RNN: Get compositional vectors from grammatical phrases only
- CNN: Compute vectors for every possible phrase
- Example: "the country of my birth" computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth
- Regardless of whether each is grammatical - many don't make sense
- Don't need parser
- But maybe not very linguistically or cognitively plausible
Relationship between CNN and RNN

Slide adapted from Manning-Socher
Relationship between CNN and RNN

representation for EVERY bigram, trigram etc.
Main CNN idea: What if we compute vectors for every possible phrase?

Example: "the country of my birth" computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth

Regardless of whether each is grammatical - not very linguistically or cognitively plausible
Convolution

- 1D discrete convolution generally:

\[(f * g)[n] = \sum_{m=-M}^{M} f[n - m]g[m].\]

- Convolution is great to extract features from images

- 2D example:
  - Yellow and red numbers show filter weights
  - Green shows input

Slide adapted from Manning-Socher
A simple variant using one convolutional layer and pooling.

Word vectors: \( \mathbf{x}_i \in \mathbb{R}^k \)

Sentence: \( \mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n \)

Convolutional filter: \( \mathbf{w} \in \mathbb{R}^{hk} \)

Could be 2 (as before) higher, e.g. 3:
Single Layer CNN

- Convolutional filter: \( \mathbf{w} \in \mathbb{R}^{hk} \)
- Window size \( h \) could be 2 (as before) or higher, e.g. 3
- To compute feature for CNN layer:

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

\[
\begin{bmatrix}
1.1 \\
0.4 \\
0.3 \\
2.1 \\
3.3 \\
7 \\
4 \\
4.5 \\
2.3 \\
3.6 \\
\end{bmatrix}
\]

the country of my birth

\text{Slide adapted from Manning-Socher}
Single Layer CNN

- Filter $w$ is applied to all possible windows (concatenated vectors)

- Sentence: $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$

- All possible windows of length $h$: $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$

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

Slide adapted from Manning-Socher
Single Layer CNN

- Filter \( w \) is applied to all possible windows (concatenated vectors)

- Sentence: \( x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n \)

- All possible windows of length \( h \): \( \{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\} \)

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

![Diagram of feature map](image)

Slide adapted from Manning-Socher
Single Layer CNN: Pooling

- New building block: Pooling
- In particular: 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: \[ \hat{c} = \max\{\mathbf{c}\} \]

- But we want more features!
Solution: Multiple Filters

- Use multiple filter weights $w$
- Useful to have different window sizes $h$
- Because of max pooling, length of $c$ is irrelevant

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

- So we can have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.
Classification after one CNN Layer

- First one convolution, followed by one max-pooling
- To obtain final feature vector: \( z = [\hat{c}_1, \ldots, \hat{c}_m] \)
  - Assuming m filters \( w \)
- Simple final softmax layer \( y = \text{softmax} \left( W^{(S)} z + b \right) \)
n words (possibly zero padded) and each word vector has k dimensions
Experiments

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

- Narrow vs. wide convolution

- Complex pooling schemes (over sequences) and deeper convolutional layers