Natural Language Processing 2

Pavel Yurlov

Higher School of Economics
Faculty of Computer Science

December 14, 2018

Outline

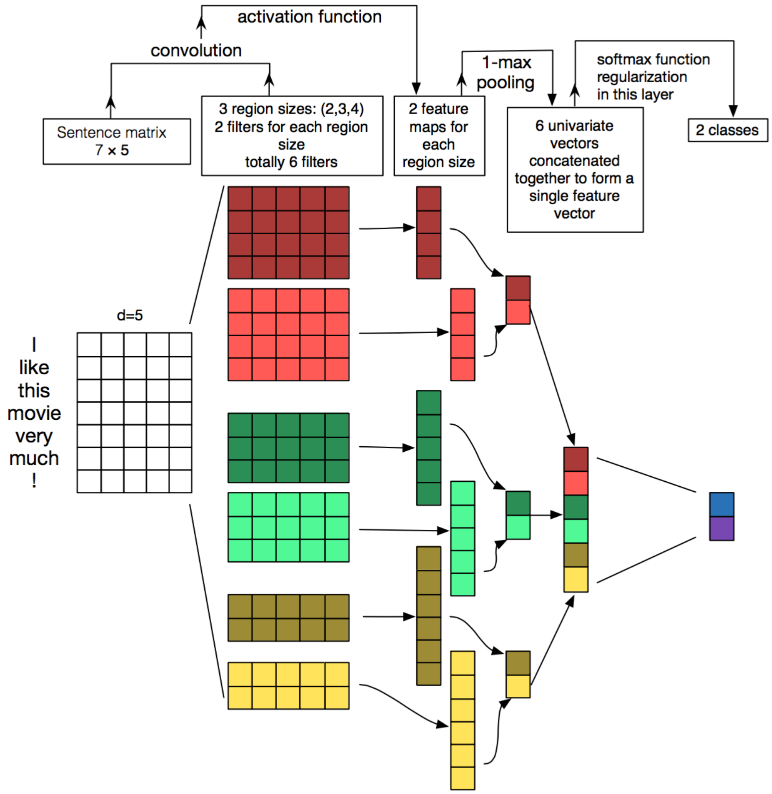
1. Convolutional networks
   - Summary
   - Interpretation
2. CNNs vs. RNNs
   - TCN
3. Graph2Seq Encoder
   - Beyond Seq2Seq
   - Model description
   - Comparison

Convolutional networks

Convolutional neural networks in NLP

Instead of pixels, the input to most NLP tasks is sentences and documents represented as matrices (e. g., each row is a vector which represents a word).
Thus, NLP filters slide over full rows of the matrix, and their width is the same as the matrix's.

Scheme



Source: www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Pooling

- Fixed output size provided by pooling allows us to use variable input or filter sizes
- Detection of specific features without their location

Channels

Different representations of the same data

- Visual recognition CNNs: RGB channels
- CNNs in NLP: different word embeddings, or different languages, or sentences phrased differently

Interpretability

Interpretability:

1. Model interpretability — structured explanation of how the model learns
2. Prediction interpretability — explanation of how the model arrived at its prediction

Compared to prediction interpretation, neural NLP model interpretation remains under-explored.

Common explanations

- 1-dimensional convolving filters detect ngrams
- max-pooling extracts the relevant ngrams
- the rest of the network classifies the text based on this information

Model for text classification

$$\text{input words } \mathrm{w}_1, ..., \mathrm{w}_n \rightarrow \text{vectors } w_1, ..., w_n \in \mathbb{R}^d$$

$$u_i = [w_i, ..., w_{i+l-1}]$$

$$F_{ij} = \langle u_i, f_j \rangle$$

$$p_j = ReLU(\max_i \{F_{ij}\})$$

$$o = softmax(Wp)$$

Datasets for Sentiment Analysis

Informative vs. uninformative ngrams

- Assumption (validated): there is a threshold for each filter, items below which are uninformative
- Deliberate vs. accidental ngrams
- Correlation between p_j and the predicted label for vector p:
  Filter f_j contributes to class $c_j = \arg\max_k W_{kj}$ (in linear case)
  Correlation label — compare c_j and the final decision by the network
  Classifier over a set of texts: pooled vectors p^i and network predictions c^i
  $\forall i : \text{dataset}(X, Y)_j = \{(p^i_j, c^i = c_j) | j < m \& i < D)\}$
  threshold t_j $purity(f, t) = \frac{|\{(x,y) \in (X,Y)_j | x \geq t \& y = true\}|}{|\{(x,y) \in (X,Y)_j | x \geq t\}|}$

Purity vs. test accuracy

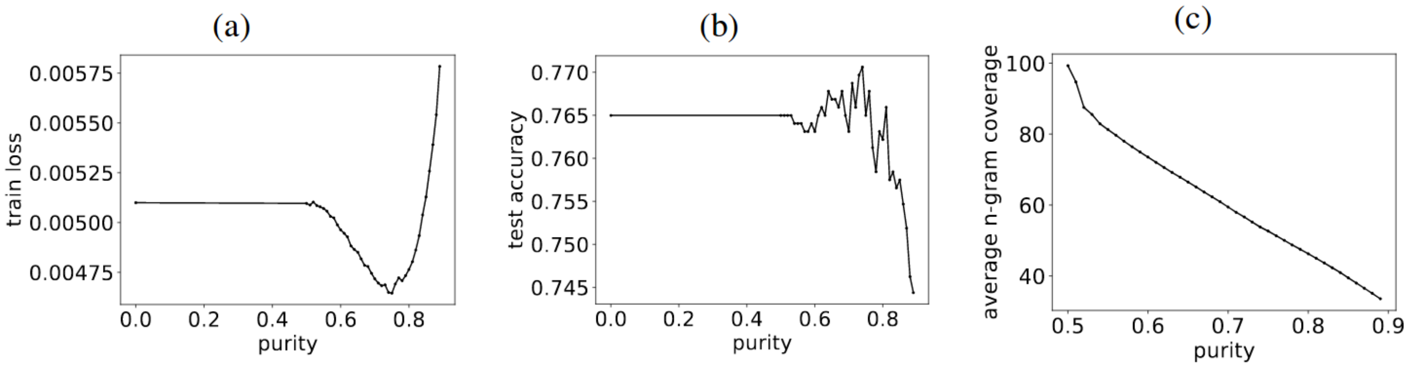


Figure 1: Evaluation results for identifying important ngrams on the MR model.

Source: arxiv.org/abs/1809.08037

What is captured by a filter?

Intuition (challenged): each feature is homogeneous.

- Slot activation vectors: $(\langle w_1, f(1)\rangle, ..., \langle w_l, f(l)\rangle)$

  Ngram score $\langle u, f\rangle = \sum_{i=0}^{l-1} \langle w_i, f_{id:i(d+1}\rangle$

  Slot i of the filter weight $f(i) = f_{id:i(d+1)}$
- On observed ngrams, the filters achieve maximum values only on some slots.

  Two hypotheses:
  - Each filter captures multiple semantic classes of ngrams (clustering deliberate ngrams according to their slot activation patterns)
  - A slot is used not to detect word existence, but lack thereof

Common explanations, refined

1. Filters
   - Heterogeneous (a single filter may detect different families of ngrams)
   - Detect negative items in ngrams
2. Max-pooling
   - Induces a threshold behaviour, with values below a given threshold ignored

Practical use

1. Model interpretability:
   "Visualise" each filter:
   - Class to which its strong signals correspond to
   - Threshold value, purity and coverage percentage
   - List of semantic patterns, each item corresponds to a slot-activations cluster, each cluster to top-k ngrams, each ngram to its total activation, slot-activating vector, list of bottom-k negative ngrams with their activations and slot activations
2. Prediction interpretability is improved by focusing on informative ngrams and taking into account the negative ones

CNNs vs. RNNs

Sequence Modelling

$f : \mathcal{X}^{T+1} \to \mathcal{Y}^{T+1}$
$\hat{y}_0, ..., \hat{y}_T = f(x_0, ..., x_T)$

Causal constraint: y_t depends only on $x_0, ..., x_t$

$L(y_0, ..., y_T, f(x_0, ..., x_T)) \to \min_f$

This formalism encompasses auto-regressive prediction, but does not capture sequence-to-sequence prediction in general, since the entire input sequence can be used to predict each output.

TCN

Temporal convolutional network:

1. can take an input of any length and map it to an output sequence of the same length
2. causal convolutions, no information “leakage” from future to past

Also, a combination of very deep networks and dilated convolutions is used to build very long effective history sizes.

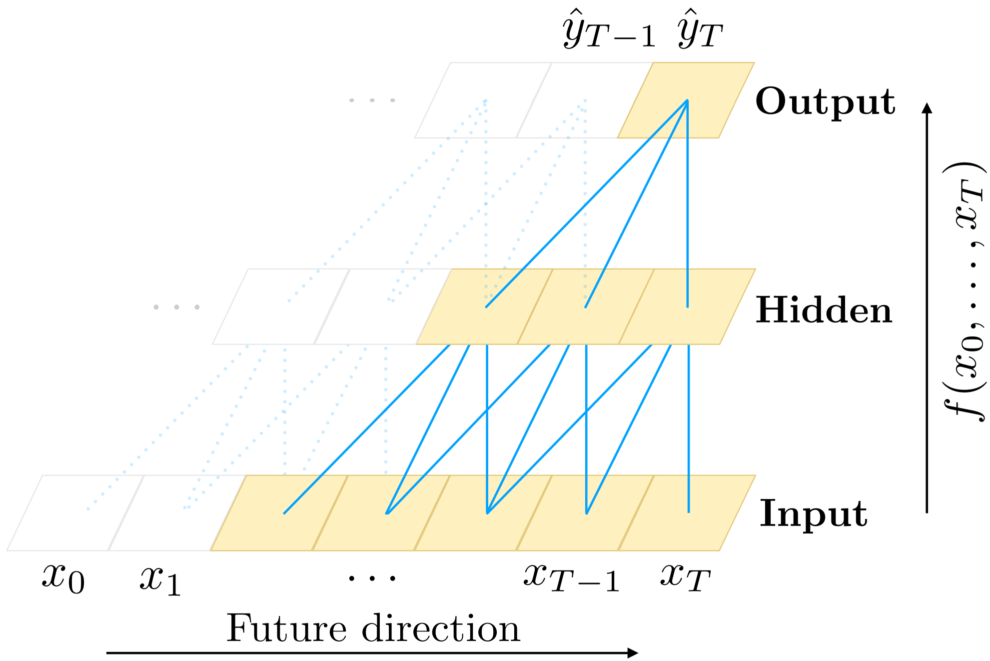
1D FCN

TCN = 1D FCN + causal convolutions

1D fully-convolutional (all layers are convolutional, none are fully-connected) network:

- each hidden layer is of the same length as the input layer
- zero padding of length (filter size - 1)

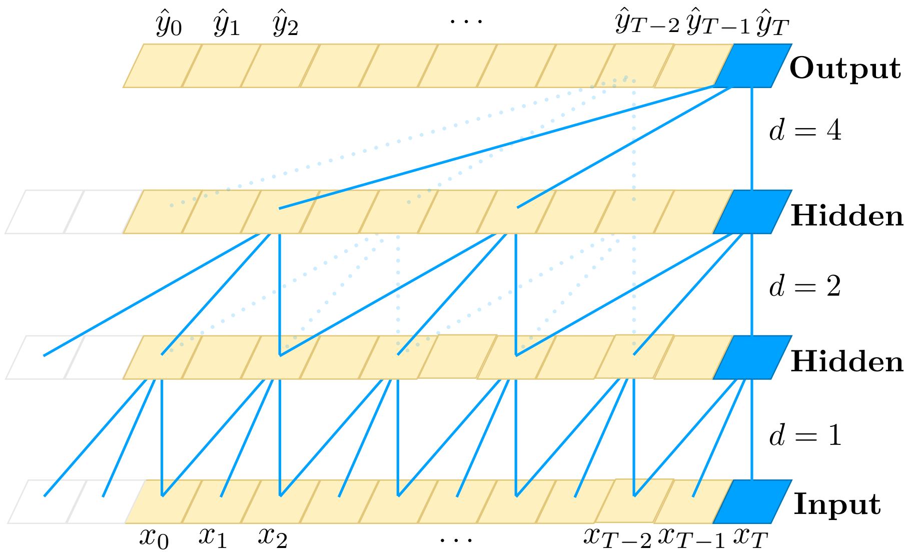
Causal convolutions



Source: arxiv.org/abs/1803.01271

Dilated convolutions

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i}$$

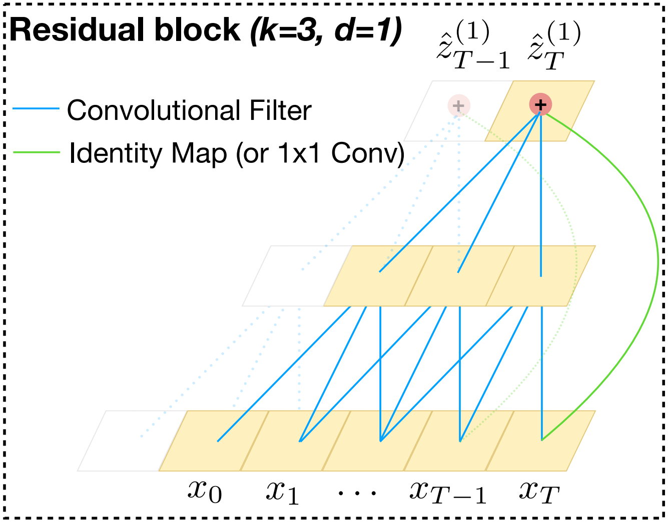


Source: arxiv.org/abs/1803.01271

Residual connections

$$o = \text{Activation}(x + \mathcal{F}(x))$$

Allows networks to learn modifications on the identity mapping which has repeatedly been shown beneficial.



Source: arxiv.org/abs/1803.01271

Pros and cons

Advantages:

- parallelism (contra RNNs)
- flexible receptive field size
- stable gradients (contra ordinary RNNs)
- low memory requirement for training (no need to store the partial results, the filters are shared across a layer)
- variable length inputs

Disadvantages:

- data storage during evaluation (need to take in the raw sequence up to the effective history length; RNNs only need to maintain a hidden state)
- potential parameter change for domain transfer (different requirements on the amount of history needed)

Tasks

1. Adding problem*
2. Sequential MNIST and P-MNIST*
3. Copy memory*
4. JSB Chorales and Nottingham
5. PennTreebank
6. Wikitext-103
7. LAMBADA
8. text8

*: synthetic stress tests

Results

Sequence Modeling Task	Model Size (≈)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

Source: arxiv.org/abs/1803.01271

Memory size of TCN and RNNs

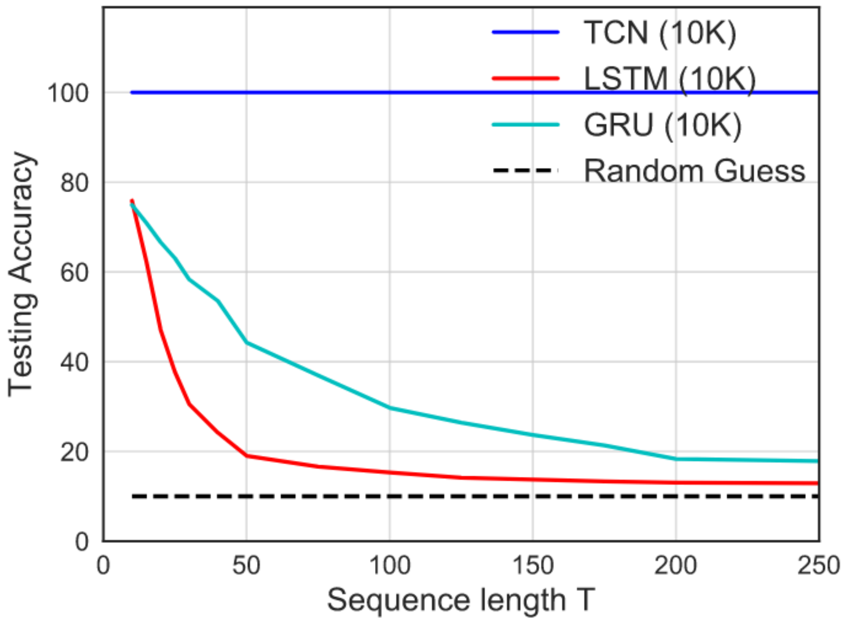


Figure 5. Accuracy on the copy memory task for sequences of different lengths T. While TCN exhibits 100% accuracy for all sequence lengths, the LSTM and GRU degenerate to random guessing as T grows.

Source: arxiv.org/abs/1803.01271

Comparison conclusions

Generic TCNs outperform generic RNNs, LSTMs and GRUs.

Advanced schemes for improving LSTMs have been proposed, while "TCN has not yet benefited from such community-wide investment".

Graph2Seq Encoder

Problems with Seq2Seq

- Sequences may be the simplest structured data
- Some objects are more naturally represented as graphs, which provide more structural information

Other encoders

- TreeLSTM
- Set2Seq
- Tree2Seq
- Graph2Seq using Gated Graph NNs
- Graph Convolutional Networks

Architecture example

1. Graph encoder
    1. Node embeddings
    2. Graph embeddings
2. Sequence decoder takes the embeddings and employs attention over node embeddings while generating sequences

Node embedding generation

Algorithm 1 Node embedding generation algorithm

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node initial feature vector $\mathbf{a}_v$, $\forall v \in \mathcal{V}$; hops K; weight matrices $\mathbf{W}^k$, $\forall k \in \{1, ..., K\}$; non-linearity σ; aggregator functions $\texttt{AGGREGATE}_k^{\vdash}$, $\texttt{AGGREGATE}_k^{\dashv}$, $\forall k \in \{1, ..., K\}$; neighborhood functions $\mathcal{N}_{\vdash}$, $\mathcal{N}_{\dashv}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_{v\vdash}^0 \leftarrow \mathbf{a}_v, \forall v \in \mathcal{V}$
$\mathbf{h}_{v\dashv}^0 \leftarrow \mathbf{a}_v, \forall v \in \mathcal{V}$
for all $k = 1...K$ **do**
  for all $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}_{\vdash}(v)}^k \leftarrow \texttt{AGGREGATE}_k^{\vdash}(\{\mathbf{h}_{u\vdash}^{k-1}, \forall u \in \mathcal{N}_{\vdash}(v)\})$
    $\mathbf{h}_{v\vdash}^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \texttt{CONCAT}(\mathbf{h}_{v\vdash}^{k-1}, \mathbf{h}_{\mathcal{N}_{\vdash}(v)}^k)\right)$
    $\mathbf{h}_{\mathcal{N}_{\dashv}(v)}^k \leftarrow \texttt{AGGREGATE}_k^{\dashv}(\{\mathbf{h}_{u\dashv}^{k-1}, \forall u \in \mathcal{N}_{\dashv}(v)\})$
    $\mathbf{h}_{v\dashv}^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \texttt{CONCAT}(\mathbf{h}_{v\dashv}^{k-1}, \mathbf{h}_{\mathcal{N}_{\dashv}(v)}^k)\right)$
  end for
end for
$\mathbf{z}_v \leftarrow \texttt{CONCAT}(\mathbf{h}_{v\vdash}^K, \mathbf{h}_{v\dashv}^K), \forall v \in \mathcal{V}$

Source: arxiv.org/abs/1804.00823

Aggregate functions

Must be invariant to input permutations.

- Element-wise mean
- LSTM on random permutations of neighbours
- Pooling:
  (neighbour's vector $\rightarrow$ fully-connected NN)$\cdot|\mathcal{N}(v)| \rightarrow$ max-pooling

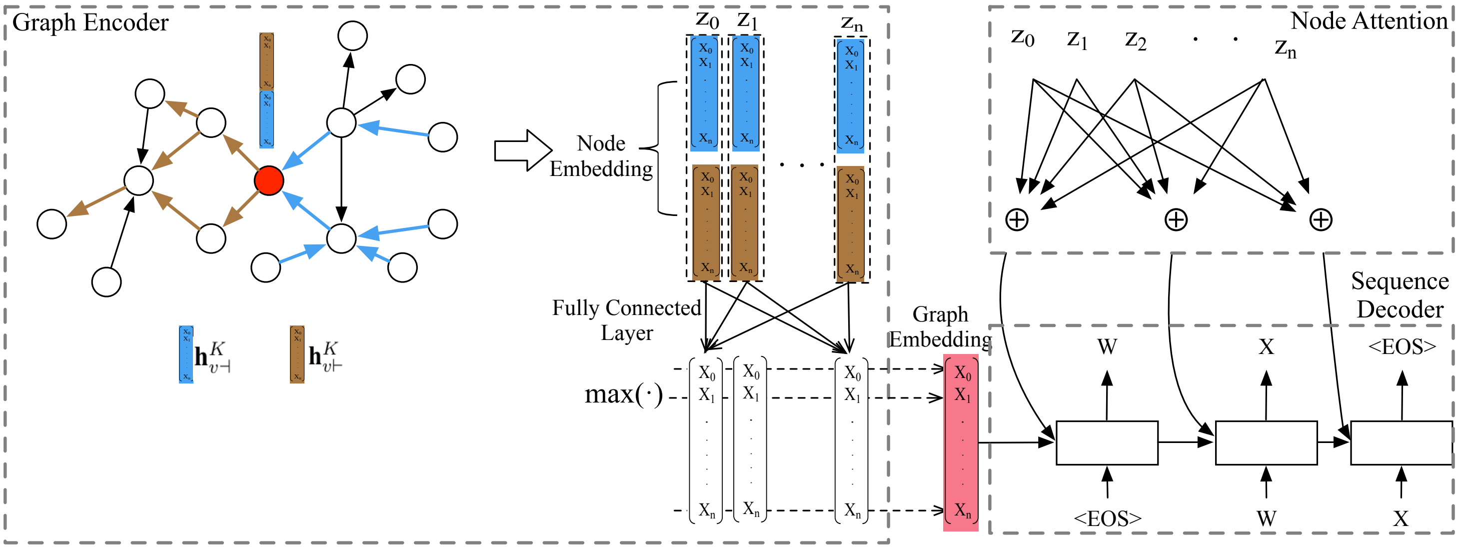
Graph embedding generation

Graph embeddings are generated from node embeddings.

Two ways:

1. Pooling-based: max-pooling, min-pooling, average-pooling (feeding them to a fully-connected NN did not reveal any significant performance difference, thus max-pooling is the default method)
2. Node-based:
   Add a super node, v_s, and direct the others to it. The node embedding of v_s obtained by aggregating the embeddings of its neighbour nodes is the target graph embedding

Scheme



Source: arxiv.org/abs/1804.00823

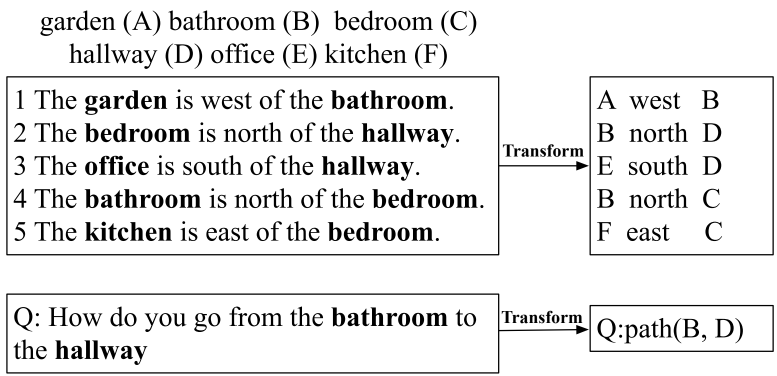
Settings

- Adam optimiser, mini-batch size $= 50$
- learning rate $= 0.001$
- dropout at the decoder level, ratio $= 0.5$
- clipped gradients, if $||\nabla|| > 20$
- hop size K $= 6$
- $\mathbf{a}_v \in \mathbb{R}^{40}$
- $\sigma =$ ReLU
- decoder: 1 layer, 80 hidden states

Default modification: mean aggregator and pooling-based graph embeddings.

bAbI Task 19

The bAbI artificial intelligence task 19 (path finding)



Source: arxiv.org/abs/1804.00823

Shortest Path Task

Goal: find the shortest directed path between two nodes in a graph.
SP-S (node size = 5), SP-L (node size = 100).

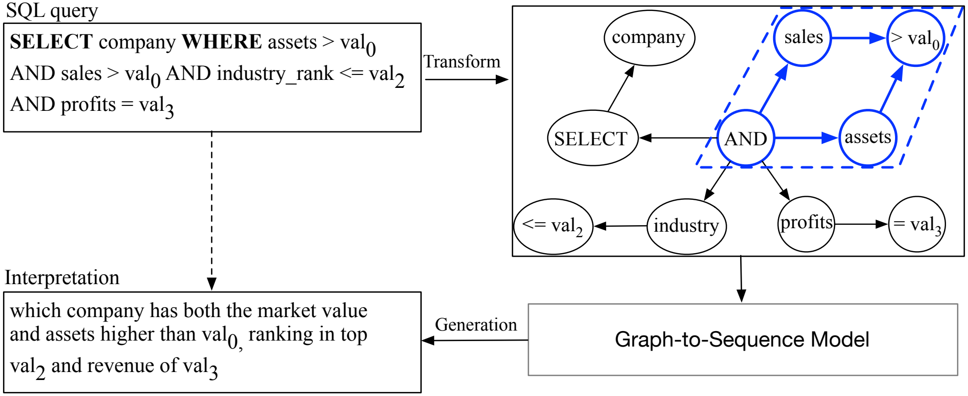
	bAbI T19	SP-S	SP-L
LSTM	25.2%	8.1%	2.2%
GGS-NN	98.1%	100.0%	95.2%
GCN	97.4%	100.0%	96.5%
Graph2Seq	**99.9%**	100.0%	**99.3%**

Table 1: Results of our model and baselines on bAbI and Shortest Directed Path tasks.

Source: arxiv.org/abs/1804.00823

Natural language generation

Goal: translate an SQL query to a natural language description expressing its meaning.



Source: arxiv.org/abs/1804.00823

	BLEU-4
Seq2Seq	20.91
Seq2Seq + Copy	24.12
Tree2Seq	26.67
GCN-PGE	35.99
GGS-NN	35.53
Graph2Seq-NGE	34.28
Graph2Seq-PGE	**38.97**

Table 2: Results on WikiSQL.

ToDo

This version of Graph2Seq encoding needs more testing on other data

Conclusions

- CNNs work well on NLP tasks
- A single filter often detects numerous semantic classes of ngrams
- Max-pooling induces a threshold behaviour
- Generic TCNs outperform generic RNNs across a broad range of sequence modelling tasks
- Graph2Seq is better suited for encoding certain types of structured data than Seq2Seq

References

- CNN in NLP in general:
  - General description http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
  - Understanding CNNs in NLP https://arxiv.org/abs/1809.08037
- CNNs vs RNNs: https://arxiv.org/abs/1803.01271
- Graph2Seq: https://arxiv.org/abs/1804.00823