Effective Attention Mechanisms for Sequence Learning

Li, Jian
Ph.D. Oral Defense
Supervisor: Prof. Michael R. Lyu
2020/06/16
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness...
Sequence Learning

• Discover valuable knowledge from sequential data.
  ▪ E.g., forecasting
  ▪ Extracting the patterns

• Manual analysis is inefficient and error-prone.
• Sequence learning automatically finds statistically relevant patterns.
Sequence Learning Tasks

- **Sequence Prediction**

  ![](It's a nice day)

  (Sentence Completion)

- **Sequence Generation**

  ![](It's a nice day)

  (Machine Translation)
The core problem of sequence learning is \textit{dependency modeling}.

[Agrawal et al., \textit{Mining Sequential Patterns}, 1995]
Traditional Sequence Learning

• Markov Models:

  - Fail to capture **long-term** dependencies
Neural Sequence Learning

- Recurrent Neural Networks (RNNs)

- Sequence-to-Sequence Learning
Attention Mechanism

• Hidden state bottleneck of RNN:
  ▪ The source sequence is encoded in one fixed-size vector.

✓ Attention adds **shortcut connections** to all source elements.

- Attention weights (connection strengths)
- Context vector (weighted summation)
Self-Attention Mechanism

• RNN’s sequential nature hinders parallel computation.

RNN

RNN with Attention

• Self-Attention Networks (SANs):
  ▪ Discard recurrent architectures
  ▪ Rely solely on attention mechanisms
  ▪ Simultaneously capture dependencies among all elements
  ▪ Positional encoding to record order information
Challenges for Attention Mechanisms

• Application domains other than text
  ▪ Source code data
  ▪ Highly structured, large vocabulary

• Model design deficiencies
  ▪ Deep self-attention involves multiple attention heads/layers.
  ▪ How to coordinate these components?
Thesis Contributions

- **Attention for Sequence Learning**
  - **Shallow Attention**
    - Code Completion (Chapter 3) [IJCAI’18]
  - **Deep Attention**
    - Multi-Head Attention (Chapter 4) [EMNLP’18, NAACL’19]
    - Representation Composition (Chapter 5) [AAAI’20]
  - **Pre-trained Attention Models**
    - Code Generation (Chapter 6) [*CIKM’20]

* In Submission
Outline

• Topic 1: Neural Attention for Code Completion

• Topic 2: Multi-Head Self-Attention

• Topic 3: Pre-trained Attention for Code Generation

• Conclusion and Future Work
Outline

• Topic 1: Neural Attention for Code Completion

• Topic 2: Multi-Head Self-Attention

• Topic 3: Pre-trained Attention for Code Generation

• Conclusion and Future Work
Code Completion

Code Suggestion

Choose a model:
neural_token

# generated code follows below

https://github.com/kootenpv/neural_complete

• Static programming language: C++, JAVA
  ▪ compile-time type information

• Dynamic programming language: Python, JavaScript
  ▪ learning-based language models
Code Completion with Language Models

• Simplified problem:
  ▪ given a sequence of code tokens, we predict the next one token.

\[
p(w_t | w_1, w_2, \ldots, w_{t-1}; \theta)
\]

• Method: adapt neural language models (e.g. RNNs) for code completion.
1. Long-range dependencies.

```javascript
// JavaScript source code

// Set Up Class
class Horse {
    constructor(name, trainer) {
        this._name = name;
        this._trainer = trainer;
    }

    get name() {
        return this._name;
    }

    get trainer() {
        return this._trainer;
    }
}

// Define Variable
txt = "";

// Create Instances
myHorse1 = new Horse('Spirit of Wedza', 'Julie Camaacho');
myHorse2 = new Horse('True North', 'Mark Prescott');

// Increment text
txt += myHorse1.name + " is trained by " + myHorse1.trainer + "<br>";
```

Challenges
Challenges

2. Out-of-Vocabulary (OoV) words.
   - Rare words
   - User-defined identifiers
Methods

• Deal with long-range dependencies:

  ✓ Abstract Syntax Tree (AST)

Problem: given a sequence of AST nodes, predict the next one AST node, including type and value.
Methods

• Deal with long-range dependencies:
  ✓ Abstract Syntax Tree (AST)

Exploit the parent-children information on program’s AST.
Attention Mechanisms

• Deal with long-range dependencies:
  ✓ Parent attention

\[
A_t = v^T \tanh(W^m M_t + (W^h h_t)^T) \\
\alpha_t = \text{softmax}(A_t) \\
c_t = M_t \alpha_t^T \\
G_t = \tanh(W^g [h_t; c_t; p_t]) \\
y_t = \text{softmax}(W^v G_t + b^v)
\]

\(p_t\) is the parent vector storing hidden state of the parent node on AST.
Methods

• Deal with OoV words:
  ✓ Locally repeated terms are prevalent.
  ✓ Intuition: **copy** from local context to predict OoVs. (Pointer Network)
Pointer Mixture Network

• Deal with OoV words:
  - Global RNN component
  - Local pointer component
    - Reuse the attention scores as the pointer distribution
  - Controller

\[ p_t = \sigma(W^p [h_t; c_t] + b^p) \]
\[ y_t = \text{softmax}([p_t w_t; (1 - p_t) l_t]) \]

Learns \textit{when} and \textit{where} to copy.
Experiments

• Datasets:
  ▪ JavaScript (JS) and Python (PY)
  ▪ Type prediction and value prediction.

<table>
<thead>
<tr>
<th></th>
<th>JavaScript</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Queries</td>
<td>$10.7 \times 10^7$</td>
<td>$6.2 \times 10^7$</td>
</tr>
<tr>
<td>Test Queries</td>
<td>$5.3 \times 10^7$</td>
<td>$3.0 \times 10^7$</td>
</tr>
<tr>
<td>Type Vocabulary</td>
<td>95</td>
<td>329</td>
</tr>
<tr>
<td>Value Vocabulary</td>
<td>$2.6 \times 10^6$</td>
<td>$3.4 \times 10^6$</td>
</tr>
</tbody>
</table>
Experiments

- Accuracies on **next value prediction with different vocabulary sizes**

<table>
<thead>
<tr>
<th>Vocab. Size</th>
<th>JS (1 K)</th>
<th>JS (10 K)</th>
<th>JS (50 K)</th>
<th>PY (1 K)</th>
<th>PY (10 K)</th>
<th>PY (50 K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OoV Rate</td>
<td>20%</td>
<td>11%</td>
<td>7%</td>
<td>24%</td>
<td>16%</td>
<td>11%</td>
</tr>
<tr>
<td>Vanilla LSTM</td>
<td>69.9%</td>
<td>75.8%</td>
<td>78.6%</td>
<td>63.6%</td>
<td>66.3%</td>
<td>67.3%</td>
</tr>
<tr>
<td>Attentional LSTM (Ours)</td>
<td>71.7%</td>
<td>78.1%</td>
<td>80.6%</td>
<td>64.9%</td>
<td>68.4%</td>
<td>69.8%</td>
</tr>
<tr>
<td>Pointer Mixture Network (Ours)</td>
<td>73.2%</td>
<td>78.9%</td>
<td><strong>81.0%</strong></td>
<td><strong>66.4%</strong></td>
<td><strong>68.9%</strong></td>
<td><strong>70.1%</strong></td>
</tr>
</tbody>
</table>

- Vanilla LSTM: without attention nor pointer.
- Attentional LSTM: context attention and parent attention.
- Pointer Mixture Network: both attention and pointer.
Experiments

- Is the learned pointer distribution meaningful?
  - Pointer Random Network: randomize the pointer distribution

<table>
<thead>
<tr>
<th>Model</th>
<th>JS_1k</th>
<th>PY_1k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer Random Network</td>
<td>71.4%</td>
<td>64.8%</td>
</tr>
<tr>
<td>Attentional LSTM</td>
<td>71.7%</td>
<td>64.9%</td>
</tr>
<tr>
<td>Pointer Mixture Network</td>
<td><strong>73.2%</strong></td>
<td><strong>66.4%</strong></td>
</tr>
</tbody>
</table>

Pointer mixture network indeed learns *when and where* to copy OoVs.
Summary

1. Propose a **parent attention** mechanism for AST-based code completion.
2. Propose a **pointer mixture network** which learns to either generate a new value or copy an OoV value from local context.
3. Demonstrate the **effectiveness** of our model via experiments.
Outline

• Topic 1: Neural Attention for Code Completion

• Topic 2: Multi-Head Self-Attention

• Topic 3: Pre-trained Attention for Code Generation

• Conclusion and Future Work
Self-Attention Mechanism

Linear Transformation

\[
\begin{bmatrix}
Q \\
K \\
V
\end{bmatrix} = H \begin{bmatrix}
W_Q \\
W_K \\
W_V
\end{bmatrix}
\]

[Vaswani et al., *Attention is All You Need*, 2017]
Self-Attention Mechanism

Linear Transformation
\[
\begin{bmatrix}
Q \\
K \\
V
\end{bmatrix} = H \begin{bmatrix}
W_Q \\
W_K \\
W_V
\end{bmatrix}
\]

Attention Weights
\[
\text{Att}(Q, K) = \text{softmax}(\frac{QK^T}{\sqrt{d}})
\]

[Vaswani et al., Attention is All You Need, 2017]
Self-Attention Mechanism

Linear Transformation

\[
\begin{bmatrix} Q \\ K \\ V \end{bmatrix} = H \begin{bmatrix} W_Q \\ W_K \\ W_V \end{bmatrix}
\]

Attention Weights

\[
\text{Att}(Q, K) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)
\]

Weighted Sum

\[
O = \text{Att}(Q, K) \cdot V
\]

[Vaswani et al., *Attention is All You Need*, 2017]
Multi-Head Self-Attention

$$[Q^h, K^h, V^h] = H \begin{bmatrix} W_{Q^h} \\ W_{K^h} \\ W_{V^h} \end{bmatrix}$$

$$\text{Att}(Q^h, K^h) = \text{softmax}\left(\frac{Q^h K^h T}{\sqrt{d_k}}\right)$$

$$O^h = \text{Att}(Q^h, K^h) \cdot V^h$$

[Vaswani et al., *Attention is All You Need*, 2017]
Multi-Head Self-Attention

\[
\begin{bmatrix}
Q^h \\
K^h \\
V^h
\end{bmatrix}
= H \begin{bmatrix}
W_{Q^h} \\
W_{K^h} \\
W_{V^h}
\end{bmatrix}
\]

\[
\text{Att}(Q^h, K^h) = \text{softmax}\left(\frac{Q^h K^h^T}{\sqrt{d_k}}\right)
\]

\[
o^h = \text{Att}(Q^h, K^h) \cdot V^h
\]

\[
o_f = \text{Concat}[o^1, \ldots, o^H]W_o
\]

Bush held a talk with Sharon.

[Vaswani et al., Attention is All You Need, 2017]
Deficiencies in Multi-Head Attention

- **Diversity**: jointly extract information from **different** representation subspaces at **different** positions.

1. There is no mechanism to guarantee that different attention heads indeed capture distinct information.
   -- Information Extraction

2. The “Concat+Linear” is not expressive enough to aggregate the diverse sub-representations.
   -- Information Aggregation
Our Solutions

1. We introduce a **disagreement regularization** to explicitly encourage the diversity.
   -- Information Extraction

2. We replace the standard linear transformation with an **advanced aggregation function**.
   -- Information Aggregation
Disagreement Regularization

• Revise the training objective for seq2seq learning (x \rightarrow y):

\[
J(\theta) = \arg \max_{\theta} \left\{ \underbrace{L(y|x; \theta)}_{\text{likelihood}} + \lambda \underbrace{D(a|x, y; \theta)}_{\text{disagreement}} \right\}
\]

- The auxiliary regularization term $D(\cdot)$ enlarges the distances among multiple attention heads.
- Do not introduce any new parameters.
Three Types of Disagreement

- **Disagreement on Subspaces** that maximizes the cosine distance among the projected values:

  \[
  D_{\text{subspace}} = -\frac{1}{H^2} \sum_{i=1}^{H} \sum_{j=1}^{H} \frac{V^i \cdot V^j}{\|V^i\| \|V^j\|}.
  \]

- **Disagreement on Positions** that disperses the attended positions predicted by different heads:

  \[
  D_{\text{position}} = -\frac{1}{H^2} \sum_{i=1}^{H} \sum_{j=1}^{H} |A^i \odot A^j|. \quad A^h = \text{softmax}\left(\frac{Q^h K^h^T}{\sqrt{d_k}}\right)
  \]

- **Disagreement on Outputs** that maximizes the cosine distance among the outputs of multiple heads:

  \[
  D_{\text{output}} = -\frac{1}{H^2} \sum_{i=1}^{H} \sum_{j=1}^{H} \frac{O^i \cdot O^j}{\|O^i\| \|O^j\|}.
  \]
Advanced Aggregation Function

- Linear transformation is a suboptimal feature fusion approach in multi-modal research [1].

We borrow the idea of routing-by-agreement from Capsule Networks [2,3].

- Iteratively update the proportion of how much a part should be assigned to a whole.

Routing-by-Agreement

- The information of $H$ input capsules is dynamically routed to $N$ output capsules.

$$\Omega_{h}^{in} = f_h(\widehat{O})$$

$$V_{h \rightarrow n} = \Omega_{h}^{in} W_{h \rightarrow n}$$

$$\Omega_{n}^{out} = \frac{\sum_{h=1}^{H} C_{h \rightarrow n} V_{h \rightarrow n}}{\sum_{h=1}^{H} C_{h \rightarrow n}}$$

- Concatenate $N$ output capsules to form the final.

$$O = [\Omega_1^{out}, \ldots, \Omega_N^{out}]$$
Routing-by-Agreement

- Two representative routing algorithms for $C_{h\rightarrow n}$:

**Algorithm 1** Iterative Simple Routing.

1: procedure ROUTING($V$, $T$):
2:   $\forall V_{h\rightarrow n}: B_{h\rightarrow n} = 0$
3:   for $T$ iterations do
4:     $\forall V_{h\rightarrow n}: C_{h\rightarrow n} = \frac{\exp(B_{h\rightarrow n})}{\sum_{n'=1}^{N} \exp(B_{h\rightarrow n'})}$
5:     $\forall \Omega_{n}^{out}: \text{compute } \Omega_{n}^{out} \text{ by Eq. 7}$
6:     $\forall V_{h\rightarrow n}: B_{h\rightarrow n} += \Omega_{n}^{out} \cdot V_{h\rightarrow n}$
   return $\Omega$

**Algorithm 2** Iterative EM Routing.

1: procedure EM ROUTING($V$, $T$):
2:   $\forall V_{h\rightarrow n}: C_{l\rightarrow n} = 1/N$
3:   for $T$ iterations do
4:     $\forall \Omega_{n}^{out}: \text{M-STEP($V$, $C$)}$ ▷ hold $C$ constant, adjust $(\mu_n, \sigma_n, A_n)$
5:     $\forall V_{h\rightarrow n}: \text{E-STEP($V$, $\mu$, $\sigma$, $A$)}$ ▷ hold $(\mu, \sigma, A)$ constant, adjust $C_{h\rightarrow n}$
6:     $\forall \Omega_{n}^{out}: \Omega_{n}^{out} = A_n * \mu_n$
   return $\Omega$

Two Complementary Work

- Disagreement Regularization: [EMNLP’18]
  - improve information extraction
  - only adjust loss function

- Advanced Aggregation Function: [NAACL’19]
  - improve information aggregation
  - modify the network architecture

They are complementary to each other and can be applied simultaneously.
Experiments

• Transformer for Seq2Seq
  ▪ Machine Translation

[Vaswani et al., Attention is All You Need, 2017]
Experiments

• Evaluation study on **disagreement regularization**:

<table>
<thead>
<tr>
<th>#</th>
<th>Regularization</th>
<th>Speed</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>×   ×  ×</td>
<td>1.21</td>
<td>24.13</td>
</tr>
<tr>
<td>2</td>
<td>✓   ×  ×</td>
<td>1.15</td>
<td>24.64</td>
</tr>
<tr>
<td>3</td>
<td>×   ✓  ×</td>
<td>1.14</td>
<td>24.42</td>
</tr>
<tr>
<td>4</td>
<td>×   ×  ✓</td>
<td>1.15</td>
<td><strong>24.78</strong></td>
</tr>
<tr>
<td>5</td>
<td>✓   ×  ✓</td>
<td>1.12</td>
<td>24.73</td>
</tr>
<tr>
<td>6</td>
<td>✓   ✓  ×</td>
<td>1.11</td>
<td>24.38</td>
</tr>
<tr>
<td>7</td>
<td>✓   ✓  ✓</td>
<td>1.05</td>
<td>24.60</td>
</tr>
</tbody>
</table>

Table 1: Effect of regularization terms, which are applied to the encoder self-attention only. “Speed” denotes the training speed (steps/second). Results are reported on the WMT17 Zh⇒En translation task using Transformer-Base.

Only employing **output disagreement** is most effective (Row 4).
Experiments

• Evaluation study on advanced aggregation function:

<table>
<thead>
<tr>
<th>#</th>
<th>Applying Aggregation to ...</th>
<th>Routing</th>
<th># Para.</th>
<th>Speed</th>
<th>BLEU</th>
<th>△</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enc-Self</td>
<td>Enc-Dec</td>
<td>Dec-Self</td>
<td>n/a</td>
<td>88.0M</td>
<td>1.92</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>Simple</td>
<td>+12.6M</td>
<td>1.23</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>EM</td>
<td>+12.6M</td>
<td>1.20</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>EM</td>
<td>+12.6M</td>
<td>1.20</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>EM</td>
<td>+12.6M</td>
<td>1.21</td>
</tr>
<tr>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>EM</td>
<td>+25.2M</td>
<td>0.87</td>
</tr>
<tr>
<td>8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>EM</td>
<td>+37.8M</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Table 2: Effect of information aggregation on different attention components, i.e., encoder self-attention (“Enc-Self”), encoder-decoder attention (“Enc-Dec”), and decoder self-attention (“Dec-Self”). “# Para.” denotes the number of parameters, and “Train” and “Decode” respectively denote the training speed (steps/second) and decoding speed (sentences/second).

Applying EM Routing at the encoder side best balances effectiveness and efficiency (Row 4).
Experiments

• Combining together and main results:

<table>
<thead>
<tr>
<th>BLEU (%)</th>
<th>En-De</th>
<th>Zh-En</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>27.31</td>
<td>24.13</td>
</tr>
<tr>
<td>27</td>
<td>28.2</td>
<td>24.85</td>
</tr>
<tr>
<td>29</td>
<td>28.26</td>
<td>24.68</td>
</tr>
<tr>
<td>29</td>
<td>28.41</td>
<td>24.9</td>
</tr>
</tbody>
</table>

- Blue: Transformer-Base
- Green: + Disagreement
- Orange: + Aggregation
- Red: + Both
Summary

1. Propose **disagreement regularization** to improve the information extraction in multi-head attention.

2. Propose **routing-by-agreement** aggregation function to adjust the information aggregation in multi-head attention.

3. The two approaches are complementary to each other.

\[
J(\theta) = \arg \max_{\theta} \left\{ L(y|x;\theta) + \lambda \cdot D(a|x,y;\theta) \right\}
\]

[Diagram of Multi-Head Attention]
Outline

• Topic 1: Neural Attention for Code Completion

• Topic 2: Multi-Head Self-Attention

• Topic 3: Pre-trained Attention for Code Generation

• Conclusion and Future Work
Semantic Parsing

- Map natural language utterances to logical forms or executable code.
  - Natural language understanding

<table>
<thead>
<tr>
<th>Player</th>
<th>Country</th>
<th>Points</th>
<th>Winnings($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Stricker</td>
<td>United States</td>
<td>9000</td>
<td>1260000</td>
</tr>
<tr>
<td>K.J. Choi</td>
<td>South Korea</td>
<td>5400</td>
<td>756000</td>
</tr>
<tr>
<td>R. Sabbatini</td>
<td>South Africa</td>
<td>3400</td>
<td>4760000</td>
</tr>
<tr>
<td>M. Calca</td>
<td>United States</td>
<td>2067</td>
<td>289333</td>
</tr>
<tr>
<td>E. Els</td>
<td>South Africa</td>
<td>2067</td>
<td>289333</td>
</tr>
</tbody>
</table>

**Question:** What is the points of South Korea player?

**SQL:** SELECT Points WHERE Country = South Korea

**Answer:** 5400

[Zhong et al., Seq2SQL, 2017]
Neural Semantic Parsing

- Neural Sequence-to-Sequence models:
  - Encoder: encode the natural language semantics
  - Decoder: generate the corresponding code
    - Sequential: bad performance due to lack of data
    - Syntax specific: external knowledge

[Image of diagrams showing the process of parsing and generating code]

Input: sort my_list in descending order

Code: sorted(my_list, reverse=True)

[Yin et al., ACL 2017]
Pre-trained Models

- Pre-trained on large-scale text corpus.
  - **Universal** language representations
  - Self-attentional Transformer nets
  - BERT, GPT, XLNet, etc.

- Fine-tune on downstream tasks.
  - Transfer pre-trained knowledge
  - Limited data
Universal pre-trained attention models

Build semantic parsers that are both effective and generalizable?
Method

• BERT-LSTM
  • Simplicity
  • Extensibility
  • Effectiveness

![Diagram showing the BERT-LSTM method]

- Minimal additional parameters
  - Pre-trained BERT
  - Attention
  - Pool
  - LSTM
  - Embedding

Natural Language:
- [CLS]
- find
- restaurants
- ...

Programming Code:
- [BOS]
- Restaurant()
- ...

Output Distribution
- copy or generate

Pointer Distribution
- Vocab Distribution

Feed Forward
Experiments

• Datasets:
  - Almond (Restaurant and People): NL question -> ThingTalk code
  - Django: NL description -> Python code
  - WiKiSQL: Table, NL Query -> SQL code

<table>
<thead>
<tr>
<th>Input</th>
<th>Table has columns: Conference Division Team City Home_Arena, which team is in the southeast with a home at Philips?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref.</td>
<td>SELECT (Team) FROM table WHERE Division = southeast AND Home_Arena = Philips</td>
</tr>
</tbody>
</table>

• Metric: **exact** match accuracy
Evaluation Study

• LSTM decoder vs. Transformer decoder
• Greedy decoding vs. Beam search
• Fine-tune BERT vs. Freeze BERT

*Experiments on Almond-Restaurant
Experiments

• Accuracies on Almond and Django:

<table>
<thead>
<tr>
<th>Model</th>
<th>ALMOND-RESTAURANT</th>
<th>ALMOND-People</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAN [102]</td>
<td>68.92%</td>
<td>75.65%</td>
</tr>
<tr>
<td>BERT-LSTM</td>
<td><strong>74.01%</strong></td>
<td><strong>81.93%</strong></td>
</tr>
<tr>
<td>- copying</td>
<td>56.76%</td>
<td>59.78%</td>
</tr>
</tbody>
</table>

Table 6.2: Code generation accuracies on the two ALMOND datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence-to-Tree Network [34]</td>
<td>39.4%</td>
</tr>
<tr>
<td>Neural Machine Translation [162]</td>
<td>45.1%</td>
</tr>
<tr>
<td>Latent Predictor Network [94]</td>
<td>62.3%</td>
</tr>
<tr>
<td>Syntax Neural Model [162]</td>
<td>71.6%</td>
</tr>
<tr>
<td>Transition-Based Syntax Parser [163]</td>
<td>73.7%</td>
</tr>
<tr>
<td>Coarse-to-Fine Decoding [35]</td>
<td>74.1%</td>
</tr>
<tr>
<td>BERT-LSTM</td>
<td><strong>76.48%</strong></td>
</tr>
<tr>
<td>- copying</td>
<td><strong>54.07%</strong></td>
</tr>
</tbody>
</table>

Table 6.3: Python code generation accuracies on DJANGO. BERT-LSTM achieves state-of-the-art result.
### Experiments

- **Accuracies on WiKiSQL:**

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence-to-Sequence [171]</td>
<td>23.4%</td>
</tr>
<tr>
<td>Sequence-to-SQL [171]†</td>
<td>48.3%</td>
</tr>
<tr>
<td>SQLNet [154]†</td>
<td>61.3%</td>
</tr>
<tr>
<td>Transition-based Syntax Parser [163]†</td>
<td>68.6%</td>
</tr>
<tr>
<td>Coarse-to-Fine Decoding [35]†</td>
<td>71.7%</td>
</tr>
<tr>
<td>Multi-Task QA Network [102]</td>
<td>75.4%</td>
</tr>
<tr>
<td>SQLova [65]†</td>
<td>83.6%</td>
</tr>
<tr>
<td>X-SQL [57]†</td>
<td>86.0%</td>
</tr>
<tr>
<td>HydraNet [100]†</td>
<td><strong>86.5%</strong></td>
</tr>
<tr>
<td>BERT-LSTM*</td>
<td>78.49%</td>
</tr>
<tr>
<td>– copying</td>
<td>35.99%</td>
</tr>
</tbody>
</table>

Table 6.4: SQL code generation accuracies on WiKiSQL. “†” denotes syntax-specific models. “*” indicates that the model employs pre-trained BERT.
Case Study

• Almond Virtual Assistant

| Input: Show me restaurants in San Francisco rated at least 4.5 stars. |
| Pred.: now => (@org.schema.Restaurant.Restaurant) filter param:geo:Location == "San Francisco" and param:ratingValue:Number >= 4.5 => notify ✓ |

*Highlighted words: copying probability > 0.9

https://almond.stanford.edu/
Summary

• Propose BERT-LSTM model for semantic parsing/code generation that is both effective and generalizable.

• Achieve state-of-the-art on three of the four experimental datasets.
Outline

• Topic 1: Neural Attention for Code Completion

• Topic 2: Multi-Head Self-Attention

• Topic 3: Pre-trained Attention for Code Generation

• Conclusion and Future Work
Conclusion

- **Parent attention on AST**
- **Pointer mixture network**
- **Disagreement regularization**
- **Routing-by-agreement aggregation function**
- **BERT-LSTM model**
Future Work

• Multi-Modal Attention Models
  ▪ Textual and visual

A **dog** is standing on a hardwood floor.

A group of **people** sitting on a boat in the water.
Future Work

• Interpretability and Reliability of Attention Models
  ▪ Adversarial attacks


Baosong Yang, Jian Li, Derek Wong, Lidia Chao, Xing Wang, Zhaopeng Tu. Context-Aware Self-Attention Networks. The 33rd AAAI Conference on Artificial Intelligence (AAAI), 2019.


Jian Li, Yue Wang, Michael R. Lyu, Irwin King. Code Completion with Neural Attention and Pointer Networks. The 28th International Joint Conference on Artificial Intelligence (IJCAI), 2018.


Silei Xu, Giovanni Campagna, Jian Li, Monica S. Lam. Schema2QA: Answering Complex Queries on the Structured Web with a Neural Model. In submission to CIKM 2020.
物有本末，事有終始。知所先後，則近道矣。
Things have their roots and branches, affairs have their end and beginning. When you know what comes first and what comes last, then you are near the Way.

-《大學》
The Great Learning
Thanks!