Contextual Memory Trees

Wen Sun
CMU —> MSR NYC
wensun@cs.cmu.edu

Joint work with Alina Beygelzimer, Hal Daumé III, Paul Mineiro, and John Langford
External Memory

External Memory
(many query-answer pairs)
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

Answer
(A Chinese sentence)
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

K memories:
(K English-Chinese pairs)

Answer
(A Chinese sentence)
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

Answer
(A Chinese sentence)

K memories:
(K English-Chinese pairs)

An external Inference/learning algorithm
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

Answer
(A Chinese sentence)

K memories:
(K English-Chinese pairs)

An external inference/learning algorithm

Answer
(A Chinese sentence)
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

K memories:
(K English-Chinese pairs)

Answer
(A Chinese sentence)

User provides
rewards (BLUE score) &
ground-truth (A Chinese sentence)

Answer
(A Chinese Sentence)

An external
Inference/learning
algorithm
External Memory

Query (e.g., An English sentence)

External Memory
(many query-answer pairs)

K memories:
(K English-Chinese pairs)

Answer
(A Chinese sentence)

User provides
rewards (BLUE score) &
ground-truth (A Chinese sentence)

An external
Inference/learning
algorithm

Answer
(A Chinese Sentence)
Desired Properties
Desired Properties

**Online**: every operation works one example at a time
Desired Properties

**Online**: every operation works one example at a time

**Linear Space**: $O(\# \text{ of examples})$
Desired Properties

**Online**: every operation works one example at a time

**Linear Space**: $O(#\ \text{of examples})$

**Fast**: Logarithmic Read & Write ($\log(#\ \text{of examples})$)
Desired Properties

Online: every operation works one example at a time

Linear Space: $O(# \text{ of examples})$

Fast: Logarithmic Read & Write (Log(# of examples))

Learning-based: do not assume, e.g., Euclidean space
Desired Properties

**Online**: every operation works one example at a time

**Linear Space**: $O(\# \text{ of examples})$

**Fast**: Logarithmic Read & Write ($\log(\# \text{ of examples})$)

**Learning-based**: do not assume, e.g., Euclidean space

**Self-consistency**: identify the item seen before
Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of Query (q) & Value (v)

e.g., (A English Sentence, A Chinese Sentence)
Learning Protocol of Contextual Memory Tree (CMT)

A memory \( (m) \): a pair of Query \( (q) \) & Value \( (v) \)
e.g., (A English Sentence, A Chinese Sentence)

1. Given a query \( q \),
Learning Protocol of Contextual Memory Tree (CMT)

A memory \( (m) \): a pair of Query \( (q) \) & Value \( (v) \)

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query \( q \),

\[ \text{QUERY}(q) \]
Learning Protocol of Contextual Memory Tree (CMT)

A memory \( (m) \): a pair of Query \( (q) \) & Value \( (v) \)

e.g., (A English Sentence, A Chinese Sentence)

1. Given a query \( q \),

\[ \text{QUERY}(q) \]

\[ (m_1, \ldots, m_k) \]
Learning Protocol of Contextual Memory Tree (CMT)

A memory \( (m) \): a pair of Query \( (q) \) & Value \( (v) \)
e.g., (A English Sentence, A Chinese Sentence)

1. Given a query \( q \),

\[
\text{QUERY}(q) \\
(m_1, \ldots, m_k)
\]

2. If reward \( r_i \) is given,
Learning Protocol of Contextual Memory Tree (CMT)

A memory \((m)\): a pair of **Query** \((q)\) & **Value** \((v)\)
e.g., (A English Sentence, A Chinese Sentence)

1. Given a query \(q\),

\[
\text{QUERY}(q) \rightarrow (m_1, \ldots, m_k)
\]

2. If reward \(r_i\) is given,

\[
\text{UPDATE}(q, m_i, r_i)
\]
Learning Protocol of Contextual Memory Tree (CMT)

A memory (m): a pair of Query (q) & Value (v)
e.g., (A English Sentence, A Chinese Sentence)

1. Given a query q,

\[
\text{QUERY}(q) \quad (m_1, \ldots, m_k)
\]

2. If reward \( r_i \) is given,

\[
\text{UPDATE}(q, m_i, r_i)
\]

3. If ground-truth value \( v \) is given,
Learning Protocol of Contextual Memory Tree (CMT)

A memory \( (m) \): a pair of Query (q) & Value (v)

\[ \text{e.g., (A English Sentence, A Chinese Sentence)} \]

1. Given a query q,

\[ \text{QUERY}(q) \]

\[ (m_1, \ldots, m_k) \]

2. If reward \( r_i \) is given,

\[ \text{UPDATE}(q, m_i, r_i) \]

3. If ground-truth value \( v \) is given,

\[ \text{INSERT}(q, v) \]
Datastructure of CMT: A Nearly Balanced Binary Tree

$\text{Datastructure of CMT: A Nearly Balanced Binary Tree}$

$\text{Datastructure of CMT: A Nearly Balanced Binary Tree}$
Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

\[ n_L, n_R \]
# of memories in the Left/Right subtree

Router (binary classifier):
\[ f : Q \to \{-1, +1\} \]
e.g., \( f(q) = \text{sign}(\beta^T \phi(q)) \)
Datastructure of CMT: A Nearly Balanced Binary Tree

**Internal node**

\[ n_L, n_R \]

# of memories in the Left/Right subtree

**Router (binary classifier):**

\[ f : Q \rightarrow \{-1, +1\} \]

e.g., \( f(q) = \text{sign}(\beta^T \phi(q)) \)

**Scorer Function:**

\[ g(q, x) = w^T \phi(q, x) \]

**Memories:**

\[ m_1, \ldots m_n \]
Datastructure of CMT: A Nearly Balanced Binary Tree

- **Internal node**
  - $n_L, n_R$
  - # of memories in the Left/Right subtree

- **Router (binary classifier):**
  - $f : Q \rightarrow \{-1, +1\}$
  - e.g., $f(q) = \text{sign}(\beta^T \phi(q))$

- **Scorer Function:**
  - $g(q, x) = w^T \phi(q, x)$

- **Leaf**

Memories: $m_1, \ldots, m_n$
Datastructure of CMT: A Nearly Balanced Binary Tree

- **Internal node**
  - $n_L, n_R$
  - # of memories in the Left/Right subtree

- **Router (binary classifier):**
  - $f : Q \rightarrow \{-1, +1\}$
  - e.g., $f(q) = \text{sign}(\beta^T \phi(q))$

- **Scorer Function:**
  - $g(q, x) = w^T \phi(q, x)$

- **Leaf**
  - Memories: $m_1, \ldots m_n$
Datastructure of CMT: A Nearly Balanced Binary Tree

**Internal node**

\[ n_L, n_R \]

# of memories in the Left/Right subtree

**Router (binary classifier):**

\[ f : Q \to \{-1, +1\} \]

e.g., \[ f(q) = \text{sign}(\beta^T \phi(q)) \]

**Memories:** \[ m_1, \ldots, m_n \]

**Scorer Function:**

\[ g(q, x) = w^T \phi(q, x) \]
Datastructure of CMT:
A Nearly Balanced Binary Tree

Router (binary classifier):
\[ f : Q \rightarrow \{-1, +1\} \]
e.g., \[ f(q) = \text{sign}(\beta^T \phi(q)) \]

Scorer Function:
\[ g(q, x) = w^T \phi(q, x) \]
Datastructure of CMT:
A Nearly Balanced Binary Tree

Internal node

$n_L, n_R$

# of memories in the Left/Right subtree

Router (binary classifier):
$f : Q \rightarrow \{-1, +1\}$
e.g., $f(q) = \text{sign}(\beta^T \phi(q))$

Memories:
$m_1, \ldots m_n$

Scorer Function:
$g(q, x) = w^T \phi(q, x)$
Datastructure of CMT: A Nearly Balanced Binary Tree

- Internal node
- Router (binary classifier): $f : Q \rightarrow \{-1, +1\}$
  - e.g., $f(q) = \text{sign}(\beta^T \phi(q))$
- Memories: $m_1, \ldots, m_n$
- Scorer Function: $g(q, x) = w^T \phi(q, x)$
- Select the most relevant memory

$$f_1(q) < 0$$
$$f_2(q) > 0$$

$n_L, n_R$
# of memories in the Left/Right subtree

$m^* = \arg \max_m g(q, m)$
Update Routers:

Using reward signals to update routers
Update Routers:

Using reward signals to update routers

\[ q \]

\[ f_1 \]

\[ f_2 \]

\[ f \]

\[
\begin{array}{c}
\text{Green nodes representing routers}
\end{array}
\]
Update Routers:

Using reward signals to update routers

\[ f_1(q) < 0 \]
Update Routers:

Using reward signals to update routers

\[ f_1(q) < 0 \]
Update Routers:

Using reward signals to update routers

\[ f_1(q) < 0 \]
Update Routers:

Using reward signals to update routers

$q$

$f_1(q) < 0$

$f$

$f_2$

$r_L$

$r_R$
Update Routers:

Using reward signals to update routers

Binary Classification

\( \{q, \text{sign}(r_R - r_L)\} \)
Update Routers:

Using reward signals to update routers

$$f_1(q) < 0$$

$$\{q, \text{sign}(r_R - r_L)\}$$

$$\epsilon$$-Greedy for exploration
Update Routers:

Using reward signals to update routers

\[ f_1(q) < 0 \]

Binary Classification
\[ \{q, \text{sign}(r_R - r_L)\} \]

\( r_L \quad r_R \)

\( \epsilon \)-Greedy for exploration
Update Routers:

Using reward signals to update routers

$\epsilon$-Greedy for exploration + Importance Weight (Propensity Score)
Update Routers:

Using reward signals to update routers

- Greedy for exploration + Importance Weight (Propensity Score)

\[ \frac{r_R}{p} \approx r_R - r_L \]
Update Routers:

Using reward signals to update routers

\[ f_1(q) < 0 \]

Binary Classification

\[ \{ q, \text{sign}(r_R - r_L) \} \]

\( \epsilon \)-Greedy for exploration + Importance Weight (Propensity Score)

\[ \frac{r_R}{p} \approx r_R - r_L \]
Update Routers:
Add Regularization for Balance

Binary Classification

\[
\{ q, \text{sign}(r_R - r_L) \} \]
Update Routers:
Add Regularization for Balance

- Internal node

\[ n_L, n_R \]

# of memories in the Left/Right subtree

Binary Classification
Update Routers: Add Regularization for **Balance**

Binary Classification
\[
\begin{cases}
q, \quad \text{sign}\left( (1 - \alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R} \right)
\end{cases}
\]

- $f_1(q) < 0$
- Internal node
- $n_L, n_R$
  - # of memories in the Left/Right subtree
- $r_L, r_R$
- $\alpha$
Update Routers: Add Regularization for Balance

Binary Classification

$$\begin{cases} q, \text{sign} \left( (1 - \alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R} \right) \end{cases}$$
Update Routers:
Add Regularization for **Balance**

\[ f_1(q) < 0 \]

Binary Classification

\[ q, \text{sign} \left( (1 - \alpha)(r_R - r_L) + \alpha \log \frac{n_L}{n_R} \right) \]
Insertion: Self-Consistency

\[ f_1 \]

\[ f_2 \]

\[ f \]

\[ m = \{ q, v \} \]
Insertion:
Self-Consistency

Same query: $q$

\[ m = \{ q, v \} \]
Insertion: Self-Consistency

Same query: $q$

$m = \{q, v\}$
Insertion:
Self-Consistency

Same query: $q$

$m = \{q, v\}$
Insertion: Self-Consistency

Same query: $q$

If a query has been seen before, we want to just retrieve it!
Insertion:
Encourage Self-Consistency

Internal node

\( n_L, n_R \)

# of memories in the Left/Right subtree
Insertion: 
Encourage Self-Consistency

A memory: \( m = \{q, v\} \)

- Internal node
- \( n_L, n_R \)
  \# of memories in the Left/Right subtree
Insertion:
Encourage **Self-Consistency**

A memory: \( m = \{q, v\} \)

- **Internal node**
- \( n_L, n_R \)
  - # of memories in the Left/Right subtree
**Insertion:**
Encourage **Self-Consistency**

A memory: $m = \{q, v\}$

- **Internal node**

- $n_L, n_R$
  - # of memories in the Left/Right subtree

$m = \{q, v\}$
Insertion:
Encourage Self-Consistency

A memory: \( m = \{q, v\} \)

Internal node

\( n_L, n_R \)

# of memories in the Left/Right subtree

\( m = \{q, v\} \)
Insertion: Encourage **Self-Consistency**

A memory: \( m = \{q, v\} \)

Internal node

\( n_L, n_R \)

# of memories in the Left/Right subtree

\( m = \{q, v\} \)

Enhance its prediction
Insertion: Encourage Self-Consistency

A memory: $m = \{q, v\}$

Binary Classification

$$\left\{ q, \text{sign}\left((1 - \alpha)f(g) + \alpha \log \left(\frac{n_L}{n_R}\right)\right) \right\}$$
Insertion: Encourage Self-Consistency

A memory: \( m = \{q, v\} \)

Enhance its prediction

Binary Classification

\[
\left\{ q, \text{sign} \left( (1 - \alpha)f(g) + \alpha \log \left( \frac{n_L}{n_R} \right) \right) \right\}
\]
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions.

1. Randomly sample a memory.
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions.

$1. \text{Randomly sample a memory}$

$f_1 \to f_2 \to f$

$m = \{q, v\}$
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions

1. Randomly sample a memory

2. Delete it & its trace
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions.

1. Randomly sample a memory
2. Delete it & its trace
3. Re-insert it

\[ m = \{ q, v \} \]
Amortized Experience Replay

Routers are updated online and may result in a lack of self-consistency for previous insertions

1. Randomly sample a memory
2. Delete it & its trace
3. Re-insert it

Apply replay constant times per insertion
Empirical Results
Empirical Results

Extreme Multi-Class Classification:

(feature, label), zero-one loss
Empirical Results

Extreme Multi-Class Classification:
(feature, label), zero-one loss

Extreme Multi-Label Classification:
(feature, set of labels), Hamming loss
Empirical Results

Extreme Multi-Class Classification:
(feature, label), zero-one loss

Extreme Multi-Label Classification:
(feature, set of labels), Hamming loss

Image Retrieval
(Caption, Image), Cosine Similarity
Extreme Multi-class Classification:
Online Progressive Performance
Extreme Multi-class Classification: Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)
Extreme Multi-class Classification: Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)
Extreme: very few examples per class!
Extreme Multi-class Classification: Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!
Extreme Multi-class Classification: Online Progressive Performance

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extremely few examples per class!

ALOI

WikiPara-3 shot

# of examples

ALOI (1K classes, 100K examples)
WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!
Extreme Multi-class Classification: Online Progressive Performance

ALOI (1K classes, 100K examples) WikiPara-3 shot (10K classes, 30K examples)

Extreme: very few examples per class!
Extreme Multi-Label Classification:
Helping an External Inference Algorithm (One-Against-Some)
Extreme Multi-Label Classification: Helping an \textbf{External} Inference Algorithm (One-Against-Some)

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)
Extreme Multi-Label Classification: Helping an External Inference Algorithm (One-Against-Some)

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence linear infer = slow)
Extreme Multi-Label Classification: Helping an **External** Inference Algorithm (One-Against-Some)

A new query $q = \{q_i, v_i\}_{i=1}^N$

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence linear infer = slow)
Extreme Multi-Label Classification: Helping an External Inference Algorithm (One-Against-Some)

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence linear infer = slow)
Extreme Multi-Label Classification: Helping an **External** Inference Algorithm (One-Against-Some)

A new query \( q \)  \[ \{ q_i, v_i \}_{i=1}^{N} \] is directed to CMT which returns \( k \) labels \( \{ q_i, v_i \}_{i=1}^{k} \). These labels are then used by OAS (OAA on the subset of labels from returned memories).

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear** infer = **slow**)
Extreme Multi-Label Classification: Helping an External Inference Algorithm (One-Against-Some)

A new query $q$ is input to the CMT algorithm, which outputs a set of $(q_i, v_i)_{i=1}^N$. This set is then input to the OAS algorithm, which returns a subset of labels from the memories.

Here is a table comparing the performance of CMT and OAA on three datasets:

<table>
<thead>
<tr>
<th>Approach</th>
<th>RCV1-1K</th>
<th>AmazonCat-13K</th>
<th>Wiki10-31K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>loss</td>
<td>Test time</td>
<td>Train time</td>
</tr>
<tr>
<td>CMT</td>
<td>2.5</td>
<td>1.4ms</td>
<td>1.9hr</td>
</tr>
<tr>
<td>OAA</td>
<td>2.6</td>
<td>0.5ms</td>
<td>1.3hr</td>
</tr>
</tbody>
</table>

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-31K (30K labels)

Extreme: very large number of labels (hence linear infer = slow)
Extreme Multi-Label Classification: Helping an **External** Inference Algorithm (One-Against-Some)

A new query $q$  

$\{q_i, v_i\}_{i=1}^N$  

$\{q_i, v_i\}_{i=1}^k$  

**OAS**  

(OAA on the **subset of labels** from returned memories)

**Computation: faster**

<table>
<thead>
<tr>
<th>Approach</th>
<th>RCV1-1K</th>
<th>AmazonCat-13K</th>
<th>Wiki10-30K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>loss</td>
<td>Test time</td>
<td>Train time</td>
</tr>
<tr>
<td>CMT</td>
<td>2.5</td>
<td>1.4ms</td>
<td>1.9hr</td>
</tr>
<tr>
<td>OAA</td>
<td>2.6</td>
<td>0.5ms</td>
<td>1.3hr</td>
</tr>
</tbody>
</table>

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence **linear infer = slow**).
Extreme Multi-Label Classification: Helping an External Inference Algorithm (One-Against-Some)

A new query $q$ is processed by CMT, which returns a set of labels $\{q_i, v_i\}_{i=1}^{N}$. These labels are then used by OAS, which operates on a subset of labels from the returned memories.

<table>
<thead>
<tr>
<th>Approach</th>
<th>RCV1-1K</th>
<th>AmazonCat-13K</th>
<th>Wiki10-30K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>loss</td>
<td>Test time</td>
<td>Train time</td>
</tr>
<tr>
<td>CMT</td>
<td>2.5</td>
<td>1.4ms</td>
<td>1.9hr</td>
</tr>
<tr>
<td></td>
<td>2.6</td>
<td>0.5ms</td>
<td>1.3hr</td>
</tr>
<tr>
<td>OAA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computation: faster

Statistical Performance: similar, sometimes even better

(1) RCV1 (1K labels), (2) AmazonCat 13K (13K labels), and (3) Wiki10-30K (30K labels)

Extreme: very large number of labels (hence linear infer = slow)
Image Retrieval: Comparison to Nearest Neighbor Approach

CMT(u):
Unsupervised version
i.e., NN (w/ Euclidean dis)
at leaf level

(1) Pascal (1K examples), (2) Flickr8k (8k examples), (2) MSCOCO (80K examples)

**Image feature**: HoG for Pascal & Flickr, Pre-trained VGG-19 for MSCOCO

**Captions feature**: Token Occurrences with hashing (high-dim, very sparse)
Image Retrieval: Comparison to Nearest Neighbor Approach

(1) Pascal (1K examples), (2) Flickr8k (8k examples), (2) MSCOCO (80K examples)

Image feature: HoG for Pascal & Flickr, Pre-trained VGG-19 for MSCOCO

Captions feature: Token Occurrences with hashing (high-dim, very sparse)

CMT(u): Unsupervised version i.e., NN (w/ Euclidean dis) at leaf level
How does computation scale wrt the size of memory tree?

On ALOI, we range in dataset size from 1K to 100K
How does computation scale wrt the size of memory tree?

On ALOI, we range in dataset size from 1K to 100K

The graph shows the increase in inference time (ms) as the size of the memory tree increases. The increase is logarithmic.
How does Self-Consistency improve?

1-shot dataset: 10K classes, 10K examples

Self-Consistency improves over time

Zero training err means perfect self-consistency
How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)
How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)
How does performance scale wrt classification difficulty?

ALOI S-shot, with S from 1 to 100
(s-shot means s examples per class)

CMT (orange) significantly outperforms
Recall Tree (Green) when s is small,
Revisit our Desired Properties
Revisit our Desired Properties

**Online:** Reduction to Online Classification
Revisit our Desired Properties

**Online**: Reduction to Online Classification

**Linear Space**: Store examples in leaves

\[ O\left(\frac{N}{\log(N)}\right) \text{many nodes} \]
Revisit our Desired Properties

**Online**: Reduction to Online Classification

**Linear Space**: Store examples in leaves

\[ O\left(\frac{N}{\log(N)}\right) \] many nodes

**Logarithmic time**: Regularization ensures a (provable) near-balanced tree
Revisit our Desired Properties

**Online**: Reduction to Online Classification

**Linear Space**: Store examples in leaves

\[ O(N/ \log(N)) \] many nodes

**Logarithmic time**: Regularization ensures a (provable) near-balanced tree

**Learning-based**: Reinforcement
Revisit our Desired Properties

**Online**: Reduction to Online Classification

**Linear Space**: Store examples in leaves

\[ O\left(\frac{N}{\log(N)}\right) \] many nodes

**Logarithmic time**: Regularization ensures a (provable) near-balanced tree

**Learning-based**: Reinforcement

**Self-consistency**: an asymptotic guarantee (due to replay)
Thanks!

CMT is in the latest Vowpal Wabbit (VW)
https://github.com/VowpalWabbit/vowpal_wabbit

Try out CMT demos!