MECTA: Memory-Economic Continual Test-Time Model Adaptation

Junyuan Hong¹*, Lingjuan Lyu², Jiayu Zhou¹, Michael Spranger²

¹Michigan State University, ²Sony AI

*Work done during internship at Sony AI
Continually Changing Environments

Of-the-shelf pre-trained model

Accuracy on CIFAR10

- On clean samples: 94%
- On samples with unseen noise: 57%

Continual Test-time Adaptation (CTA)

Of-the-shelf pre-trained model

Fine-tune

A batch of unlabeled test samples

Accuracy on CIFAR10

94%

87%

57%

On clean samples

On samples with unseen noise

Continual Test-time Adaptation (CTA)

Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

Car 90%

Continual Test-time Adaptation (CTA)

Entropy Minimization (Tent)

\[ \theta_t = \text{Optimize}_{\theta \in \Theta_t} \left( \mathbb{E}_{x \sim P_t(x)} [H(f_\theta(x))], \theta_{t-1} \right), \]

Car 90%

CTA:
- Unsupervised finetuning.

Continual Test-time Adaptation (CTA)

Entropy Minimization (Tent)
\[ \theta_t = \underset{\theta \in \Theta}{\text{Optimize}} \left( \mathbb{E}_{x \sim P_t(x)} [H(\theta(x))], \theta_{t-1} \right), \]

Baseline
Dense cache

Car 90%

CTA:
- Unsupervised finetuning.
- Parameter efficient: Update BN only.

Continual Test-time Adaptation (CTA)

Entropy Minimization (Tent)
\[ \theta_t = \text{Optimize}_{\theta \in \Theta_t} (\mathbb{E}_{x \sim P_t(x)}[H(f_\theta(x))], \theta_{t-1}), \]  
Baseline

Dense cache

Car 90%

Batch Norm (BN)
\[ z_{n,i,j,k}^l = \frac{x_{n,i,j,k}^l - \mu_i^l}{\sqrt{\sigma_i^2 + \epsilon_0}} \]
\[ a_{n,i,j,k}^l = \gamma_i z_{n,i,j,k}^l + b_i^l \]

CTA:
- Unsupervised finetuning.
- Parameter efficient: Update BN only.
- Batch-estimated BN statistics \((\mu, \sigma)\) for capturing new environment.


Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches
Continual Test-time Adaptation (CTA)

Entropy Minimization (Tent)
\[ \theta_t = \text{Optimize}_{\theta \in \Theta_t} \left( \mathbb{E}_{x \sim P_t(x)} \left[ H(f_\theta(x)) \right], \theta_{t-1} \right), \] Baseline

Dense cache

Car 90%

Batch Norm (BN)
\[ z^l_{n,i,j,k} = \frac{x^l_{n,i,j,k} - \mu^l_i}{\sqrt{\sigma^2_i + \epsilon_0}} \quad a^l_{n,i,j,k} = \gamma^l_i z^l_{n,i,j,k} + b^l_i \]

CTA:
- Unsupervised finetuning.
- Parameter efficient: Update BN only.
- Batch-estimated BN statistics \((\mu, \sigma)\) for capturing new environment.
- (EATA by Niu et al., 2022)
  Computation efficient.

High memory load for CTA on edge

Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

model.init  156 Mb
Cache for inference  +786 Mb

model.forward

Forward  
Backward

Batch Norm

Conv

Batch Norm

Conv

Raspberry Pi: 1-4Gb RAM
* Enough for inference
High memory load for CTA on edge

Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

Raspberry Pi: 1-4Gb RAM
* Enough for inference but not for adaptation
High cache memory for back-propagation

Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

model.init 156 Mb
optimizer.init +98 Mb Cache for inference +786 Mb Cache for backward +4.5 Gb
model.forward Cache for backward +1186 Mb
loss.backward +0 Mb
optimizer.step

Basel 
Dense cache

Batch-norm

\[ z_{n,i,j,k}^l = \frac{x_{n,i,j,k}^l - \mu_i^l}{\sqrt{\sigma_i^2 + \epsilon_0}} \]
\[ a_{n,i,j,k}^l = \gamma_i^l z_{n,i,j,k}^l + b_i^l \]

\[ \sum_{n=1}^{B} \frac{\partial \ell_n}{\partial z_{n,i,j,k}^l} = \sum_{n=1}^{B} \sum_{j=1}^{W} \sum_{k=1}^{H} \frac{\partial \ell_n}{\partial a_{n,i,j,k}^l} \]

Batch Norm

Conv

Cache
Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

Baseline

Dense cache

Forward

Backward

Cache size

High cache memory for back-propagation

Batch-norm

$z_{n,i,j,k}^l = \frac{x_{n,i,j,k}^l - \mu_i^l}{\sqrt{\sigma_i^2 + \epsilon_0}}$

$\alpha_{n,i,j,k}^l = \gamma_i^l z_{n,i,j,k}^l + b_i^l$

$\sum_{n=1}^{B} \frac{\partial \ell_n}{\partial \gamma_i^l} = \sum_{n=1}^{B} \sum_{j=1}^{W} \sum_{k=1}^{H} \frac{\partial \ell_n}{\partial \alpha_{n,i,j,k}^l} z_{n,i,j,k}^l$.

$R_{\text{fwd}} = \max_{l \in \{1, \ldots, L\}} B \times C^l \times W^l \times H^l$.

$R_{\text{bwd}} = \sum_{l=1}^{L} B \times C^l \times W^l \times H^l \geq R_{\text{fwd}}$

Traditional CTA cannot fit into low-memory devices.

- **(B)** Require large batch size for statistic estimation.
- **(L&C)** The cache tensor $z$ scales by number of layers and channels.
Memory-Efficient Adaptation by MECTA

\[ z_{n,i,j,k}^{l} = \frac{x_{n,i,j,k}^{l} - \mu_{\hat{z}}^{l}}{\sqrt{\sigma_{\hat{z}}^{2} + \epsilon_0}} \]

\[ R_{\text{bwd}} = \sum_{l=1}^{L} B \times C^{l} \times W^{l} \times H^{l} \geq R_{\text{fwd}} \]
Memory-Efficient Adaptation by MECTA

• (B) Reduce batch size and maintain accurate statistic estimation.

\[ \mu = \frac{1}{B} \frac{1}{W} \frac{1}{H} \sum_{n,j,k} x_{n,i,j,k} \]

Accurate statistics \( \phi = [\mu, \sigma] \) require more samples in a batch (larger B).

\[ z_{n,i,j,k}^l = \frac{x_{n,i,j,k}^l - \mu_i^l}{\sqrt{\sigma_i^l + \epsilon_0}} \]

\[ R_{\text{bwd}} = \sum_{l=1}^{L} B \times C^l \times W^l \times H^l \geq R_{\text{fwd}} \]
Memory-Efficient Adaptation by MECTA

• (B) Reduce batch size and maintain accurate statistic estimation.
• (C) Reduce channels to update.
• (L) Reduce layers to update.

\[
z_{n,i,j,k}^l = \frac{x_{n,i,j,k}^l - \mu_k^l}{\sqrt{\sigma_k^2 + \epsilon_0}}
\]

\[
R_{\text{bwd}} = \sum_{l=1}^{L} B \times C_l \times W_l \times H_l \geq R_{\text{fwd}}
\]

Drop cache ⇔ Drop gradient
Maintain effective adaptation requires enough gradient information.
Memory-Efficient Adaptation by MECTA

- **(B)** Reduce batch size and maintain accurate statistic estimation per BN layer.

Exponential Moving Average (EMA)

\[ \phi \text{ represents } [\mu, \sigma] \]

\[ \hat{\phi}_t \]

Statistic of current batch

\[ \mu = \frac{1}{B W H} \sum_{n,j,k} x_{n,i,j,k} \]

Inaccurate if \( B \downarrow \)
Memory-Efficient Adaptation by MECTA

• **(B)** Reduce batch size and maintain accurate statistic estimation per BN layer.
  • Moving average accommodate small batch sizes towards robust and accurate statistic estimations.

Exponential Moving Average (EMA)

\[ \phi_t = (1 - \beta)\phi_{t-1} + \beta\hat{\phi}_t, \]

\( \phi \) represents \([\mu, \sigma]\)

Previous memory

Statistic of current batch

\[ \mu = \frac{1}{B} \frac{1}{W} \frac{1}{H} \sum_{n,j,k} x_{n,i,j,k} \]
Memory-Efficient Adaptation by MECTA

- (B) Reduce batch size and **maintain** accurate statistic estimation.
  - Moving average accommodate small batch sizes towards robust and accurate statistic estimations.

Exponential Moving Average (EMA)

\[ \phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t, \]

- \( \phi \) represents \([\mu, \sigma]\)
- Previous memory
- Statistic of current batch

\( \beta = 1 \): Only using current **small** batch result in **inaccurate** estimation.

How to set \( \beta \) in changing env?
(B) Reduce batch size and maintain accurate statistic estimation.

- Moving average accommodate small batch sizes towards robust and accurate statistic estimations.

\[
\phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t,
\]

\(\beta = 1\): Only using current small batch result in inaccurate estimation.

\(\beta = 0\): Only using training/past stat (without updates) result in non-adaptive/non-robust estimation.
Memory-Efficient Adaptation by MECTA

• (B) Reduce batch size and maintain accurate statistic estimation.
  • Moving average accommodate small batch sizes towards robust and accurate statistic estimations.

Exponential Moving Average (EMA)

\[ \phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t, \]

\( \phi \text{ represents } [\mu, \sigma] \)

\( \beta = 1 : \text{Only using current small batch result in inaccurate estimation.} \)

\( \beta = 0 : \text{Only using training batch (without updates) result in non-adaptive estimation.} \)

Trade-off between accuracy and adaptivity

\( 0 < \beta < 1 \)
Memory-Efficient Adaptation by MECTA

• (B) Reduce batch size and maintain accurate statistic estimation.
  • Moving average accommodate small batch sizes towards robust and accurate statistic estimations.
  • Adaptive memory by time-varying \( \beta \).

Exponential Moving Average (EMA)

\[
\phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t,
\]

\( \phi \) represents \([\mu, \sigma]\)

\( \beta = 1 \): Only using current **small** batch result in **inaccurate** estimation.

\( \beta = 0 \): Only using **training** batch (without updates) result in **non-adaptive** estimation.

\( P_{t-1} = P_t \): Stable \( \Rightarrow \beta \to 0 \) (accurate)

\( P_{t-1} \neq P_t \): On change. \( \Rightarrow \beta \to 1 \) (fast adapt)

\( P_{t-1} \neq P_t \): On change. \( \Rightarrow \beta \to 1 \) (fast adapt)
Memory-Efficient Adaptation by MECTA

- (B) Reduce batch size and maintain accurate statistic estimation.
  - Moving average accommodate small batch sizes towards robust and accurate statistic estimations.
  - Adaptive memory by time-varying $\beta$.

$$\beta_t = 1 - e^{-D(\phi_{t-1}, \hat{\phi}_t)}$$

$$D(\phi_{t-1}, \hat{\phi}_t) = \frac{1}{C} \sum_{i=1}^{C} KL(\phi_{t-1,i} \| \hat{\phi}_{t,i}) + KL(\hat{\phi}_{t,i} \| \phi_{t-1,i})$$

Exponential Moving Average (EMA)

$\phi$ represents $[\mu, \sigma]$,

$$\phi_t = (1 - \beta)\phi_{t-1} + \beta\hat{\phi}_t,$$

Previous memory Statistic of current batch

$\beta = 1$ : Only using current small batch result in inaccurate estimation.

$\beta = 0$ : Only using training batch (without updates) result in non-adaptive estimation.

$\phi_{t-1} = \hat{\phi}_t$ : Stable $\Rightarrow$ $\beta \rightarrow 0$ (accurate)

$\phi_{t-1} \neq \hat{\phi}_t$ : On change. $\Rightarrow$ $\beta \rightarrow 1$ (fast adapt)
Memory-Efficient Adaptation by MECTA

• (B) Reduce batch size and maintain accurate statistic estimation.
• (C) Reduce channels to update.
• (L) Reduce layers to update.
Memory-Efficient Adaptation by MECTA

- Reduce channels to update.
- Drop $q \times 100\%$ channels in cache.

Drop cache $\Leftrightarrow$ Drop gradient
Maintain effective adaptation requires enough gradient information.
Memory-Efficient Adaptation by MECTA

• (C) Reduce channels to update.
  • Stochastically drop $q \times 100\%$ channels in cache.

Stoc. drop cache $\Leftrightarrow$ **Not** drop gradient. Maintain effective adaptation requires **enough gradient** information.

$$E[M_{i}z_{n,i,k}] = qE[z_{n,i,j,k}]$$

Expectation over continual batches can **restore** the dropped channels.
Memory-Efficient Adaptation by MECTA

• (C) Reduce channels to update.
  • Stochastically drop $q \times 100\%$ channels in cache.
  • Implicit gradient regularization which mitigates forgetting.

\[
\begin{align*}
B \times C^l \times W^l \times H^l & \\
\tilde{z}_{n, i, j, k} & = M_i z_{n, i, j, k} \\
B \times qC^l \times W^l \times H^l & \\
\|\theta_t - \tilde{\theta}_0\| & = \|\sum_t g_t\| \leq O(\sum_t \|g_t\|)
\end{align*}
\]

Smaller $\Rightarrow$ less forgetting

Memory-Efficient Adaptation by MECTA

• (Reduce B) Adaptive and online statistic estimation on dynamic distributions for accurate statistics on small batch sizes.

• (Reduce C) Channel-sparse gradients via stochastically-pruned caches.

• (Dynamic L) Cache and train layers on demand.
Memory-Efficient Adaptation by MECTA

• (Dynamic L) Cache and train layers on demand.
  • If environment is **stable**, there is no need to continually adapt.
  • Stop gradient to save caches.

![Diagram showing baseline and MECTA cache mechanisms](image)
Memory-Efficient Adaptation by MECTA

- **(Dynamic L)** Cache and train layers on demand.
  - If environment is stable, there is no need to continually adapt.
  - Stop gradient to save caches.

\[
\phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t,
\]

*Previous memory*  *Statistic of current batch*

- When to stop? Environment is stable.
- When to restart? Environment changes.
Memory-Efficient Adaptation by MECTA

• (Dynamic L) Cache and train layers on demand.
  • If environment is stable, there is no need to continually adapt.
  • Stop gradient to save caches.

\[
\phi_t = (1 - \beta)\phi_{t-1} + \beta\hat{\phi}_t,
\]

• When to stop? Environment is stable.
• When to restart? Environment changes.

Previous memory $P_{t-1} = P_t$ : Stable.
$P_{t-1} \neq P_t$ : On change.
Memory-Efficient Adaptation by MECTA

• (Dynamic L) Cache and train layers on demand.
  • If environment is **stable**, there is no need to continually adapt.
  • Stop gradient to save caches.

• When to stop? Environment is stable.
• When to restart? Environment changes.

\[ \phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t, \]

Previous memory
Statistic of current batch

\[ P_{t-1} = P_t : \text{Stable} \quad \beta \rightarrow 0 \]
\[ P_{t-1} \neq P_t : \text{On change.} \quad \beta \rightarrow 1 \]
Memory-Efficient Adaptation by MECTA

• (Dynamic L) Cache and train layers on demand.
  • If environment is stable, there is no need to continually adapt.
  • Stop gradient to save caches.

\[ \phi_t = (1 - \beta)\phi_{t-1} + \beta \hat{\phi}_t, \]

Previous memory
Statistic of current batch

\[ P_{t-1} = P_t : \text{Stable} \quad \iff \beta \to 0 \]
\[ P_{t-1} \neq P_t : \text{On change.} \quad \iff \beta \to 1 \]

\[ \beta_t = 1 - e^{-D(\phi_{t-1}, \hat{\phi}_t)} \]

• When to stop? Environment is stable: \( \beta_t < \beta_{th} \)
• When to restart? Environment changes: \( \beta_t > \beta_{th} \)
Memory-Efficient Adaptation by MECTA

- **(Reduce B)** Adaptive and online statistic estimation on dynamic distributions for accurate statistics on small batch sizes.
- **(Reduce C)** Channel-sparse gradients via stochastically-pruned caches.
- **(Dynamic L)** Cache and train layers on demand.
Continual adaptation via Tent/EATA of ResNet50 w/ 64-sized batches

Model.init
- 156 Mb
- optimizer.init
- +98 Mb
  Cache for inference
  +786 Mb
  Cache for backward
  +1186 Mb
- optimizer.step
- +0 Mb

model.forward
- +4.5 Gb

Memory-econmic continual adaptation via MECTA of ResNet50 w/ 16-sized batches

Memory-economic continual adaptation via MECTA of ResNet50 w/ 16-sized batches

MECTA Norm
- Batch Norm
- Conv
- Forward
- Backward

Ours
- Sparse cache

Baseline
- Dense cache

MECTA Norm
- Batch Norm
- Conv

MECTA greatly reduce running memory.
### Benchmark: Accuracy & Memory Efficiency

#### Changing test-time noise

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Noise</th>
<th>Blur</th>
<th>Weather</th>
<th>Digital</th>
<th>Acc.</th>
<th>Cache</th>
<th>#fwd</th>
<th>#bwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTT (GN+JT)</td>
<td>31.0</td>
<td>33.6</td>
<td>33.4</td>
<td>28.1</td>
<td>7.8</td>
<td>33.2</td>
<td>36.8</td>
<td>40.9</td>
</tr>
<tr>
<td>BN</td>
<td>15.5</td>
<td>16.1</td>
<td>16.3</td>
<td>20.0</td>
<td>20.0</td>
<td>28.5</td>
<td>40.0</td>
<td>34.8</td>
</tr>
<tr>
<td>TTA</td>
<td>4.1</td>
<td>4.9</td>
<td>4.5</td>
<td>12.5</td>
<td>8.2</td>
<td>12.9</td>
<td>25.8</td>
<td>14.0</td>
</tr>
<tr>
<td>MEMO</td>
<td>7.5</td>
<td>8.7</td>
<td>9.0</td>
<td>19.7</td>
<td>13.0</td>
<td>20.7</td>
<td>27.6</td>
<td>25.3</td>
</tr>
</tbody>
</table>

**ResNet50: Reset model per perturbation**

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Noise</th>
<th>Blur</th>
<th>Weather</th>
<th>Digital</th>
<th>Acc.</th>
<th>Cache</th>
<th>#fwd</th>
<th>#bwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoTTA(+GC)</td>
<td>16.9</td>
<td>20.3</td>
<td>22.8</td>
<td>20.6</td>
<td>22.0</td>
<td>31.7</td>
<td>42.4</td>
<td>34.5</td>
</tr>
<tr>
<td>Tent(+GC)</td>
<td>28.4</td>
<td>34.1</td>
<td>31.7</td>
<td>19.3</td>
<td>12.2</td>
<td>6.9</td>
<td>4.0</td>
<td>1.4</td>
</tr>
<tr>
<td>Tent+MECTA</td>
<td>24.5</td>
<td>29.5</td>
<td>28.3</td>
<td>22.0</td>
<td>23.5</td>
<td>27.4</td>
<td>37.2</td>
<td>28.2</td>
</tr>
<tr>
<td>EATA(+GC)</td>
<td>35.0</td>
<td>38.1</td>
<td>36.8</td>
<td>33.8</td>
<td>34.2</td>
<td>47.3</td>
<td>53.2</td>
<td>51.1</td>
</tr>
<tr>
<td>EATA+MECTA</td>
<td>33.7</td>
<td>30.1</td>
<td>37.8</td>
<td>31.7</td>
<td>33.1</td>
<td>42.2</td>
<td>50.3</td>
<td>46.3</td>
</tr>
</tbody>
</table>

**Best trade-off between memory and accuracy.**
MECTA dynamically cache data on demand

Cache on demand of environment change
Benchmark with Constrained Cache

MECTA norm avoid forgetting of TENT.

Better accuracy on all noise, reasonable computation load.
Which component matters more?

Trade-off between robust accuracy and memory (cache size)

B, C are more effective while L is useful in low memory region.
MECTA: Memory-Economic Continual Test-Time Model Adaptation

- **New Problem**: We initiate the study on the memory efficiency of continual test-time adaptation (CTA), revealing the substantial obstacle in practice.

- **New Method**: We propose a novel method with a simple plug-in MECTA Norm layer that improves the memory efficiency of different CTA methods.

- **Better Memory-Robustness Trade-off**: Our method maintains comparable performance to full back-propagation methods while significantly reducing the dynamic and maximal cache overheads.

Improve **on-device** machine learning memory efficiency on **changing** environments.
Thank you!

paper

code