Adapter Parameter Generation for Multi-task & Continual Learning based on LMs

Haoran Yang

hryang@se.cuhk.edu.hk

Aug 24, 2021

Papers

Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks

Rabeeh Karimi Mahabadi*
EPFL University, Idiap Research Institute
rabeeh.karimimahabadi@epfl.ch

Sebastian Ruder
DeepMind
ruder@google.com

Mostafa Dehghani
Google Brain
dehghani@google.com

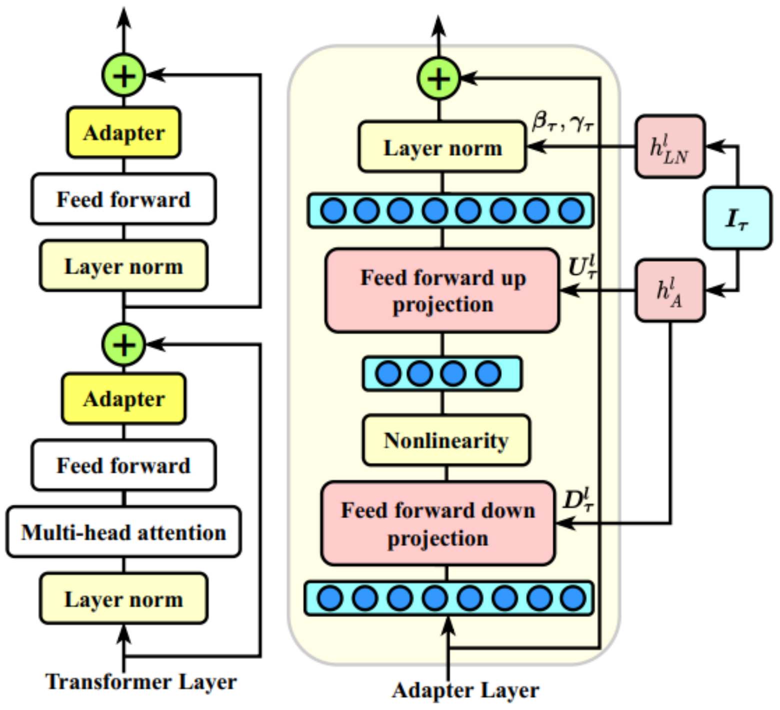
James Henderson
Idiap Research Institute
james.henderson@idiap.ch

Lifelong Learning of Few-shot Learners across NLP Tasks

Xisen Jin[§] **Mohammad Rostami**[†] **Xiang Ren**[§]
§University of Southern California, †Information Sciences Institute
{xisenjin, xiangren}@usc.edu {mrostami}@isi.edu

Background

- Adapter is a module that can be inserted into each layer of LMs.
    - Avoid fine-tuning the entire model (only fine-tuning the adapter while keeping other parameters fixed)
    - 3.6% additional parameters



Each transformer layer has two additional adapter layers

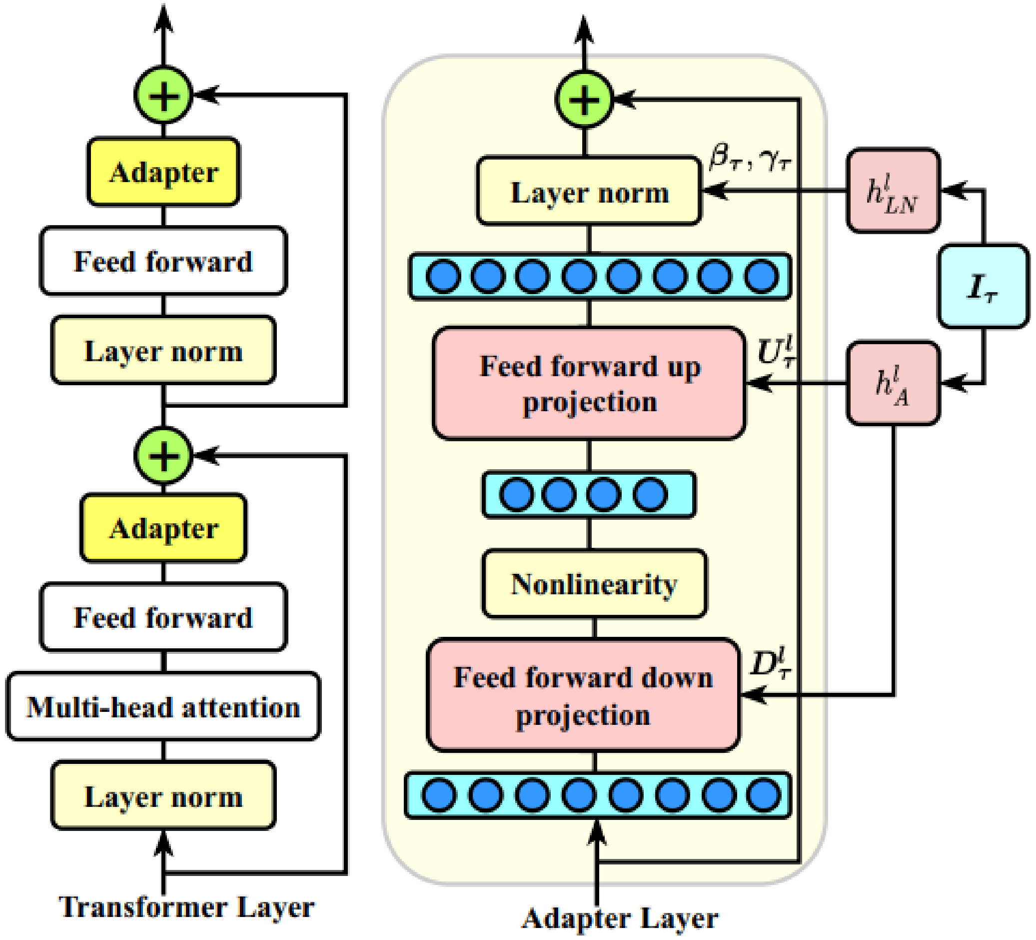
Multi-task: Normal FT VS Adapter-based FT

- Normal FT the model across multiple tasks allows sharing information between the different tasks and positive transfer to other related tasks.
- result in models underperforming on high-resource tasks due to constrained capacity
- task interference or negative transfer, where achieving good performance on one task can hinder performance on another

- Adapter-based FT can overcome the above two shortcomings
- Training each task with a separate adapter does not enable sharing information across tasks.

- Solution: design a hypernetwork and it learns to generate task and layer-specific adapter parameters, conditioned on task and layer id embeddings.

Multi-task: HYPERFORMER



τ: task id, l: layer number, h:hypernetwok

$$A_\tau^l(\boldsymbol{x}) = LN_\tau^l\left(\boldsymbol{U}_\tau^l(\text{GeLU}(\boldsymbol{D}_\tau^l(\boldsymbol{x})))\right) + \boldsymbol{x},$$

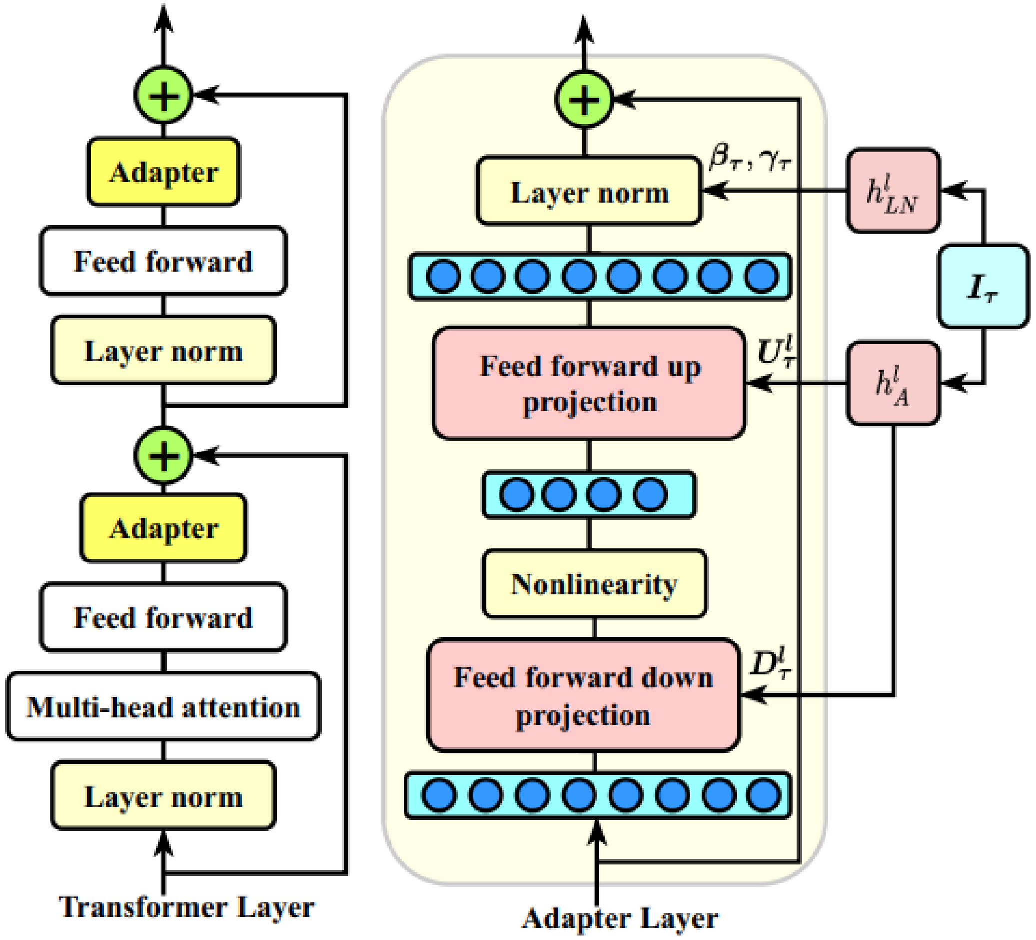
$$LN_\tau^l(\boldsymbol{x}_\tau^i) = \boldsymbol{\gamma}_\tau^l \odot \frac{\boldsymbol{x}_\tau^i - \boldsymbol{\mu}_\tau}{\boldsymbol{\sigma}_\tau} + \boldsymbol{\beta}_\tau^l,$$

$$\boldsymbol{I}_\tau = h_I(\boldsymbol{z}_\tau),$$

$$(\boldsymbol{U}_\tau^l, \boldsymbol{D}_\tau^l) := h_A^l(\boldsymbol{I}_\tau) = \left(\boldsymbol{W}^{U^l}, \boldsymbol{W}^{D^l}\right)\boldsymbol{I}_\tau$$

$$(\boldsymbol{\gamma}_\tau^l, \boldsymbol{\beta}_\tau^l) := h_{LN}^l(\boldsymbol{I}_\tau) = \left(\boldsymbol{W}^{\gamma^l}, \boldsymbol{W}^{\beta^l}\right)\boldsymbol{I}_\tau,$$

Multi-task: HYPERFORMER++



τ: task id, l: layer number

A downside of introducing a separate hypernetwork in each layer of the Transformer is that it increases the overall number of parameters.
So how to share the hypernetworks in different layers and different adapters?

$$\mathcal{I} = \{\boldsymbol{l_i}\}_{i=1}^{L}$$

$$\mathcal{P} = \{\boldsymbol{p_j}\}_{j=1}^{2}$$

$$\boldsymbol{I_\tau} = h'_I(\boldsymbol{z_\tau}, \boldsymbol{l_i}, \boldsymbol{p_j})$$

Results: main

Model	#Total params	#Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
					Single-Task Training						
$T5_{SMALL}$	8.0×	100%	**46.81**	**90.47**	**86.21/90.67**	**91.02/87.96**	**89.11/88.70**	**82.09**	**90.21**	**59.42**	**82.06**
$Adapters_{SMALL}$ ❣	1+8×0.01	**0.74%**	40.12	89.44	85.22/89.29	90.04/86.68	83.93/83.62	81.58	89.11	55.80	79.53
$T5_{BASE}$	8.0×	100%	54.85	92.19	**88.18/91.61**	**91.46/88.61**	**89.55/89.41**	**86.49**	91.60	67.39	84.67
$Adapters_{BASE}$ ❣	1+8×0.01	**0.87%**	**59.49**	**93.46**	88.18/91.55	90.94/88.01	87.44/87.18	86.38	**92.26**	**68.84**	**84.88**
					Multi-Task Training						
$T5_{SMALL}$ ♠	1.0×	12.5%	50.67	**91.39**	84.73/88.89	**89.53/86.31**	**88.70/88.27**	81.04	89.67	59.42	81.69
Adapters†$_{SMALL}$	1.05×	0.68%	39.87	90.01	88.67/91.81	88.51/84.77	88.15/87.89	79.95	89.60	60.14	80.85
$HYPERFORMER_{SMALL}$	1.45×	5.80%	47.64	**91.39**	**90.15/92.96**	88.68/85.08	87.49/86.96	**81.24**	**90.39**	65.22	82.47
$HYPERFORMER{++}_{SMALL}$	1.04×	**0.50%**	**53.96**	90.59	84.24/88.81	88.44/84.46	87.73/87.26	80.69	**90.39**	**71.01**	**82.51**
$T5_{BASE}$ ♠	1.0×	12.5%	54.88	92.54	90.15/93.01	**91.13/88.07**	88.84/88.53	85.66	92.04	75.36	85.47
Adapters†$_{BASE}$	1.07×	0.82%	61.53	93.00	90.15/92.91	90.47/87.26	89.86/89.44	86.09	**93.17**	70.29	85.83
$HYPERFORMER_{BASE}$	1.54×	6.86%	61.32	93.80	**90.64/93.33**	90.13/87.18	89.55/89.03	**86.33**	92.79	**78.26**	**86.58**
$HYPERFORMER{++}_{BASE}$	1.02×	**0.29%**	**63.73**	**94.03**	89.66/92.63	90.28/87.20	**90.00/89.66**	85.74	93.02	75.36	86.48

- HYPERFORMER++ obtains similar performance as HYPERFORMER, while being more than an order of magnitude parameter efficient.
- Compared to single-task fine-tuning, proposed model improves the performance.
    - substantial improvement on low-resource dataset CoLA and RTE

Results: few shot domain transfer

- They assess how well a trained HYPERFORMER can generalize to new tasks.
- New task list: NLI, QA, sentiment analysis datasets IMDB and Yelp Polarity, PAWS
- They use the adapter and the task embedding respectively trained on the most similar GLUE task for initialization, i.e. MNLI for NLI, QNLI for QA, SST-2 for sentiment analysis, and QQP for PAWS. Few-shot fine-tuning the model on each target training data.

Dataset	# Samples	$T5_{BASE}$	$Adapters^{\dagger}_{BASE}$	$HYPERFORMER{+}{+}_{BASE}$
Natural Language Inference				
SciTail	4	$79.60_{\pm 3.3}$	$79.54_{\pm 2.8}$	$\mathbf{82.00}_{\pm 4.9}$
	16	$80.03_{\pm 2.3}$	$83.25_{\pm 1.7}$	$\mathbf{86.55}_{\pm 1.4}$
	32	$81.97_{\pm 1.3}$	$85.06_{\pm 1.1}$	$\mathbf{85.85}_{\pm 1.4}$
	100	$84.04_{\pm 0.7}$	$88.22_{\pm 1.3}$	$\mathbf{88.52}_{\pm 0.7}$
	500	$88.07_{\pm 0.7}$	$91.27_{\pm 0.8}$	$\mathbf{91.44}_{\pm 0.6}$
	1000	$88.77_{\pm 1.0}$	$91.75_{\pm 0.8}$	$\mathbf{92.34}_{\pm 0.5}$
	2000	$91.01_{\pm 1.0}$	$92.72_{\pm 0.5}$	$\mathbf{93.40}_{\pm 0.2}$
CB	4	$57.78_{\pm 10.9}$	$51.11_{\pm 9.2}$	$\mathbf{60.74}_{\pm 16.66}$
	16	$\mathbf{77.04}_{\pm 7.2}$	$74.81_{\pm 5.4}$	$76.29_{\pm 4.45}$
	32	$80.0_{\pm 7.6}$	$74.81_{\pm 5.9}$	$\mathbf{81.48}_{\pm 6.2}$
	100	$85.93_{\pm 5.4}$	$80.74_{\pm 7.6}$	$\mathbf{87.41}_{\pm 2.96}$
	250	$85.19_{\pm 4.7}$	$86.67_{\pm 5.0}$	$\mathbf{89.63}_{\pm 4.32}$

Continual Learning

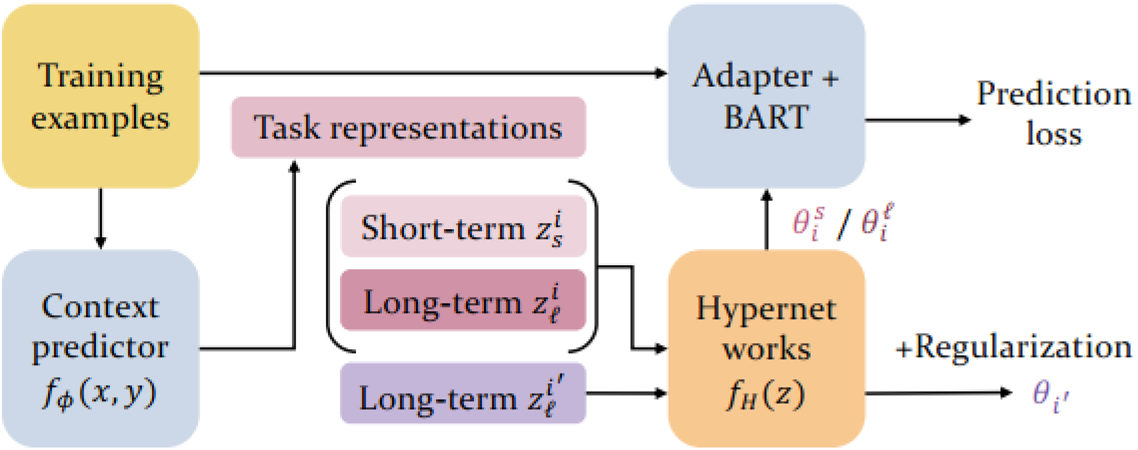
- Enable a single base model to expand its knowledge continually to learn the new future tasks while still remembering previous tasks
- Notations:

$\{\mathcal{T}_{tr}^i\}_{i=1}^T$	Task sequence
$\mathcal{D}^i = \{(\boldsymbol{x}_j^i, \boldsymbol{y}_j^i)\}_{j=1}^{N_{tr}^i}$	Training set
$\mathcal{D}_{te}^i = \{\boldsymbol{x}_j^i\}_{j=1}^{N_{te}^i}$	Testing set
$\{\mathcal{T}_f^k\}_{k=1}^K$	Few-shot task sequence
$\mathcal{D}^k$ and $\mathcal{D}_{te}^k$	Few-shot training and testing set

Aim to address the following questions:

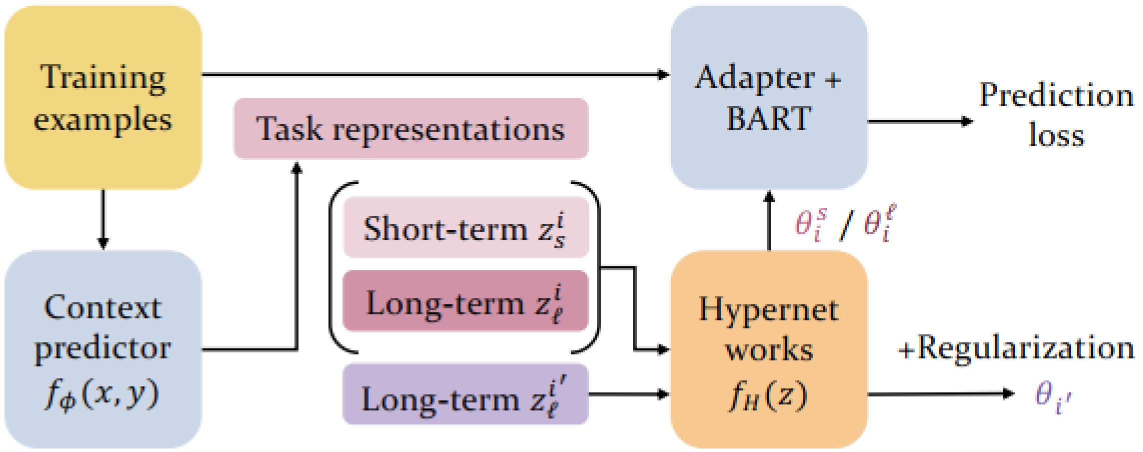
- can continually trained models retain performance effectively on seen tasks
- can models transfer knowledge to learn new tasks better
- can the resulting continually trained models learn new tasks with only a few training examples

Models: Long-Short Term Hypernetwork with Regularization (HNET+Reg)



1. **Context Prediction:** output a task representation vector
   - Long-term representation $z_s^i = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{x}_i \in \mathcal{D}_i} f_\phi(\boldsymbol{x}_i, \boldsymbol{y}_i)$
   - Short-term representation $z_l^i = \frac{1}{N} \sum_{i=i'}^{i'-N+1} f_\phi(\boldsymbol{x}_i, \boldsymbol{y}_i)$
   - f_ϕ is a frozen BART encoder
2. **Adapter Generation with Hypernetworks** f_H:
   - It generates parameters of adapters for each layers. $f_H(z)$

Models: Long-Short Term Hypernetwork with Regularization (HNET+Reg)



More specifically, before learning a new task $\mathcal{T}_i$, the hypernetwork generates adapter weights for each seen task $\mathcal{T}_{1..i-1}$, noted as $\boldsymbol{\theta}^i_{1..i-1}$. While updating the hypernetwork on the new task i, the model generate adapters for a random seen task from $1..i-1$ with stored low-dimensional long-term task representations, noted as $\boldsymbol{\theta}_{i'}$. It then penalizes the ℓ_2 distance between the adapter weights generated at the current time step and those generated before learning the new task, *i.e.*, $||\boldsymbol{\theta}_{i'} - \boldsymbol{\theta}^i_{i'}||^2_2$.

3. Mitigating Catastrophic Forgetting

- when the hypernetwork tries to learn current task adapter parameters, it may forget how to generate previous tasks adapter parameters.
- The idea is to regularize the change of generated model weights for previous tasks using the stored task representations.
- For example, before learning the 2nd task, f_H firstly generates the parameters θ_1 of the 1st task by inputting the 1st task representation vector, then when learning the 2nd task, a regularization loss is added between θ_1 and current generated parameters $\theta_{1'}$

Experiments setting & Evaluation

- They employ the GLUE tasks as the sequential training tasks in their experiments.
- To evaluate few-shot learning ability, they employ the 17 few-shot learning tasks.
- Evaluation
    - Instant Accuracy: evaluate right after learning each task
    - Final Accuracy: evaluate after learning all tasks

- **Measuring Catastrophic Forgetting**

Task	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Average
Continual learning										
BART-Vanilla	0.00	0.00	0.00	0.00	0.00	4.04	49.46	52.71	43.66	20.68
BART-MbPA++	60.40	73.74	61.27	66.47	69.50	45.56	63.31	55.96	39.44	59.52
BART-meta-MbPA	30.87	70.41	68.38	49.53	67.06	32.75	76.50	62.09	43.66	55.69
HNET-Vanilla	0.00	49.54	31.37	49.53	63.18	0.00	63.92	70.76	56.34	42.75
HNET-EWC	0.00	80.16	0.00	0.00	0.00	0.00	51.18	60.65	57.74	27.76
HNET-Reg	30.87	89.90	63.23	82.13	81.04	75.55	75.03	62.09	60.56	68.93
HNET$_{\backslash ST}$-Reg	60.11	91.28	83.82	81.47	81.25	73.02	70.09	62.45	57.74	**73.47**
HNET$_{d=4}$-Reg	78.72	89.68	31.62	49.80	78.98	44.38	57.66	63.54	59.15	61.50
Single-task learning										
HNET-Single	78.52	90.25	85.54	85.00	82.31	75.77	86.40	53.43	56.34	**77.06**
Adapter-Single	69.13	91.28	77.94	84.20	82.53	75.19	85.50	52.71	56.34	74.98
Majority	69.13	50.92	68.38	50.47	63.18	35.33	50.54	52.71	56.34	55.22

Experiments setting & Evaluation

- **Measuring Knowledge Transfer**
  - Previous tasks help to improve current task performance ?
  - Using instant accuracy.

Task	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	Average
BART-Vanilla	77.56	88.07	80.88	84.47	80.54	71.28	83.10	59.21	43.66	74.31
BART-MbPA++	80.15	91.97	80.64	84.73	83.25	75.50	85.56	58.84	56.34	77.48
BART-meta-MbPA	81.69	91.62	82.84	85.53	85.25	75.94	85.25	63.18	56.34	78.63
HNET-Vanilla	79.70	91.51	84.06	86.20	79.55	77.26	87.74	71.48	59.15	79.67
HNET-EWC	79.70	91.74	73.53	83.13	81.51	75.42	87.11	70.76	57.75	77.85
HNET-Reg	79.70	90.94	86.76	85.73	82.43	76.22	86.56	69.68	60.56	**79.84**
HNET$_{\backslash ST}$-Reg	78.04	90.82	86.52	86.00	81.95	75.52	87.06	62.45	57.74	78.46
HNET$_{d=4}$-Reg	79.19	92.43	70.09	83.20	79.13	75.83	86.40	70.40	59.15	74.01

Experiments setting & Evaluation

- Measuring Few-shot Learning

Task	**Entity Typing**	**Text Classification**	**NL Inference**	**Sentiment Analysis**	**Average**
Task Num.	2	10	1	4	17
Continual learning					
BART-Vanilla	59.57	54.21	64.35	78.83	61.23
BART-MBPA	60.88	54.39	69.59	85.58	63.39
BART-meta-MbPA	63.26	54.14	71.93	84.80	63.47
HNET-Vanilla	67.40	58.75	78.46	85.83	67.30
HNET-EWC	66.54	58.19	69.04	84.73	66.06
HNET-Reg	66.60	58.78	73.75	87.88	**67.43**
$\text{HNET}_{\backslash \text{ST}}$-Reg	61.44	57.62	68.39	84.73	65.08
Single-task learning					
HNET-Single	64.71	56.70	57.90	71.95	61.30
BART-Adapter	61.76	54.32	58.61	71.87	59.58

Enhancing Content Preservation in Text Style Transfer Using Reverse Attention and Conditional Layer Normalization

Dongkyu Lee **Zhiliang Tian** **Lanqing Xue** **Nevin L. Zhang**
Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology
{dleear, ztianac, lxueaa, lzhang}@cse.ust.hk

Background

- Text style transfer aims to alter the style (e.g., sentiment) of a sentence while preserving its content.
- A common approach is to map a given sentence to content representation that is free of style, and the content representation is fed to a decoder with a target style.
- How to distill content representation from a sentence?
  - Previous works: remove style tokens (may incur the loss of the content information)
- Proposed methods:
  - implicitly removing the style information of each token with reverse attention
  - Making the style dynamic with respect to the content (conditional layer normalization)

Method

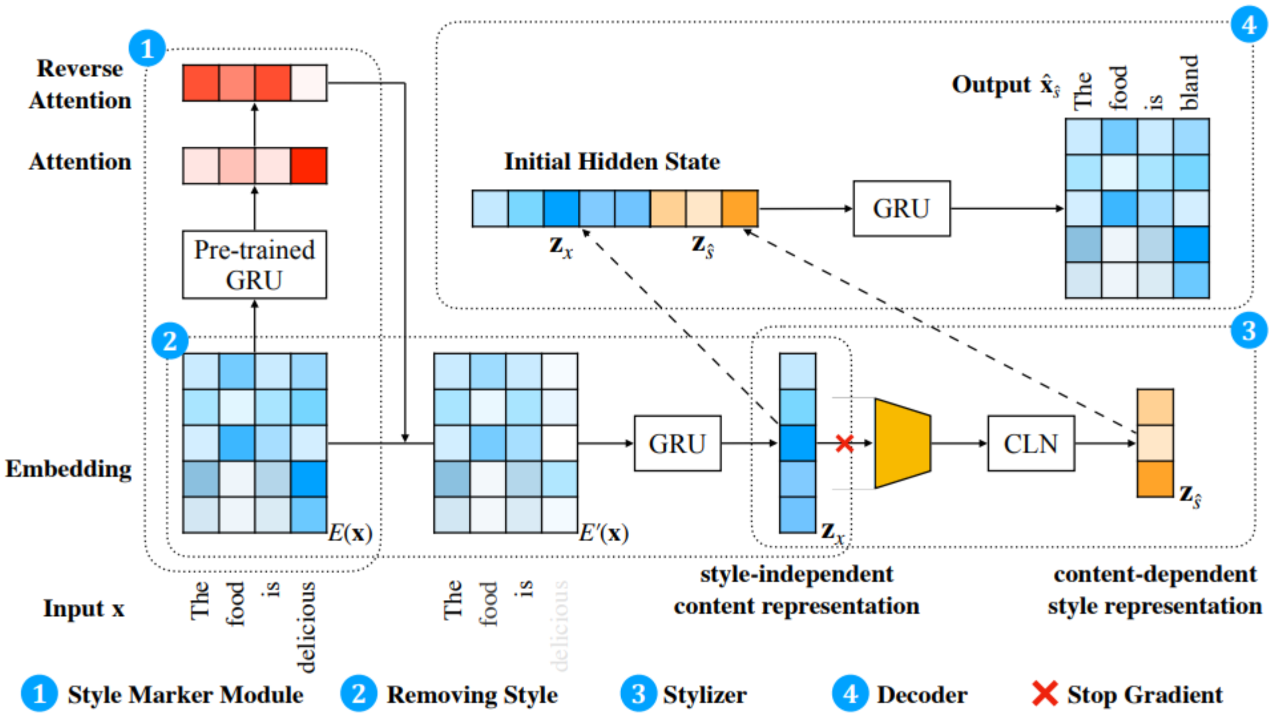
Reverse attention:

Pretrained GRU is a pretrained style classifier with self-attention layer.

Attention assigns the style tokens larger weights.

Reverse Attention = 1 – Attention assigns style tokens smaller weights.

Reverse Attention x Embeddings reduces the influence of style tokens while keeping the content information



Content-Dependent Style:

$$\tilde{\mathbf{z}}_{\mathbf{x}} = \mathbf{W}_z \mathbf{z}_{\mathbf{x}} + \mathbf{b}_z \qquad \mathbf{z}_{\hat{s}} = CLN(\tilde{\mathbf{z}}_{\mathbf{x}}; \hat{s}) = \gamma^{\hat{s}} \odot N(\tilde{\mathbf{z}}_{\mathbf{x}}) + \beta^{\hat{s}}$$

$$N(\tilde{\mathbf{z}}_{\mathbf{x}}) = \frac{\tilde{\mathbf{z}}_{\mathbf{x}} - \mu}{\sigma}$$

Our model learns separate γ^s (gain) and β^s (bias) parameters for different styles.

Method: Loss function

Self Reconstruction Loss:

$$\mathcal{L}_{self} = -\mathbb{E}_{(\mathbf{x},s)\sim\mathcal{D}}[\log p_D(\mathbf{x}|\mathbf{z}_{\mathbf{x}}, \mathbf{z}_s)]$$

Cycle Reconstruction Loss:

$$\mathcal{L}_{cycle} = -\mathbb{E}_{(\mathbf{x},s)\sim\mathcal{D}}[\log p_D(\mathbf{x}|\mathbf{z}_{\hat{\mathbf{x}}_{\hat{s}}}, \mathbf{z}_s)]$$

Content Loss:

$$\mathcal{L}_{content} = \mathbb{E}_{(\mathbf{x},s)\sim\mathcal{D}}||\mathbf{z}_{\mathbf{x}} - \mathbf{z}_{\hat{\mathbf{x}}_{\hat{s}}}||_2^2$$

Style Transfer Loss:

$$\mathcal{L}_{style} = -\mathbb{E}_{(\mathbf{x},s)\sim\mathcal{D}}[\log p_C(\hat{s}|\hat{\mathbf{x}}_{\hat{s}})]$$

Thanks