Microsoft

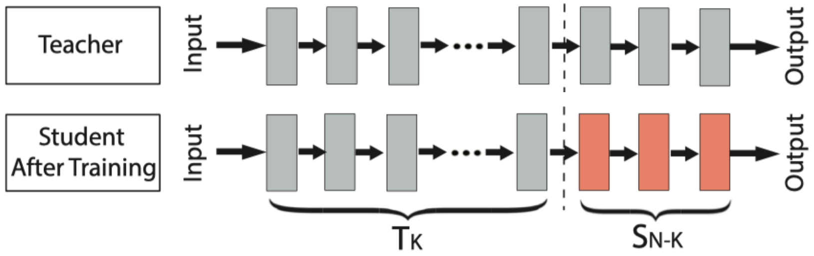
迁移学习前沿探究探讨：低资源、领域泛化与安全迁移

王晋东　微软亚洲研究院
https://jd92.wang/
2022.04.08

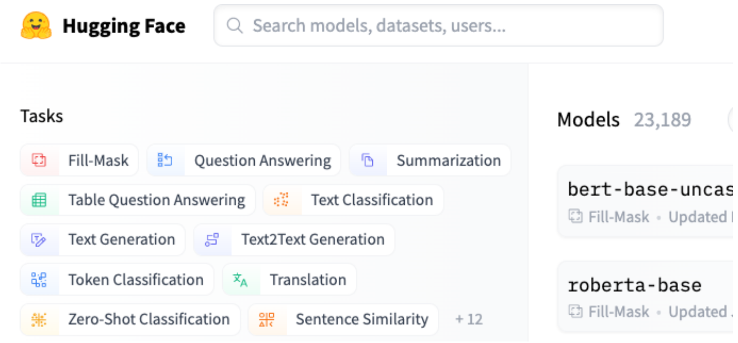
This talk will not be possible without my interns and collaborators.

Background: transfer learning

- Transfer learning: 迁移学习
  - Reuse *pre-trained* models by *fine-tuning* it for downstream tasks
  - Today's ML applications widely adopt the pre-train and fine-tune paradigm
    - Why? Because training from scratch is extremely time-consuming
  - Model highlights: ResNet for CV, BERT and RoBERTa for NLP

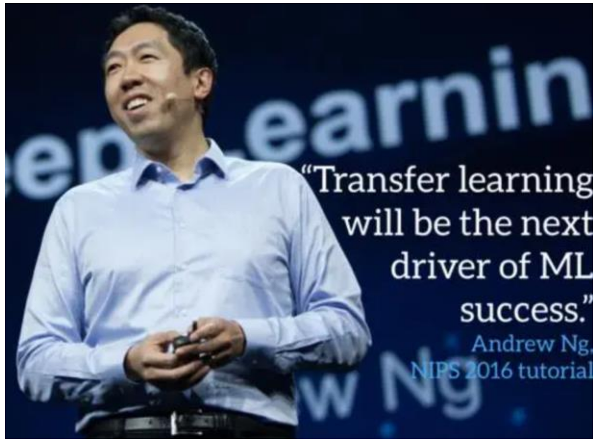


Primitive Models	# Repositories
GoogLeNet [56]	466
AlexNet [33]	303
Inception.v3 [57]	190
ResNet [27]	341
VGG [54]	931
Total	2,220

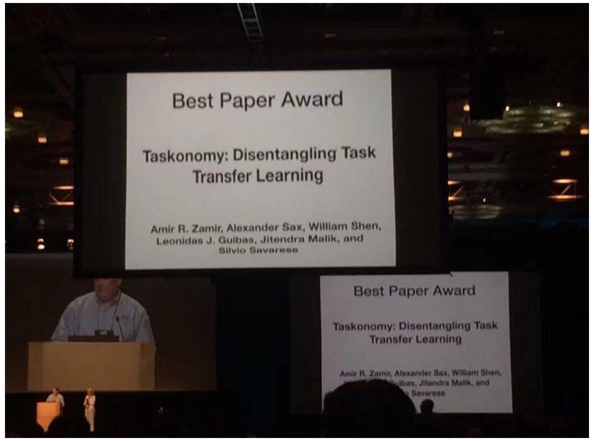


- Industrial applications
  - Google Cloud ML tutorial suggests using Google's Inception V3 model as a pre-trained model
  - Microsoft Cognitive Toolkit (CNTK) suggests using ResNet18 as a pre-trained model for tasks such as flower classification

Transfer learning: always the research frontier

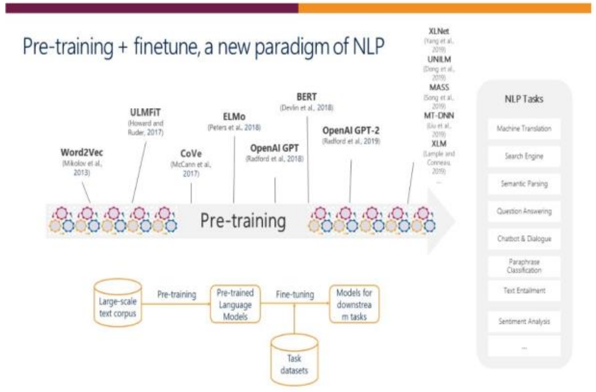


NIPS’16 tutorial



CVPR’18 best paper

IJCAI’18 Ads challenge winner



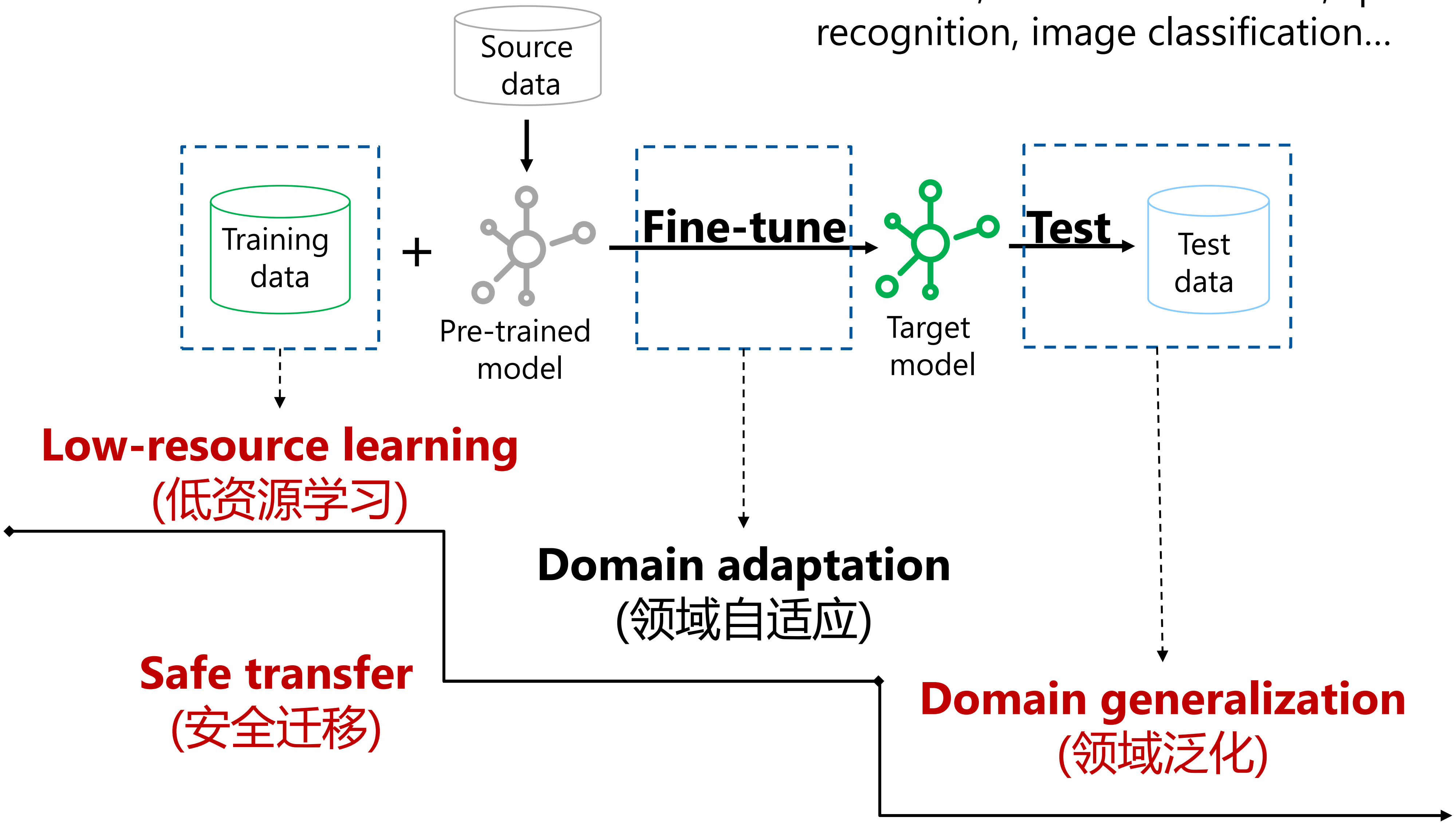
ACL’19 opening keynote

ACL’20 best paper nominee

Statements from *Turing Award* winners in 2021

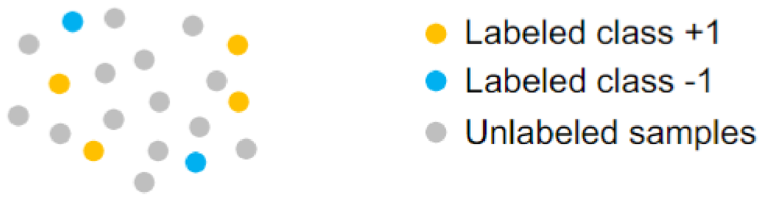
Roadmap

Applications: Time series analysis, anomaly detection, Parkinson's disease, speech recognition, image classification…



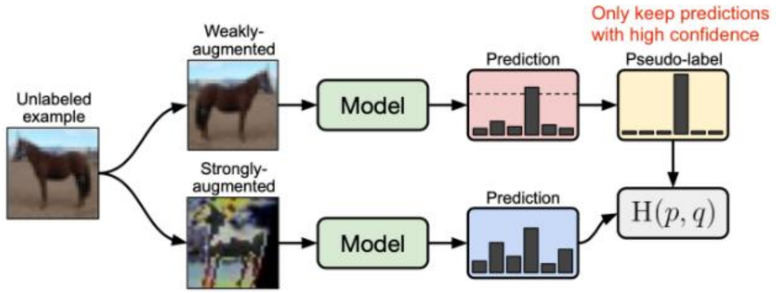
Contents

Low-resource Learning



- Research background
  - Learn a generalized model by relying on a small amount of labeled data
- Problem
  - How to guarantee that knowledge can seamlessly transfer from labeled to unlabeled data?
  - Transfer criterion: a ***fixed*** threshold by Google's FixMatch[NeurIPS'20]

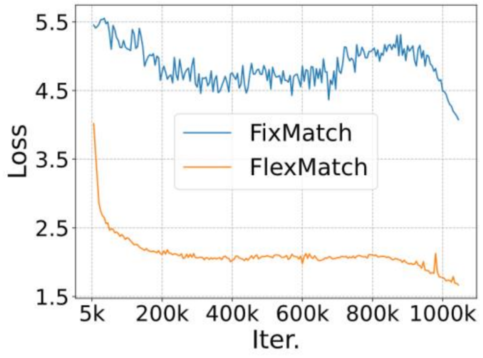
$$\frac{1}{\mu B}\sum_{b=1}^{\mu B}\mathbb{1}(\max(p_m(y|\omega(u_b))) > \boxed{\tau})H(\hat{p}_m(y|\omega(u_b)), p_m(y|\Omega(u_b)))$$

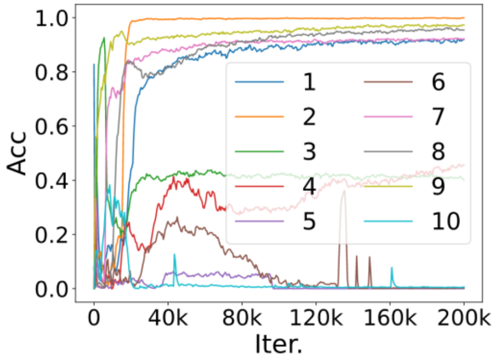


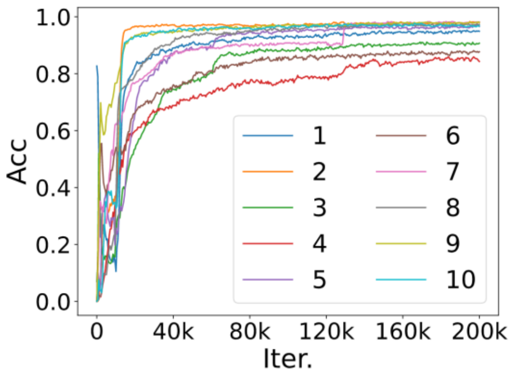
- Research challenge
  - Is the pre-defined fixed threshold for semi-supervised learning enough?
  - Can design better thresholding for semi-supervised learning?

Low-resource learning

- Fixed vs. flexible threshold
  - We should learn different thresholds for different classes







FlexMatch

- Our proposal: FlexMatch
  - Different for different classes → per-class *adaptation*
  - Lower down thresholds for hard-to-learn classes → encourage *difficult* classes
  - Raise thresholds if already well-learned → keep strict to ensure final acc
  - Dynamically adjusted for every class at every time step → automate the process

Low-resource learning: FlexMatch

- Technical details
  - A *curriculum pseudo labeling (CPL)* strategy that gradually learn the difficulties of classes
  - Cluster assumption: The learning effect of a class can be reflected by the number of samples whose predictions fall above the high fixed threshold and into this class.

$$\sigma_t(c) = \sum_{n=1}^{N} \mathbb{1}(\max(p_{m,t}(y|u_n)) > \tau) \cdot \mathbb{1}(\arg\max(p_{m,t}(y|u_n)) = c).$$

$$\beta_t(c) = \frac{\sigma_t(c)}{\max\left\{\max_c \sigma_t, N - \sum_c \sigma_t\right\}},$$

Warm-up&Norm

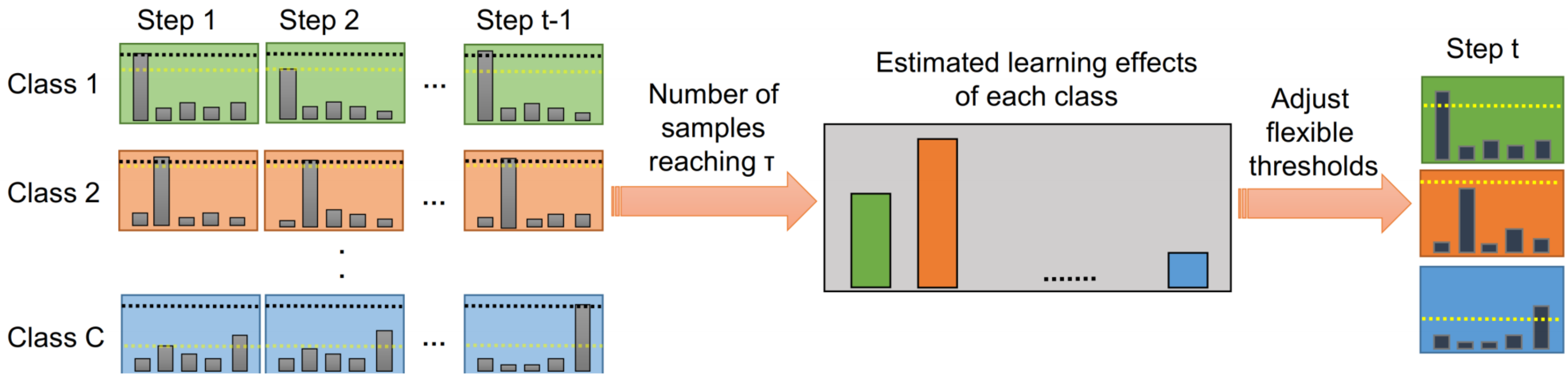
$$\mathcal{T} \xrightarrow{\text{Flexible}} \mathcal{T}_t(c) = a_t(c) \cdot \tau \xrightarrow{\text{Warm-up}} \mathcal{T}_t(c) = \mathcal{M}(\beta_t(c)) \cdot \tau,$$

$$\mathcal{M}(x) = \frac{x}{2-x}$$

Mapping function

Method: CPL

- Curriculum Pseudo Labeling (CPL)



Results

Dataset	CIFAR-10			CIFAR-100			STL-10			SVHN	
Label Amount	40	250	4000	400	2500	10000	40	250	1000	40	1000
PL	74.61±0.26	46.49±2.20	15.08±0.19	87.45±0.85	57.74±0.28	36.55±0.24	74.68±0.99	55.45±2.43	32.64±0.71	64.61±5.60	**9.40**±0.32
Flex-PL	**73.74**±1.96	**46.14**±1.81	**14.75**±0.19	**85.72**±0.46	**56.12**±0.51	**35.60**±0.15	**73.42**±2.19	**52.06**±2.50	**32.05**±0.37	**63.21**±3.64	12.05±0.54
UDA	10.62±3.75	5.16±0.06	4.29±0.07	46.39±1.59	27.73±0.21	22.49±0.23	37.42±8.44	9.72±1.15	6.64±0.17	5.12±4.27	**1.89**±0.01
Flex-UDA	**5.44**±0.52	**5.02**±0.07	**4.24**±0.06	**45.17**±1.88	**27.08**±0.15	**21.91**±0.10	**29.53**±2.10	**9.03**±0.45	**6.10**±0.25	**3.42**±1.51	2.02±0.05
FixMatch	7.47±0.28	**4.86**±0.05	4.21±0.08	46.42±0.82	28.03±0.16	22.20±0.12	35.97±4.14	9.81±1.04	6.25±0.33	**3.81**±1.18	**1.96**±0.03
FlexMatch	**4.97**±0.06	4.98±0.09	**4.19**±0.01	**39.94**±1.62	**26.49**±0.20	**21.90**±0.15	**29.15**±4.16	**8.23**±0.39	**5.77**±0.18	8.19±3.20	6.72±0.30
Fully-Supervised	4.62± 0.05			19.30± 0.09			-			2.13± 0.02	

- Significant improvement with **limited labels.**
- Significant improvement with **complicated tasks.**
- Significant improvement on **convergence speed**.
- **No new hyperparameter** introduced.
- **No additional computation** introduced.

Table 2: Error rate results on ImageNet after 2^{20} iterations.

Method	Top-1	Top-5
FixMatch	43.66	21.80
FlexMatch	**41.85**	**19.48**

- B. Zhang et al. Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling. NeurIPS 2021. https://arxiv.org/abs/2110.08263

TorchSSL

- A unified Pytorch library for semi-supervised learning

  https://github.com/TorchSSL/TorchSSL

Supported algorithms: In addition to fully-supervised (as a baseline), TorchSSL supports the following popular algorithms:

1. PiModel (NeurIPS 2015) [1]
2. MeanTeacher (NeurIPS 2017) [2]
3. PseudoLabel (ICML 2013) [3]
4. VAT (Virtual adversarial training, TPAMI 2018) [4]
5. MixMatch (NeurIPS 2019) [5]
6. UDA (Unsupervised data augmentation, NeurIPS 2020) [6]
7. ReMixMatch (ICLR 2019) [7]
8. FixMatch (NeurIPS 2020) [8]
9. FlexMatch (NeurIPS 2021) [9]

Besides, we implement our Curriculum Pseudo Labeling (CPL) method for Pseudo-Label (Flex-Pseudo-Label) and UDA (Flex-UDA).

Supported datasets: TorchSSL currently supports 5 popular datasets in SSL research:

1. CIFAR-10
2. CIFAR-100
3. STL-10
4. SVHN
5. ImageNet

TorchSSL/TorchSSL Public

A PyTorch-based library for semi-supervised learning (NeurIPS'21)

Python 691 100

Run the experiments

It is convenient to perform experiment with TorchSSL. For example, if you want to run FlexMatch algorithm:

1. Modify the config file in `config/flexmatch/flexmatch.yaml` as you need
2. Run `python flexmatch.py --c config/flexmatch/flexmatch.yaml`

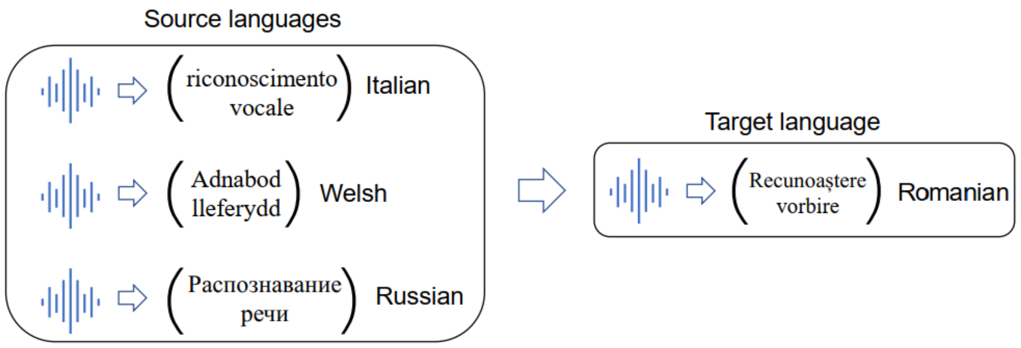
Customization

If you want to write your own algorithm, please follow the following steps:

1. Create a directory for your algorithm, e.g., `SSL`, write your own model file `SSl/SSL.py` in it.
2. Write the training file in `SSL.py`
3. Write the config file in `config/SSL/SSL.yaml`

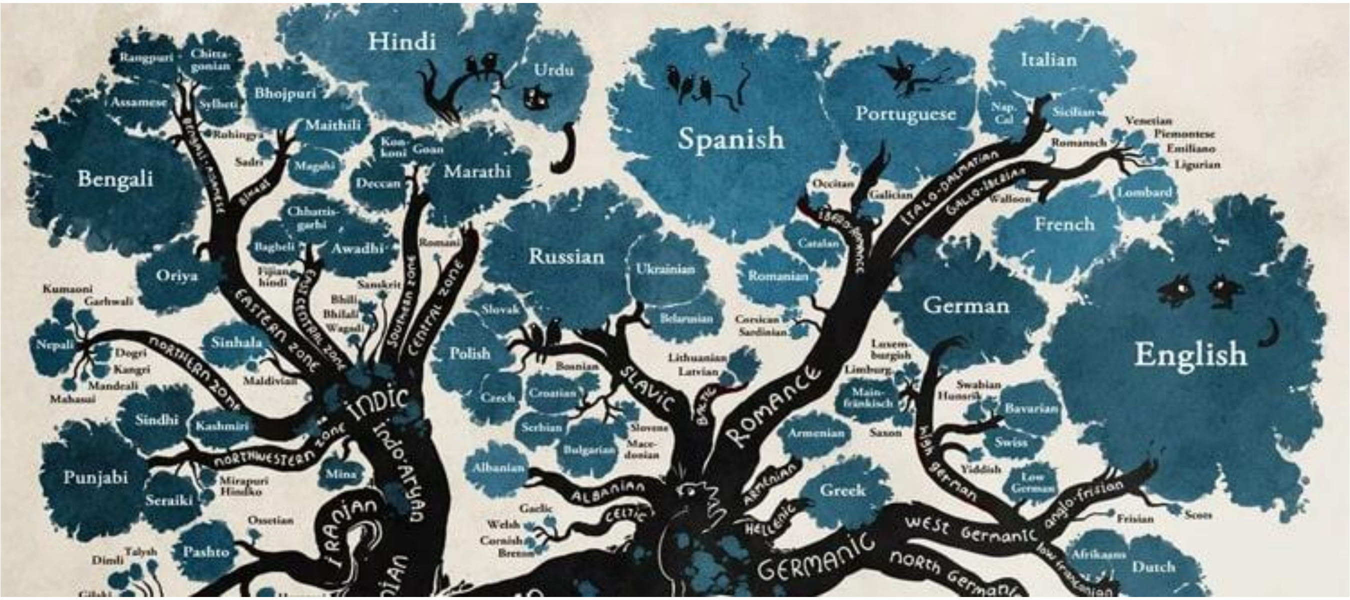
Application: speech recognition

- Background
  - There are around **7,000** languages in the world, most of which do not have large amount of labelled data
  - Automatic speech recognition (ASR) for the low-resource languages remains a challenge for end-to-end (E2E) models
- Existing methods
  - *Pre-training* on the rich-resource languages and fine-tuning on the low-resource languages
  - Performing *multi-task learning* on rich- and low-resource languages simultaneously
  - *Meta-learning* on the rich-resource languages for rapid adaptation to the low-resource languages
- Limitations
  - Low parameter-efficiency
    - Speech-Transformer models have huge amounts of parameters
  - Overfitting problem
    - Heavy models can get easily overfit on low-resource languages



Motivation

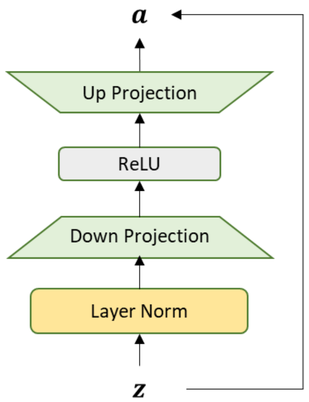
- Learn the relationship between different languages
  - Different languages have different vocabulary, but may share same representations
- Relationship
  - Implicit: make no assumptions on their relationship, use a network to learn it directly
  - Explicit: assume languages have a linear relation, simplify the algorithm
- Reduce overfitting
  - Freeze most of the parameters
  - Only tune a small part

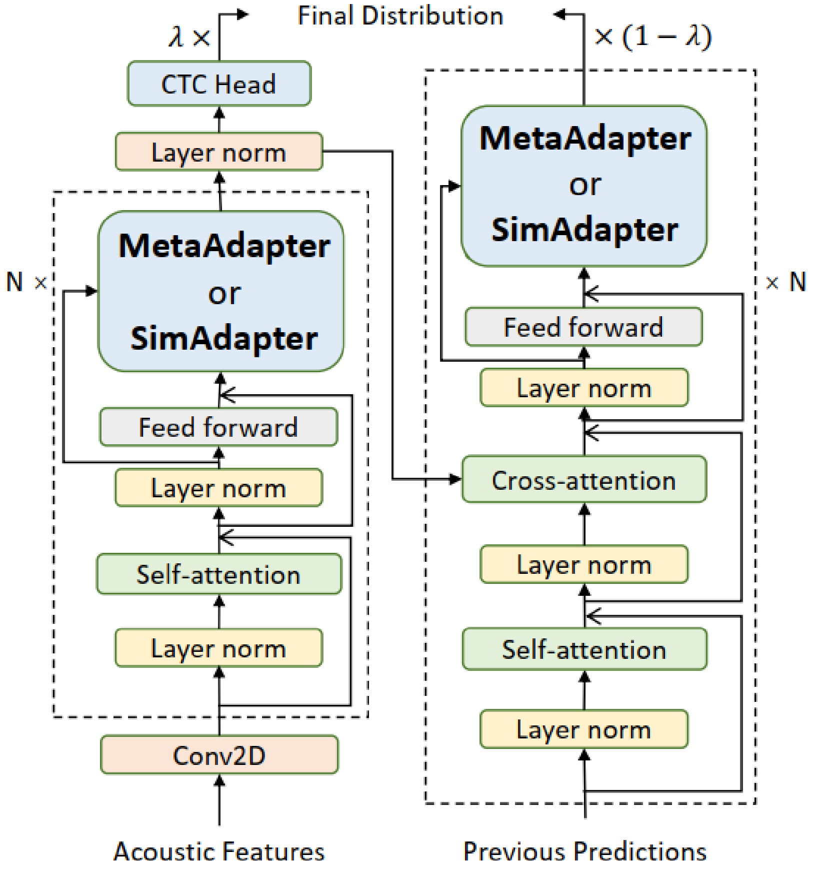


Our approach

- Exploiting adapters for cross-lingual low-resource ASR
  - **MetaAdapter**: (implicit relationship)
    - Directly learn the relationship between different languages
  - **SimAdapter**: (explicit relationship)
    - Assume they have a linear relationship, learn it using attention
  - **SimAdapter+**: (implicit + explicit)
    - Combine MetaAdapter and SimAdapter for better results

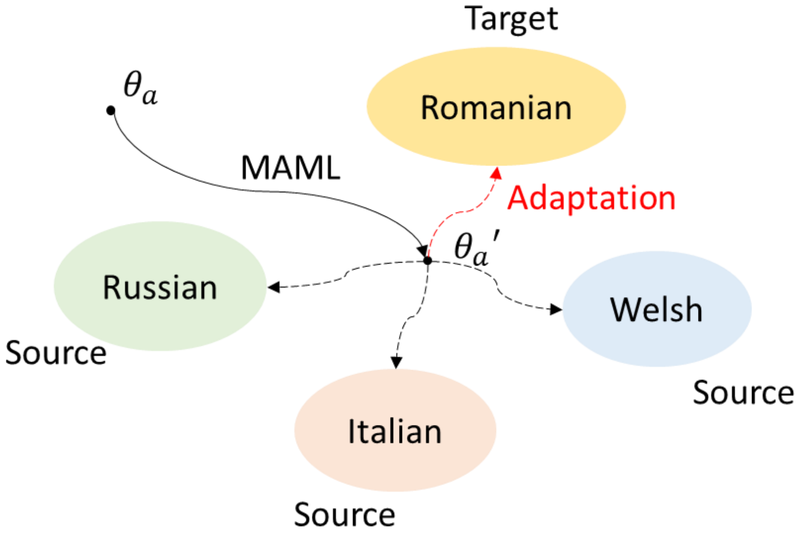
$$\mathbf{a}^l = \mathrm{Adapter}(\mathbf{z}^l) = \mathbf{z}^l + \mathbf{W}_u^l \mathrm{ReLU}\left(\mathbf{W}_d^l\left(\mathrm{LN}\left(\mathbf{z}^l\right)\right)\right)$$





MetaAdapter for Cross-lingual ASR

- Motivation: obtain a proper initialization for fast adaptation



Algorithm 1 Learning algorithm of the MetaAdapter

Input: Rich-resource languages $\{S_1, \cdots, S_N\}$, low-resource task L_T.

1: Train language-specific heads on source languages S_i.
2: Initialize the MetaAdapter.
3: **while** meta-learning not done **do**
4: Optimizing the MetaAdapter using Eq. (6).
5: **end while**
6: Train the target head on target language L_T.
7: Fine-tune the MetaAdapter using ASR loss Eq. (1).
8: **return** Cross-lingual ASR model.

- Pre-train the Adapters using Model-Agnostic Meta-Learning (MAML):

$$\theta'_{a,i} = \theta_a - \epsilon \nabla \mathcal{L}_{S_i^{tr}}\left(f_{\theta_a}\right)$$

$$\mathcal{L}_{S_i^{val}}(f_{\theta'_{a,i}}) = \mathcal{L}_{S_i^{val}}\left(f_{\theta_a - \epsilon \nabla_{\theta_a} \mathcal{L}_{S_i^{tr}}(f_{\theta_a})}\right)$$

$$\theta_a = \theta_a - \mu \sum_{i=1}^{N} \nabla_{\theta_a} \mathcal{L}_{S_i^{val}}\left(f_{\theta'_{a,i}}\right)$$

SimAdapter for Cross-lingual ASR

- Formulation
  - SimAdapter:

$$\mathrm{SimAdapter}(\mathbf{z}, \mathbf{a}_{\{S_1, S_2, \ldots, S_N\}}) = \sum_{i=1}^{N} \mathrm{Attn}(\mathbf{z}, \mathbf{a}_{S_i}) \cdot (\mathbf{a}_{S_i} \mathbf{W}_V)$$

$$\mathrm{Attn}(\mathbf{z}, \mathbf{a}) = \mathrm{Softmax}\left(\frac{(\mathbf{z}\mathbf{W}_Q)(\mathbf{a}\mathbf{W}_K)^\top}{\tau}\right)$$

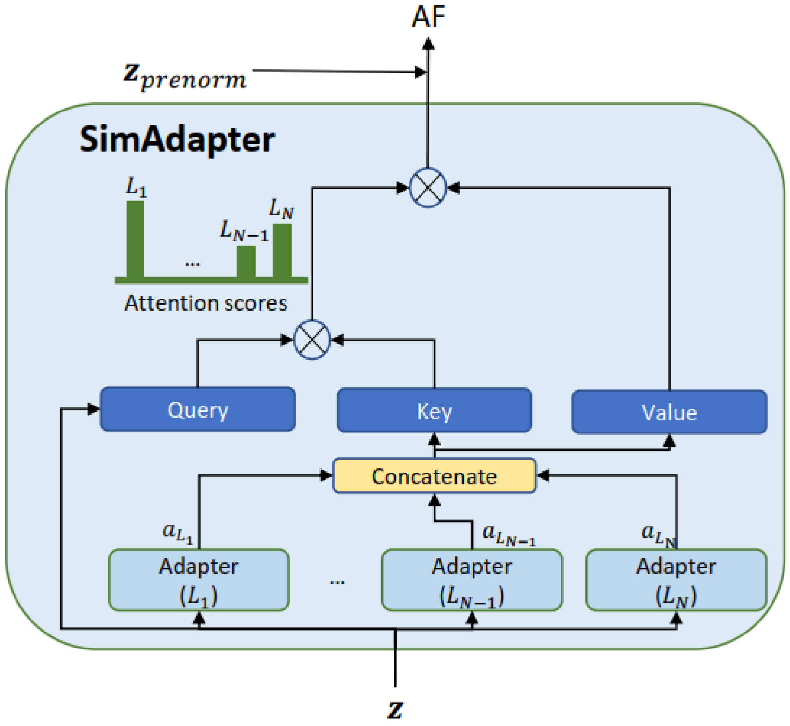
  - Stability regularization: $\mathcal{L}_{\mathrm{reg}} = \sum_{i,j} \left((\mathbf{I}_V)_{i,j} - (\mathbf{W}_V)_{i,j}\right)^2$
  - Fusion-guide loss:
    - Encourage the model to focus on the corresponding adapters for the

$$\mathcal{L}_{\mathrm{guide}}^{f} = -\frac{1}{K \times S} \sum_{s=1}^{S} \sum_{k=1}^{K} \log \alpha_{f,k}^{s},$$

Algorithm 2 Learning algorithm of SimAdapter

Input: Rich-resource languages $\{S_1, \cdots, S_N\}$, low-resource task L_T.

Train language-specific heads on the source languages S_i and the target language.
Train Adapters A_t on top of language-specific heads.
Initialize SimAdapter layers.
while not done **do**
  Optimizing SimAdapter layers using Eq. (12).
end while
return Target ASR model.



Experiment Results

- Main results

Target Language	Hybrid DNN/HMM	Transformer	Head	Full-FT	Adapter	SimAdapter	MetaAdapter	SimAdapter +
Romanian (ro)	70.14	97.25	63.98	53.90	48.34	47.37	**44.59**	47.29
Czech (cs)	63.15	48.87	75.12	34.75	37.93	35.86	37.13	**34.72**
Breton (br)	-	97.88	82.80	61.71	58.77	**58.19**	58.47	59.14
Arabic (ar)	69.31	75.32	81.70	47.63	47.31	47.23	46.82	**46.39**
Ukrainian (uk)	77.76	64.09	82.71	**45.62**	50.84	48.73	49.36	47.41
Average	-	76.68	77.26	48.72	48.64	47.48	47.27	**46.99**
Weighted Average	-	72.28	77.54	46.72	47.38	46.08	46.12	**45.45**

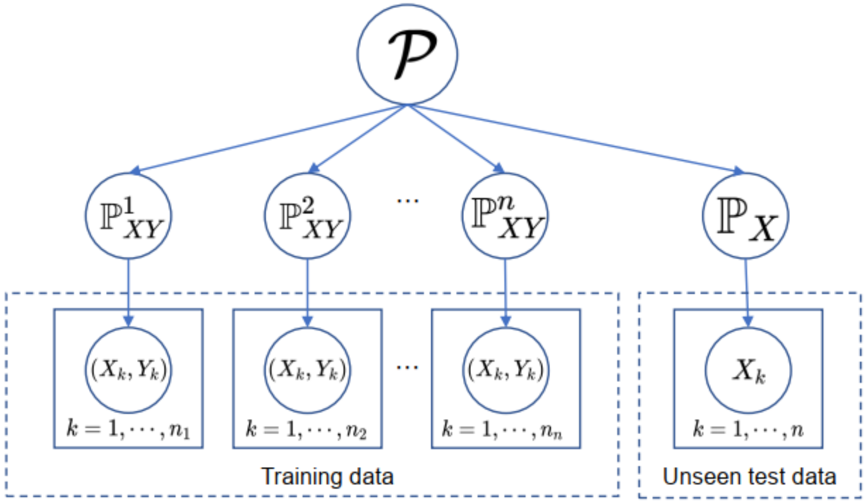
- Impact of adapter training strategies

Target	Joint	+SimAdapter	Two-stage	+SimAdapter
ro	52.92	53.88	48.34	47.37
cs	39.16	36.79	37.93	35.86
br	65.10	63.37	58.77	58.19
ar	50.53	49.31	47.31	47.23
uk	52.27	48.84	50.84	48.73
Average	52.00	50.44	48.64	47.48
+Weighted	50.35	48.57	47.38	46.08

Hou et al. Exploiting Adapters for Cross-lingual Low-resource Speech Recognition. TASLP 2022.
https://arxiv.org/abs/2105.11905

Domain generalization

- Research background
  - Leverage multiple training distributions to learn a generalized model on *unseen* domains
- Problem
  - Data properties are dynamically changing over time
  - Leading to dynamic distribution change
- Research challenge
  - How to capture the dynamical distribution change?
  - E.g., how to quantify the distributions in time series?



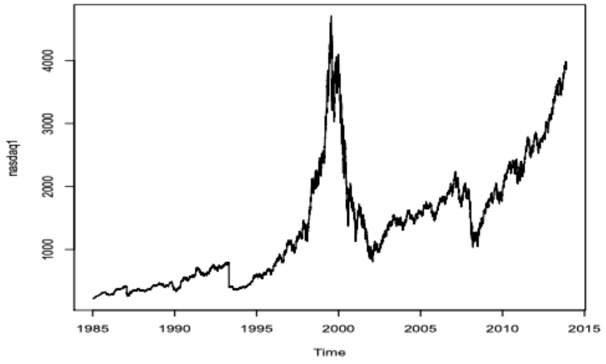
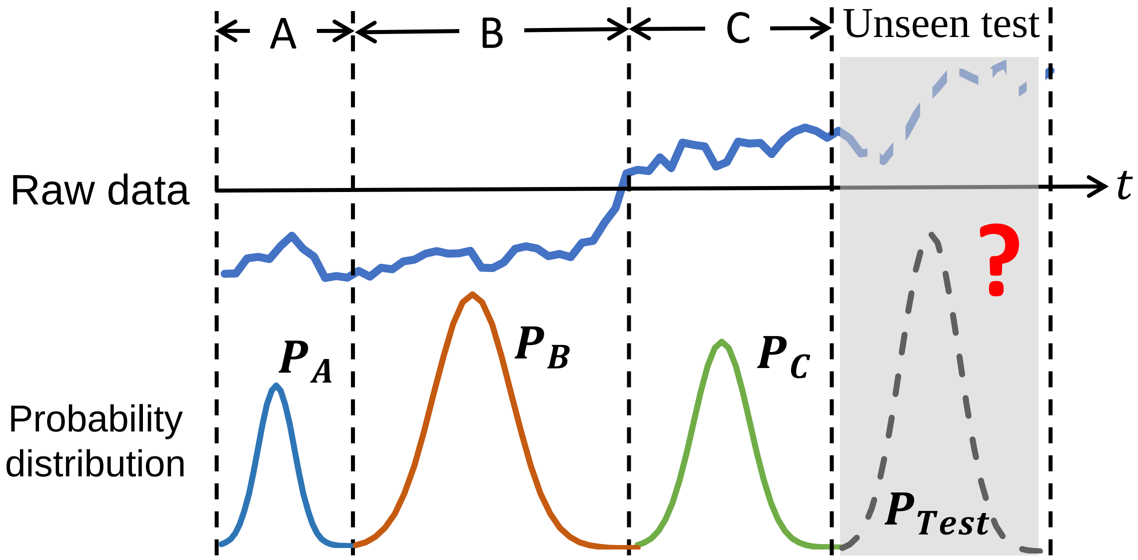


Figure 1.3: Plot of daily closing price of Nasdaq 1985-2014

- Wang et al. Generalizing to unseen domains: a survey on domain generalization. IJCAI 2021 survey track. https://arxiv.org/abs/2103.03097

Domain generalization

- Our proposal: AdaRNN (adaptive RNNs)
  - Formulate the problem as **Temporal Covariate Shift (TCS)**
  - Design a framework to solve TCS in continuous data



Temporal Covariate Shift: $P_A \neq P_B \neq P_C \neq P_{Test}$

How to solve TCS?

Construct worst-case distribution scenario

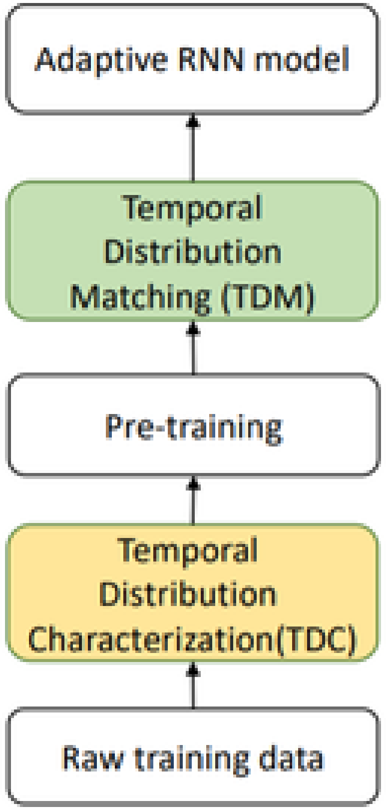
Match the big distribution gap

Good model

AdaRNN: Adaptive RNNs

Temporal Distribution Characterization: characterizing the distribution information in original TS

Temporal Distribution Matching: build a temporally-invariant model



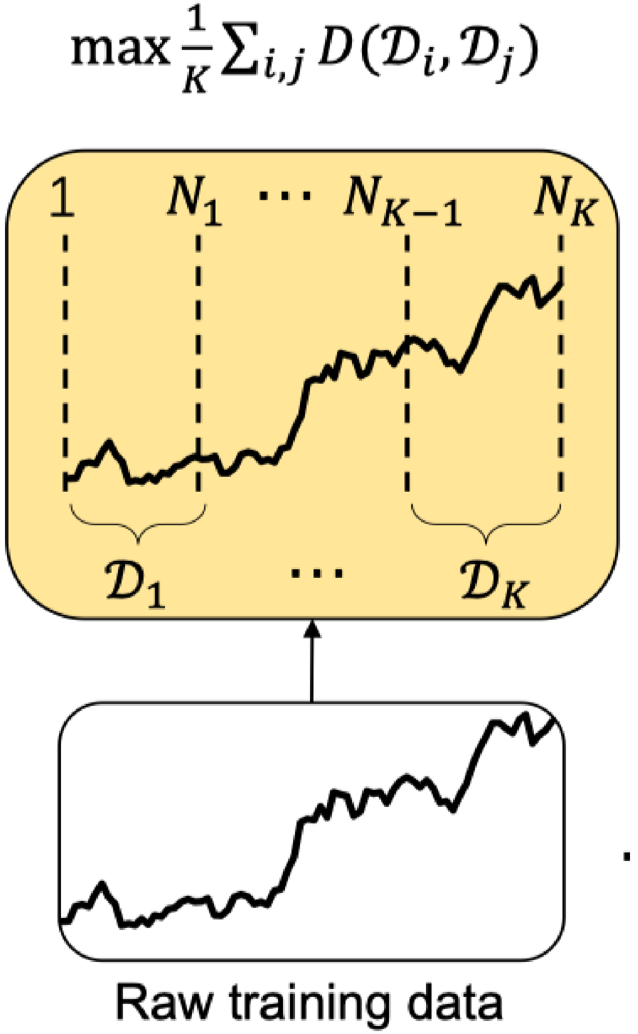
(a) Overview of AdaRNN

Temporal Distribution Characterization

- TDC
  - Find the K **most dissimilar** segments
  - How to define similarity?
    - Distribution distance D
  - Why most dissimilar segments?
    - Diverse distribution information helps generalization
  - Objective

$$\max_{0<K\leq K_0} \max_{N_1,\cdots,N_K} \frac{1}{K} \sum_{1\leq i\neq j\leq K} D(\mathcal{D}_i, \mathcal{D}_j)$$

$$\text{s.t. } \forall\, i,\ \Delta_1 < N_i < \Delta_2;\ \sum_{i=1}^{K} N_i = N,$$



It is similar to a dynamic programming problem, but we solve it using greedy algorithm for efficiency.

Temporal Distribution Matching

- Plain method
  - A plain domain generalization (DG) problem with K domains

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{K} \mathcal{L}_{pred}(\mathbf{y}_i, \hat{\mathbf{y}}_i) + \lambda \sum_{1 \leq i,j \leq K} D(\mathcal{D}_i, \mathcal{D}_j)$$

  - Plain DG ignores the importance of each hidden representation's distribution
- TDM
  - Our solution: an adaptive weight matrix for each hidden state

$$\theta^*, \boldsymbol{\alpha}^* = \arg\min_{\theta, \boldsymbol{\alpha}} \mathcal{L}_{pred}(\theta) + \lambda \sum_{1 \leq i,j \leq K} \mathcal{L}_{tdm}(\mathbf{H}_i, \mathbf{H}_j; \boldsymbol{\alpha}_{i,j}, \theta)$$

$$\mathcal{L}_{tdm}(\mathbf{H}_i, \mathbf{H}_j) = \sum_{t=1}^{T} \alpha_{i,j}^t D(\mathbf{h}_i^t, \mathbf{h}_j^t)$$

Temporal Distribution Matching (Cont.)

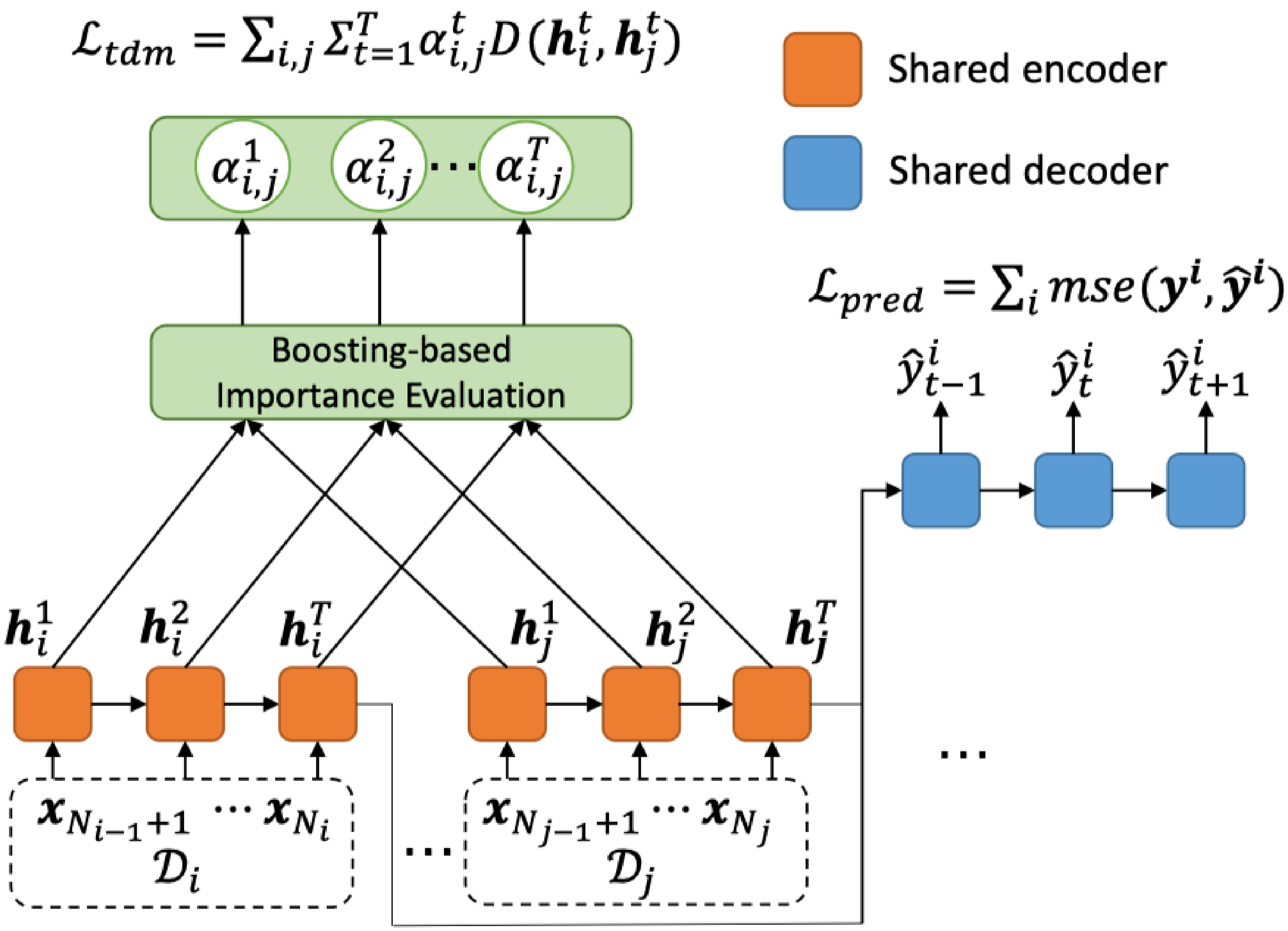
- How to learn the weight matrix?
  - A naïve way of attention layer will fail since:
    - At early stage, the hidden state representations learned by θ are less meaningful, which will result in insufficient learning of weights
    - Network can easily get stuck since it is very complex and time-consuming
  - Our solution
    - A boosting-based importance evaluation

$$\alpha_{i,j}^{t,(n+1)} = \begin{cases} \alpha_{i,j}^{t,(n)} \times G(d_{i,j}^{t,(n)}, d_{i,j}^{t,(n-1)}) & d_{i,j}^{t,(n)} \geq d_{i,j}^{t,(n-1)} \\ \alpha_{i,j}^{t,(n)} & \text{otherwise} \end{cases}$$

where

$$G(d_{i,j}^{t,(n)}, d_{i,j}^{t,(n-1)}) = (1 + \sigma(d_{i,j}^{t,(n)} - d_{i,j}^{t,(n-1)}))$$

Temporal Distribution Matching (Cont.)



Results

Method	ACC	P	R	F1	AUC
LightGBM	84.11	83.73	83.63	84.91	90.23
GRU	85.68	85.62	85.51	85.46	91.33
MMD	86.39	86.80	86.26	86.38	91.77
DANN	85.88	85.59	85.62	85.56	91.41
AdaRNN (Adv.)	87.31	87.19	87.18	87.17	92.32
AdaRNN (Cos.)	**88.44**	**88.71**	**88.59**	**88.63**	**93.19**

Table 2: Results on UCI time series classification dataset

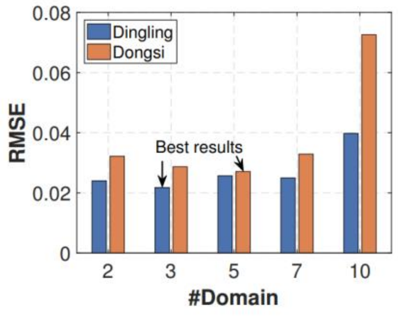
Method	IC	ICIR	RankIC	RankICIR	RawIC	RawICIR
LightGBM	0.063	0.551	0.056	0.502	0.064	0.568
STRIPE	0.108	0.987	0.101	0.946	0.109	0.993
GRU	0.106	0.965	0.098	0.925	0.109	0.986
MMD	0.107	0.962	0.101	0.926	0.108	0.967
AdaRNN	**0.115**	**1.071**	**0.110**	**1.035**	**0.116**	**1.077**

Table 3: Results on stock price prediction

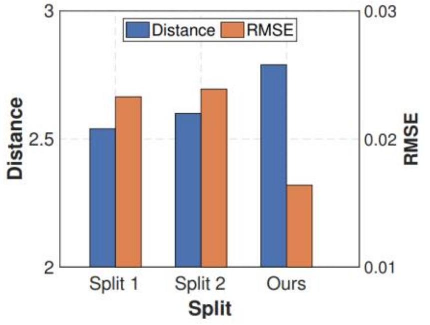
	Dongsi		Tiantan		Nongzhanguan		Dingling		Δ(%)	Electric Power
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE		
FBProphet [10]	0.1866	0.1403	0.1434	0.1119	0.1551	0.1221	0.0932	0.0736	-	0.080
ARIMA	0.1811	0.1356	0.1414	0.1082	0.1557	0.1156	0.0922	0.0709	-	-
GRU	0.0510	0.0380	0.0475	0.0348	0.0459	0.0330	0.0347	0.0244	0.00	0.093
MMD-RNN	0.0360	0.0267	0.0183	0.0133	0.0267	0.0197	0.0288	0.0168	-61.31	0.082
DANN-RNN	0.0356	0.0255	0.0214	0.0157	0.0274	0.0203	0.0291	0.0211	-59.97	0.080
LightGBM	0.0587	0.0390	0.0412	0.0289	0.0436	0.0319	0.0322	0.0210	-11.08	0.080
LSTNet [23]	0.0544	0.0651	0.0519	0.0651	0.0548	0.0696	0.0599	0.0705	-	0.080
Transformer [45]	0.0339	0.0220	0.0233	0.0164	0.0226	0.0181	0.0263	0.0163	-61.20	0.079
STRIPE [24]	0.0365	0.0216	0.0204	0.0148	0.0248	0.0154	0.0304	0.0139	-64.60	0.086
AdaRNN	**0.0295**	**0.0185**	**0.0164**	**0.0112**	**0.0196**	**0.0122**	**0.0233**	**0.0150**	**-73.57**	**0.077**

Analysis

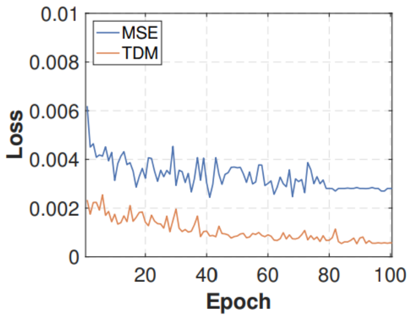
- Segment number matters!
  - Different numbers of segments reflect different distribution information
- Our TDC algorithm give the best segmentation results
  - Better than random split and reverse
- Convergence
  - Our method can converge within a few iterations
- Training time
  - Our method will not bring significant computational burden and even more efficient than SOTA



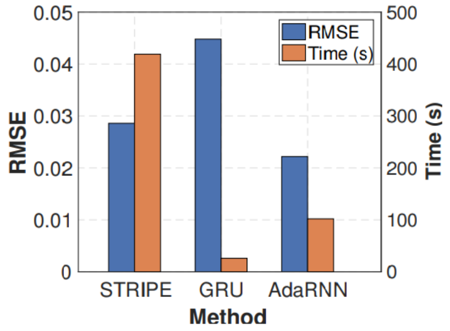
(a) #Domain in TDC



(b) Different domain split



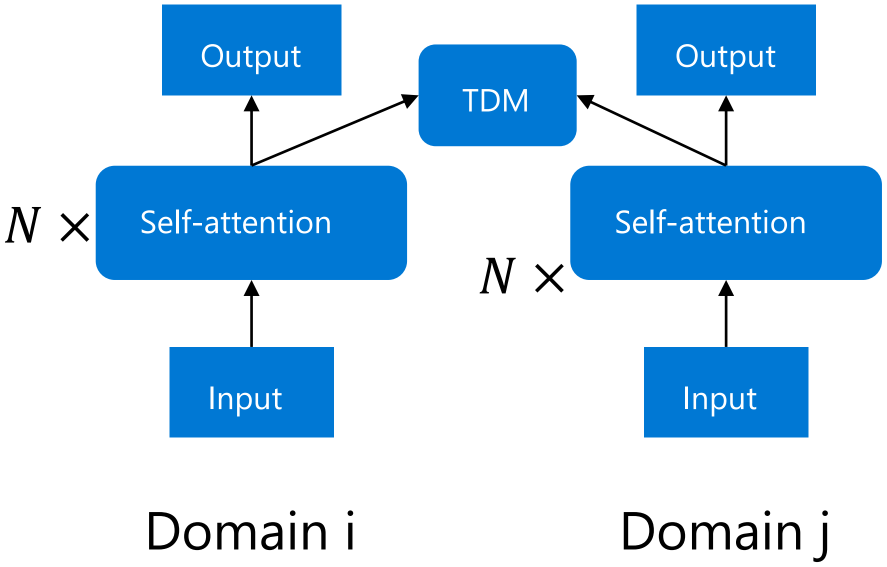
(d) Convergence



(e) Training time

One more thing: Extension to Transformer

- The structure can also apply to Transformers…
  - AdaTransformer: Adaptive Transformer



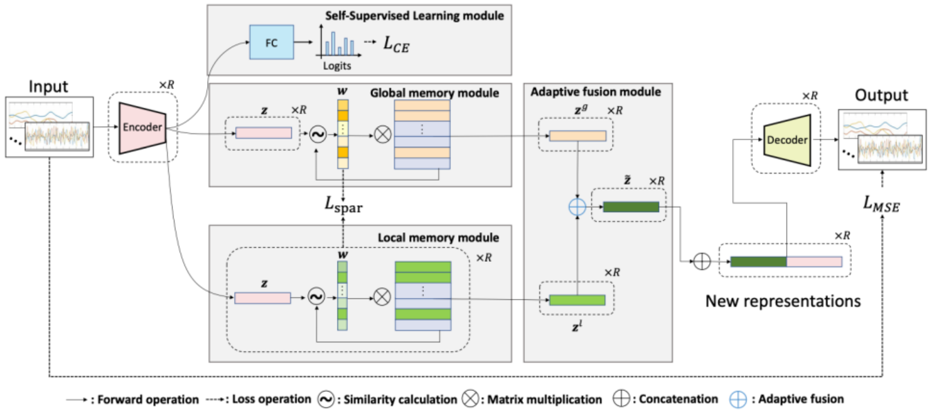
Method	Station 1	Station 2
Vanilla Transformer	0.028	0.035
AdaTransformer	**0.025**	**0.030**

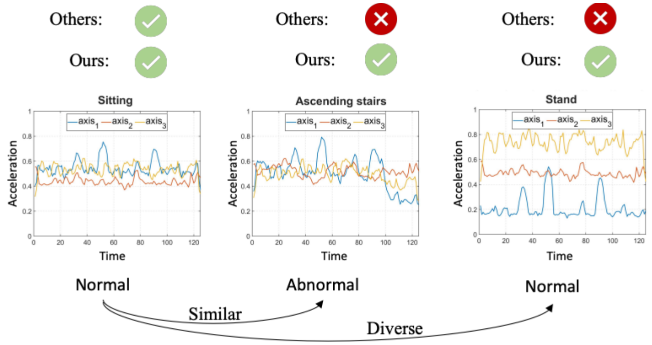
We did not tune hyperparameters heavily; better results will come if you tune them carefully
Research on transformer is left for future work.

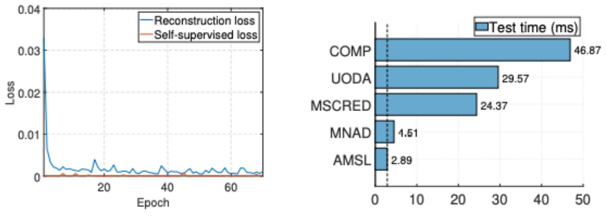
- Du et al. AdaRNN: adaptive learning and forecasting for time series. CIKM 2021. https://arxiv.org/abs/2108.04443

Application to anomaly detection

- AMSL: adaptive memory network with self-supervised learning
  - Limited normal data: lack of representation patterns → Self-supervised learning
  - Unseen abnormal data: needs to learn from normal data → adaptive memory network
  - 2-5% improvement over best baselines
  - Efficient and simple to implement







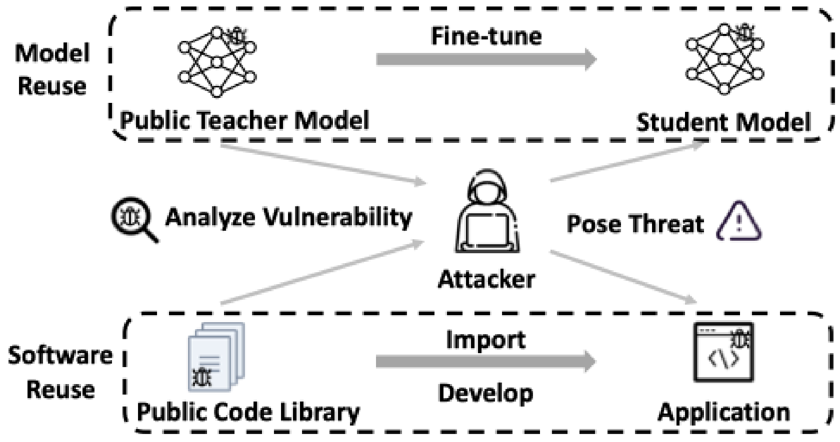
(a) Convergence (b) Inference time

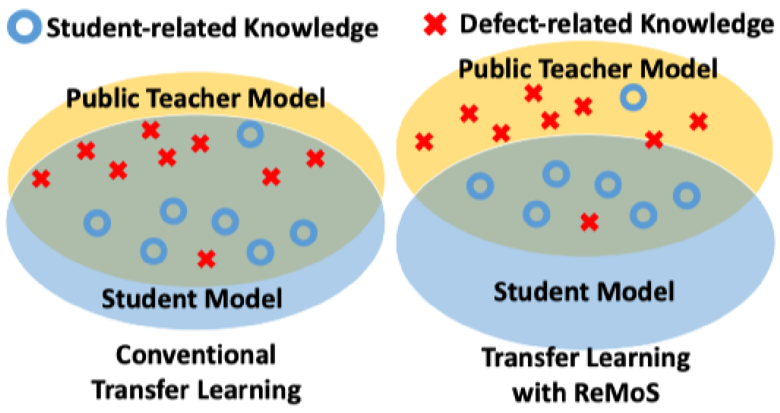
Method	DSADS dataset				PAMAP2 dataset				WESAD dataset				CAP dataset			
	mPre	mRec	mF1	Acc	mPre	mRec	mF1	Acc	mPre	mRec	mF1	Acc	mPre	mRec	mF1	Acc
Kernel PCA [11]	0.6184	0.6182	0.6183	0.6186	0.7236	0.6579	0.6892	0.5645	0.5496	0.5495	0.5495	0.5486	0.7603	0.5847	0.6611	0.5892
ABOD [50]	0.6880	0.6510	0.6690	0.6554	0.8653	0.9022	0.8834	0.8985	0.8782	0.8786	0.8784	0.8783	0.7867	0.6365	0.7037	0.6326
OCSVM [15]	0.7608	0.7277	0.7439	0.7312	0.7600	0.7204	0.7397	0.7679	0.6092	0.5631	0.5852	0.5518	0.9267	0.9259	0.9263	0.9257
HMM [51]	0.6959	0.6917	0.6937	0.6901	0.6950	0.6553	0.6745	0.5725	0.6123	0.6060	0.6097	0.6018	0.8238	0.8078	0.8157	0.8090
CNN-LSTM [52]	0.6845	0.6270	0.6545	0.6425	0.6680	0.5392	0.5968	0.6131	0.5883	0.5440	0.5653	0.5318	0.6159	0.5217	0.5649	0.5762
LSTM-AE [8]	0.8471	0.7729	0.8083	0.7852	0.8619	0.7997	0.8296	0.8426	0.2353	0.4762	0.3150	0.4599	0.7147	0.6253	0.6671	0.6286
MSCRED [9]	0.7540	0.5602	0.6428	0.6192	0.6997	0.7301	0.7146	0.7517	0.8850	0.8124	0.8471	0.8420	0.6410	0.5784	0.6081	0.5819
ConvLSTM-AE [16]	0.8164	0.6951	0.7509	0.7121	0.7359	0.7361	0.7360	0.7341	0.9733	0.9698	0.9716	0.9709	0.8150	0.8194	0.8172	0.8165
ConvLSTM-COMP [16]	0.8229	0.7379	0.7781	0.7518	0.8844	0.8842	0.8843	0.8844	0.9626	0.9629	0.9627	0.9619	0.8367	0.8377	0.8372	0.8394
BeatGAN [53]	0.9517	0.5663	0.7100	0.7818	0.7981	0.7420	0.7691	0.8369	0.7586	0.5000	0.6027	0.5172	0.5251	0.5002	0.5123	0.8437
MNAD-P [43]	0.5816	0.5783	0.5799	0.5721	0.8198	0.8176	0.8186	0.8135	0.7600	0.6938	0.7254	0.6849	0.7742	0.7489	0.7613	0.6960
MNAD-R [43]	0.8337	0.7694	0.8003	0.7811	0.8350	0.8355	0.8353	0.8334	0.7426	0.6677	0.7031	0.6579	0.8189	0.8235	0.8212	0.7871
GDN [54]	0.8706	0.8151	0.8419	0.8251	0.8129	0.8104	0.8116	0.8123	0.7520	0.5434	0.6309	0.5590	0.6831	0.6237	0.6520	0.6569
UODA [23]	0.8679	0.8281	0.8475	0.8365	0.8957	0.8513	0.8730	0.8823	0.7623	0.6314	0.6907	0.6191	0.7557	0.5124	0.6107	0.5173
AMSL (Ours)	**0.9407**	**0.9298**	**0.9352**	**0.9332**	**0.9788**	**0.9713**	**0.9750**	**0.9770**	**0.9953**	**0.9949**	**0.9951**	**0.9951**	**0.9771**	**0.9736**	**0.9753**	**0.9756**
Improvement	7.28%	10.17%	8.77%	9.67%	9.44%	8.71%	9.07%	9.26%	2.20%	2.51%	2.35%	2.42%	5.04%	4.77%	4.90%	4.99%

Y. Zhang, **J. Wang#**, Y. Chen, H. Yu, T. Qin. Adaptive Memory Networks with Self-supervised Learning for Unsupervised Anomaly Detection. **IEEE TKDE 2022**.

Safe transfer learning

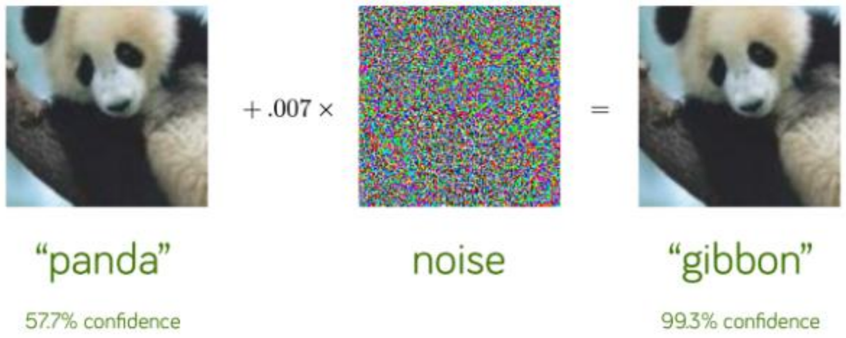
- Motivation
  - Software reuse is popular in software engineering → wide usage of pretrained models in ML
  - Malicious program/code cause damage → Fine-tuned models can inherit vulnerabilities from PT. models
  - The defects can easily be propagated from the teacher to the students, with the inheritance rate ranging from **52.58%** to **97.85%**
- Research question
  - Reducing the defect inheritance of PT model, while
  - Preserving its benefits (e.g., performance)
- Challenges
  - Attack is unknown
  - DNN models are diverse and lack of interpretability



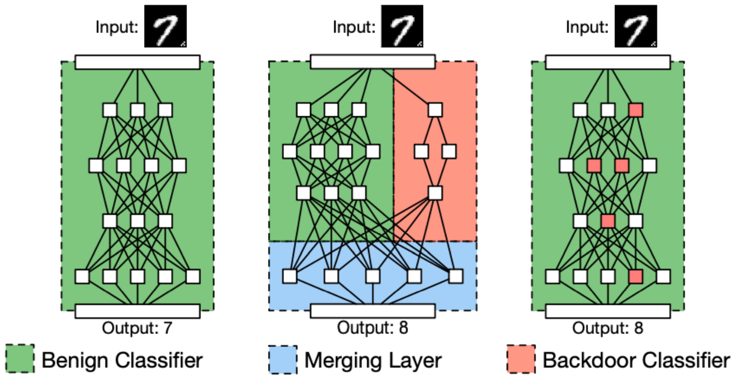


Background: DNN model attack

- DNN models are not safe:
  - Adversarial attack: Obtain adversarial examples using adversarial training to fool the model [1]



  - Backdoor attack: Hidden malicious logic is injected into the model purposely [2]



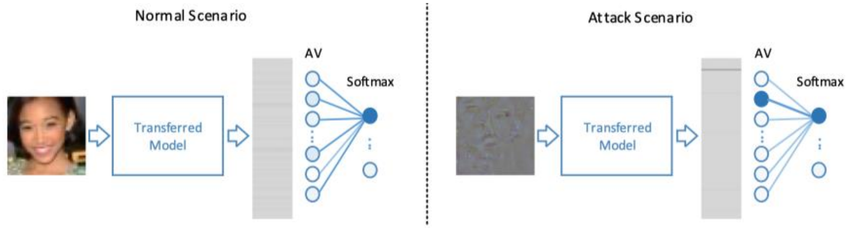
Task		Defect Type	Inheritance Rate
CV	Adversarial Vulnerability	Penultimate-Layer Guided [51]	58.01%
		Neuron-Coverage Guided [21, 48]	52.58%
	Backdoor	Latent Data Poisoning [62]	72.91%
NLP	Adversarial Vulerability	Greedy Word Swap [31]	64.86%
		Word Importance Ranking [29]	94.73%
	Backdoor	Data Poisoning [20]	96.72%
		Weight Poisoning [32]	97.85%

[1] https://towardsdatascience.com/adversarial-attacks-in-machine-learning-and-how-to-defend-against-them-a2beed95f49c

[2] Gu et al. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. arXiv 1708.06733.

Related work

- Can transfer learning models be attacked? Yes!
  - Given several target images and pretrained model, we can attack it by perturbing the input student images [Wang et al.'18]
  - Generate salient features, then perturb the inputs [Ji et al.'18]
  - Softmax layer is easy to attack [Rezaei et al.'20]



- How to defend?
  - Train from scratch: best defense, worst performance
  - Fine-tune: worst defense, best performance
  - Fix-after-transfer
    - fine-tune, then use defense: expensive and poor effectiveness due to small data
  - Fix-before-transfer: randomly initialize the student, then extract teacher knowledge
    - Renofeation [Chin et al.'21]: add dropout, feature regularization, and stochastic weight average; not end-to-end

- [Wang et al.'18] Wang B, Yao Y, Viswanath B, et al. With great training comes great vulnerability: Practical attacks against transfer learning. USENIX Security'18.
- [Ji et al.'18] Ji Y, Zhang X, Ji S, et al. Model-reuse attacks on deep learning systems. CCS'18.
- [Rezaei et al.'20] Rezaei S, Liu X. A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning. ICLR'20.
- [Chin et al.'21] Chin et al. Renofeation: A Simple Transfer Learning Method for Improved Adversarial Robustness. CVPR'21 workshop.

Our approach

- ReMoS: Relevant Model Slicing
  - Given a DNN model M and a target domain dataset D, ReMoS is to compute a <u>subset</u> of model <u>weights</u> that are more relevant (bounded by a threshold) to the inference of samples in D and less relevant to the samples outside D.
  - Relevant slicing: from traditional software engineering
  - Assumption:
    - Pre-trained model as a white-box (i.e., architecture and weights)
    - Target dataset for student task
  - Formulation:

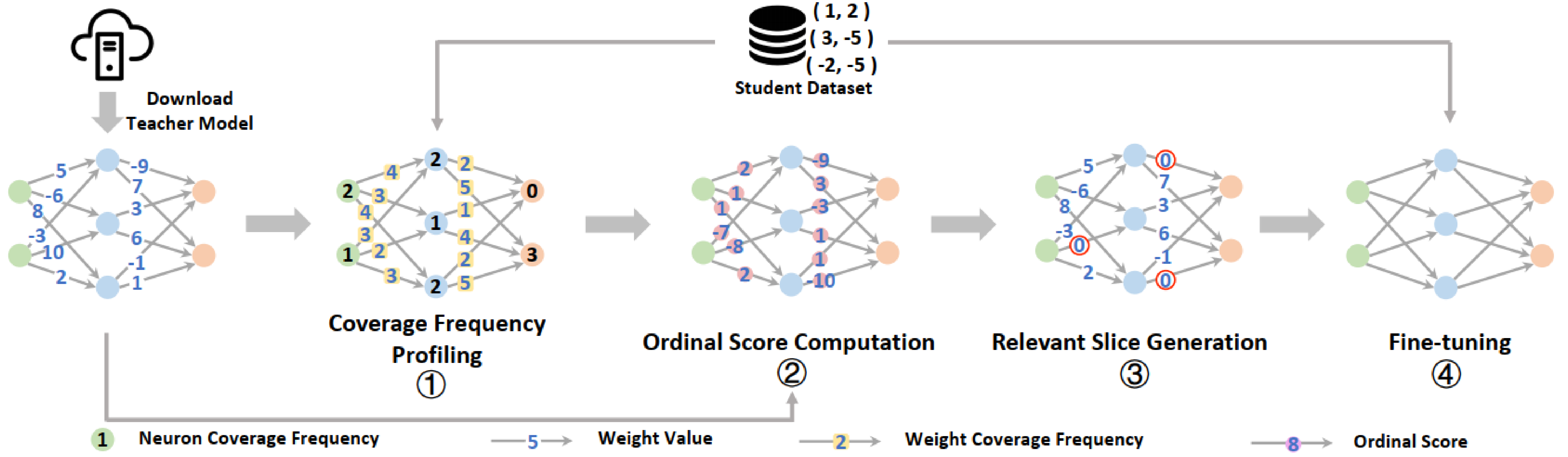
$$\max_{\mathbf{w} \subset \mathbf{w}^T} \sum_{(x,y) \in D^S} \mathbb{I}[f(x; T(\mathbf{w})) = y] + \mathbb{I}[f(\tilde{x}; T(\mathbf{w})) = y)]$$

```
read( a, b )
int x=0, y=0
x = a + 1
y = b + 1
int w = 0
if x > 3 then
    if y > -3 then
        w = w / b
    endif
endif
if y > 5 then
    w = w + 1
endif
write( w )
```

```
read( a, b )
int x=0, y=0
x = a + 1

int w = 0
if x > 3 then



endif
if y > 5 then
    w = w + 1
endif
write( w )
```

Our approach

- ReMoS
  - Coverage frequency profiling: compute coverage frequency of each weight → support of student task
  - Ordinal score computation: compute score of each weight based on teacher weight and coverage frequency
  - Relevant slice generation: identify the relevant weights
  - Fine-tuning: vanilla fine-tune



Our approach

- Coverage frequency profiling
  - Neuron coverage: find the neuron whose activation value is large than a threshold α

$$\begin{aligned}\mathrm{Cov}(x) &= \mathrm{Cov}(\ \mathrm{Run}(M, x)\ ) = \mathrm{Cov}(\{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_K\}) \\ &= \{\mathbf{v}_i | \mathbf{v}_i = \mathbb{I}[\mathbf{h}_i > \alpha]\}.\end{aligned}$$

  - On dataset D^S

$$\mathrm{Cov}(D^S) = \{\sum_{x \in D^S} \mathrm{Cov}(x)_i | i = 2, 3, \cdots, K\}$$

  - Weight coverage:
    - Sum of the neuron coverage frequency of two neurons that this weight connects

$$\mathrm{CovW}(D^S)_{k,i,j} = \mathrm{Cov}(D^S)_{k-1,i} + \mathrm{Cov}(D^S)_{k,j}$$

- Ordinal score computation
  - Formulation:

$$\mathbf{w}^{ReMoS} = \arg\max_{\mathbf{w} \subset \mathbf{w}^T} ACC(T(\mathbf{w}), D^S) - \sum_{w \in \mathbf{w}} |w|$$

  - To unify the value range

$$ord\_mag_{k,i,j} = rank(|w_{k,i,j}|),$$

$$ord\_cov_{k,i,j} = rank(\mathrm{CovW}(D^S)_{k,i,j})$$

$$ord_{k,i,j} = ord\_cov_{k,i,j} - ord\_mag_{k,i,j}$$

Our approach

- Relevant slice generation
  - Identify which weights should be included based on ordinal scores (t_θ is the slice size, not value size)

$$slice(D^S) = \{w_{k,i,j} | ord_{k,i,j} > t_\theta\}$$

- Fine-tune
  - Traditional fine-tune

$$\begin{cases} \bullet \text{ Weights inside } slice(D^S)\text{: Fine-tune from teacher} \\ \bullet \text{ Weights outside } slice(D^S)\text{: Random initialization} \end{cases}$$

- Advantage
  - Less computation overhead: only *forward-pass* using the student dataset once
  - No need to know the student task in advance
  - Agnostic to DNN architectures

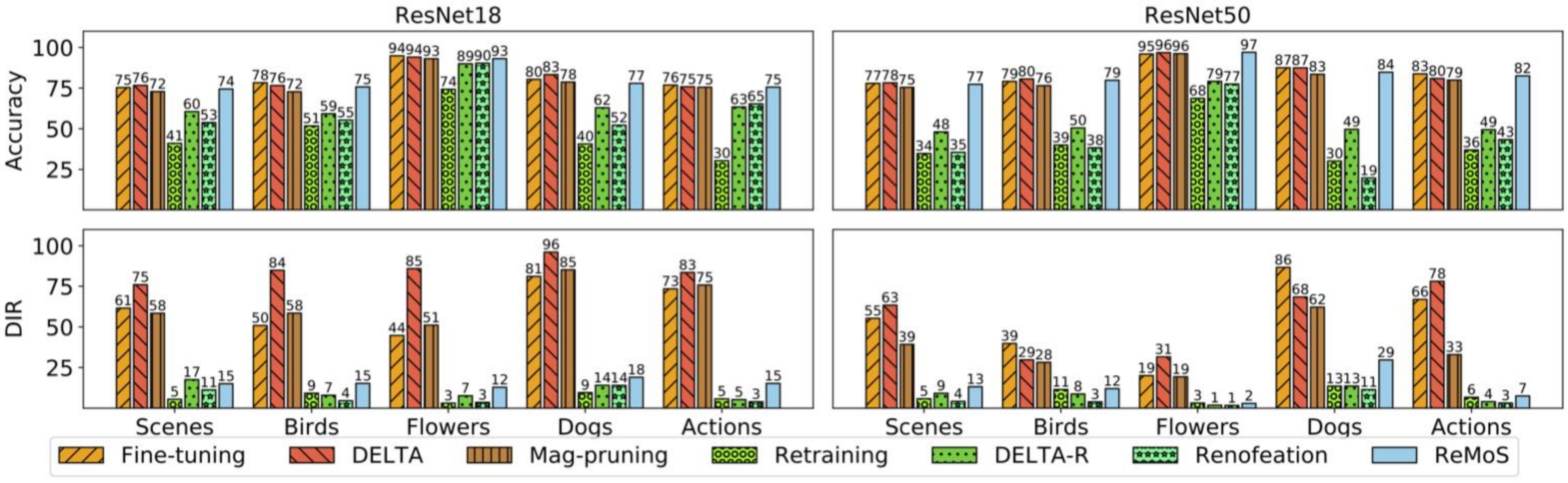
Experiments

- Research questions to be answered
  - Defect mitigation effectiveness
    - Performance sacrifice vs. defect mitigation
  - Generalizability
    - Generalize to different tasks (CV and NLP)
  - Efficiency
    - How much time does it cost compared to traditional transfer learning
  - Interpretability
    - Why is ReMoS effective? What does the student model look like?

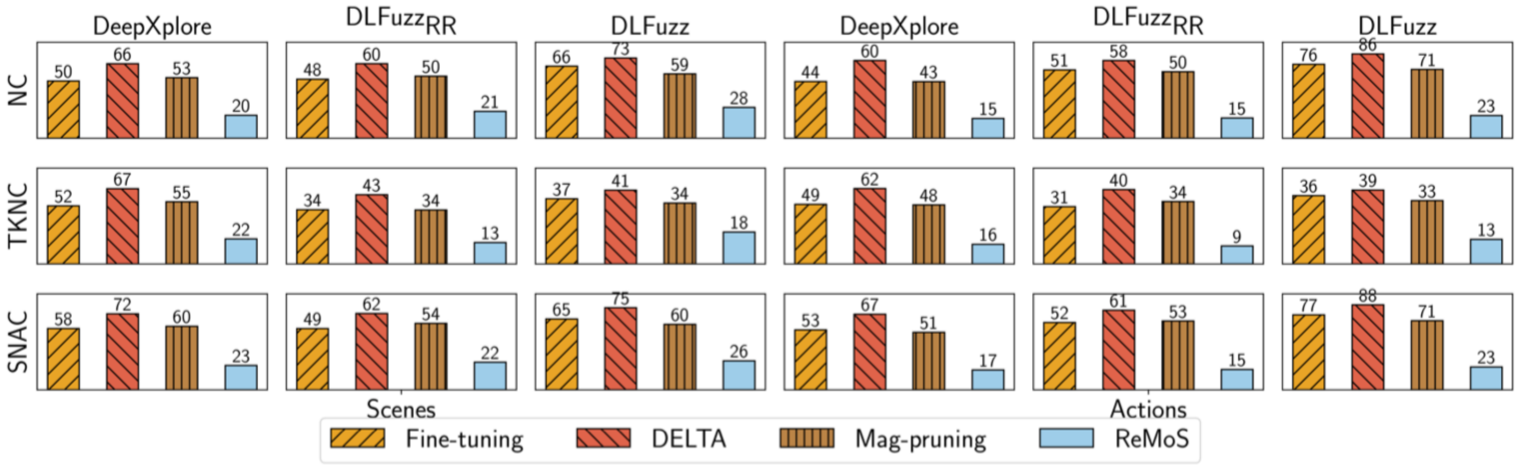
Effectiveness

- CV results
  - Better ACC
  - Lower DIR

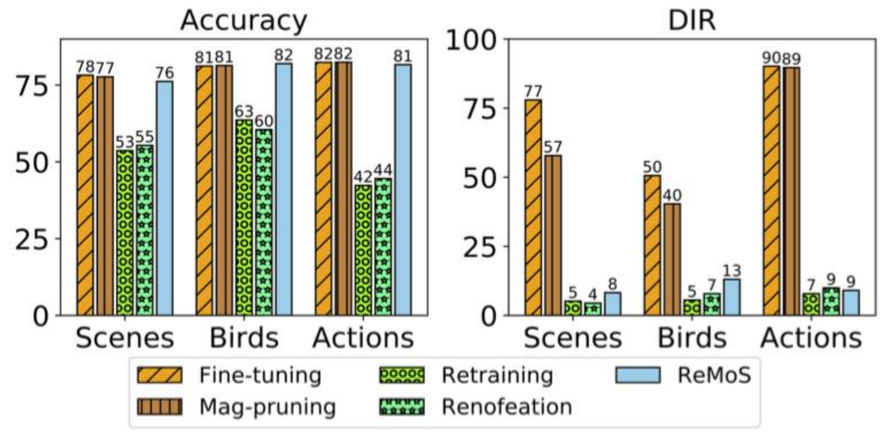
ReMoS only sacrifices less than 2% model accuracy and reduces over 75% inherited defects than the conventional fine-tuning



Adversarial attack



Neuron coverage attack



Backdoor attack

Effectiveness

- NLP results
  - Defect is more severe in NLP
  - Ours is significantly better
  - ReMoS reduces 50% to 61% DIR, only sacrificing 3% acc at most

Adversarial attack

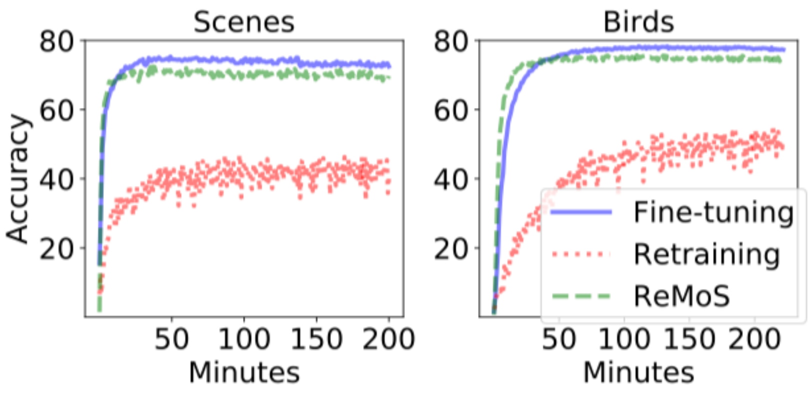
Model	Dataset		Fine-tune	Mag-prune	ReMoS
BERT	SST-2	*ACC*	92.20	93.43	92.03
		DIR	85.30	67.70	62.23
	QNLI	*ACC*	89.42	88.84	88.45
		DIR	74.94	62.11	45.16
RoBERTa	SST-2	*ACC*	94.40	94.06	92.08
		DIR	84.94	58.48	49.46
	QNLI	*ACC*	90.25	91.09	89.60
		DIR	74.02	56.61	36.36
Average Relative Value		$rACC_m$	-	1.00	0.99
		$rDIR_m$	-	0.76	0.60

Backdoor attack

Model		Dataset	Data Poisoning						Weight Poisoning					
			Fine-tune		Mag-prune		ReMoS		Fine-tune		Mag-prune		ReMoS	
			ACC	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*
BERT	FDK	SST-2 to SST-2	94.19	100.00	93.70	100.00	91.27	39.09	93.37	100.00	93.19	98.93	90.92	29.82
		IMDB to IMDB	90.60	93.52	89.54	95.24	85.53	61.73	89.05	96.53	88.76	92.05	87.00	37.72
	DS	SST-2 to IMDB	92.11	99.88	92.27	100.00	90.04	74.67	91.85	100.00	90.82	99.53	87.42	61.48
		IMDB to SST-2	93.52	88.15	92.65	85.26	91.15	27.71	93.85	93.93	93.57	91.21	91.94	21.55
RoBERTa	FDK	SST-2 to SST-2	92.70	100.00	92.35	100.00	91.17	29.82	92.29	100.00	92.44	100.00	90.70	24.94
		IMDB to IMDB	87.96	96.11	88.24	96.15	85.74	70.19	89.34	96.15	89.48	96.09	86.34	85.91
	DS	SST-2 to IMDB	90.53	100.00	91.26	100.00	90.32	24.14	91.67	100.00	91.16	100.00	88.71	30.83
		IMDB to SST-2	93.21	96.17	92.46	96.17	92.17	61.26	92.80	96.22	92.58	96.02	89.95	18.07
Average Relative Value			-	-	0.99	0.99	0.97	0.50	-	-	0.99	0.98	0.97	0.39

Efficiency

- Efficiency
  - Same speed as fine-tuning
  - Better performance than training from scratch



- Interpretability
  - Compute w^S/w^T: ReMoS reduces the number of weights that require large-range adjustment
  - Distribution of magnitude: more weights with higher magnitude are excluded, which is intuitive

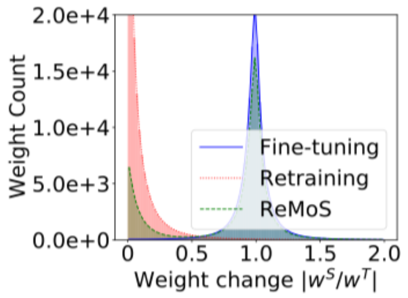


Figure 9: The distribution of weight changes during training.

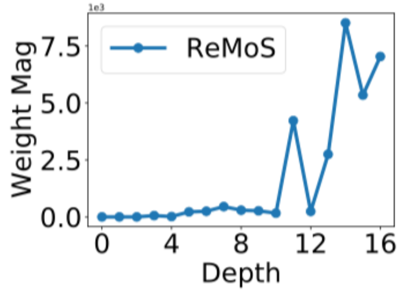


Figure 10: The magnitude of weights excluded by ReMoS in each layer.

- Zhang et al. ReMoS: reducing defect inheritance in transfer learning by relevant model slicing. ICSE 2022.

Application to federated healthcare

- Personalization in federated learning
  - <u>Non-i.i.d</u>: Data from different client have totally different distributions
    - Hobbies, lifestyles, body shapes, devices…
  - Preserve the <u>specificity</u> of each client, while leveraging the commonality of all other clients

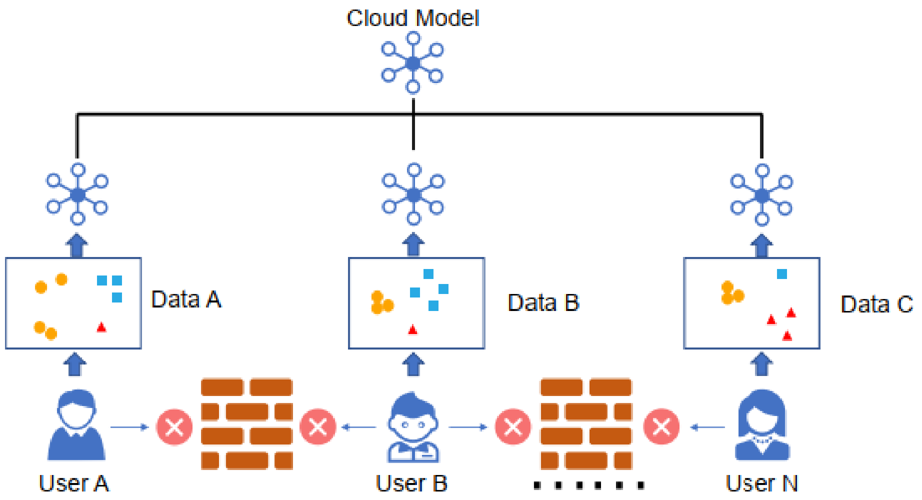
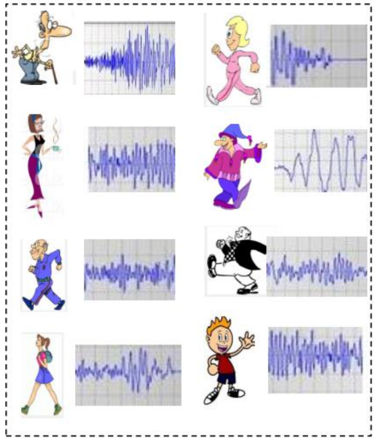
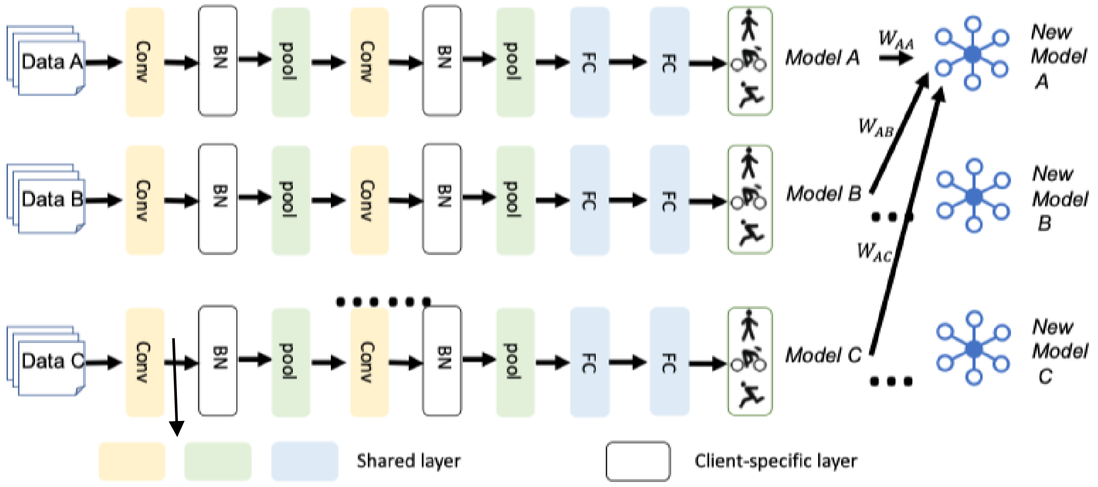


Figure 1: Data islanding and personalization in healthcare.



Similarity-guided aggregation

- How to compute similarity between clients?
  - Motivation: **batch norm (BN)** layers contains sufficient statistics of the data
  - We can use BN's statistics after local data inputs to compute similarity
- AdaFed [2]
  - Adaptive batch norm for federated learning



Train a server model and distribute it to clients

↓

Client computes local statistics and update local models

↓

Server obtains client similarity $\boldsymbol{W}$ to guide aggregation

↓

Aggregate models to the server

[2] Chen Y, Lu W, Wang J, et al. Federated Learning with Adaptive Batchnorm for Personalized Healthcare[J]. arXiv preprint arXiv:2112.00734, 2021.

Obtain similarity matrix

- BN statistics at each layer: $(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i) = [(\boldsymbol{\mu}^{i,1}, \boldsymbol{\sigma}^{i,1}), (\boldsymbol{\mu}^{i,2}, \boldsymbol{\sigma}^{i,2}), \cdots, (\boldsymbol{\mu}^{i,L}, \boldsymbol{\sigma}^{i,L})]$
- Similarity between two clients:

$$W_2^2(\mathcal{N}(\boldsymbol{\mu}^{i,l}, \boldsymbol{\sigma}^{i,l}), \mathcal{N}(\boldsymbol{\mu}^{j,l}, \boldsymbol{\sigma}^{j,l})) = ||\boldsymbol{\mu}^{i,l} - \boldsymbol{\mu}^{j,l}||^2 + tr(\boldsymbol{\sigma}^{i,l} + \boldsymbol{\sigma}^{j,l} - 2((\boldsymbol{\sigma}^{i,l})^{1/2}\boldsymbol{\sigma}^{j,l}(\boldsymbol{\sigma}^{i,l})^{1/2})^{1/2})$$

- Then we get the distance:

$$d_{i,j} = \sum_{l=1}^{L} W_2(\mathcal{N}(\boldsymbol{\mu}^{i,l}, \boldsymbol{\sigma}^{i,l}), \mathcal{N}(\boldsymbol{\mu}^{j,l}, \boldsymbol{\sigma}^{j,l})) = \sum_{l=1}^{L}(||\boldsymbol{\mu}^{i,l} - \boldsymbol{\mu}^{j,l}||^2 + ||\sqrt{\mathbf{r}^{i,l}} - \sqrt{\mathbf{r}^{j,l}}||_2^2)^{1/2}$$

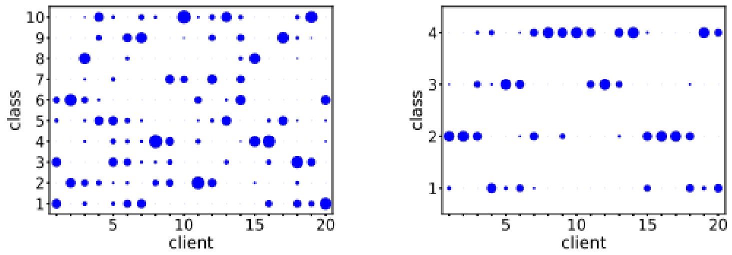
- Model aggregation:

$$w_{i,j} = \begin{cases} \lambda, & i = j, \\ (1-\lambda) \times \hat{w}_{i,j}, & i \neq j. \end{cases} \qquad \begin{cases} \boldsymbol{\phi}_i^{t+1} = & \boldsymbol{\phi}_i^{t*} \\ \boldsymbol{\psi}_i^{t+1} = & \sum_{j=1}^{N} w_{ij}\boldsymbol{\psi}_j^{t*} \end{cases}$$

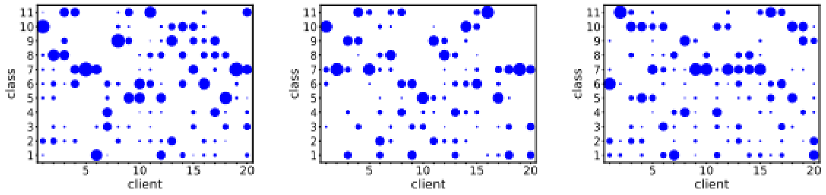
Experiments of AdaFed

- Different datasets from healthcare

Dataset	Type	#Class	#Sample
PAMAP	Sensor-based time series	18	3,850,505
COVID-19	Image	4	9,208
OrganAMNIST	Image	11	58,850
OrganCMNIST	Image	11	23,660
OrganSMNIST	Image	11	25,221



(a) PAMAP (b) COVID-19



(c) OrganAMNIST (d) OrganCMNIST (e) OrganSMNIST

- Comparison methods
  - Base: Each client uses local data to train local models.
  - FedAvg (McMahan et al. 2017): The cloud aggregate all client models without any particular operations for non-iid data.
  - FedBN (Li et al. 2021): Each client preserves the local batch normalization.
  - FedProx (Li et al. 2018): Allow partial information aggregation and add a proximal term to FedAvg.
  - FedPer (Arivazhagan et al. 2019): Each client preserves some local layers.

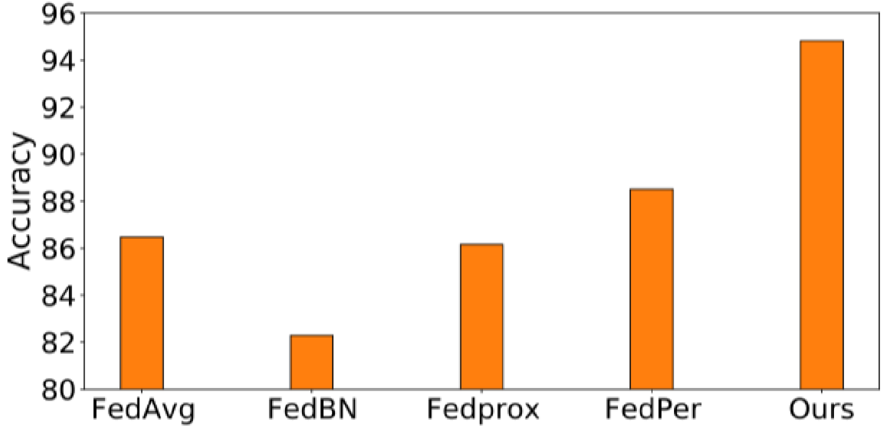
Results

- PAMAP2 (time-series)
  - 10%+ better than FedBN (ICLR-21)

Client	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	avg
Base	**92.86**	17.68	**100.00**	**83.52**	18.78	77.66	**95.05**	17.58	**92.39**	**93.37**	29.12	**84.78**	**98.90**	24.18	**98.91**	**98.90**	41.44	**93.62**	**85.71**	37.02	69.07
FedAvg	60.27	62.36	50.56	73.98	74.27	62.90	64.03	87.78	74.49	64.71	65.24	63.35	68.33	64.79	63.12	85.26	66.21	59.64	67.87	72.46	67.58
FedBN	60.72	62.59	50.34	73.53	74.72	62.44	62.90	88.24	74.27	64.48	65.69	62.90	68.33	65.24	62.44	85.94	65.99	59.64	68.10	72.69	67.56
FedProx	60.50	62.36	50.34	73.98	73.81	61.76	63.57	87.78	74.27	64.71	66.37	63.12	68.33	65.69	62.44	85.49	66.21	59.41	67.87	72.46	67.52
FedPer	48.31	**97.51**	61.40	47.29	58.47	23.98	49.55	91.86	51.24	77.60	89.16	57.92	42.53	49.44	58.60	86.62	77.32	52.38	73.08	**97.52**	64.59
WFedBN	77.20	77.55	77.43	79.64	**81.94**	79.86	86.20	**95.02**	85.33	69.23	**91.42**	79.41	74.43	69.75	81.67	94.10	**82.77**	75.74	77.15	86.91	**81.14**
d-WFedBN	64.33	77.55	78.33	77.38	79.91	**80.77**	85.52	92.53	86.23	69.23	87.58	80.09	74.66	**70.43**	83.71	93.88	80.50	74.60	78.73	87.13	80.16
f-WFedBN	64.11	77.78	69.53	79.86	77.88	74.43	84.62	93.67	74.04	81.00	79.91	71.95	74.21	62.98	78.05	89.57	79.59	68.71	71.95	87.81	77.08

Table 1: Activity recognition results of 20 clients on PAMAP. Bold means the best result while underline means the second best result.

- COVID-19 diagnosis
  - 4%+ better than FedPer



Related materials

- Book
  - 迁移学习导论
  - http://jd92.wang/tlbook
- Code library
  - Github repo: the most popular transfer learning repo on Github
  - Papers, codes, datasets, applications…

transferlearning Public

Transfer learning / domain adaptation / domain generalization / multi-task learning etc. Papers, codes, datasets, applications, tutorials.-迁移学习

Python 9.1k 3.2k

transferlearning-tutorial Public

《迁移学习简明手册》LaTex源码

TeX 2.2k 463

TorchSSL/TorchSSL Public

A PyTorch-based library for semi-supervised learning (NeurIPS'21)

Python 691 100

Deep-learning-activity-recognition Public

A tutorial for using deep learning for activity recognition (Pytorch and Tensorflow)

Python 185 71

Conclusions

- We are pushing the frontier of transfer learning in 3 aspects:
  - Low-resource learning: FlexMatch for efficient and effective learning (NeurIPS’21)
    - Design better framework for SSL?
  - Domain generalization: AdaRNN for generalized time series learning (CIKM’21, TKDE’22)
    - Can the method be one-stage?
  - Safety: Safe transfer learning and personalized federated learning (ICSE’22)
    - More formal way to do this
- Next
  - Develop theories and algorithms for domain generalization
  - Safe transfer learning

Thanks for your attention
Discussions and collaborations are welcomed!
Jindong.wang@microsoft.com

王晋东不在家

王晋东不在家

微信扫描二维码，关注我的公众号