Learning a Low-Rank Feature Representation: Achieving Better Trade-Off Between Stability and Plasticity in Continual Learning

Zhenrong Liu[1,2], Yang Li[3], Yi Gong[1], and Yik-Chung Wu[2]

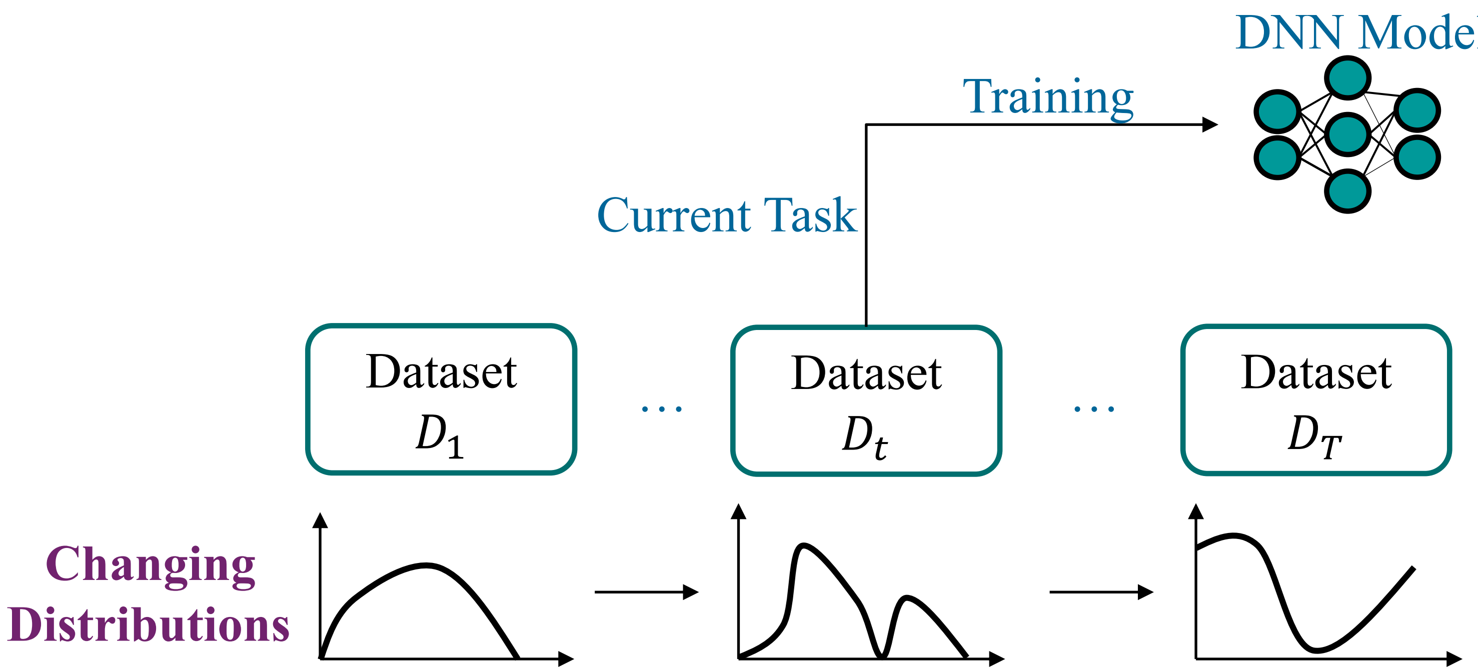
Presenter: Yang Li[3]

Apr. 2024

[1] Southern University of Science and Technology, Shenzhen, China
[2] The University of Hong Kong, Hong Kong
[3] Shenzhen Research Institute of Big Data, Shenzhen, China

What is continual learning?

- Continual learning is the ability of a model to learn from **a stream of data**
- Each batch may **differ significantly** from the previous one
- A DNN model must **adapt to new** subjects, one after another, **without forgetting the old** ones (also, what we aim to achieve with our approach)



Why we need continual learning?

- The need for continual learning arises because, in real-world applications, data **evolves over time**
    - For instance, consider a facial recognition system that must **adapt to new looks without forgetting old ones**

- Traditional DNN models struggle with this, as retraining them on the **entire dataset periodically** is not only **resource-intensive** but also **impractical**

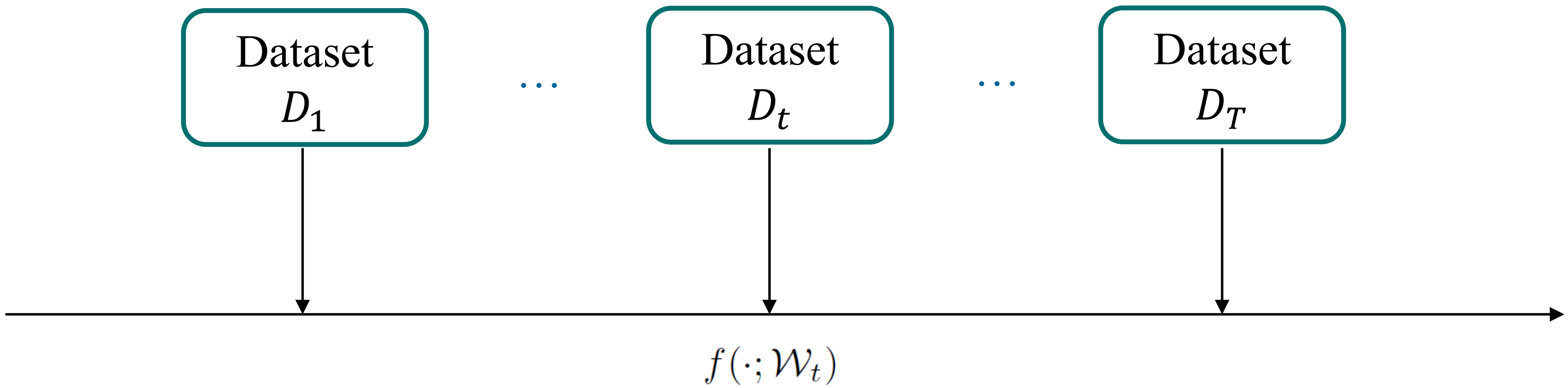
Two core concepts in continual learning: stability and plasticity

- **Stability:** the model's ability to **retain** what it has learned from previous tasks
  - Without stability, a DNN is like a student who forgets everything learned in the previous term when a new term starts (this phenomenon is also known as **catastrophic forgetting**)
- **Plasticity:** the model's capacity to learn **new** tasks and **adapt** to changes

Understanding and balancing these two aspects is key to our research, as it allows us to create models that are **both knowledgeable of the past and adaptable to the future**.

How to achieve stability?

Motivation: after the ***t***-th training episode, our DNN, denoted by $f(\cdot;\mathcal{W}_t)$ is expected to perform well on **all the previous datasets** $\mathcal{D}_1$ to $\mathcal{D}_{t-1}$



This can be guaranteed if

$$f(\mathbf{h}_{i,q};\mathcal{W}_t) = f(\mathbf{h}_{i,q};\mathcal{W}_{t-1}),$$
$$\forall \mathbf{h}_{i,q} \in \mathcal{D}_q, \ \forall q = 1, 2, ..., t-1,$$

If this condition is satisfied, it ensures that the DNN **retains** the knowledge from previous tasks **without catastrophic forgetting**.

How to guarantee the stability condition?

Examine a single-layer MLP, which is a basic neural network model

$f(\mathbf{x}; \mathbf{W}) = \mathbf{W}^{\mathrm{T}}\mathbf{x}$

- After the 1-sth training episode, the output of an input $\mathbf{x}_1$ is $\mathbf{W}_1^{\mathrm{T}}\mathbf{x}_1$
- After the 2-nd training episode, the parameter becomes $\mathbf{W}_2 = \mathbf{W}_1 + \Delta\mathbf{W}_2$
- To maintain the output for $\mathbf{x}_1$ unchanged, we require that

$$(\mathbf{W}_1 + \Delta\mathbf{W}_2)^{\mathrm{T}}\, \mathbf{x}_1 = \mathbf{W}_1^{\mathrm{T}}\mathbf{x}_1$$

This leads to the condition: $\Delta\mathbf{W}_2^{\mathrm{T}}\mathbf{x}_1 = \mathbf{0}$

How to guarantee the stability condition?

- Recall that we need the following condition to guarantee the output of $\mathbf{x}_1$ unchanged:

$$\Delta \mathbf{W}_2^{\mathrm{T}} \mathbf{x}_1 = \mathbf{0}$$

- When dealing with **multiple samples** from datasets $\mathcal{D}_1$ to $\mathcal{D}_{t-1}$, we can represent the data as a **feature representation** matrix $\mathbf{F}_{t-1}$, where each column corresponds to a data sample

- To maintain the output across all data, the following condition must be met:

$$\mathbf{F}_{t-1}^{\mathrm{T}} \Delta \mathbf{W}_t = \mathbf{0}$$

How to guarantee the stability condition?

- Given the typically **vast number** of data samples, storing the entire $\mathbf{F}_{t-1}$ matrix is not feasible. To alleviate the memory load, we store

$$\bar{\mathbf{F}}_{t-1} = \mathbf{F}_{t-1}\mathbf{F}_{t-1}^{\mathrm{T}}$$

- This leads us to an **equivalent** condition:

$$\bar{\mathbf{F}}_{t-1}\Delta\mathbf{W}_t = \mathbf{0}$$

- Extend this condition to the case of **multi-layer** DNNs
  - For the l-th layer of the DNN, the condition becomes: $\bar{\mathbf{F}}_{t-1}^{l}\Delta\mathbf{W}_t^{l} = \mathbf{0}$

This ensures that the updates to the parameters do not affect the output for previous data **across all layers** of the DNN.

How to achieve plasticity?

- Plasticity is achieved when the parameter update $\Delta\mathbf{W}_t^l$ aligns with the **negative gradient** of the loss with respect to $\mathbf{W}_t^l$, denoted as $\mathbf{G}_t^l$

- Mathematically, this is when their inner product is **greater than** zero:

$$< \Delta\mathbf{W}_t^l, \mathbf{G}_t^l > \geq 0$$

- This condition means that $\Delta\mathbf{W}_t^l$ will decrease the training loss, allowing the DNN to integrate **new knowledge from the latest data**

Trade-off between stability and plasticity

- To maintain **stability**, we restrict the parameter update within **the null space** of the covariance matrix $\bar{\mathbf{F}}_{t-1}^{l}$, i.e., $\bar{\mathbf{F}}_{t-1}^{l}\Delta\mathbf{W}_{t}^{l} = \mathbf{0}$
- However, as the number of data increases, the dimension of the null space **decreases** due to the **increasing** rank of $\bar{\mathbf{F}}_{t-1}^{l}$

$$\mathrm{NULL}\left(\bar{\mathbf{F}}_{t-1}^{l}\right) = \underbrace{a_l}_{\text{row/column number of } \bar{\mathbf{F}}_{t-1}^{l}} - \mathrm{RANK}(\bar{\mathbf{F}}_{t-1}^{l})$$

- This **shrinking** null space limits the options for $\Delta\mathbf{W}_{t}^{l}$ that can satisfy the **plasticity** condition, i.e., $< \Delta\mathbf{W}_{t}^{l}, \mathbf{G}_{t}^{l} > \geq 0$

By observing and understanding this **trade-off**, we can design DNNs that are both **robust to forgetting** and **adaptable to new information**.

Proposed low-rank covariance approach

- The key to improving the trade-off between stability and plasticity is to **decrease the rank** of the feature covariance matrix $\bar{\mathbf{F}}_{t-1}^{l}$

$$\mathrm{NULL}\left(\bar{\mathbf{F}}_{t-1}^{l}\right) = a_l - \mathrm{RANK}(\bar{\mathbf{F}}_{t-1}^{l})$$

- However, $\bar{\mathbf{F}}_{t-1}^{l}$ is generated by the DNN itself

- To construct a low-rank $\bar{\mathbf{F}}_{t-1}^{l}$, we can construct a low-rank feature representation matrix $\mathbf{F}_{q}^{l}$ at each training episode q

$$\bar{\mathbf{F}}_{t-1}^{l} = \sum_{q=1}^{t-1} \mathbf{F}_{q}^{l} {\mathbf{F}_{q}^{l}}^{\mathrm{T}}$$

Feature representation matrix corresponding to the data in $\mathcal{D}_q$

Proposed low-rank covariance approach

- To construct a **low-rank feature representation $\mathbf{F}_t^l$** during each training episode t, the proposed approach involves **two stages**

Pretraining → Training

Aims to induce **sparsity** in the weights of each layer

$$\bar{\mathcal{W}}_t = \arg\min_{\mathcal{W}} \sum_{\mathbf{h}_{i,t}\in\mathcal{D}_t} \ell\left(\mathbf{h}_{i,t}, f(\mathbf{h}_{i,t};\mathcal{W})\right) + \mu \sum_{l=1}^{L}\sum_{j=1}^{b_l} \|\mathbf{w}_j^l\|_2,$$
$$\text{s.t.}\quad \bar{\mathbf{F}}_{t-1}^l \Delta \mathbf{W}_{t,s}^l = \mathbf{0}, \quad \forall s = 1,2,\ldots,S,\ \forall l = 1,2,\ldots,L$$

induces the columns of the weight matrix to become sparse

Prunes the DNN by setting the smaller weights to zero and **fine-tuning** the **larger** weights

$$\mathcal{W}_t = \arg\min_{\mathcal{W}} \sum_{\mathbf{h}_{i,t}\in\mathcal{D}_t} \ell\left(\mathbf{h}_{i,t}, f(\mathbf{h}_{i,t};\mathcal{W})\right)$$
$$\text{s.t.}\quad \bar{\mathbf{F}}_{t-1}^l \Delta \mathbf{W}_{t,s}^l = \mathbf{0}, \quad \forall s = 1,2,\ldots,S,\ \forall l = 1,2,\ldots,L$$
$$\mathbf{w}_j^l = \mathbf{0},\ \forall j \in \{1,2,\ldots,b_l\} \text{ such that } \|\bar{\mathbf{w}}_j^l\|_2 < \lambda_l,$$

It is important to note that during both stages, weight parameter updates are conducted by projecting the gradient **onto the null space** of $\bar{\mathbf{F}}_{t-1}^l$, to ensure the **stability condition**.

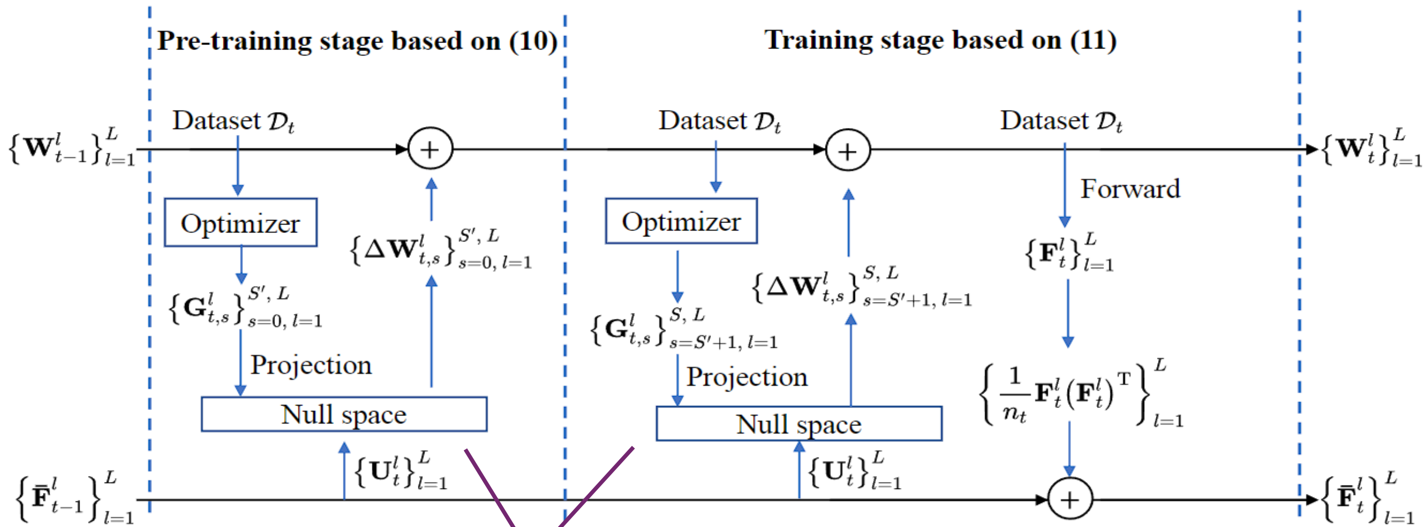
Proposed low-rank covariance approach

- Workflow of the *t*-th training episode

Generate **sparse** weights in each layer

The **larger** weights are further **fine-tuned**

Input the dataset $\boldsymbol{D_t}$ into the DNN to derive the **low-rank feature representation** and update $\bar{\mathbf{F}}_t^l$



$$\Delta\mathbf{W}_{t,s}^l = \mathbf{U}_t^l\left(\mathbf{U}_t^l\right)^{\mathrm{T}}\mathbf{G}_{t,s}^l$$

whose columns are the eigenvectors associated with the **zero singular values** of $\bar{\mathbf{F}}_{t-1}^l$

Simulation setup

- Dataset: 10-split-CIFAR-100, 20-split-CIFAR-100 and 25-split-TinyImageNet
- Architecture: Resnet-18 (Pre-Activation)
- Batch size: 32 (CIFAR), 16 (Tiny ImageNet)
- Learning Rate & Epoch: 5×10^{-5} , halving it at epochs 30 and 60 over 80 total epochs.
- Penalty Parameter: $\mu = 0.1$.
- Pruning Ratio: disable 50% neurons.

Our code has been made open source on GitHub (https://github.com/Dacaidi/LRFR)

If details are not fully covered in this paper and presentation, you can download our **code** to understand the **specifics**.

Simulation results

- In our simulation analysis, we measure the performance using two key metrics:
  - **ACC** (Average Classification Accuracy): reflects the **overall accuracy** of the model across all tasks
  - **BWT** (Backward Transfer): indicates the model's ability to **retain knowledge** from previous tasks.

    A **higher BWT** value means **greater stability** and **less forgetting**

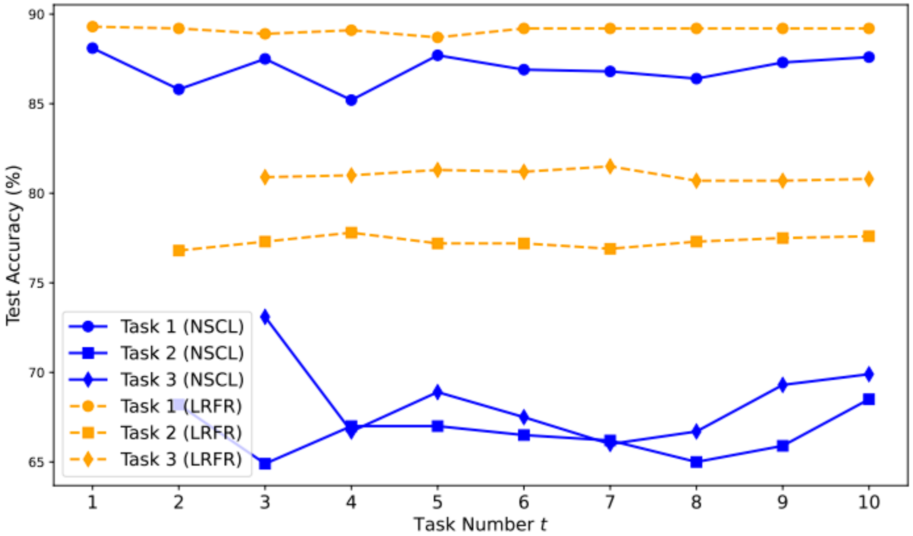
Method	10-CIFAR	20-CIFAR	25-Tiny
	ACC (BWT)	ACC (BWT)	ACC (BWT)
EWC [6]	70.77 (-2.83)	71.66 (-3.72)	52.33 (-6.17)
MAS [15]	66.93(-4.03)	63.84(-6.29)	47.96 (-7.04)
GEM [7]	49.48 (2.77)	68.89 (-1.2)	N/A
A-GEM [10]	49.57 (-1.13)	61.91 (-6.88)	53.32 (-7.68)
MEGA [16]	54.17 (-2.19)	64.98 (-5.13)	57.12 (-5.90)
OWM [17]	68.89 (-1.88)	68.47 (-3.37)	49.98 (-3.64)
GPM [8]	73.66 (-2.20)	75.20 (-7.58)	58.96 (-6.96)
NSCL[9]	73.77 (-1.6)	75.95 (-3.66)	58.28 (-6.05)
AdNS [11]	77.21(-2.32)	77.33(-3.25)	59.77(-4.58)
Proposed → LRFR	**81.30** (0.11)	**82.95** (-1.37)	**62.28** (-4.05)

The proposed approach **outperforms** the benchmarks in **both metrics** across various datasets, which demonstrates its effectiveness in achieving a **balance between stability and plasticity**.

Simulation results

- **NSCL** (Null Space Continual Learning): This is a vanilla null space projection approach **without using the low-rank feature representation** as in our work

Test accuracy of the two approaches on the **first three tasks** in 10-split-CIFAR dataset



The proposed approach achieves **much higher test accuracy** and **more stable**, which demonstrates the effectiveness of the proposed **low-rank feature representation** compared with the vanilla null space projection approach.

Conclusions

- We began by introducing the two core concepts of **stability** and **plasticity** in the realm of continual learning.
- We explored the **inherent trade-off** between these concepts, highlighting the challenges they present in model training.
- In response to this trade-off, we proposed our novel **Low-Rank Feature Representation (LRFR)** approach.
- Through simulations, we demonstrated that the proposed approach outperforms state-of-the-art approaches with **superior average accuracy** and **robustness against forgetting**.

Thank you very much！

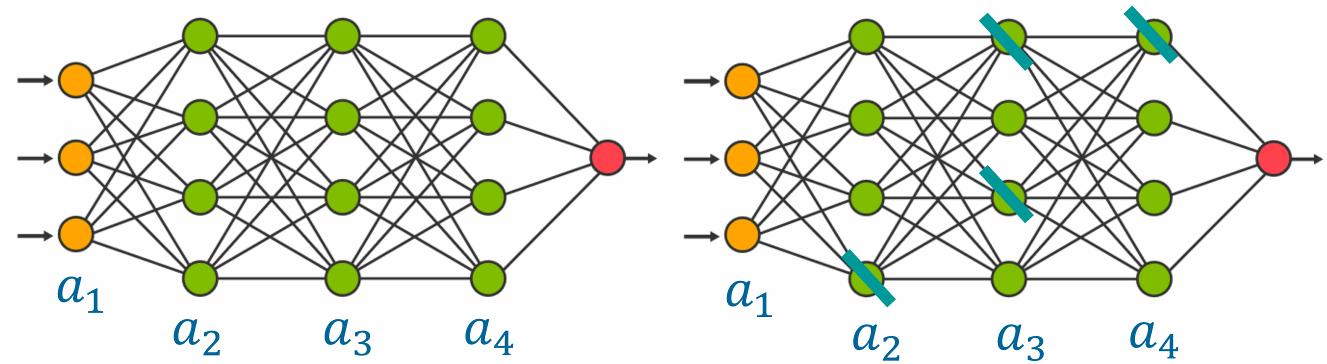
Continual learning methods fall into the following four categories:

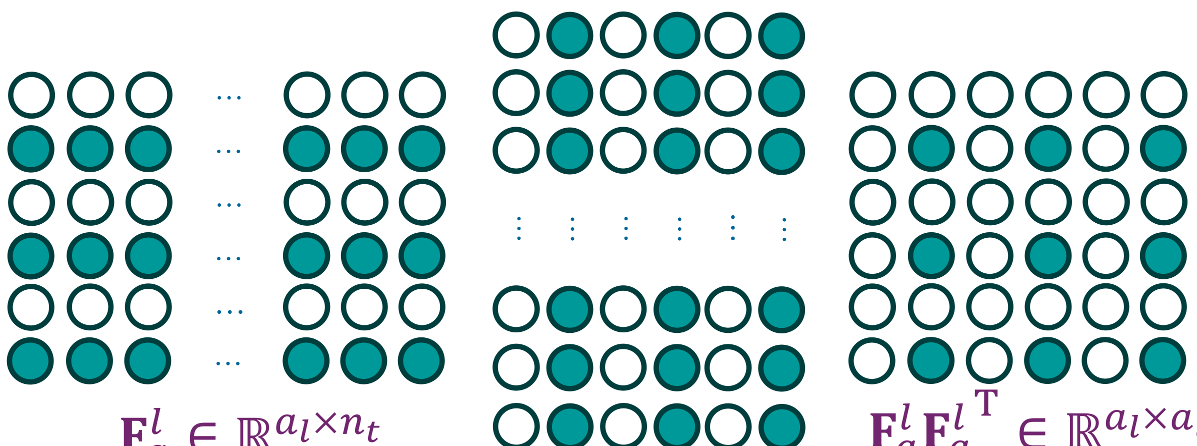
Four elements of DNN training:

Data	Model	Loss	Optimizer
↑	↑	↑	↑
replay-based	architecture-based	regularization-based	algorithm-based

Low Rank $\bar{\mathbf{F}}_{t-1}^{l}$

Structure Pruning:





$\mathbf{F}_q^l \in \mathbb{R}^{a_l \times n_t}$

${\mathbf{F}_q^l}^{\mathrm{T}} \in \mathbb{R}^{n_t \times a_l}$

$\mathbf{F}_q^l {\mathbf{F}_q^l}^{\mathrm{T}} \in \mathbb{R}^{a_l \times a_l}$

Proposed Approach (LRFR):

- Each task t training has two stages:

Pretraining → Training

Find a optimal subnetwork

Training this subnetwork to update low rank $\bar{\mathbf{F}}_t^l$

DNN over-parameterized for each continual learning task

Selecting a subnetwork to learn a single task, as opposed to using the entire network, does not affect training performance (The Lottery Ticket Hypothesis[2])

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In ICLR, 2019.