ECE 693 – Special Topics:
AI for Radar System Design

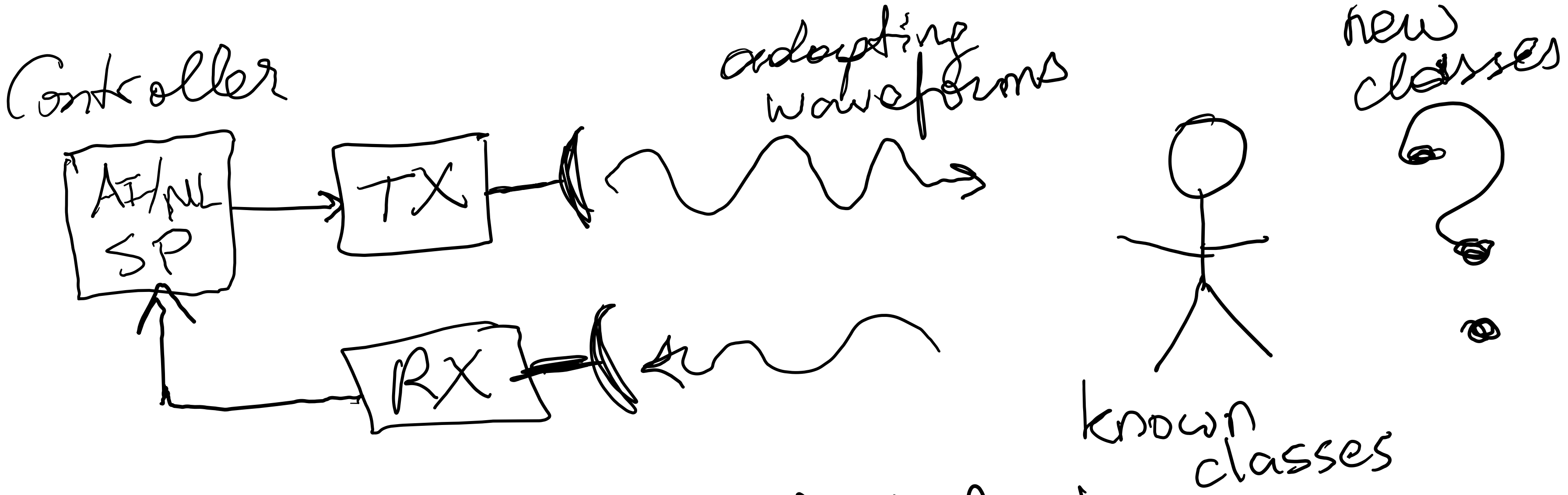
Continual and Incremental Learning

Dr. Sevgi Zubeyde Gurbuz
szgurbuz@ua.edu

Mar. 30, 2022

THE UNIVERSITY OF ALABAMA®

Context: ATR in Cognitive Radar



1) Pre-deployment batch training
2) Post-deployment continual learning
   - learn new classes   - generalization

Can We Simply Train on New Data?

Task A: $D_A = \{(x_i, y_i)\}$

Task B: $D_B = \{(x_i, y_i)\}$

First: $\min\limits_{\theta} \sum\limits_{x_i, y_i \in D_A} \mathcal{L}(\hat{y}_\theta(x_i), y_i)$

Then: $\min\limits_{\theta} \sum\limits_{x_i, y_i \in D_B} \mathcal{L}(\hat{y}_\theta(x_i), y_i)$

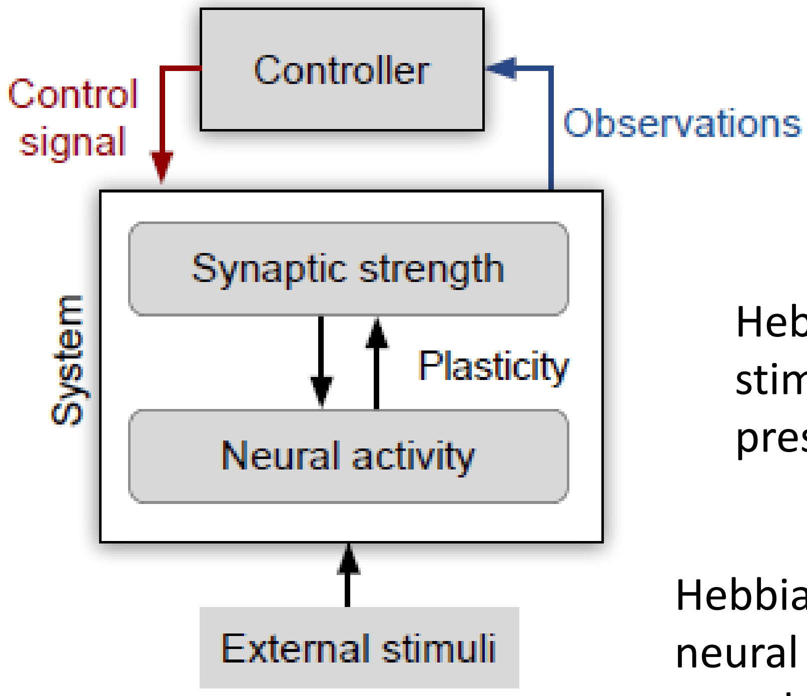
Problem: Catastrophic Forgetting

- Good performance on B, but worse performance on A!

- What if we train using both new AND old data?
  – Long time to re-train networks
  – Waste of power and computations
  – Original training data may be unavailable

Plasticity-Stability Dilemma

- Hebbian Plasticity and Stability



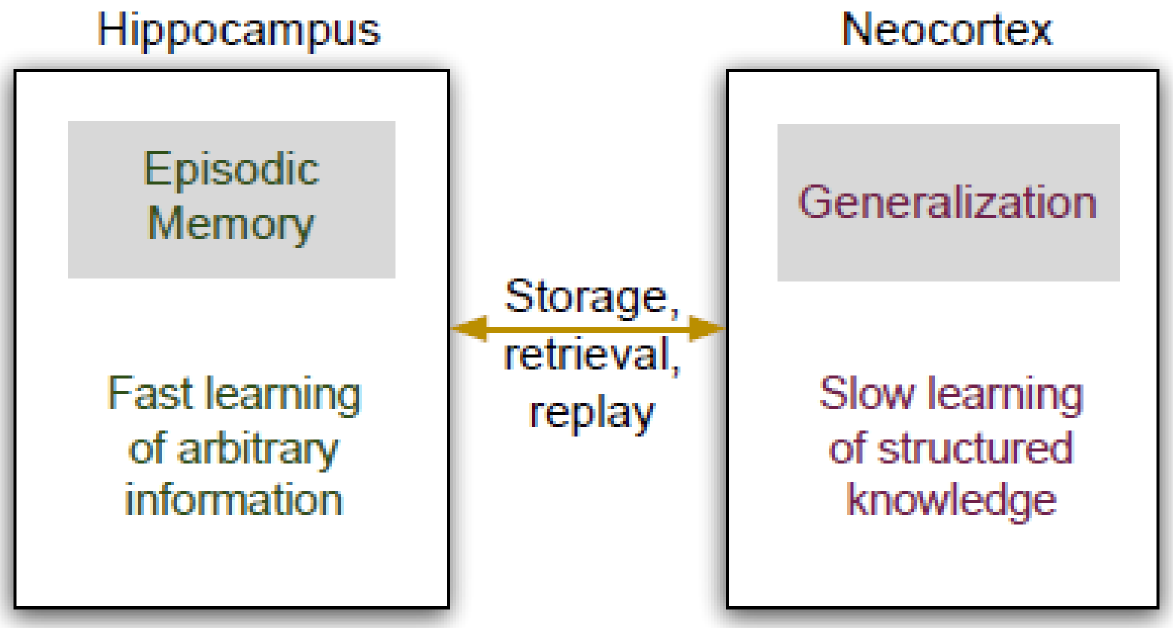
(Parisi, 2019)

Neurosynaptic plasticity is an essential feature of the brain yielding physical changes in the neural structure and allowing us to learn, remember, and adapt to dynamic environments

Hebb's rule states that the repeated and persistent stimulation of the postsynaptic cell from the presynaptic cell leads to an increased synaptic efficacy

Hebbian plasticity alone is unstable and leads to runaway neural activity, thus requiring compensatory mechanisms to stabilize the learning process

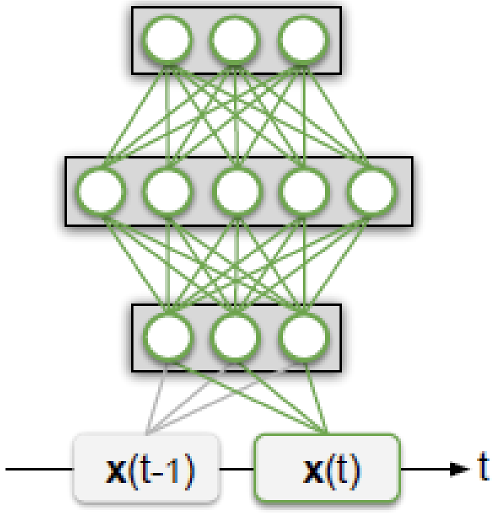
Complementary Learning Systems


Hippocampus
Neocortex
Episodic Memory
Generalization
Storage, retrieval, replay
Fast learning of arbitrary information
Slow learning of structured knowledge

(Parisi, 2019)

Approaches for Continual Learning

- Re-train the whole network with regularization



a) Retraining with regularization

Update the network weights, but penalize changes in order to minimize forgetting

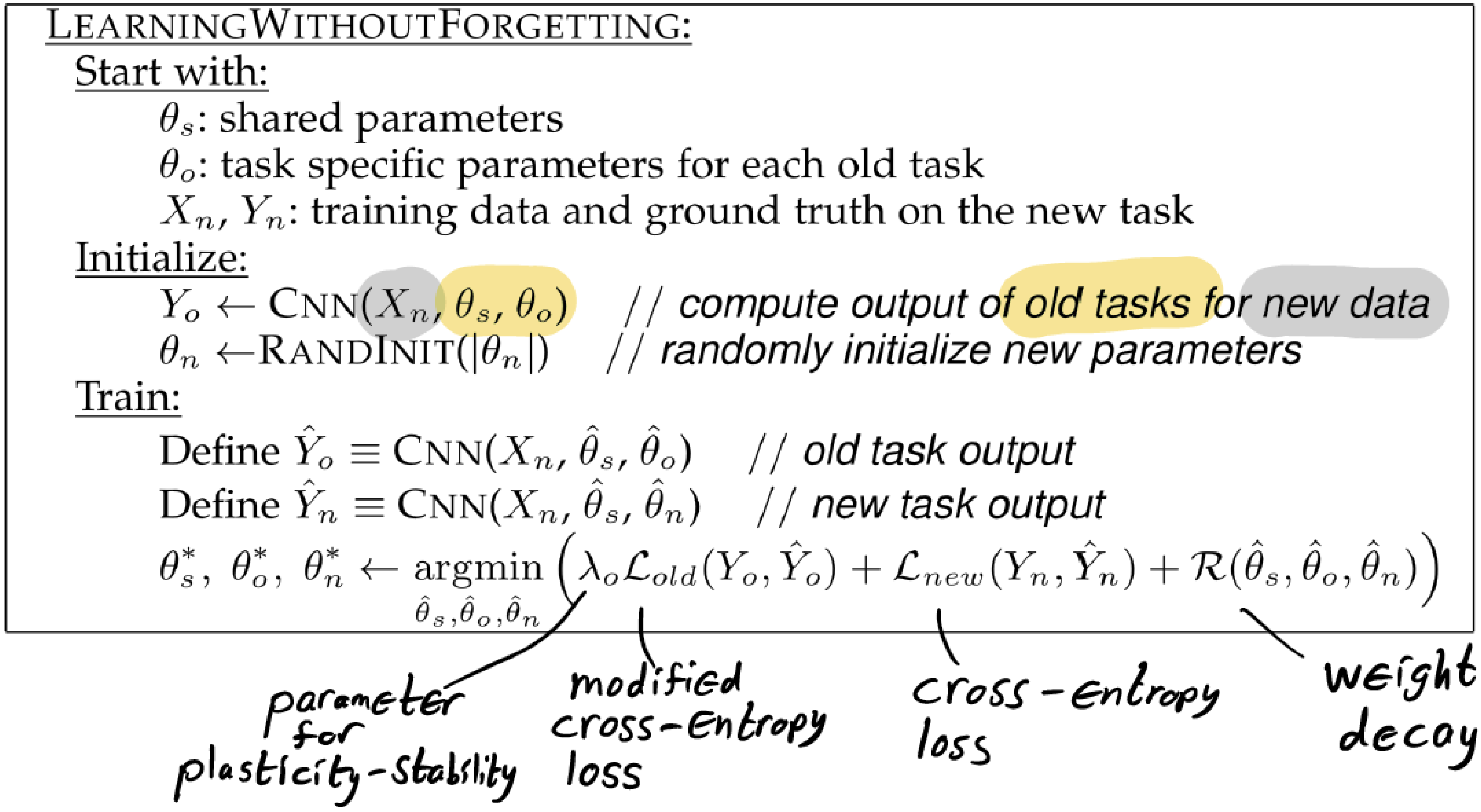
- Learning without Forgetting (LwF)
- Elastic Weight Consolidation (EWC)
- Estimation of Importance of Individual Synapses
- AR1 Model – combination of regularization and architectural modifications

Learning without Forgetting

- Consider a predictor with shared parameters across tasks and some task specific parameters
- At the new task, update
  - Shared parameters
  - New parameters
  - Old parameters

  So that output of old task on new data doesn't change too much.

Learning without Forgetting (2)



```
LEARNINGWITHOUTFORGETTING:
  Start with:
        $\theta_s$: shared parameters
        $\theta_o$: task specific parameters for each old task
        $X_n, Y_n$: training data and ground truth on the new task
  Initialize:
        $Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$    // compute output of old tasks for new data
        $\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$    // randomly initialize new parameters
  Train:
        Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$    // old task output
        Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$    // new task output
        $\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n)\right)$
```

parameter for plasticity-stability

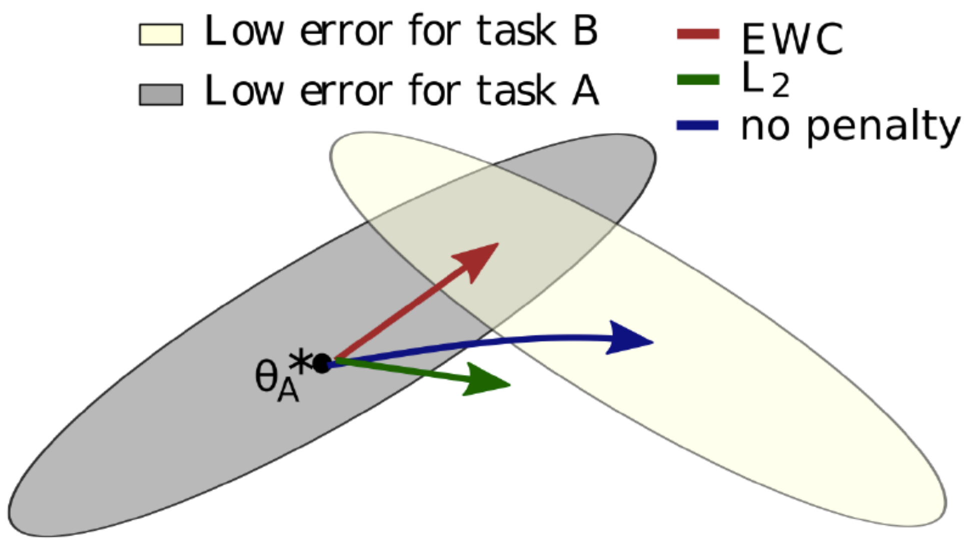
modified cross-entropy loss

cross-entropy loss

weight decay

Elastic Weight Consolidation (EWC)

- When training on Task B, identify weights that were important to A and penalize updates to those weights



Elastic Weight Consolidation (2)

Minimize

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\overset{\text{parameter}}{\lambda}}{2} F_i \left(\theta_i - \overset{\text{solution to Task A}}{\theta_{A,i}}\right)^2$$

where $F_i = E\left(\partial_{\theta_i} \log p_\theta(x)\right)^2$

diagonal entries give precision of θ_A^* (updated weights)

Estimation of Importance of Individual Synapses

- Zenke, Poole, and Ganguli (2017):
  - Individual synapses estimate their importance for solving a learned task (learned in online fashion)
  - Penalizes changes to most relevant synapses so that new tasks can be learned with minimal forgetting

weighting to balance old & new data

$$\underbrace{\tilde{L}_n^*}_{\text{modified cost}} = L_n + C \sum_k \underbrace{\Omega_k^n}_{\text{regularization strength per parameter}} \left(\theta_k^* - \theta_k\right)^2$$

Other Approaches

- Maltoni and Lomonaco (2018):
  - Progressively reduce magnitude of weight changes from batch to batch + architectural modifications
- Fernando (2017):
  - Genetic algorithm used to find the optimal path through a neural network of fixed size for replication and mutation.
  - Discovers which parts of network can be reused for learning new tasks while freezing task-relevant paths to avoid catastrophic forgetting

Example: MNIST Permutation

- Task 1: Training data is MNIST data (x_i, y_i)
- Task 2: Fix an image permutation P2

  Training data is MNIST $(P_2 x_i, y_i)$

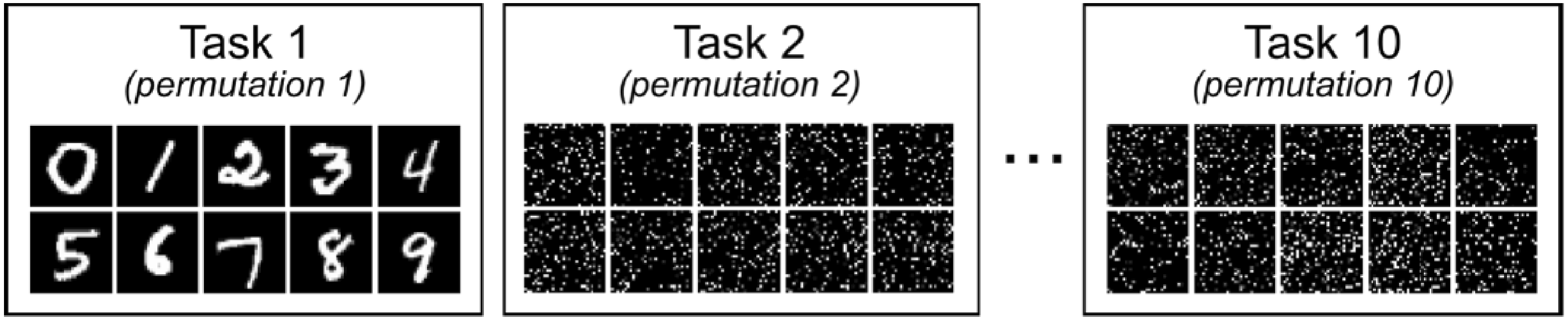
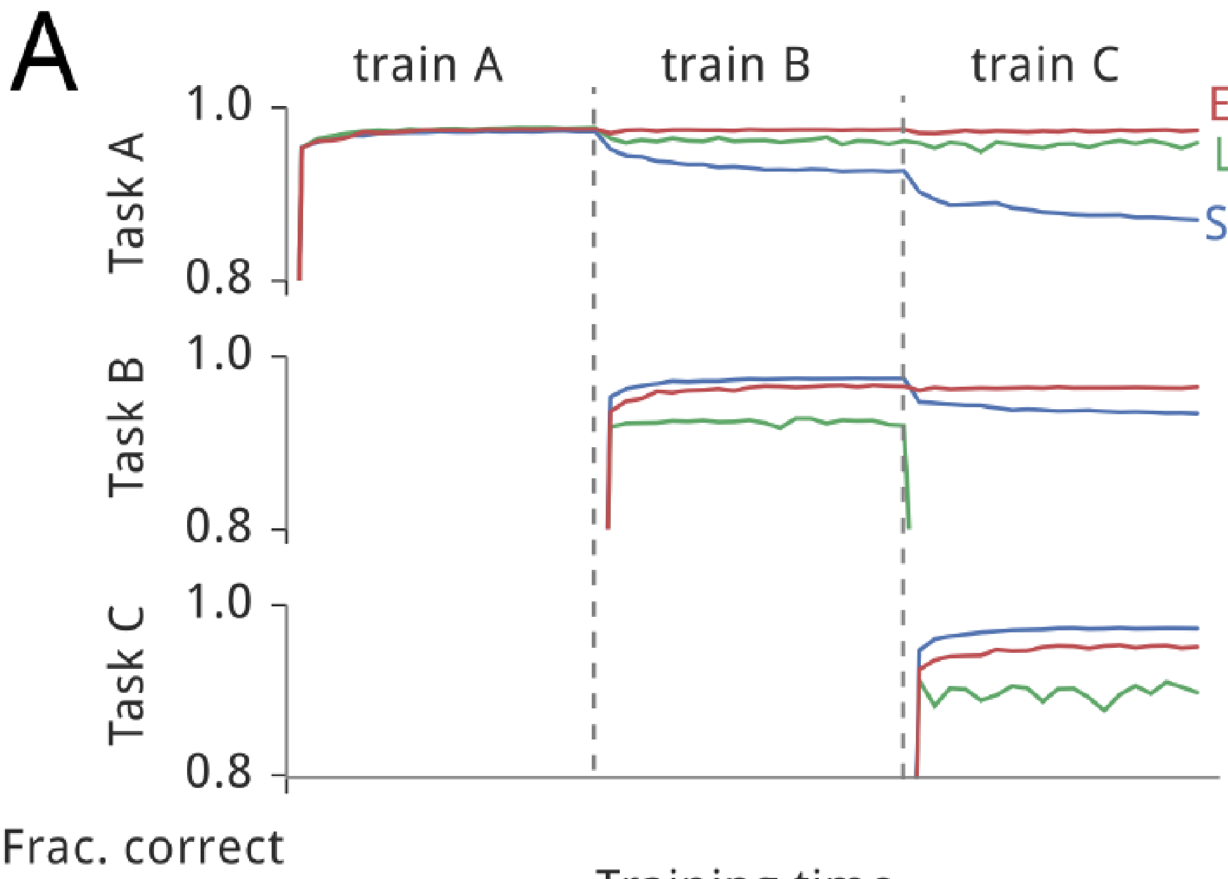


Figure 2: Schematic of permuted MNIST task protocol.

Comparison on Permutation-MNIST



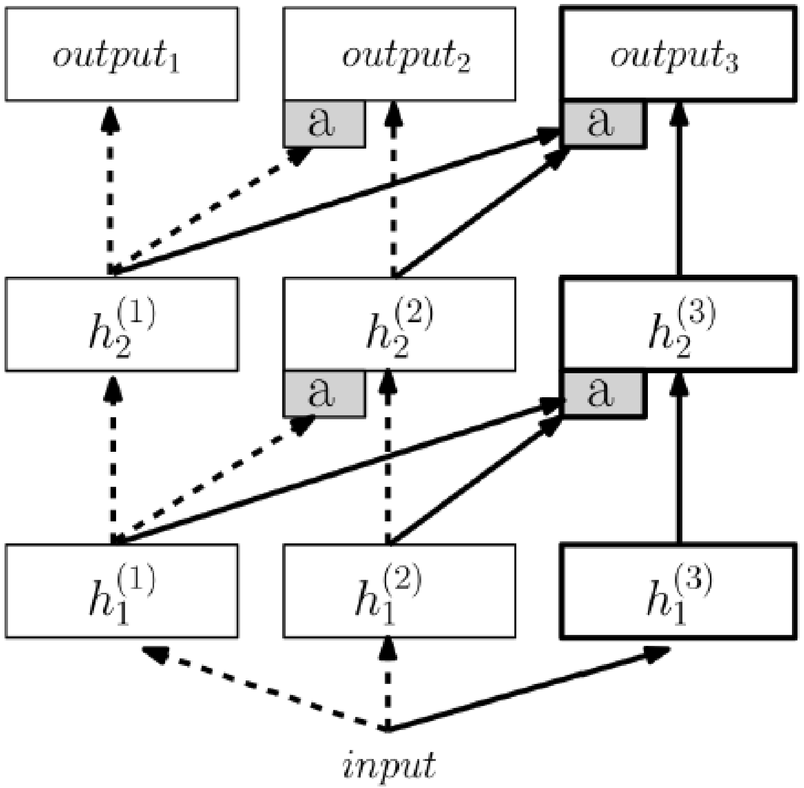
penalize with F_i

penalize equally

train on new data

Progressive Neural Networks

- Each iteration
  - – Add neurons
  - – Add output layer
  - – Add lateral connections
  - – Don't modify weights!

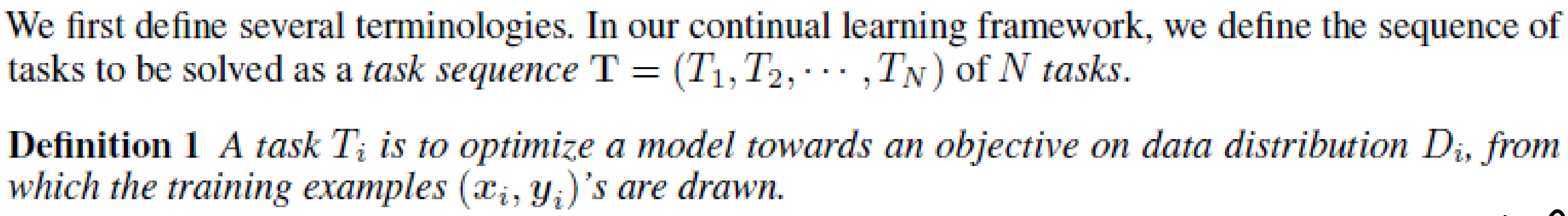


(Rusu, 2016)

Continuous Learning with Deep Generative Replay

- Shin, Lee, Kim, & Kim (2017)

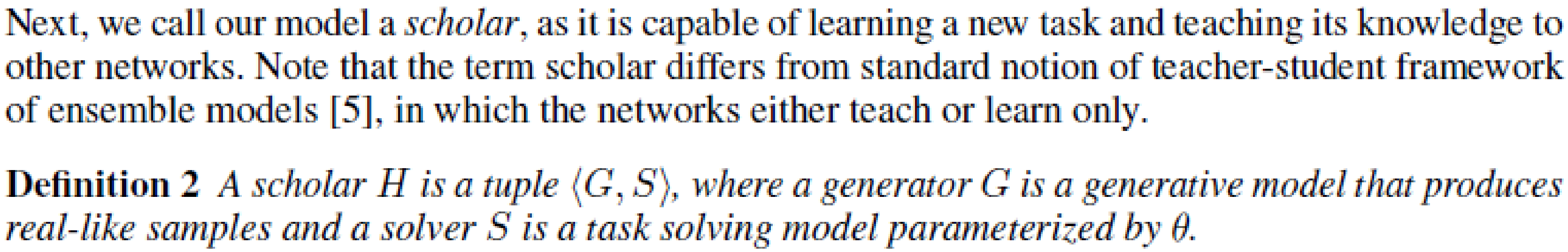
Hippocampus Model



We first define several terminologies. In our continual learning framework, we define the sequence of tasks to be solved as a *task sequence* $\mathbf{T} = (T_1, T_2, \cdots, T_N)$ of N *tasks.*

Definition 1 *A task T_i is to optimize a model towards an objective on data distribution D_i, from which the training examples (x_i, y_i)'s are drawn.*

Neo-Cortex Model



Next, we call our model a *scholar*, as it is capable of learning a new task and teaching its knowledge to other networks. Note that the term scholar differs from standard notion of teacher-student framework of ensemble models [5], in which the networks either teach or learn only.

Definition 2 *A scholar H is a tuple $\langle G, S \rangle$, where a generator G is a generative model that produces real-like samples and a solver S is a task solving model parameterized by θ.*

Sequential Training of Scholar Models

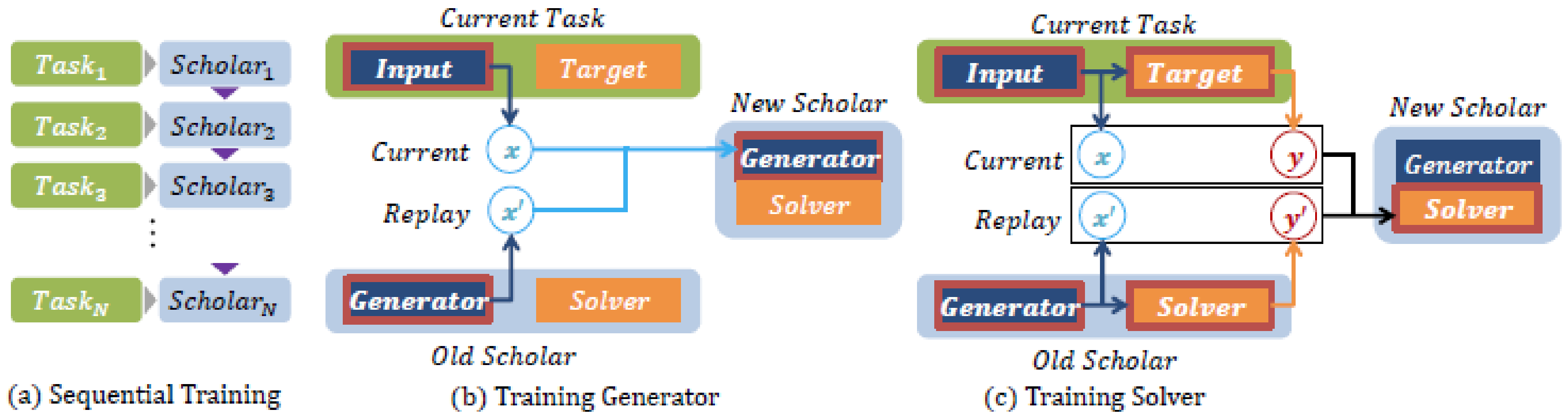
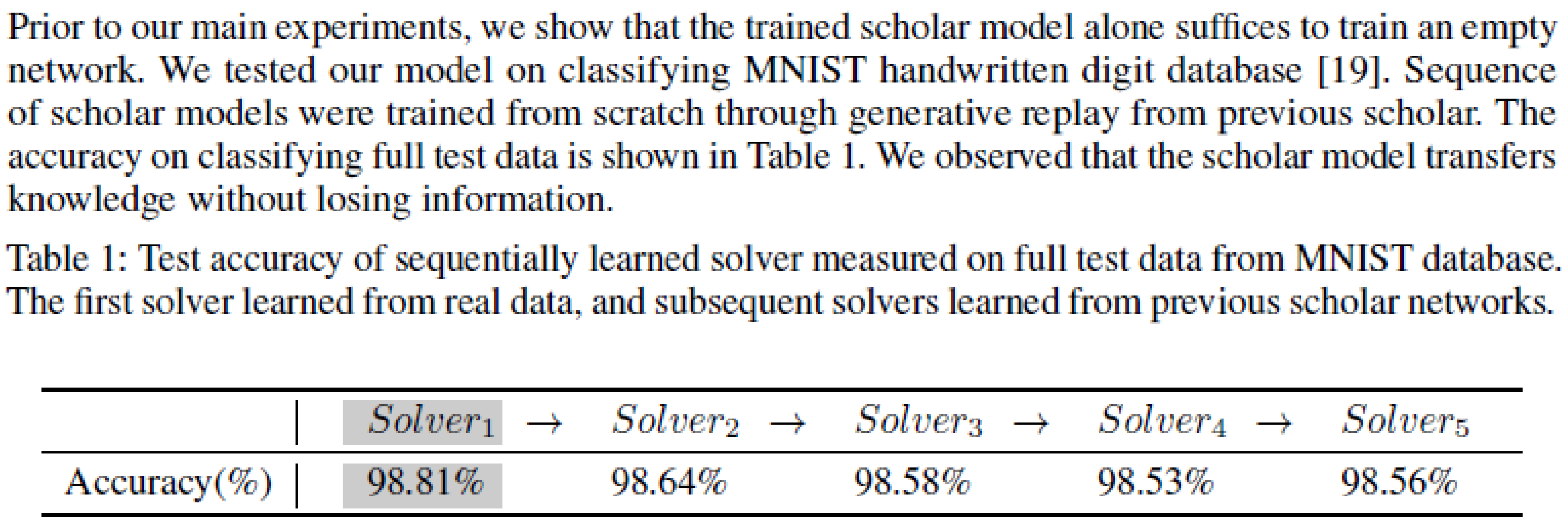


Figure 1: Sequential training of scholar models. (a) Training a sequence of scholar models is equivalent to continuous training of a single scholar while referring to its most recent copy. (b) A new generator is trained to mimic a mixed data distribution of real samples x and replayed inputs x' from previous generator. (c) A new solver learns from real input-target pairs (x, y) and replayed input-target pairs (x', y'), where replayed response y' is obtained by feeding generated inputs into previous solver.

Results on MNIST

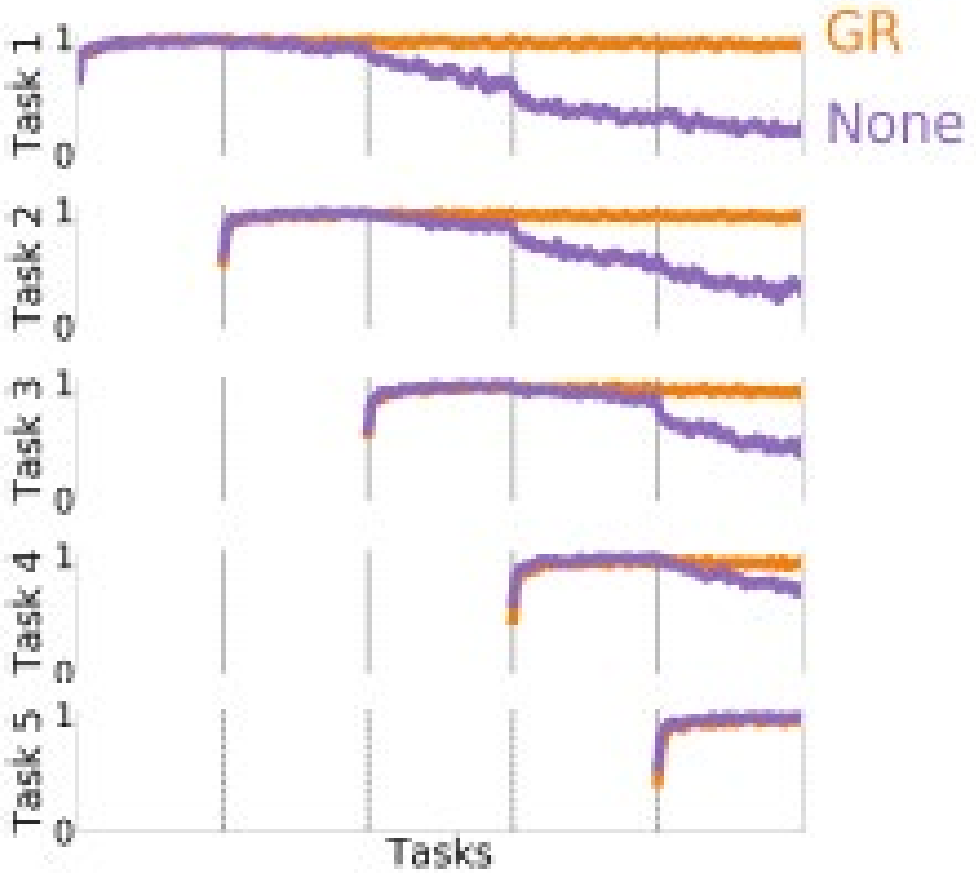


Prior to our main experiments, we show that the trained scholar model alone suffices to train an empty network. We tested our model on classifying MNIST handwritten digit database [19]. Sequence of scholar models were trained from scratch through generative replay from previous scholar. The accuracy on classifying full test data is shown in Table 1. We observed that the scholar model transfers knowledge without losing information.

Table 1: Test accuracy of sequentially learned solver measured on full test data from MNIST database. The first solver learned from real data, and subsequent solvers learned from previous scholar networks.

	$Solver_1$	$\rightarrow$	$Solver_2$	$\rightarrow$	$Solver_3$	$\rightarrow$	$Solver_4$	$\rightarrow$	$Solver_5$
Accuracy(%)	98.81%		98.64%		98.58%		98.53%		98.56%

Results on MNIST (2)


GR
None
Task 1
Task 2
Task 3
Task 4
Task 5
Tasks