The initializing module and the learning sequence

國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

Where we are now...

Representation and development

(Algorithm - Wikipedia)

- Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, drakon-charts, programming languages or control tables (processed by interpreters).
  - ✓ Natural language expressions of algorithms tend to be verbose and ambiguous, and are rarely used for complex or technical algorithms.
  - ✓ Pseudocode, flowcharts, drakon-charts and control tables are structured ways to express algorithms that avoid many of the ambiguities common in the statements based on natural language.
  - ✓ Programming languages are primarily intended for expressing algorithms in a form that can be executed by a computer, but are also often used as a way to define or document algorithms.

- Typical steps in the development of algorithms:
  - ✓ Problem definition
  - ✓ Development of a model
  - ✓ Specification of the algorithm
  - ✓ Designing an algorithm
  - ✓ Checking the correctness of the algorithm
  - ✓ Analysis of algorithm
  - ✓ Implementation of algorithm
  - ✓ Program testing
  - ✓ Documentation preparation

- For the AI application, the new learning mechanism is designed to get a good performance (i.e., the effectiveness and the efficiency) in the inferencing phase.
- Therefore, the new learning mechanism is not a product from an AI fundamental study.

Checking the correctness of the new mechanism

- Cannot validate the new learning mechanism through the **mathematical proof**.
- To validate the new learning mechanism, you need to make it and then to set up an AI application experiment with the real data, the proposed learning mechanism, and the computation capability.
- Check whether the corresponding learning process does display the proposed ideas/concepts. This is an AI fundamental study issue regarding the learning mechanism.
- Check whether the proposed learning mechanism does lead to good performances in the AI application. This is an AI application study issue regarding the AI system.

Program debugging of the new mechanism

- What should I do if the code cannot be run, the learning process is weird, or the performance is unsatisfied? ← Debug! Debug! And Debug!

➢ Debug each module/block through printing out some information associated with the module/block.

For example, print out values of $L(\mathbf{w})$ and all $f(\mathbf{x}^c, \mathbf{w})$ and y^c before and after the block.

➢ Debug the consistency amongst several consecutive modules/blocks through printing out some data flow between these consecutive modules/blocks.

For example, the learning goals used in all modules (i.e., the weight-tuning module, the cramming module, the regularizing module, and the reorganizing module) should be consistent.

➢ Debug the logic of the whole mechanism through printing out some performance results.

An optimization mechanism

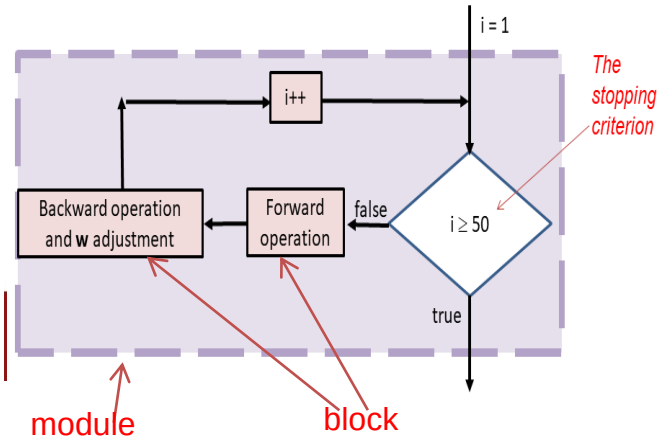
The weight-tuning_EU

PyTorch: nn

Higher-level wrapper for working with neural nets

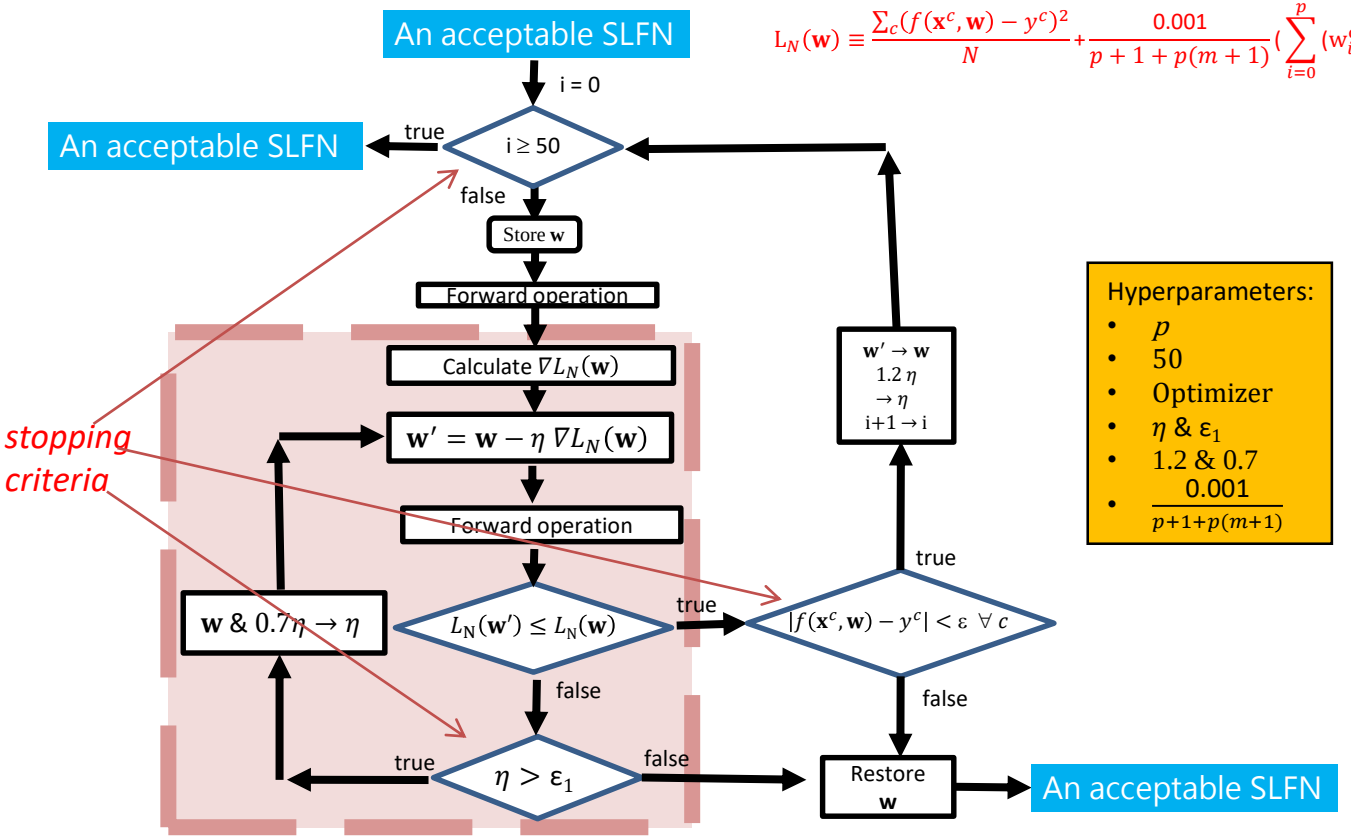
Use this! It will make your life easier

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```



An optimization mechanism

The regularizing_EU_LG_UA



Program debugging of blocks/modules

- Cannot merely double check the correctness of codes
- Simple checks (for a block/module):

1) whether the evolution of $L(\mathbf{w})$ values is reasonable?

2) whether the tuning of $\mathbf{w}$ is reasonable?

3) whether the evolution of all $f(\mathbf{x}^c, \mathbf{w})$ values is reasonable?

- Complicated checks (for several blocks/modules):

  New learning mechanism → new learning process ← Double check the learning process, not the performance, which is related with the inferencing.

An optimization mechanism

The reorganizing_ALL_r_EU_LG_UA_w_EU_LG_UA

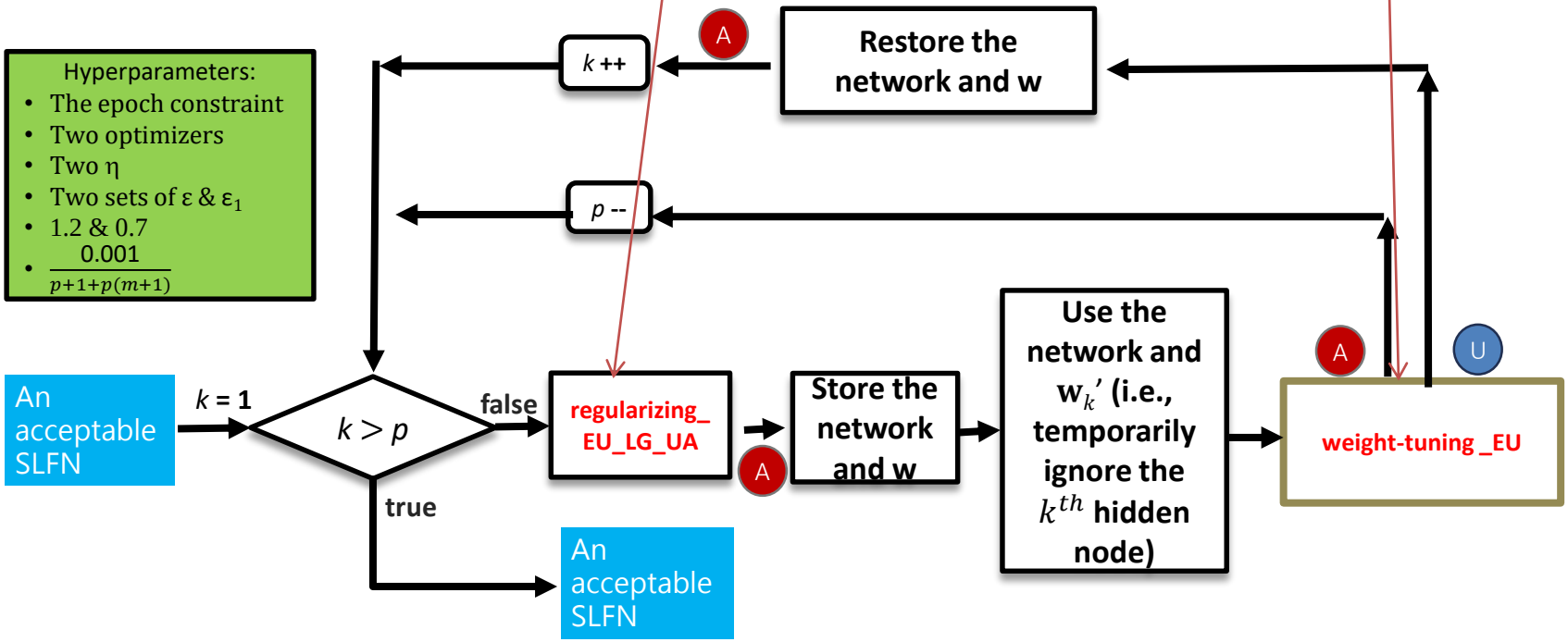
$$L_N(\mathbf{w}) \equiv \frac{\sum_c (f(\mathbf{x}^c, \mathbf{w}) - y^c)^2}{N} + \frac{0.001}{p + 1 + p(m+1)} \left(\sum_{i=0}^{p} (w_i^o)^2 + \sum_{i=1}^{p} \sum_{j=0}^{m} (w_{ij}^H)^2 \right)$$

$$L_N(\mathbf{w}) \equiv \frac{\sum_c (f(\mathbf{x}^c, \mathbf{w}) - y^c)^2}{N}$$

Note that there are two optimizers:

One for the regularizing purpose

Another for the pruning purpose



A rule-based mechanism

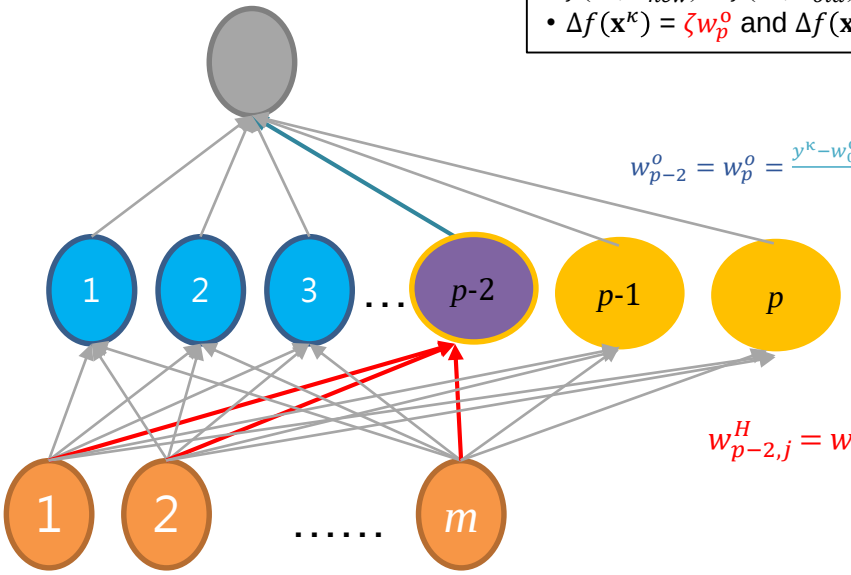
The cramming_ReLU_RI_SO_RE_SU

Step 1: Pick up a tiny number ζ and then use the random-number generation method to create an m-vector $\boldsymbol{\gamma}$ of length one such that $\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa}) \neq 0 \ \forall\ c \in \mathbf{I}-\{\kappa\}$ AND $(\zeta+\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa}))*(\zeta-\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa})) < 0 \ \forall\ c \in \mathbf{I}-\{\kappa\}$.

Step 2: Let p+3 → p, add three new hidden nodes p-2th, p-1th and pth to the existing SLFN, and then assign their associated weights in the following way to make $(f(\mathbf{x}^{c},\mathbf{w})-y^{c})^{2} \leq \varepsilon^{2}\ \forall\ c \in \mathbf{I}$ true:

- $\mathbf{w}^{H}_{p-2} = \mathbf{w}^{H}_{p-2} = \mathbf{w}^{H}_{p-2} = \boldsymbol{\gamma}$
- $w^{H}_{p-2,0} = \zeta-\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$, $w^{H}_{p-1,0} = -\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$, $w^{H}_{p,0} = -\zeta-\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$
- $w^{o}_{p-2} = w^{o}_{p} = \frac{y^{\kappa}-w^{o}_{0}-\sum_{i=1}^{p-3} w^{o}_{i} a^{\kappa}_{i}}{\zeta}$; $w^{o}_{p-1} = \frac{-2(y^{\kappa}-w^{o}_{0}-\sum_{i=1}^{p-3} w^{o}_{i} a^{\kappa}_{i})}{\zeta}$

- $f(\mathbf{x}^{\kappa},\mathbf{w}_{old}) = w^{o}_{0} + \sum_{i=1}^{p-3} w^{o}_{i} a^{\kappa}_{i}$
- $f(\mathbf{x}^{c},\mathbf{w}_{new}) = f(\mathbf{x}^{c},\mathbf{w}_{old}) + w^{o}_{p}$*[ReLU($\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa})+\zeta$) - 2ReLU($\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa})$) + ReLU($\boldsymbol{\gamma}^{\mathrm{T}}(\mathbf{x}^{c}-\mathbf{x}^{\kappa})-\zeta$)]
- $\Delta f(\mathbf{x}^{\kappa}) = \zeta w^{o}_{p}$ and $\Delta f(\mathbf{x}^{c}) = 0\ \forall\ c \in \mathbf{I}-\{\kappa\}$

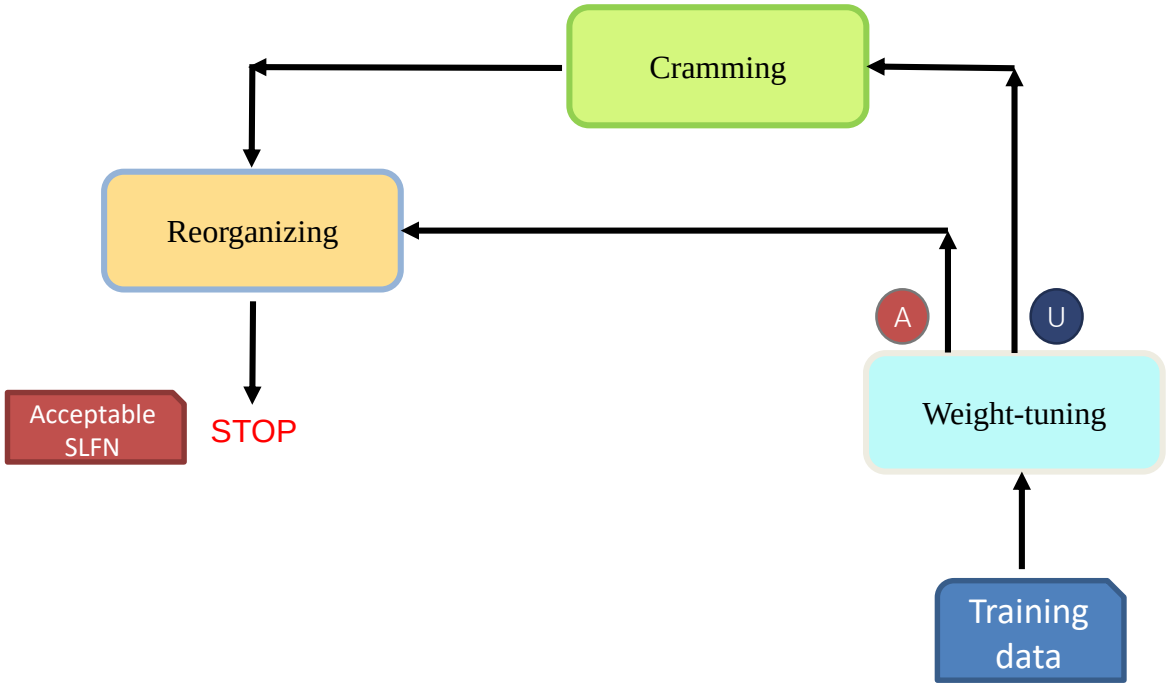


$w^{o}_{p-2} = w^{o}_{p} = \frac{y^{\kappa}-w^{o}_{0}-\sum_{i=1}^{p-3} w^{o}_{i} a^{\kappa}_{i}}{\zeta}$; $w^{o}_{p-1} = \frac{-2(y^{\kappa}-w^{o}_{0}-\sum_{i=1}^{p-3} w^{o}_{i} a^{\kappa}_{i})}{\zeta}$

$w^{H}_{p-2,0} = \zeta-\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$; $w^{H}_{p-1,0} = -\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$; $w^{H}_{p,0} = -\zeta-\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{x}^{\kappa}$

$w^{H}_{p-2,j} = w^{H}_{p-1,j} = w^{H}_{p,j} = \gamma_{j} \quad \forall\ j$

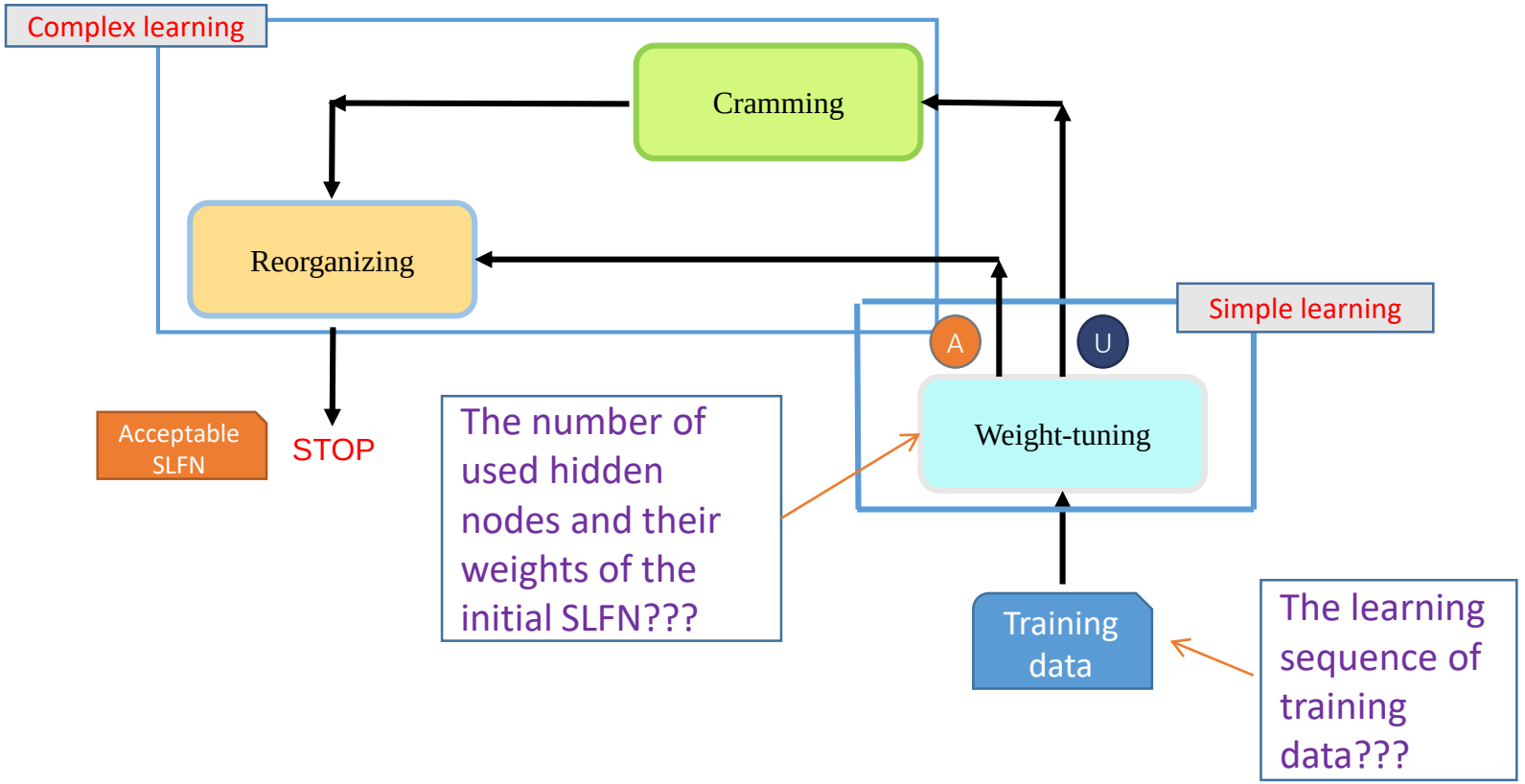
A new learning mechanism (incomplete)



Fix the mechanism with debugging

1. Debug each module/block through printing out some information associated with the module/block. ← Fix the bugs of the module/block or replace the module/block.
2. Debug the consistency amongst several consecutive modules/blocks through printing out some data flow between these consecutive modules/blocks. ← Fix the inconsistency via fine-tuning or replacing some modules/blocks.
3. Debug the logic of the whole mechanism through printing out some results. ← Fix the logic via fine-tuning or replacing some modules/blocks. ← This is the core of validating the new mechanism.

A new learning mechanism (incomplete)



The initializing module

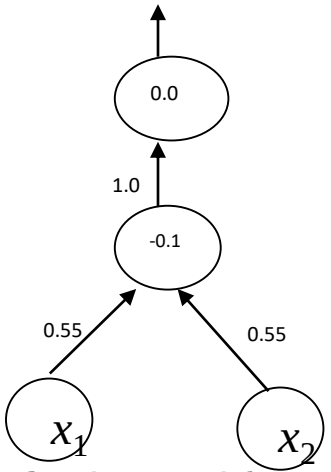
- Initialize an SLFN with one hidden node: $f(\mathbf{x}^c,\mathbf{w}) \equiv w_0^o + w_1^o ReLU(w_{10}^H + \sum_{j=1}^m w_{1j}^H x_j^c)$
- Initialize an SLFN with two hidden nodes:

$$f(\mathbf{x}^c,\mathbf{w}) \equiv w_0^o + \sum_{i=1}^2 w_i^o \, ReLU\left(w_{i0}^H + \sum_{j=1}^m w_{ij}^H x_j^c\right)$$

- Initialize an SLFN with *p* hidden nodes:

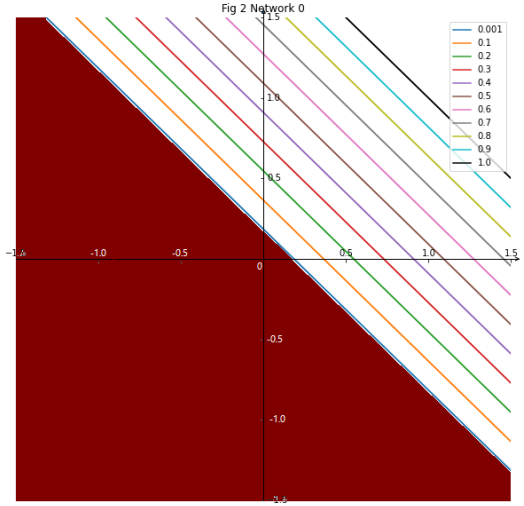
$$f(\mathbf{x}^c,\mathbf{w}) \equiv w_0^o + \sum_{i=1}^p w_i^o \, ReLU\left(w_{i0}^H + \sum_{j=1}^m w_{ij}^H x_j^c\right)$$

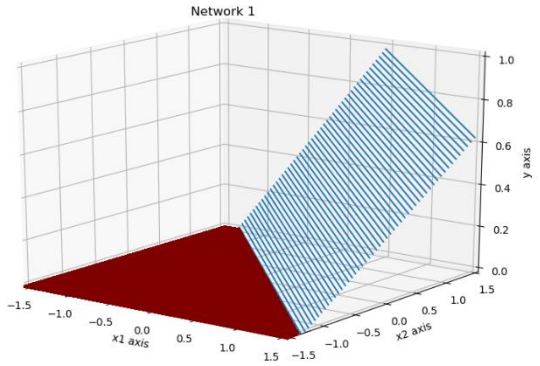
The SLFN with one hidden node



$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$





The initializing module_1_ReLU_LR

Step 1: Apply the linear regression method to the data set $\{(\mathbf{x}^c, y^c - \min_{u \in \mathbf{I}} y^u): c \in \mathbf{I}\}$ to obtain a set of $m + 1$ weights $\{w_j: j = 0, 1, \ldots, m\}$.

Step 2: Set up the SLFN with one hidden node whose w_{1j}^H equals $w_j \; \forall \; j = 1, \ldots, m$, w_{10}^H equals w_0, w_1^o equals 1 and w_0^o equals $\min_{u \in \mathbf{I}} y^u$.

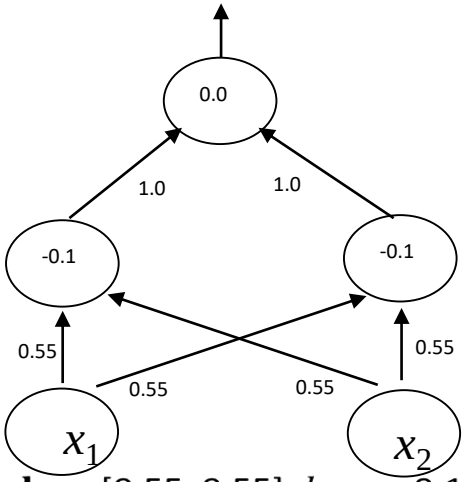
The initializing module_1_ReLU_WT

Step 1: Set up the SLFN with one hidden node with random weights.

Step 2: Use a weight-tuning module to tune up the weights.

The number of weights = $1 + 1 + m + 1$

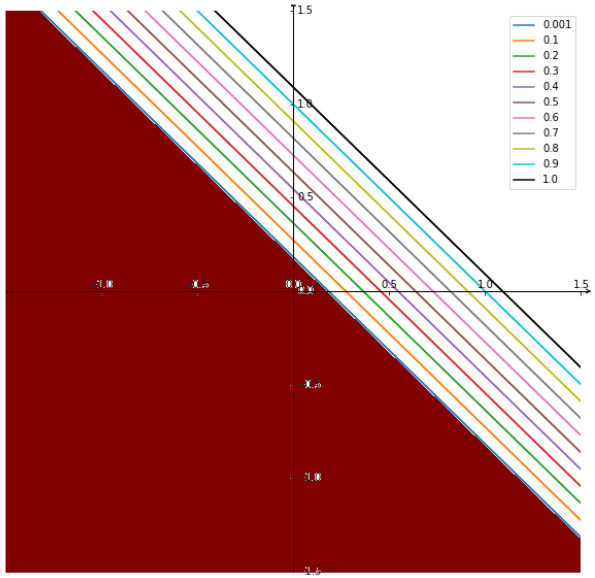
The SLFN with two hidden nodes

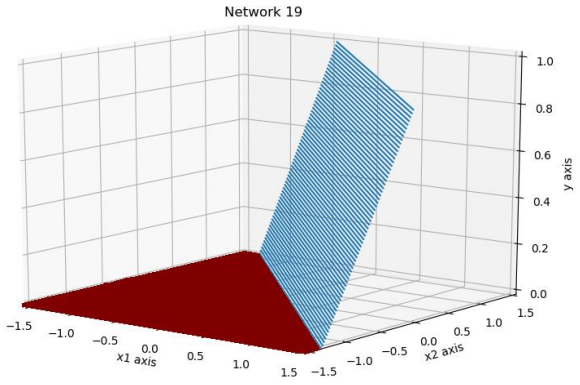


$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

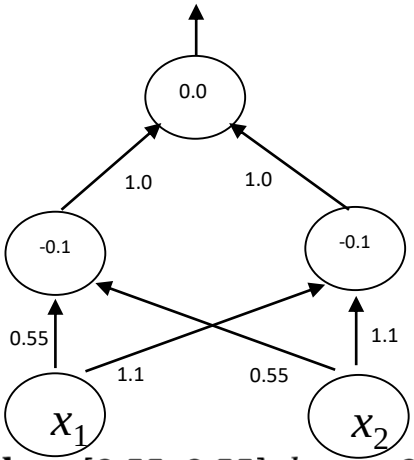
$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = 1*$\mathbf{b}_1$, $w_{2,0}^H$ = 1*$b_{1,0}$





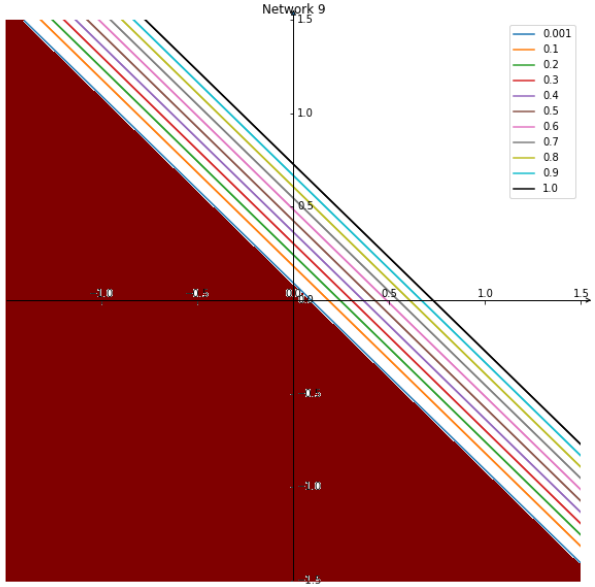
The SLFN with two hidden nodes

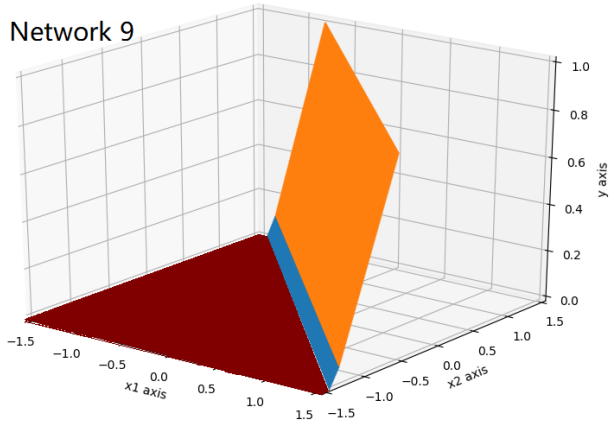


$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

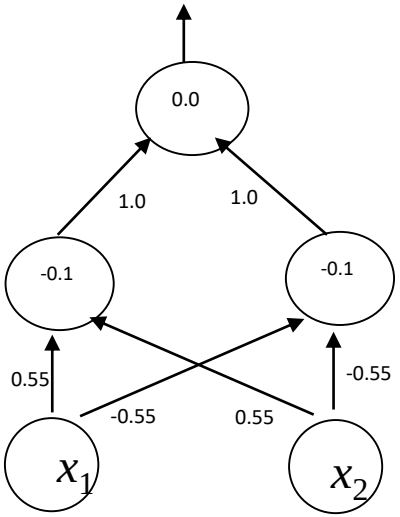
$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = 2*$\mathbf{b}_1$, $w_{2,0}^H$ = 1*$b_{1,0}$





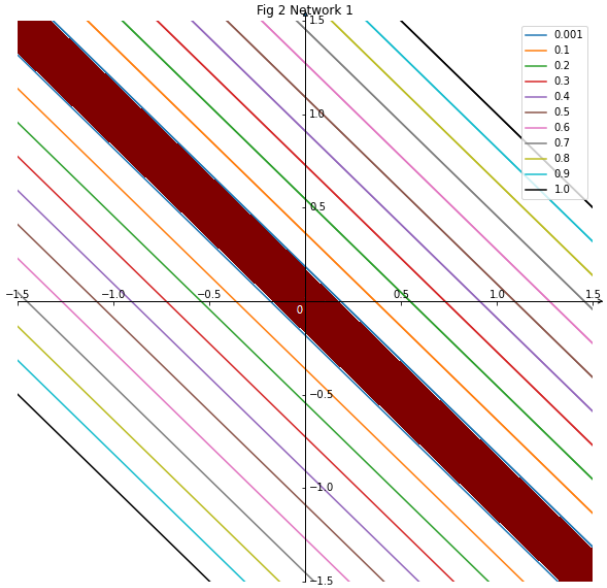
The SLFN with two hidden nodes

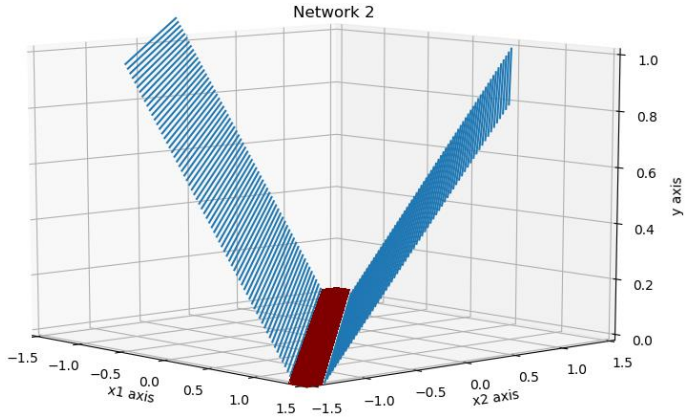


$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

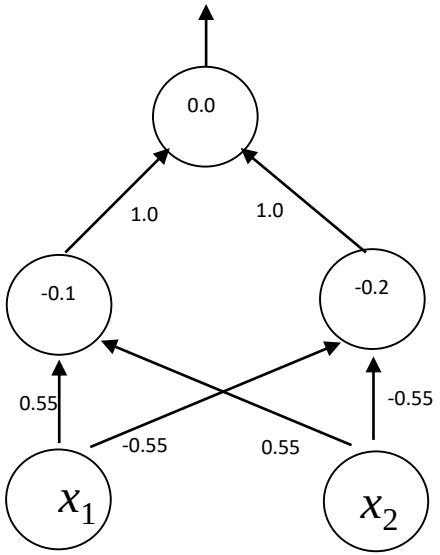
$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = -1*$\mathbf{b}_1$, $w_{2,0}^H$ = 1*$b_{1,0}$





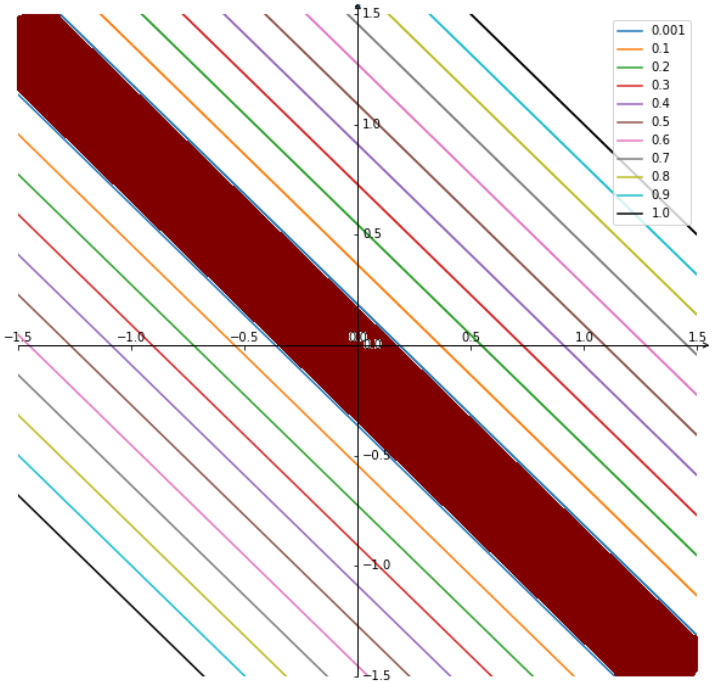
The SLFN with two hidden nodes



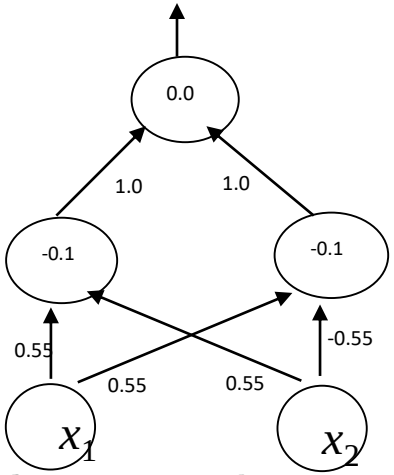
$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = -1*$\mathbf{b}_1$, $w_{2,0}^H$ = 2*$b_{1,0}$



The SLFN with two hidden nodes

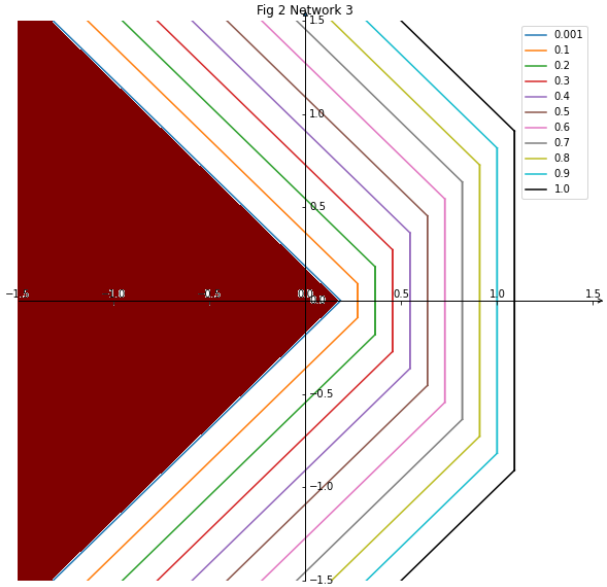


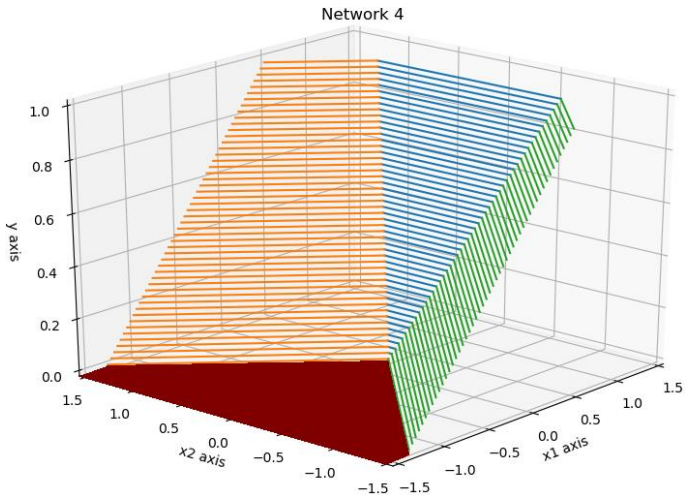
$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

$\mathbf{b}_2$ = [0.55, -0.55], $b_{2,0}$ = -0.1

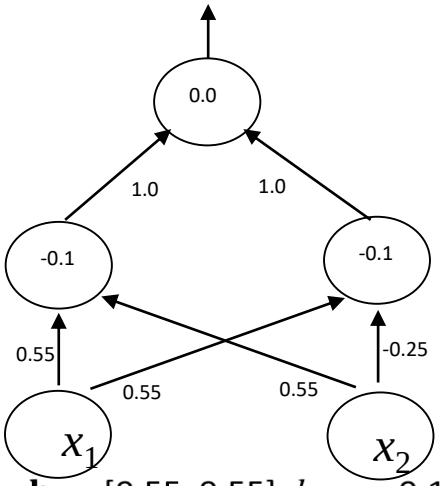
$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = 1*$\mathbf{b}_2$, $w_{2,0}^H$ = 1*$b_{2,0}$





The SLFN with two hidden nodes

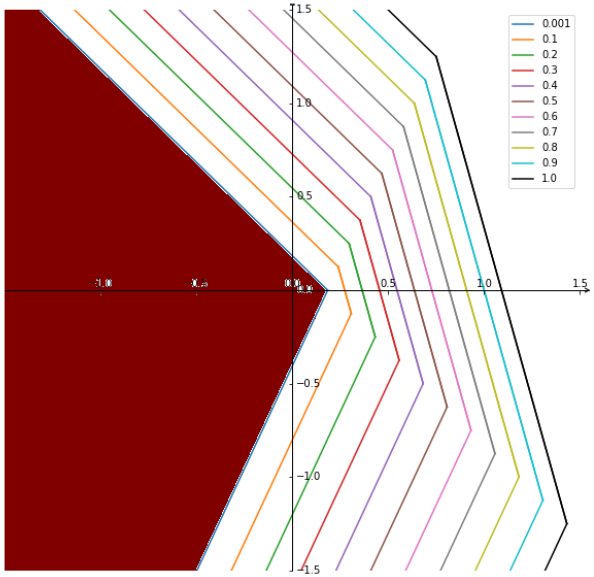


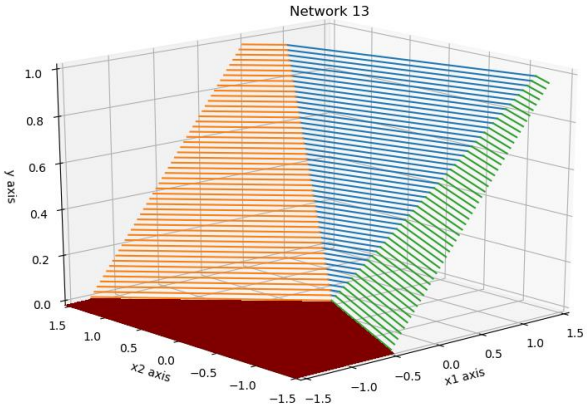
$\mathbf{b}_1$ = [0.55, 0.55], $b_{1,0}$ = -0.1

$\mathbf{b}_2$ = [0.55, -0.25], $b_{2,0}$ = -0.1

$\mathbf{w}_1^H$ = 1*$\mathbf{b}_1$, $w_{1,0}^H$ = 1*$b_{1,0}$

$\mathbf{w}_2^H$ = 1*$\mathbf{b}_2$, $w_{2,0}^H$ = 1*$b_{2,0}$





The initializing module_2_ReLU_WT

Step 1: Set up the SLFN with two hidden nodes with random weights.

Step 2: Use a weight-tuning module to tune up the weights.

The number of weights = $2 + 1 + 2(m + 1)$

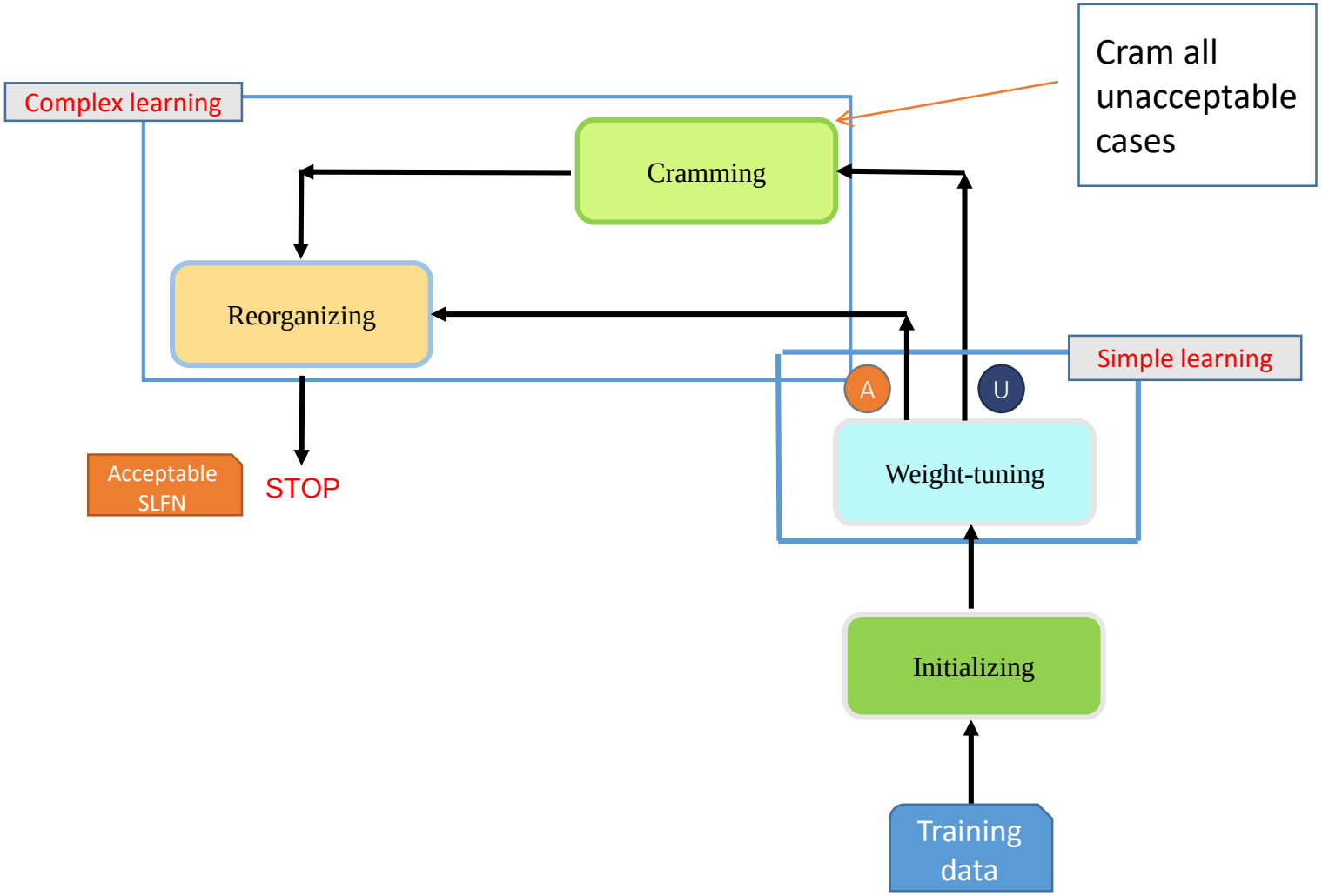
The initializing module_*p*_ReLU_WT

Step 1: Set up the SLFN with p hidden nodes with random weights.

Step 2: Use a weight-tuning module to tune up the weights.

The number of weights = $p + 1 + p(m + 1)$

A new learning mechanism
(in flowchart)



When the weight-tuning module leads to an unacceptable SLFN

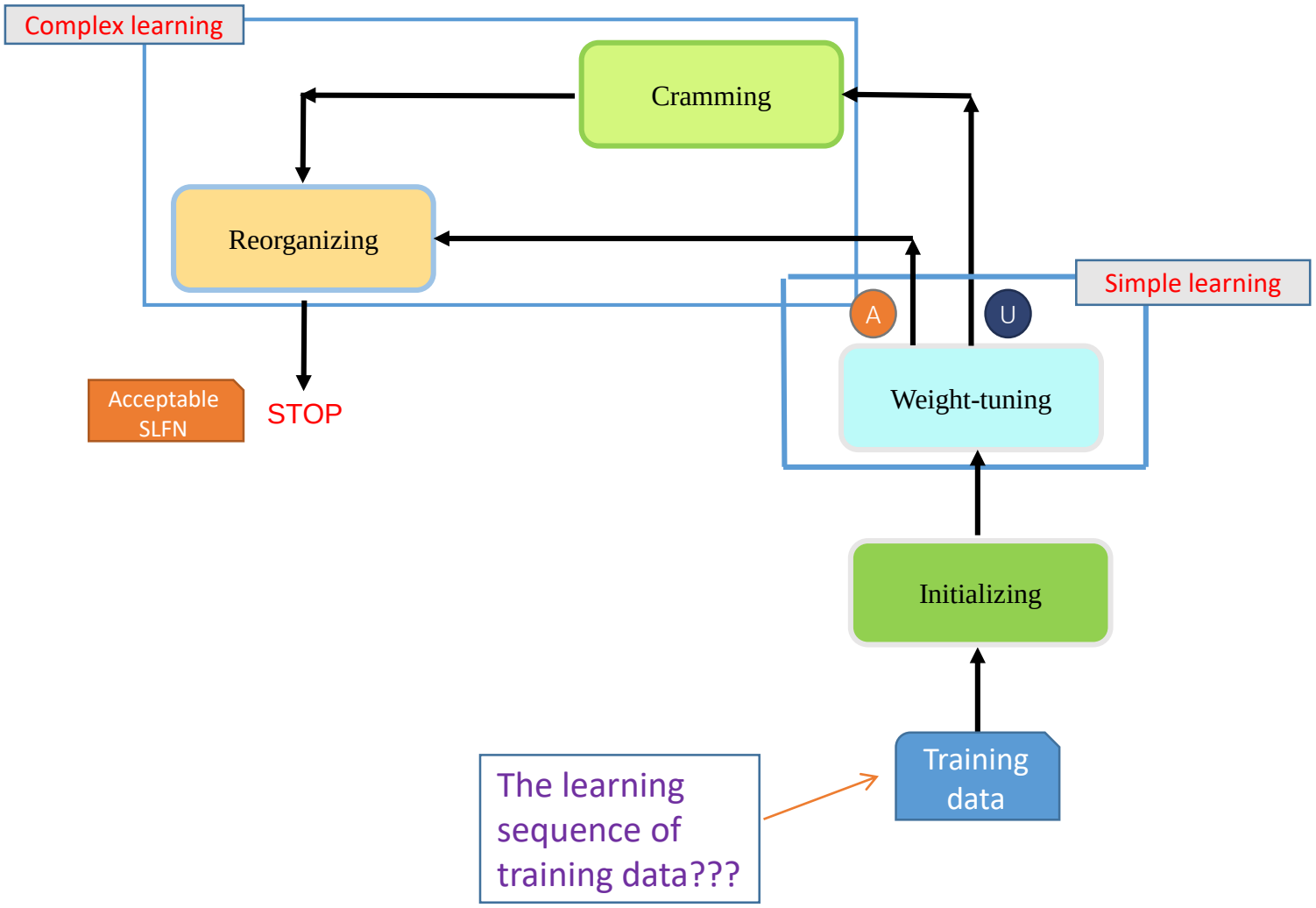
Apply the cramming module to cram

- all unacceptable cases, or
- one unacceptable case,
- some unacceptable cases,
- …

The learning sequence of training data???

A new learning mechanism (incomplete)

(in flowchart)



The learning sequence of training data

- The whole-batch learning
- The mini-batch learning ← add randomness
- The trimmed-batch learning ← add robustness
- The learning one by one ← the sequence order (random order OR specific order)

The whole-batch: the regression process and the least square estimator (LSE)

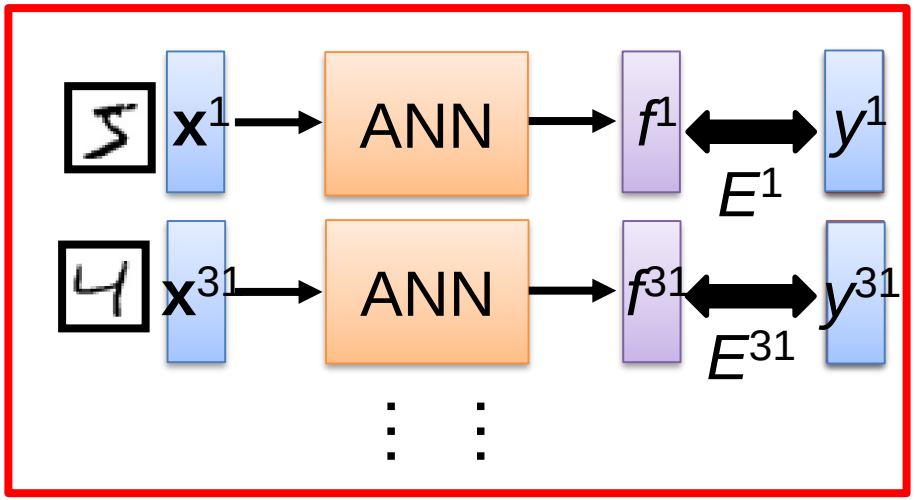
- The response is modeled as $f(\mathbf{x}, \mathbf{w}) + \delta$, where $\mathbf{w}$ is the parameter vector and δ is the error term.
- In the (linear or non-linear) regression process, the proposed functional form f is pre-specified and fixed during the process of deriving its associated $\mathbf{w}$ from a set of given samples $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$.
- Normally, the least square estimate (LSE) is used to estimate $\mathbf{w}$ from the given model $f(\mathbf{x}^c, \mathbf{w})$ and set of given samples $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$.
- If $\hat{\mathbf{w}}$ denotes any estimate of $\mathbf{w}$, then LSE is defined to be the $\hat{\mathbf{w}}$ that minimizes $E_N(\mathbf{w}) \equiv \sum_{c=1}^{\mathrm{N}} (e^c)^2$, where $e^c = f(\mathbf{x}^c, \mathbf{w}) - y^c$.

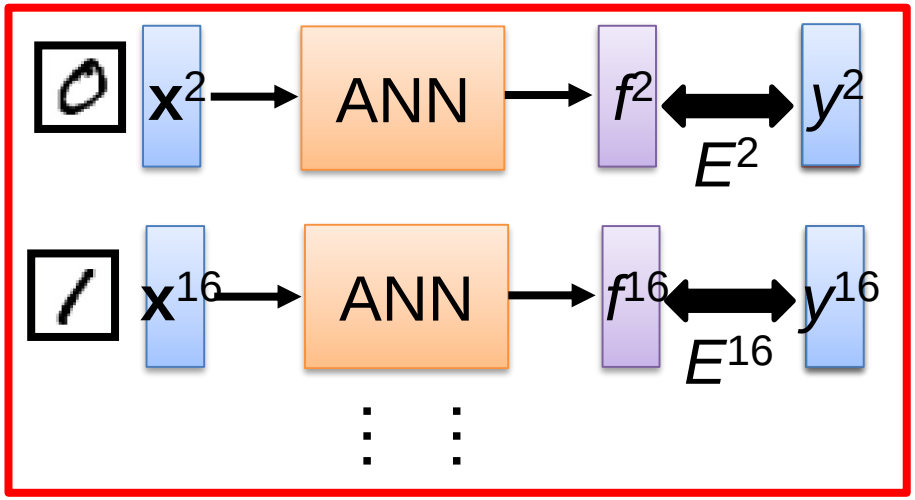
Regarding the deep learning application

The mini-batch

Faster

Better!





- Randomly initialize $\mathbf{w}^0$
- Pick the 1st batch

  $E = E^1 + E^{31} + \dots$

  $\mathbf{w}^1 \leftarrow \mathbf{w}^0 - \eta \nabla E(\mathbf{w}^0)$
- Pick the 2nd batch

  $E = E^2 + E^{16} + \dots$

  $\mathbf{w}^1 \leftarrow \mathbf{w}^0 - \eta \nabla E(\mathbf{w}^0)$

  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

The trimmed-batch: the robustness analysis and the least trimmed squares (LTS) estimator

- Robustness analysis is used to limit the attention to a "trimmed" sum of squared residuals instead of all the squared residuals stated in the LSE.
- $(e^{[c]})^2$ denotes the ordered squared residuals; that is, $(e^{[1]})^2 \leq (e^{[2]})^2 \leq \ldots \leq (e^{[N]})^2$.
- Only the smallest q cases of the ordered squared residuals are included in the summation $E_q(\mathbf{w}) \equiv \sum_{c=1}^{q} (e^{[c]})^2$.
- The least trimmed squares (LTS) estimator is defined as the estimate $\hat{\mathbf{w}}$ that minimizes $E_q(\mathbf{w}) \equiv \sum_{c=1}^{q} (e^{[c]})^2$.

The sequence learning

(Sequence learning - Wikipedia)

- ✓ In cognitive psychology, **sequence learning** is inherent to human ability because it is an integrated part of conscious and nonconscious learning as well as activities.
- ✓ Sequences of information or sequences of actions are used in various everyday tasks: "from sequencing sounds in speech, to sequencing movements in typing or playing instruments, to sequencing actions in driving an automobile."
- ✓ Sequence learning can be used to study skill acquisition and in studies of various groups ranging from neuropsychological patients to infants.
- ✓ According to Ritter and Nerb, "The order in which material is presented can strongly influence what is learned, how fast performance increases, and sometimes even whether the material is learned at all."
- ✓ Sequence learning, more known and understood as a form of explicit learning, is now also being studied as a form of implicit learning as well as other forms of learning.
- ✓ Sequence learning can also be referred to as sequential behavior, behavior sequencing, and serial order in behavior.

The incremental learning

(Incremental learning - Wikipedia)

- ✓ In computer science, the incremental learning is a method of machine learning in which input data is continuously used to extend the existing model's knowledge (i.e., to further train the model).
- ✓ It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time or its size is out of system memory limits.
- ✓ Algorithms that can facilitate incremental learning are known as incremental machine learning algorithms.

The obtaining module_LTS

Step 1: Sort all training data $\{(\mathbf{x}^c, y^c)\ c \in \mathbf{I}(N)\}$ by their squared residuals in ascending order as $(e^{[1]})^2 \leq (e^{[2]})^2 \leq \ldots \leq (e^{[N]})^2$.

The learning goal

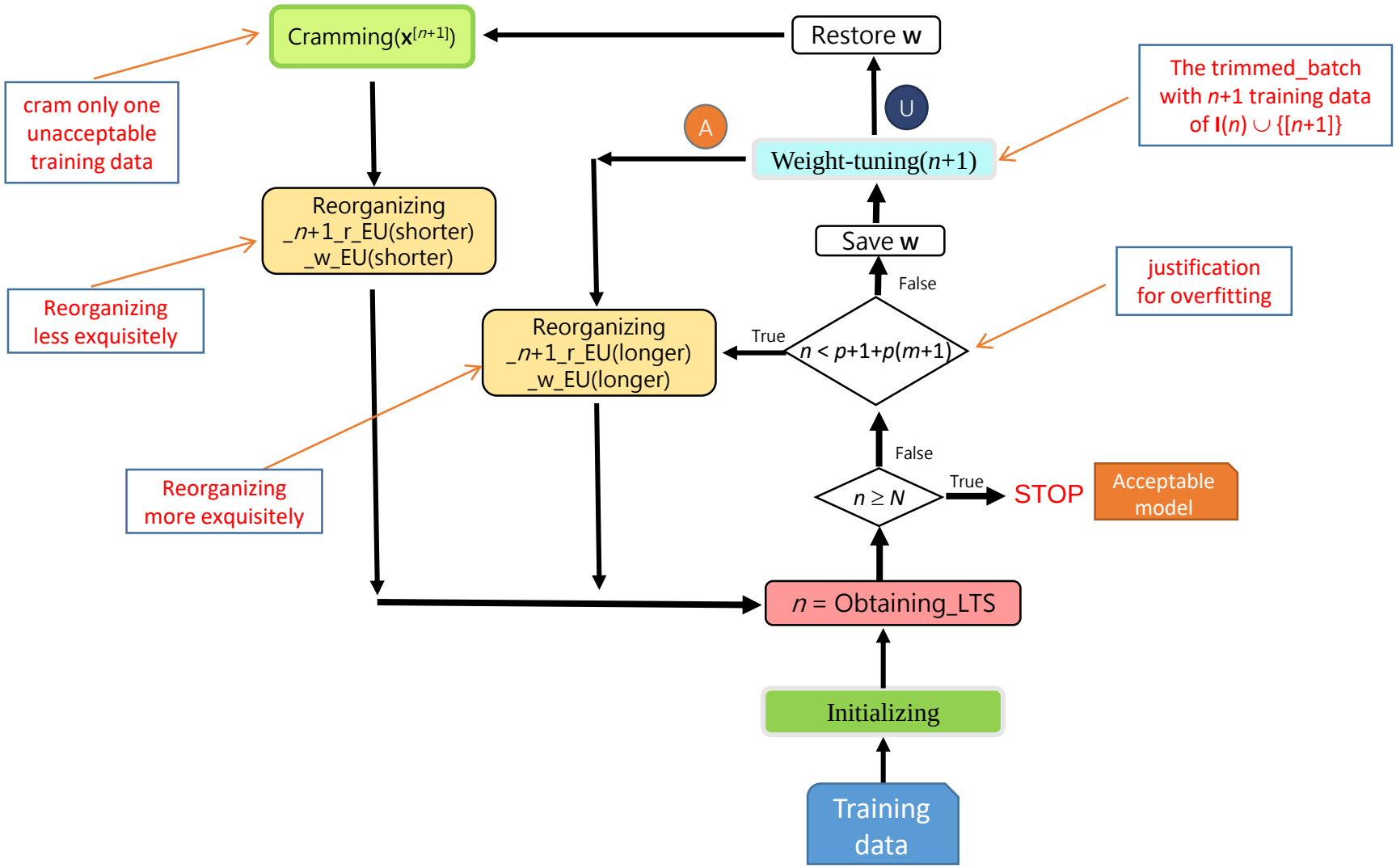
Step 2: $n = \arg\max_c((e^{[c]})^2 \leq \varepsilon^2)$.

Step 3: Obtain the n training data $\{(\mathbf{x}^c, y^c)\}$ that are the ones with the smallest n squared residuals among current N squared residuals.

Step 4: Let $\mathbf{I}(n)$ be the set of indices of these data.

$e^c \equiv f(\mathbf{x}^c, \mathbf{w}) - y^c$

The second new learning mechanism
(in flowchart)



The third new learning mechanism
(in nature language)

- Select the training data one by one according to the LTS principle.

  Small errors first, larger errors later

- When a new training data is presented, check first to see whether it is acceptable.
  - ✓ If acceptable, reorganize the network to more concisely integrate the learnt knowledge.
  - ✓ If unacceptable, weight-tune the network to learn it.
    - ➢ If it can be successfully learned, then reorganize the network to more concisely integrate the learnt knowledge.
    - ➢ If it cannot be successfully learned, then cram it. Later, reorganize the network to more concisely integrate the learnt knowledge.

The selecting module_LTS

At the n^{th} stage,

Step 1: Sort all training data $\{(\mathbf{x}^c, y^c)\ c \in \mathbf{I}(N)\}$ by their squared residuals in ascending order as $(e^{[1]})^2 \leq (e^{[2]})^2 \leq \ldots \leq (e^{[N]})^2$.

Step 2: Select the n training data $\{(\mathbf{x}^c, y^c)\}$ that are the ones with the smallest n squared residuals among current N squared residuals.

Step 3: Let $\mathbf{I}(n)$ be the set of indices of these data.

$e^c \equiv f(\mathbf{x}^c, \mathbf{w}) - y^c$

The third new learning mechanism
(in Pseudocode)

Step 1: Initialize an SLFN with one hidden node. — Initializing

Step 2: n = obtaining_LTS and then n+1→ n. — Obtaining

Step 3: If $n > N$, STOP. — The stopping criterion of the learning mechanism

Step 4: Run Selecting_LTS(n). Let $\mathbf{I}(n)$ be the set of indices of the picked data. — Selecting

Step 5: If the learning goal regarding $\{f(\mathbf{x}^c, \mathbf{w}), \forall\ c \in \mathbf{I}(n)\}$ is satisfied, go to Step 8; otherwise, there is one and only one $\kappa \in \mathbf{I}(n)$ that causes the contradiction and $\kappa = [n]$. — Check the learning goal

Step 6: Save $\mathbf{w}$.

Step 7: Weight tune the current SLFN. At the end, — Weight-tuning

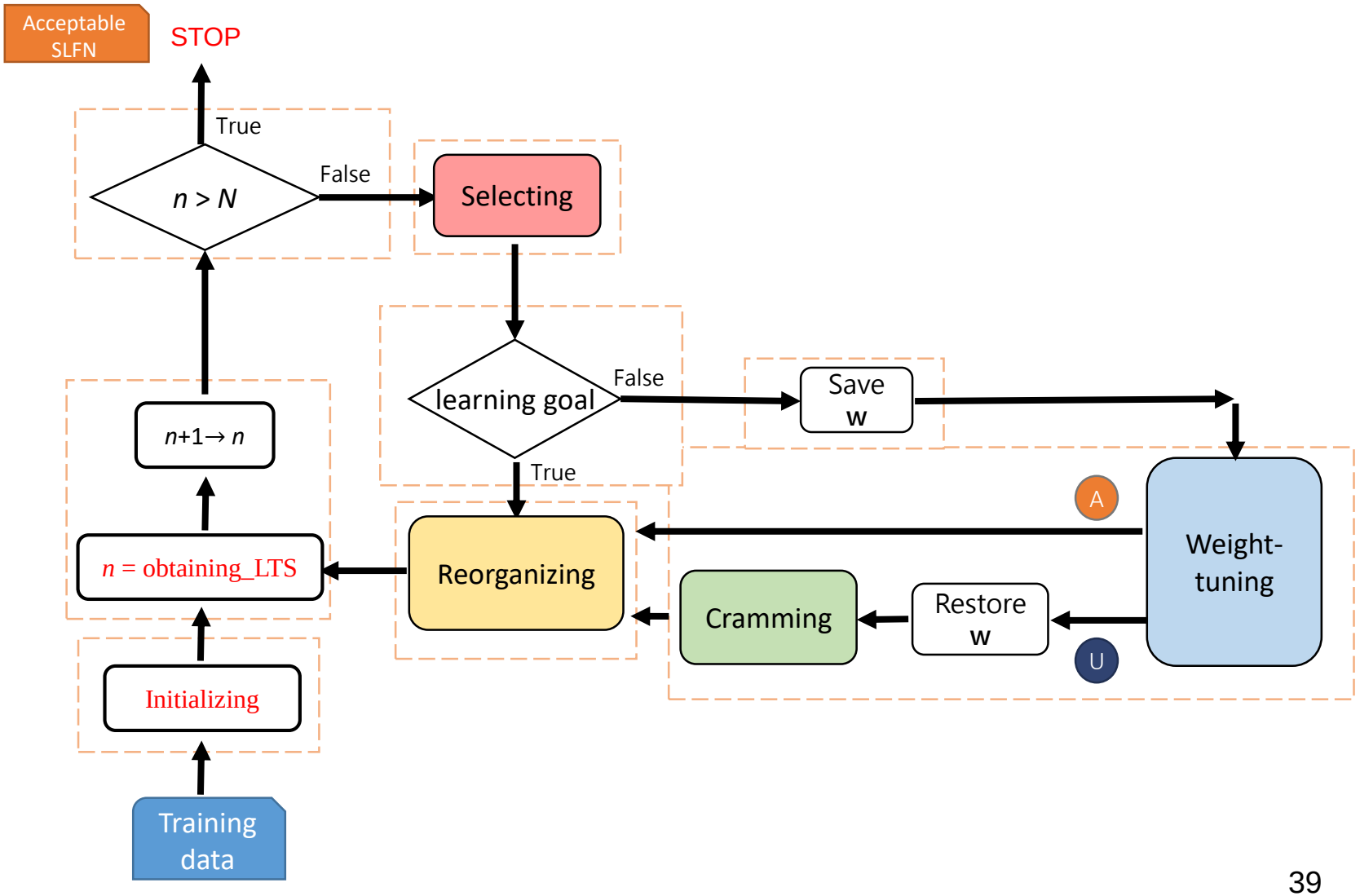
(1) if the obtained SLFN is acceptable, go to Step 8;

(2) otherwise, restore $\mathbf{w}$ and then cram $(\mathbf{x}^{[n]}, y^{[n]})$ to obtain a new acceptable SLFN. — Cramming

Step 8: Reorganize the SLFN to regularize the weights and then identify and remove the potentially irrelevant hidden node. — Reorganizing

Step 9: Go to Step 2. — Loop

The third new learning mechanism
(in flowchart)



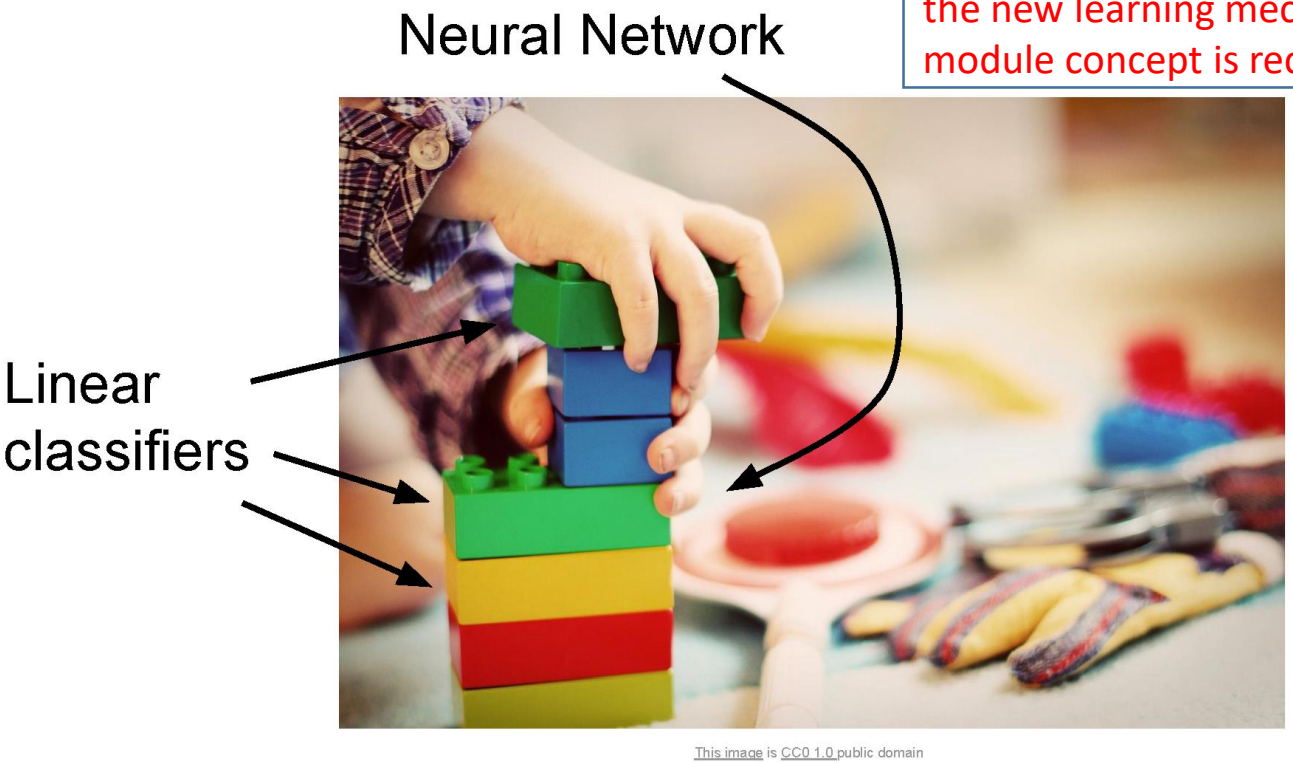
Homework #4

1. Pick up one of the following data files Copper_forecasting_data.csv (real number inputs) or SPECT_data.txt (binary number inputs) to decide your AI application problem.
2. Pick up one of the learning mechanisms stated in page 25, page 35 or page 39.
3. Based upon the picked-up, write down the details of your own learning mechanism with ppt.
4. Based upon the details of your own learning mechanism, make the coding of your own learning mechanism with PyTorch or TensorFlow.

- Note that the learning goals used in all modules (i.e., the initializing module, the obtaining module, the selecting module, the weight-tuning module, the cramming module, the regularizing module, and the reorganizing module) should be consistent.

Developing a new learning algorithm is like playing with Lego – lots of (pre-built or self-built) modules

For the AI application, some AI framework (e.g., PyTorch or TensorFlow) is used to implement the new learning mechanism. Thus, using the module concept is recommended.



The module list

- ✓Weight-tuning
- ✓Regularizing
- ✓Reorganizing

Optimization mechanisms: much harder to be proved by mathematical proofs, but much easier to be tried by CS code.

- ✓Cramming
- ✓Initializing
- ✓Obtaining
- ✓Selecting
- ✓…

Rule-based mechanisms: much easier to be proved by mathematical proofs, but the validation is usually required.

Two-layer nets / SLFN with one output node

The notations and indexes

- $ReLU(x) \equiv max(0, x)$;
- N: the number of data;
- m: the number of input nodes;
- $\mathbf{x}^c \equiv (x_1^c, x_2^c, \ldots, x_m^c)^{\mathrm{T}}$: the c^{th} input;
- p: the number of adopted hidden nodes; p is adaptable within the training phase;
- $w_{i,0}^{\mathrm{H}}$: the bias value of i^{th} hidden node;
- $w_{i,j}^{\mathrm{H}}$: the weight between the j^{th} input node and the i^{th} hidden node, j = 1, 2, ..., m;
- $\mathbf{w}_i^{\mathrm{H}} \equiv (w_{i,1}^{\mathrm{H}}, w_{i,2}^{\mathrm{H}}, \ldots, w_{i,m}^{\mathrm{H}})^{\mathrm{T}}$, i=1, 2, …, p;
- $\mathbf{w}^H \equiv (\mathbf{w}_1^H, \mathbf{w}_2^H, \ldots, \mathbf{w}_p^H)^{\mathrm{T}}$;
- $\mathbf{w}_0^H \equiv (w_{1,0}^H, w_{2,0}^H, \ldots, w_{p,0}^H)^{\mathrm{T}}$;
- w_0^o: the bias value of output node;
- w_i^o: the weight between the i^{th} hidden node and the output node;
- $\mathbf{w}^o \equiv (w_1^o, w_2^o, \ldots, w_p^o)^{\mathrm{T}}$;
- $\mathbf{w} \equiv \{\mathbf{w}^H, \mathbf{w}_0^H, \mathbf{w}^o, w_0^o\}$;
- a_i^c: the activation value of i^{th} hidden node corresponding to $\mathbf{x}^c$;
- $\mathbf{a}^c \equiv (a_1^c, a_2^c, \ldots, a_p^c)^{\mathrm{T}}$;
- $f(\mathbf{x}^c, \mathbf{w}) \in \mathrm{R}$: the output value of SLFN corresponding to $\mathbf{x}^c$;
- y^c: the desired output value corresponding to $\mathbf{x}^c$;
- $e^c \equiv f(\mathbf{x}^c, \mathbf{w}) - y^c$.

Should specify (binary or real numbers)

The adaptive SLFN, if you adopt the new learning mechanism

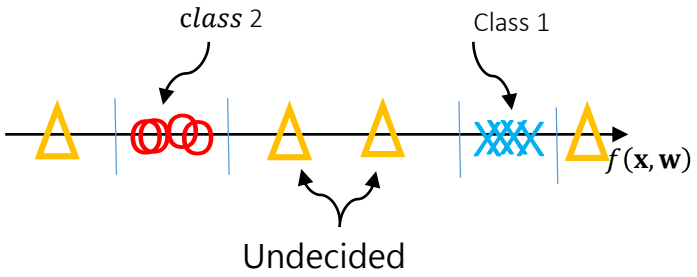
Depend on the application!

Different stopping criteria result in different length of training time and different model.

Should specify (binary or real numbers)

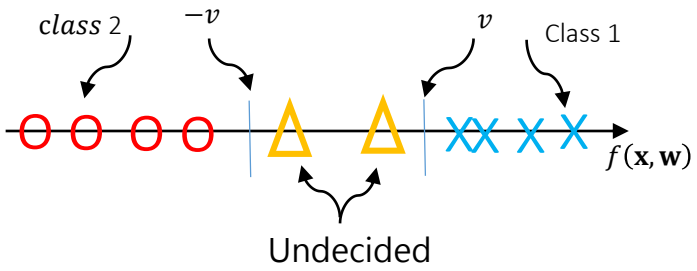
Stopping criteria (also the learning goals) for the SLFN with each output node whose output values are real numbers for the two-class classification application

$y^c = 1 \; \forall \; c \in \mathbf{I}_1; y^c = 0 \; \forall \; c \in \mathbf{I}_2$ X : $f(\mathbf{x}^c, \mathbf{w}), \; \forall \; c \in \mathbf{I}_1$ O : $f(\mathbf{x}^c, \mathbf{w}), \; \forall \; c \in \mathbf{I}_2$



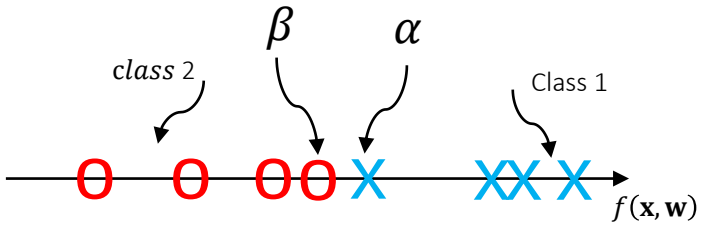
learning goal type 1
(also inferencing mechanism):
$|f(\mathbf{x}^c, \mathbf{w}) - y^c| \leq \varepsilon \; \forall \; c \in \mathbf{I}_1;$
$|f(\mathbf{x}^c, \mathbf{w}) + y^c| \leq \varepsilon \; \forall \; c \in \mathbf{I}_2$

ε Is a hyperparameter regarding the learning!



learning goal type 2
(also inferencing mechanism):
$f(\mathbf{x}^c, \mathbf{w}) \geq \nu \; \forall \; c \in \mathbf{I}_1;$
$f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \; \forall \; c \in \mathbf{I}_2$

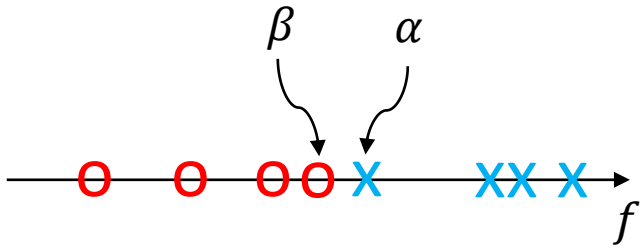
ν Is a hyperparameter regarding the learning and the inferencing!



learning goal type 3: LSC
inferencing mechanism:
$f(\mathbf{x}^c, \mathbf{w}) \geq \nu \; \forall \; c \in \mathbf{I}_1;$
$f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \; \forall \; c \in \mathbf{I}_2$

ν Is a hyperparameter regarding the learning!

Stopping criteria (also the learning goals) for the SLFN with each output node whose output values are real numbers for the two-class classification application



$$\alpha \equiv \min_{c \epsilon \mathbf{I}_1} f(\mathbf{x}^c, \mathbf{w}) \,;\; \beta \equiv \max_{c \epsilon \mathbf{I}_2} f(\mathbf{x}^c, \mathbf{w})$$

learning goal type 3: LSC

When LSC ($\alpha > \beta$) is true, the inferencing mechanism

$$f(\mathbf{x}^c, \mathbf{w}) \geq v \;\; \forall\, c \in \mathbf{I}_1 \text{ and } f(\mathbf{x}^c, \mathbf{w}) \leq -v \;\forall\, c \in \mathbf{I}_2$$

can be set by directly adjusting $\mathbf{w}^o$ according to the following formula:

$$\frac{2v}{\alpha - \beta} w_i^o \rightarrow w_i^o \;\; \forall\, i,$$

The weight vector between the hidden layer and the output node

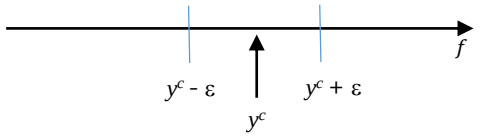
then $v - \min_{c \in \mathbf{I}_1} \sum_{i=1}^{p} w_i^o a_i^c \rightarrow w_0^o$

The threshold of the output node

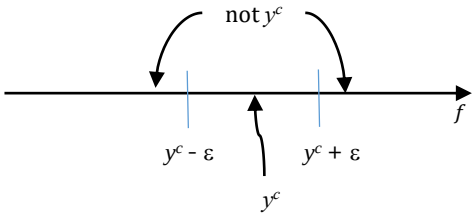
The regression applications

The learning goal

$$|f(\mathbf{x}^c, \mathbf{w}) - y^c| \leq \varepsilon \;\forall\; c \in \mathbf{I}$$



The inferencing mechanism

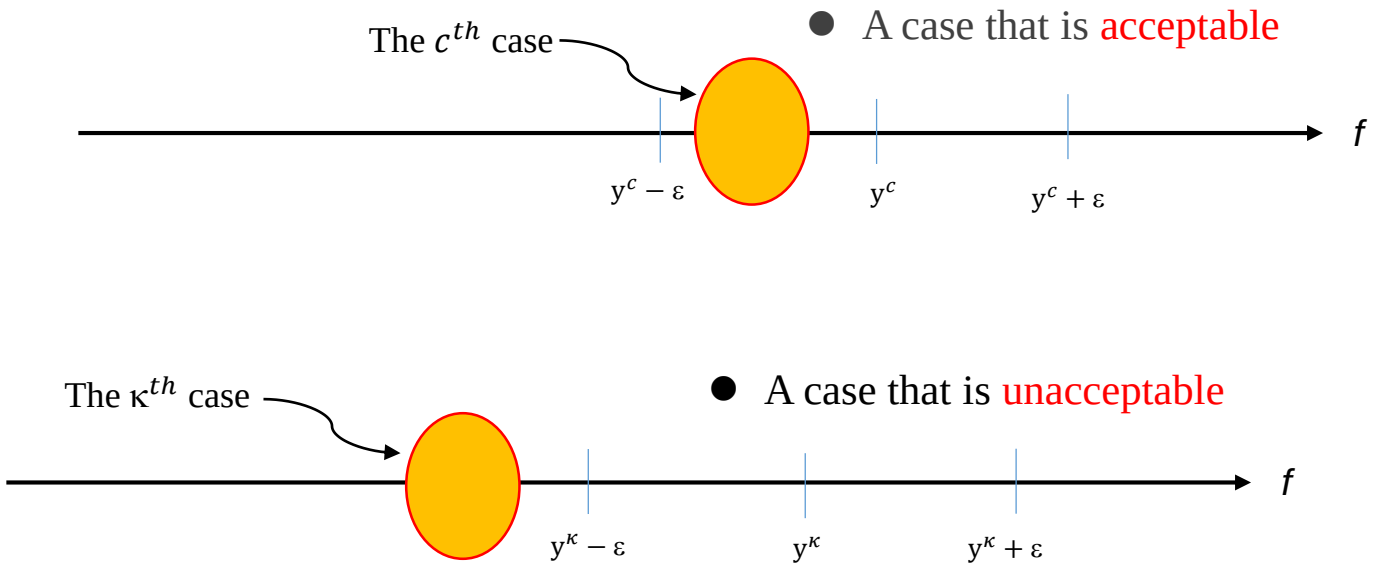


This learning goal and the associated inferencing mechanism is similar to LGT1: $|f(\mathbf{x}^c, \mathbf{w}) - 1| \leq \varepsilon \;\forall\; c \in \mathbf{I}_1$ and $|f(\mathbf{x}^c, \mathbf{w})| \leq \varepsilon \;\forall\; c \in \mathbf{I}_2$

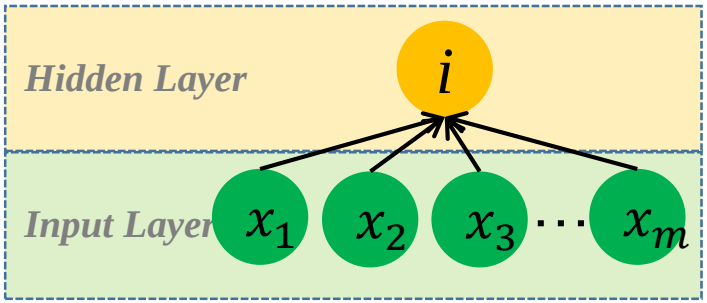
The unacceptable case

The acceptability of each case is related with the learning goal.

For example, the learning goal is $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \; \forall \; c \in \mathbf{I}$



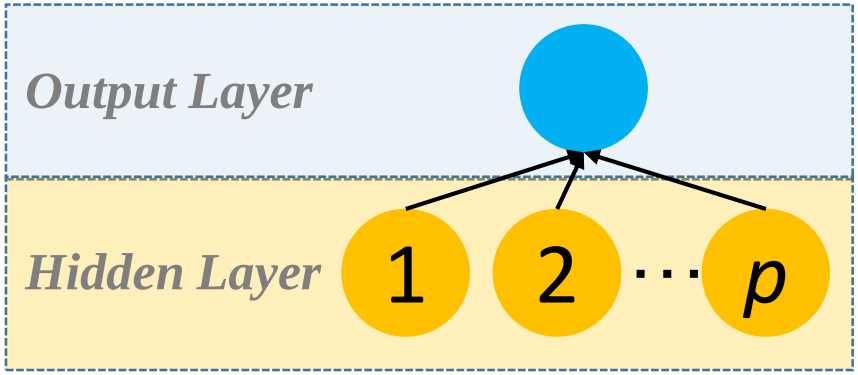
The forward operation SLFN with one output node



The hidden layer:

$$a_i^c \equiv ReLU\left(w_{i0}^H + \sum_{j=1}^{m} w_{ij}^H x_j^c\right)$$

$$\mathbf{a} \equiv ReLU(\mathbf{W}^H\mathbf{x} + \mathbf{w}_0^H)$$



The output layer:

$$f(\mathbf{x}^c,\mathbf{w}) \equiv w_0^o + \sum_{i=1}^{p} w_i^o a_i^c$$

$$\boldsymbol{f}(\mathbf{x}^c, \mathbf{w}) \equiv \mathbf{W}^o\mathbf{a} + \mathbf{w}_0^o$$

$E_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c\in\mathbf{I}}(f(\mathbf{x}^c,\mathbf{w})-y^c)^2$: the loss function;

$E_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c\in\mathbf{I}}(f(\mathbf{x}^c,\mathbf{w})-y^c)^2 + \lambda(\sum_{i=0}^{p}(w_i^o)^2 + \sum_{i=1}^{p}\sum_{j=0}^{m}(w_{ij}^H)^2)$: the loss function with the regularization term.

An optimization mechanism

The weight-tuning module

- The weight-tuning module helps tune up the weights to decrease the loss function value to obtain an acceptable SLFN.
- There are four weight-tuning modules
  - ✓ the weight-tuning module_EU

The simplest and the learning time length is expected

  - ✓ the weight-tuning module_EU_LG

Shorter learning time length than the weight-tuning module_EU

  - ✓ the weight-tuning module_EU_LG_UA

The learning time length may be longer than the weight-tuning module_EU_LG

  - ✓ the weight-tuning module_LG_UA

The learning time length is not an issue

An optimization mechanism

The regularizing module

- After obtaining an acceptable SLFN, the regularizing module helps further regularize weights of the acceptable SLFN while keeping the learning goal satisfied.
- A well-regularized SLFN can alleviate the overfitting tendency.

✓the regularizing module_LG_UA
The regularizing time length is expected

✓the regularizing module_EU_LG_UA
The regularizing time length may be much longer

✓the regularizing module_EU
The simplest and the regularizing time length is expected

✓the regularizing module_DO

✓the regularizing module_BN

✓Your creative idea

An optimization mechanism

The reorganizing module

The reorganizing module helps regularize weights of an acceptable SLFN and then identify and prune some potentially irrelevant hidden nodes.

- ✓ The reorganizing module_ALL_r_EU_LG_UA_w_EU_LG_UA that regularizes weights of an acceptable SLFN while keeping the learning goal satisfied as well as examines all hidden nodes one by one to see any of them is potentially irrelevant. Remove potentially irrelevant hidden nodes identified within the process.
- ✓ The reorganizing module_R3_r_EU_LG_UA_w_EU_LG_UA that regularizes weights of an acceptable SLFN while keeping the learning goal satisfied as well as randomly picks up 3 hidden nodes and examines whether they are potentially irrelevant. Remove potentially irrelevant hidden nodes identified within the process.
- ✓ The reorganizing module_PCA_r_EU_LG_UA_w_EU_LG_UA that regularizes weights of an acceptable SLFN while keeping the learning goal satisfied as well as uses PCA to pick up a hidden node and examines whether it is potentially irrelevant. If yes, remove it and then repeat the process; otherwise, stop the process.
- ✓ The reorganizing module_MAW_r_EU_LG_UA_w_EU_LG_UA that regularizes weights of an acceptable SLFN while keeping the learning goal satisfied as well as uses $k = \arg\min_i |w_i^o|$ to pick up a hidden node and examines whether it is potentially irrelevant. If yes, remove it and then repeat the process; otherwise, stop the process.
- ✓ The reorganizing module_ETP_r_EU_LG_UA_w_EU_LG_UA that regularizes weights of an acceptable SLFN while keeping the learning goal satisfied as well as calculates the entropy of each hidden node and then, based on the obtained entropy, picks up a hidden node and examines whether it is potentially irrelevant. If yes, remove it and then repeat the process; otherwise, stop the process.
- ✓ Your creative idea

A rule-based mechanism

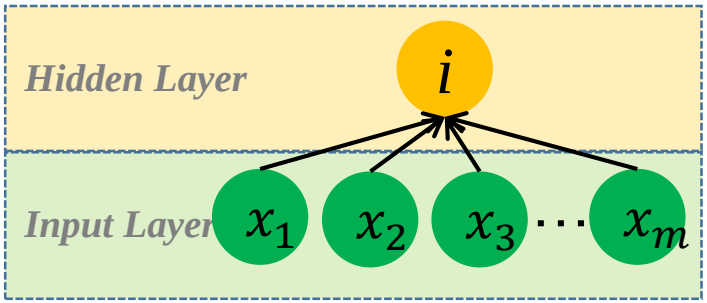
The cramming module for SLFN with single output node

The cramming module helps add extra hidden nodes with proper weights to the existing SLFN to make the learning goal satisfied immediately.

- ✓The cramming module_ReLU_BI_SO_LGT1_SU; The cramming module_ReLU_RI_SO_LGT1_SU
- ✓The cramming module_ReLU_BI_SO_LGT3_SU; The cramming module_ReLU_RI_SO_LGT3_SU
- ✓The cramming module_ReLU_BI_SO_RE_SU; The cramming module_ReLU_RI_SO_RE_SU
- ✓The cramming module_ReLU_BI_SO_RE_MU; The cramming module_ReLU_RI_SO_RE_MU
- ✓The cramming module_ReLU_BI_SO_LGT1_MU; The cramming module_ReLU_RI_SO_LGT1_MU
- ✓The cramming module_ReLU_BI_SO_LGT3_MU; The cramming module_ReLU_RI_SO_LGT3_MU
- ✓Your creative idea

You may derive the red parts by yourself.

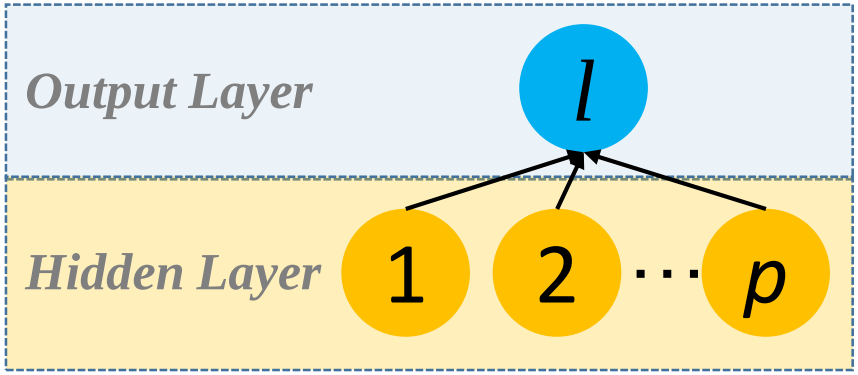
The forward operation SLFN with multiple output nodes



The hidden layer:

$$a_i^c \equiv ReLU\left(w_{i0}^H + \sum_{j=1}^{m} w_{ij}^H x_j^c\right)$$

$$\mathbf{a} \equiv ReLU(\mathbf{W}^H\mathbf{x} + \mathbf{w}_0^H)$$



The output layer:

$$f_l(\mathbf{x}^c,\mathbf{w}) \equiv w_{l0}^o + \sum_{i=1}^{p} w_{li}^o a_i^c$$

$$\boldsymbol{f}(\mathbf{x}^c, \mathbf{w}) \equiv \mathbf{W}^o\mathbf{a} + \mathbf{w}_0^o$$

$E_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c\in\mathbf{I}}\sum_{l=1}^{q}(f_l(\mathbf{x}^c,\mathbf{w})-y_l^c)^2$: the loss function;

$E_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c\in\mathbf{I}}\sum_{l=1}^{q}(f_l(\mathbf{x}^c,\mathbf{w})-y_l^c)^2 + \lambda(\sum_{l=1}^{q}\sum_{i=0}^{p}(w_{li}^o)^2 + \sum_{i=1}^{p}\sum_{j=0}^{m}(w_{ij}^H)^2)$: the loss function with the regularization term.

A rule-based mechanism

The cramming module for SLFN with multiple output nodes

The cramming module helps add extra hidden nodes with proper weights to the existing SLFN to make the learning goal satisfied immediately.

- ✓The cramming module_ReLU_BI_MO_RE_SU; The cramming module_ReLU_RI_MO_RE_SU
- ✓The cramming module_ReLU_BI_MO_LGT1_SU; The cramming module_ReLU_RI_MO_LGT1_SU
- ✓The cramming module_ReLU_BI_MO_LGT3_SU; The cramming module_ReLU_RI_MO_LGT3_SU
- ✓The cramming module_ReLU_BI_MO_RE_MU; The cramming module_ReLU_RI_MO_RE_MU
- ✓The cramming module_ReLU_BI_MO_LGT1_MU; The cramming module_ReLU_RI_MO_LGT1_MU
- ✓The cramming module_ReLU_BI_MO_LGT3_MU; The cramming module_ReLU_RI_MO_LGT3_MU
- ✓Your creative idea

You may derive the red parts by yourself.