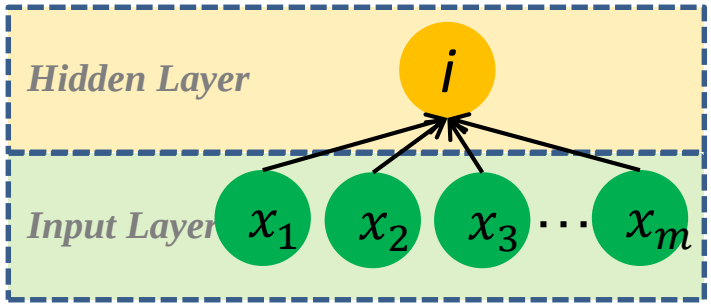
Coping with the learning dilemma issue: cramming with ReLU for binary-number inputs

國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

Where we are now...

The AI application →
ideas/concepts →
(pre-existed & extra) modules →
new learning algorithm →
codes →
intelligent system for the AI application

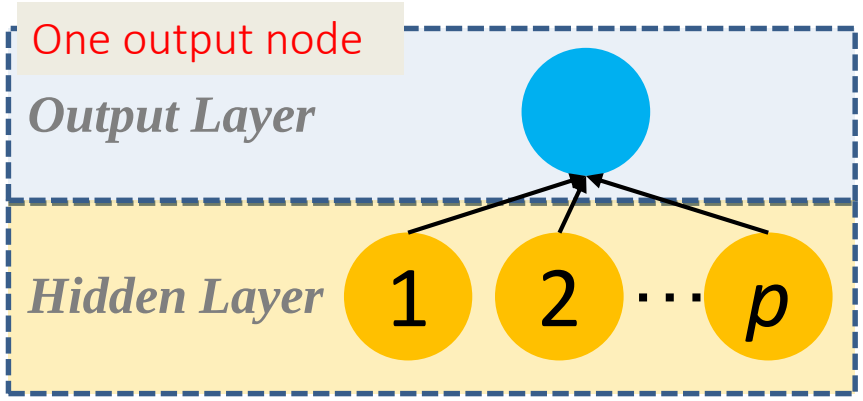
Activation Values of Nodes



The activation value of i^{th} hidden node:

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

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



The value of the output node:

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

Output Layer: $f \equiv \mathbf{w}^o\mathbf{a} + w_0^o$

Where we are now...

The notations and indexes

- $ReLU(x) \equiv max(0, x)$;
- N: the number of data;
- m: the number of input nodes;
- $\mathbf{x}^c \equiv (x_1^c, x_2^c, \ldots, x_m^c)^{\mathrm{T}} \in \mathrm{R}^m$: 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;
- $\mathbf{w}^o \equiv \left(w_1^o, w_2^o, \ldots, w_p^o\right)^{\mathrm{T}}$;
- $\mathbf{w} \equiv \{\mathbf{w}^H, \mathbf{w}_0^H, \mathbf{w}^o, w_0^o\}$;
- a_i^c: the activation value of i^{th} hidden node corresponding to $\mathbf{x}^c$;
- $\mathbf{a}^c \equiv \left(a_1^c, a_2^c, \ldots, a_p^c\right)^{\mathrm{T}}$;
- $f(\mathbf{x}^c, \mathbf{w})$: the output value of SLFN corresponding to $\mathbf{x}^c$;
- $y^c \in \mathrm{R}$: the target output value corresponding to $\mathbf{x}^c$;
- $e^c \equiv f(\mathbf{x}^c, \mathbf{w}) - y^c$.

The weight-tuning and the learning – Hebb's rule

- **Hebbian theory** is a neuroscientific theory claiming that an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell. It is an attempt to explain synaptic plasticity, the adaptation of brain neurons during the learning process.
- It was introduced by Donald Hebb in his 1949 book *The Organization of Behavior.* The theory is also called **Hebb's rule**, **Hebb's postulate**, and **cell assembly theory**. Hebb states it as follows:

Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. ... When an axon of cell *A* is near enough to excite a cell *B* and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that *A*' s efficiency, as one of the cells firing *B*, is increased.

- The theory is often summarized as "Cells that fire together wire together."

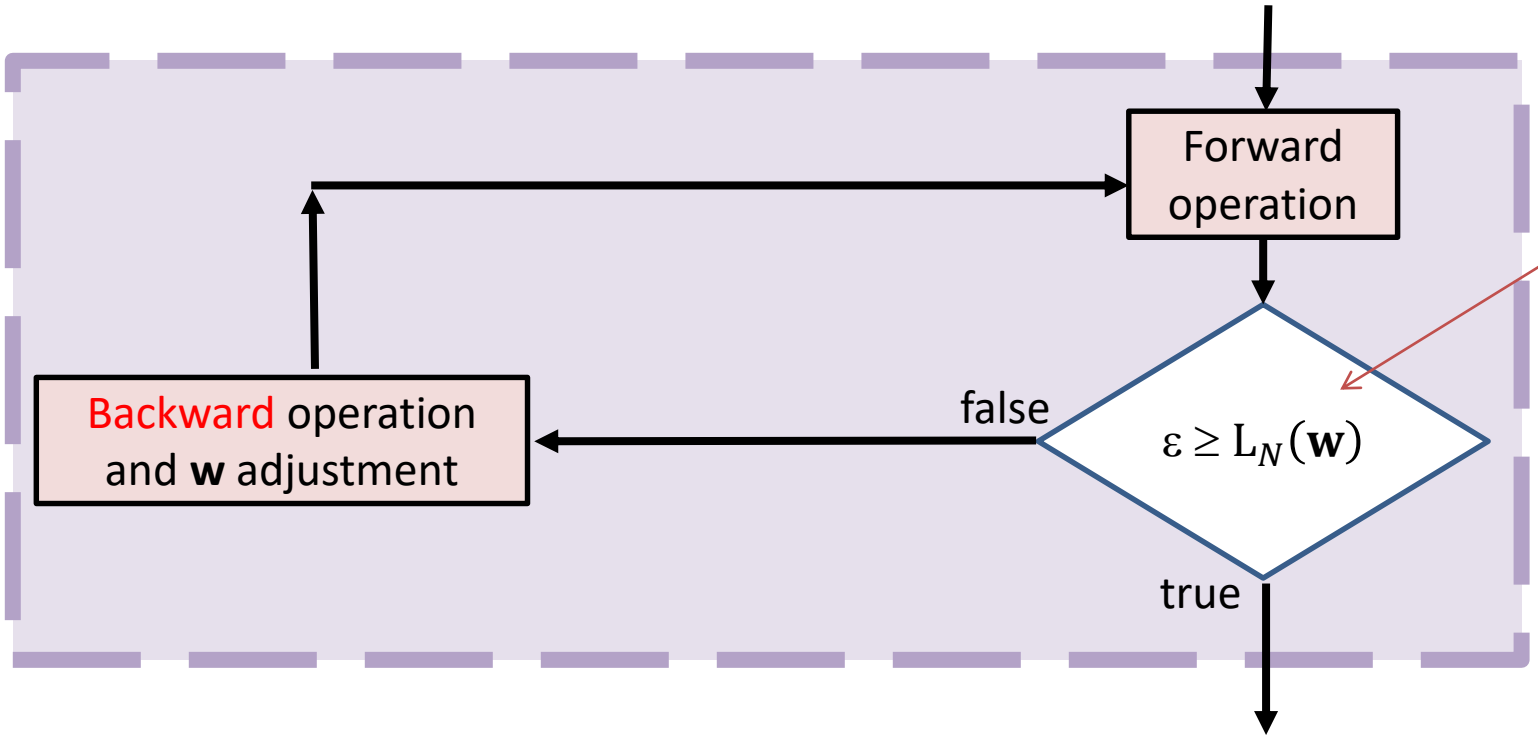
Hebbian theory - Wikipedia

The weight-tuning and the learning – back propagation learning algorithm

- In machine learning, **backpropagation** (**backprop**, **BP**) is a widely used algorithm for training feedforward neural networks. Generalizations of backpropagation exist for other artificial neural networks (ANNs), and for functions generally. These classes of algorithms are all referred to generically as "backpropagation."
- In fitting a neural network, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single input–output example, and does so efficiently, unlike a naive direct computation of the gradient with respect to each weight individually. This efficiency makes it feasible to use gradient methods for training multilayer networks, updating weights to minimize loss; gradient descent, or variants such as stochastic gradient descent, are commonly used. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.
- The term *backpropagation* strictly refers only to the algorithm for computing the gradient, not how the gradient is used; however, the term is often used loosely to refer to the entire learning algorithm, including how the gradient is used, such as by stochastic gradient descent. Backpropagation generalizes the gradient computation in the delta rule, which is the single-layer version of backpropagation, and is in turn generalized by automatic differentiation, where backpropagation is a special case of reverse accumulation (or "reverse mode").
- The term *backpropagation* and its general use in neural networks was announced in Rumelhart, Hinton & Williams (1986a), then elaborated and popularized in Rumelhart, Hinton & Williams (1986b), but the technique was independently rediscovered many times, and had many predecessors dating to the 1960s.

Backpropagation - Wikipedia

The weight-tuning module_BP



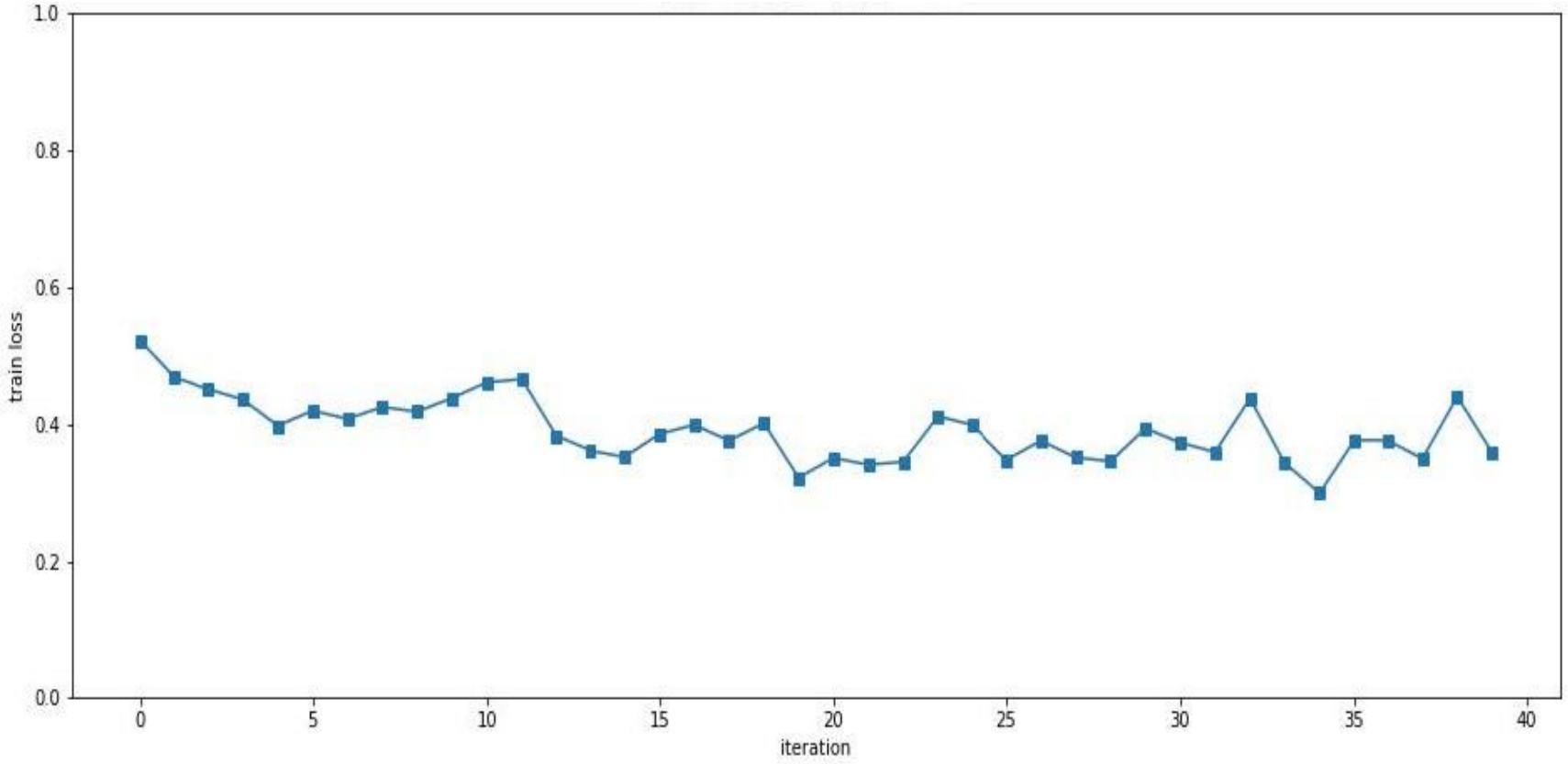
The stopping criterion is related with the loss function that sums up deviations of all training cases

Hyperparameters:
- p
- Optimizer: SGD
- Learning rate: 0.1
- ε

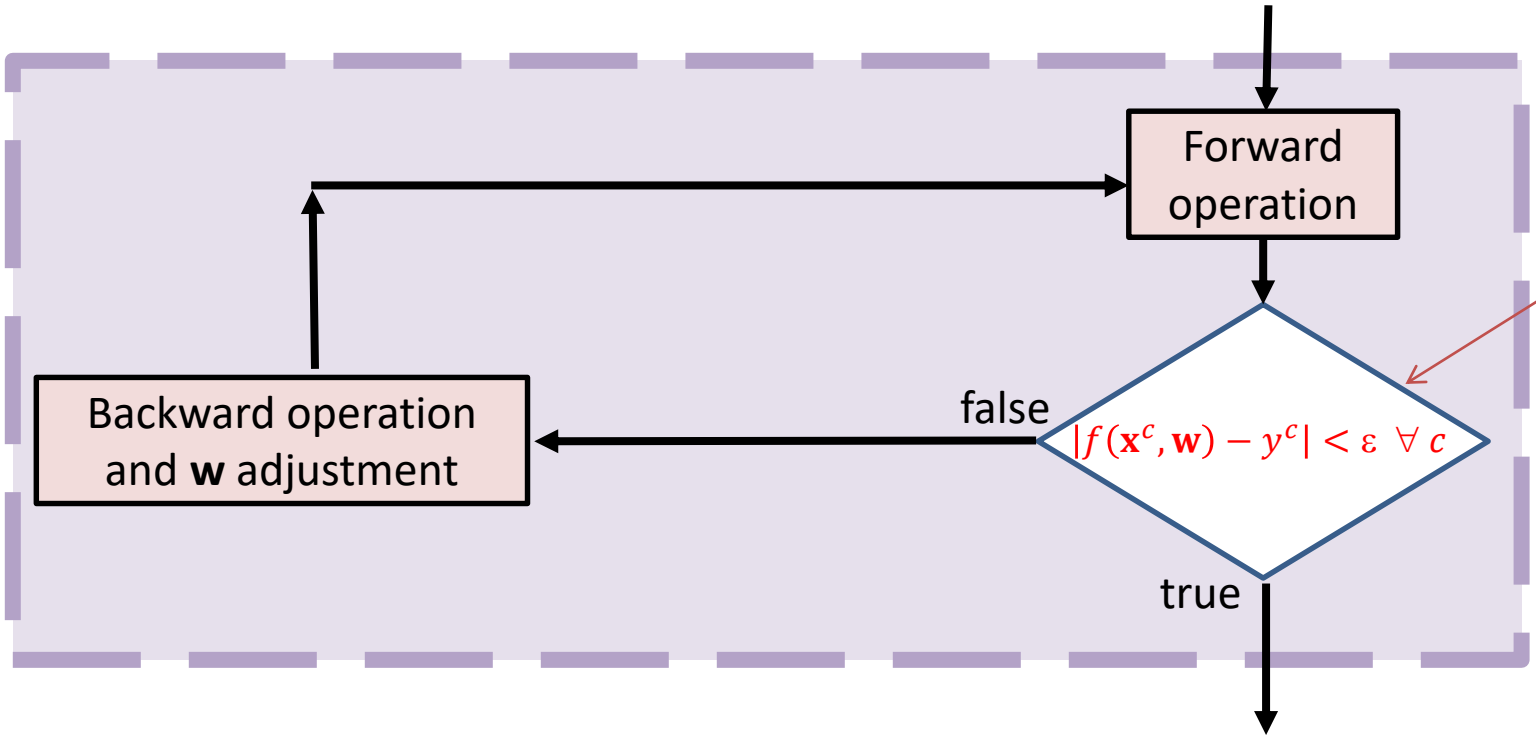
Q: Can this the learning process stop through satisfying the stopping criterion?
A: May not be! So, this stopping criterion is not good and thus this algorithm is not good.

Where we are now...

The evolution of loss function value



The weight-tuning module_BP



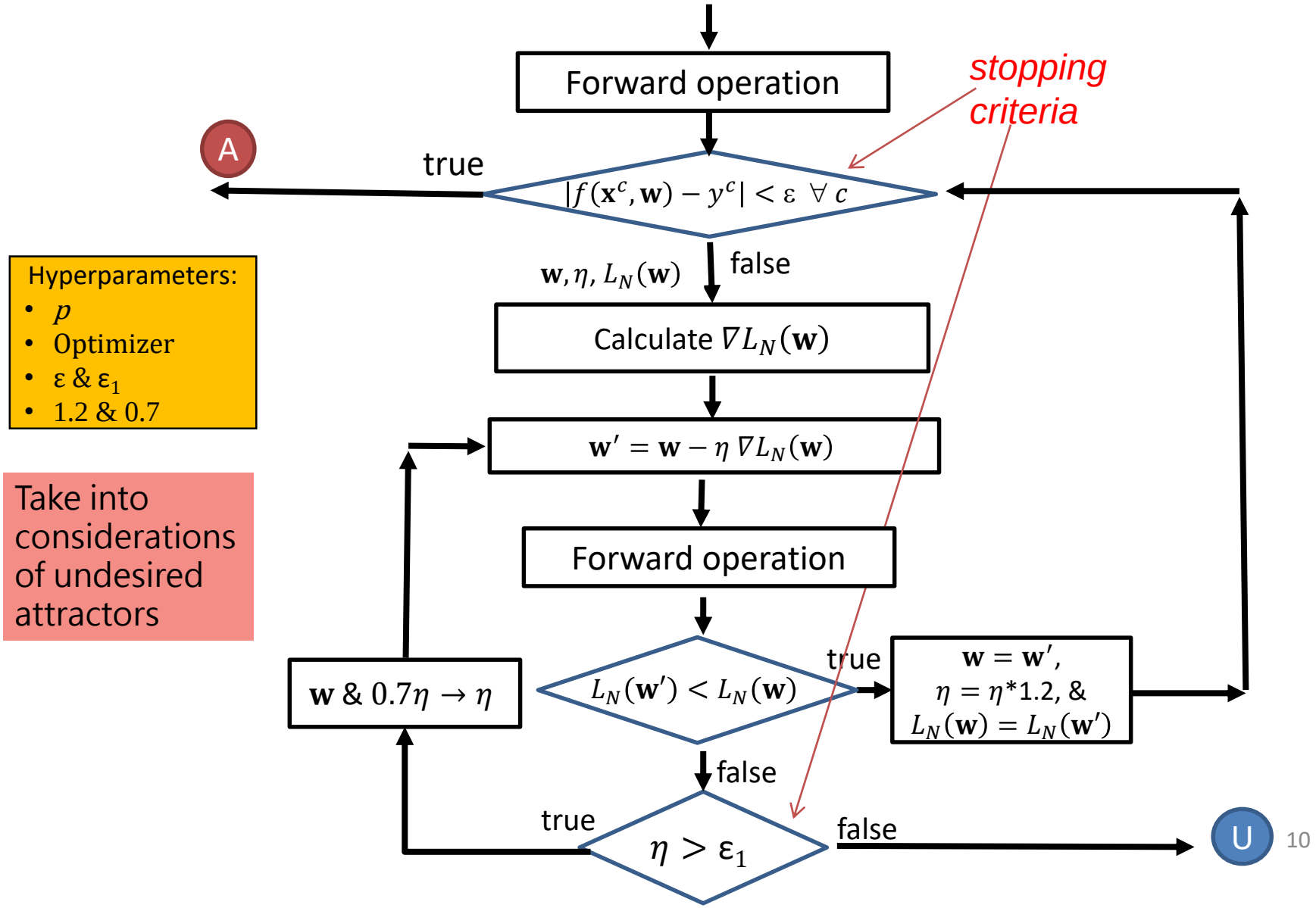
Focus on the deviation regarding each individual training case

Hyperparameters:

- p
- Optimizer: SGD
- Learning rate: 0.1
- ε

Where we are now...

The weight-tuning module_LG_UA

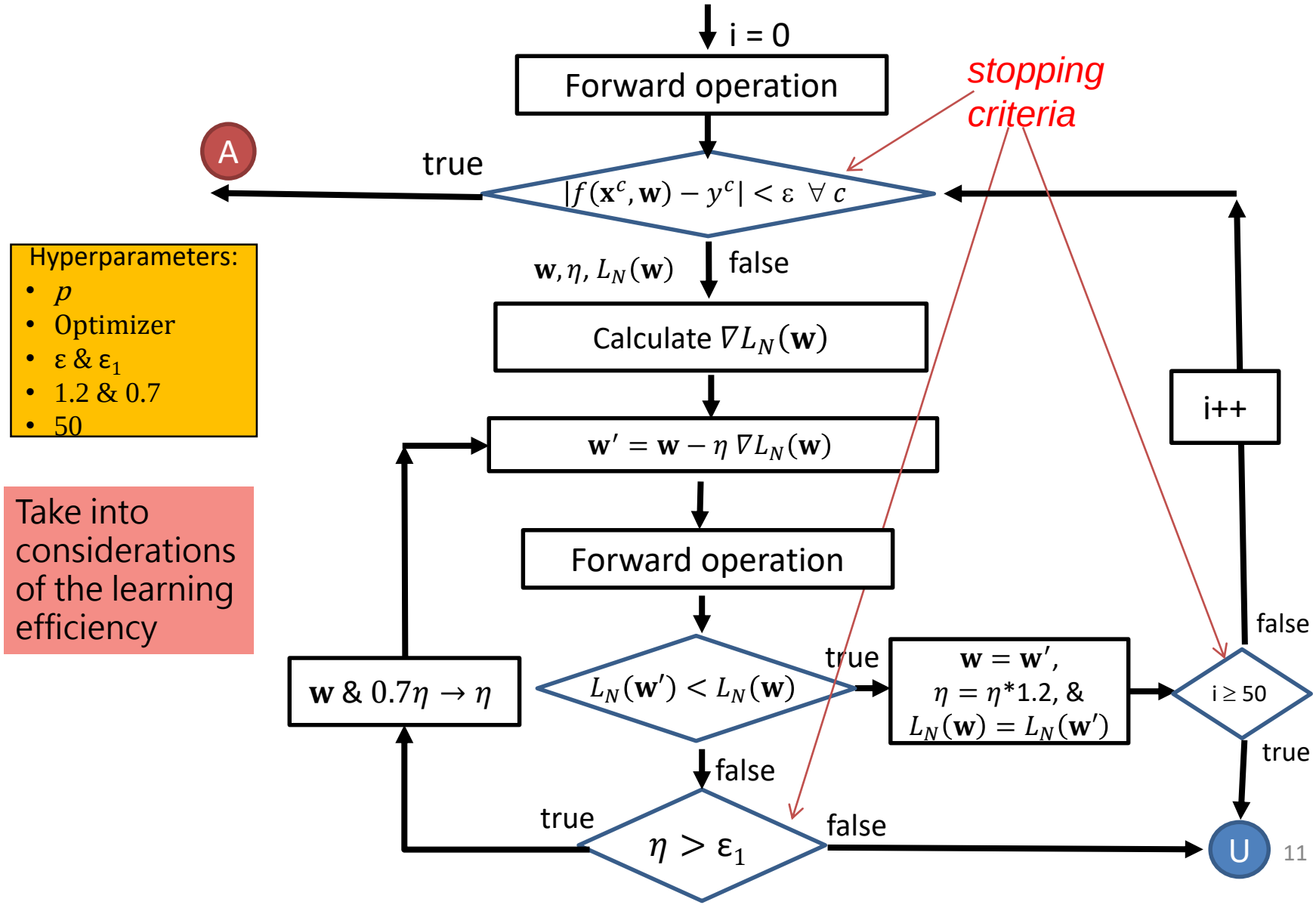


Hyperparameters:
- p
- Optimizer
- ε & ε_1
- 1.2 & 0.7

Take into considerations of undesired attractors

Where we are now...

The weight-tuning module_EU_LG_UA

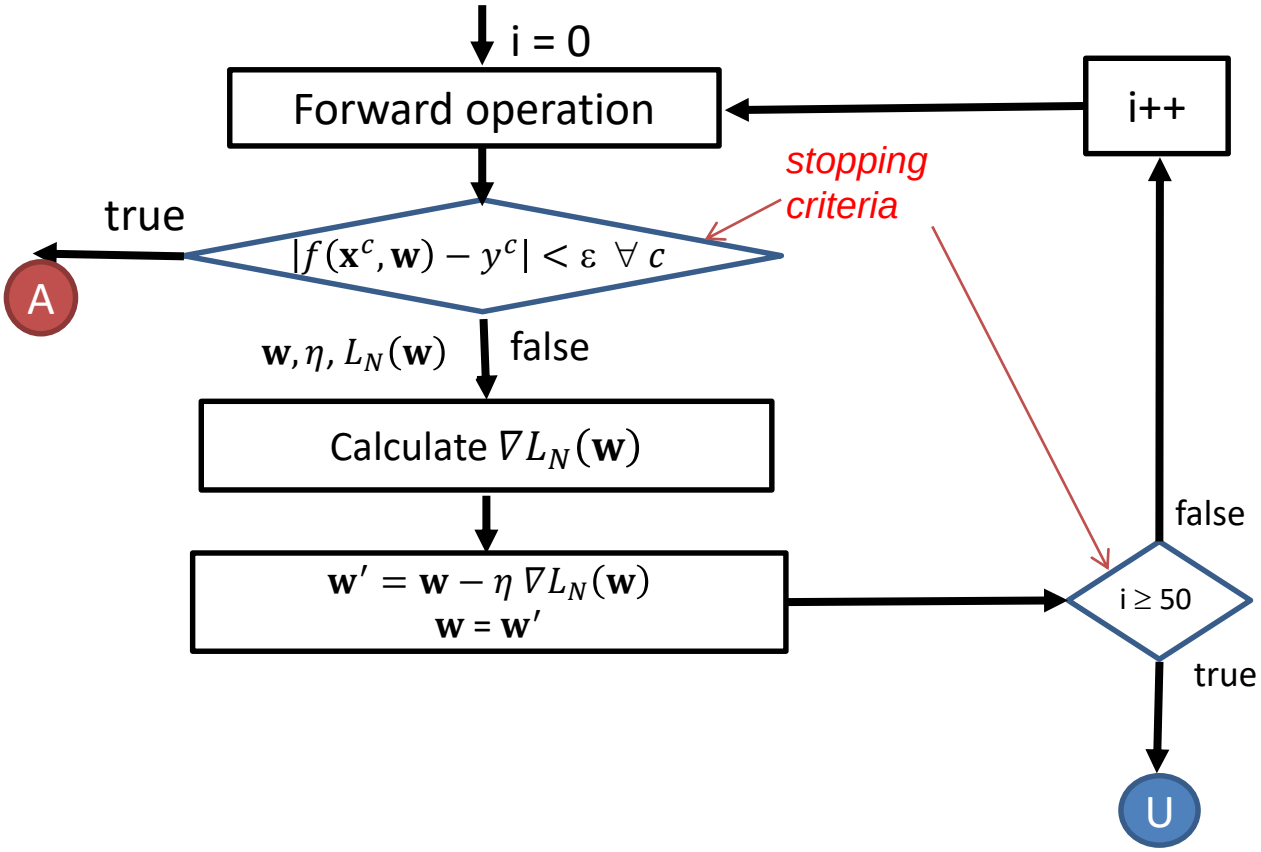


Where we are now...

The weight-tuning module_EU_LG

Hyperparameters:
- p
- Optimizer
- ε
- 50
- η

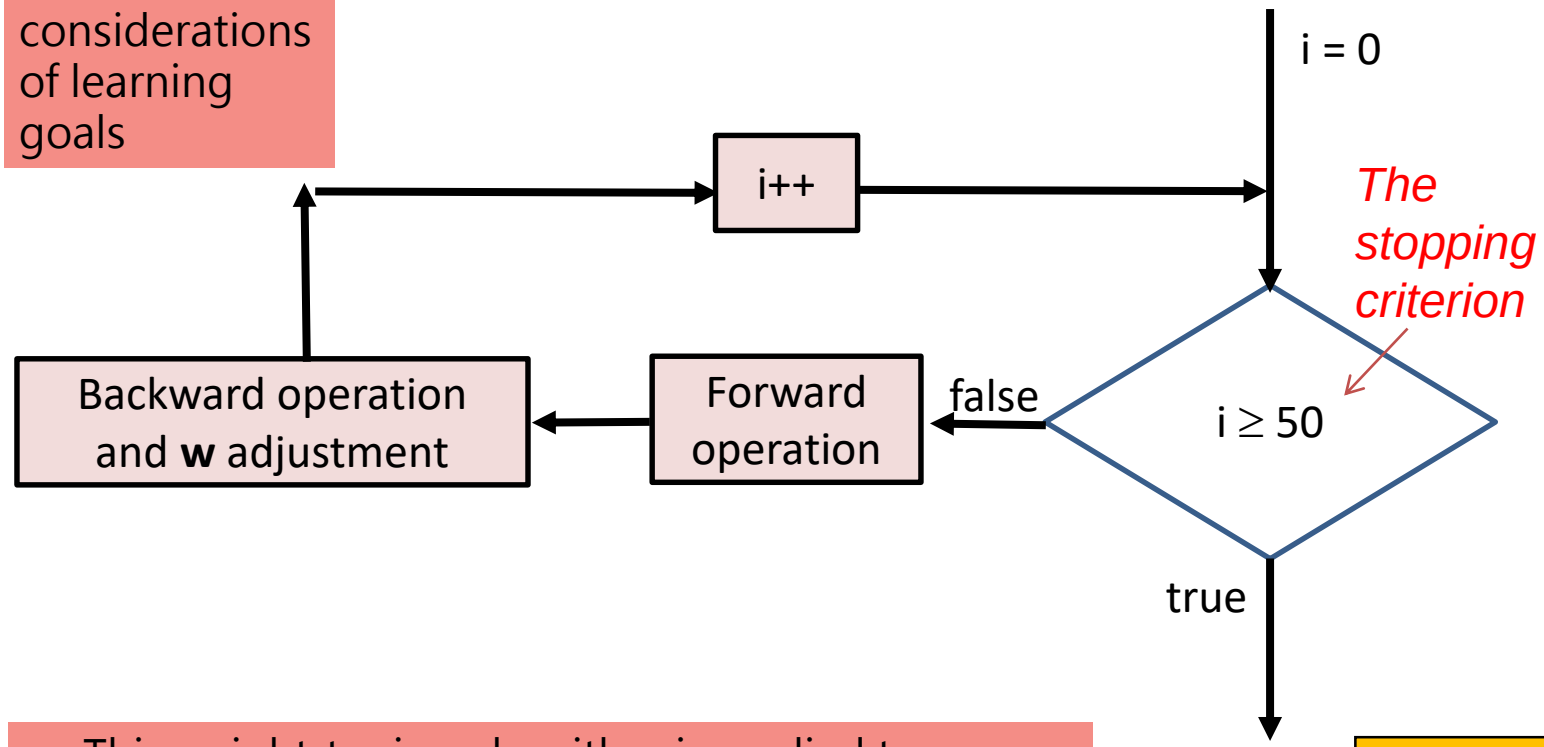
Release the considerations of undesired attractors



Where we are now...

The weight-tuning module_EU

Further release the considerations of learning goals



- This weight-tuning algorithm is applied to many kinds of Neural Networks, including 2-layer neural networks, CNN, RNN, reinforcement learning, GAN, BERT, and so on.
- The weight-tuning process stops when the stopping criterion is satisfied.

Hyperparameters:

- p
- Optimizer: SGD
- Epoch upper bound: 50
- Learning rate: 1e-6

The weight-tuning modules

- The weight-tuning module helps tune up the weights to decrease the data error 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

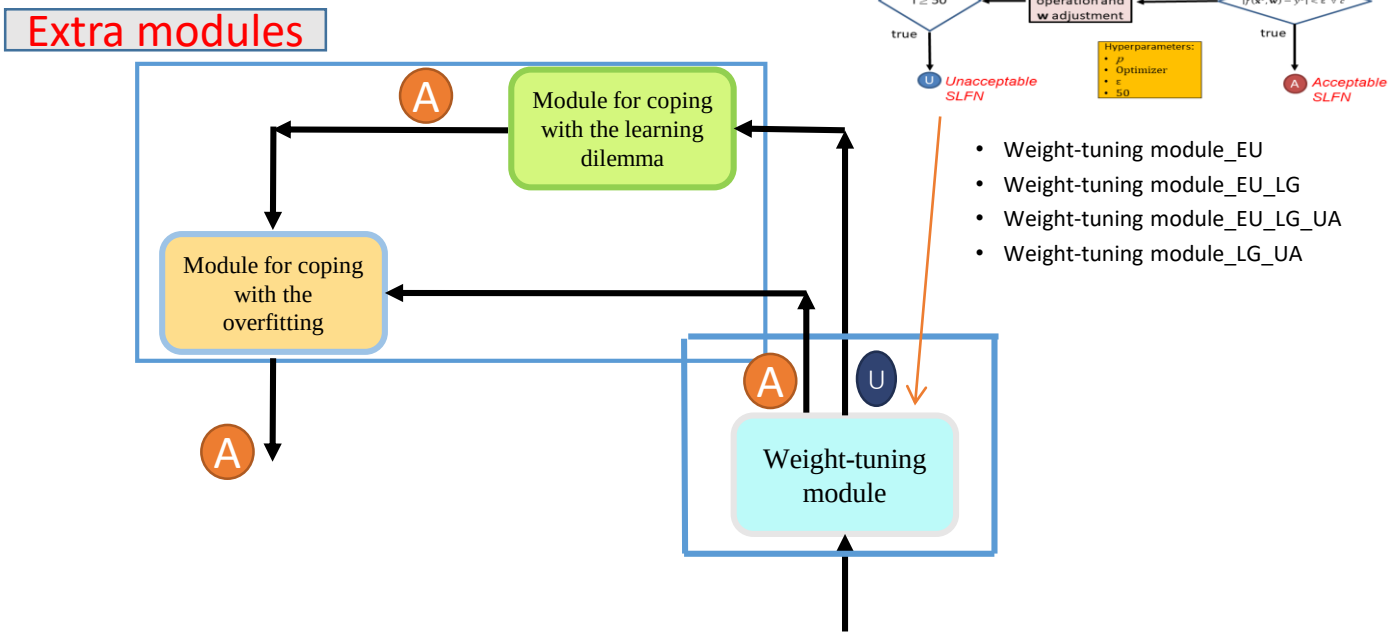
  - ✓ the weight-tuning module_EU_LG_UA

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

Cope with the overfitting and the learning dilemma



The overfitting due to big weights

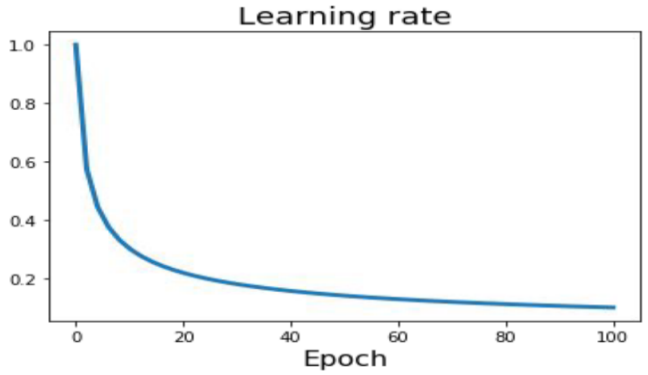
- Adopt a regularization term in the loss function to penalize big weights:

$$L_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c=1}^{N}(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 + \lambda\|\mathbf{w}\|^2$$

  - Decay coefficient: tiny λ
  - Regularization strength: arbitrary λ

- The regularization strength (RS) λ determines how dominant the regularization is during gradient computation: Bigger λ → bigger penalty for big weights
- You may have a RS scheduling (a study issue) like the LR scheduling. But the RS should start from a tiny value.

Learning Rate Decay



Vaswani et al, "Attention is all you need", NIPS 2017

Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0\,(1 + \cos(t\pi/T))$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

The regularizing modules

- 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

The reorganizing modules

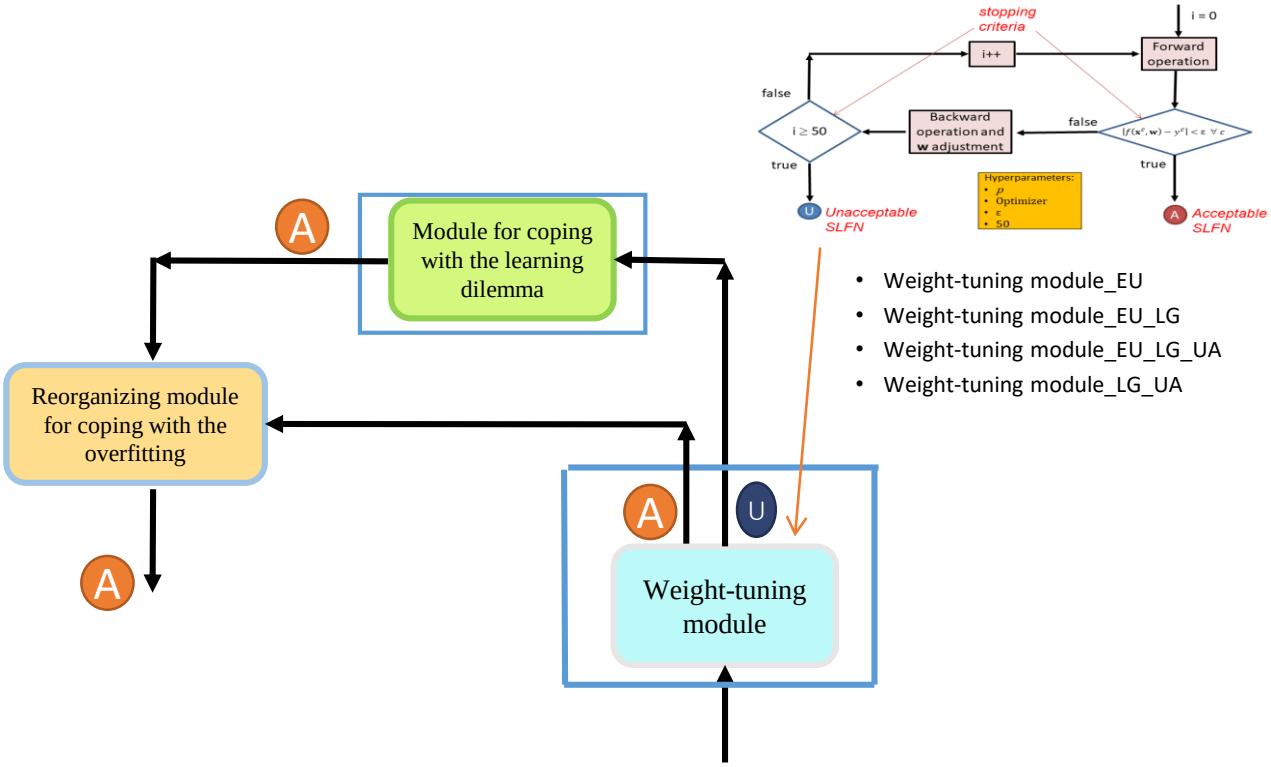
The reorganizing module helps regularize weights of an acceptable SLFN and then examines some hidden nodes one by one.

- ✓ 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

The AI application

- The AI expertise (developing a new AI system is like playing with Lego – lots of pre-built or self-built modules; transfer learning; learning algorithms: supervised, unsupervised, or reinforcement; …)
- The math expertise / the computer expertise (HW, SW, cloud, edge computing, …)
- The application domain know-how
- The performance evaluation

Cope with the learning dilemma



- Weight-tuning module_EU
- Weight-tuning module_EU_LG
- Weight-tuning module_EU_LG_UA
- Weight-tuning module_LG_UA

Cramming (education)

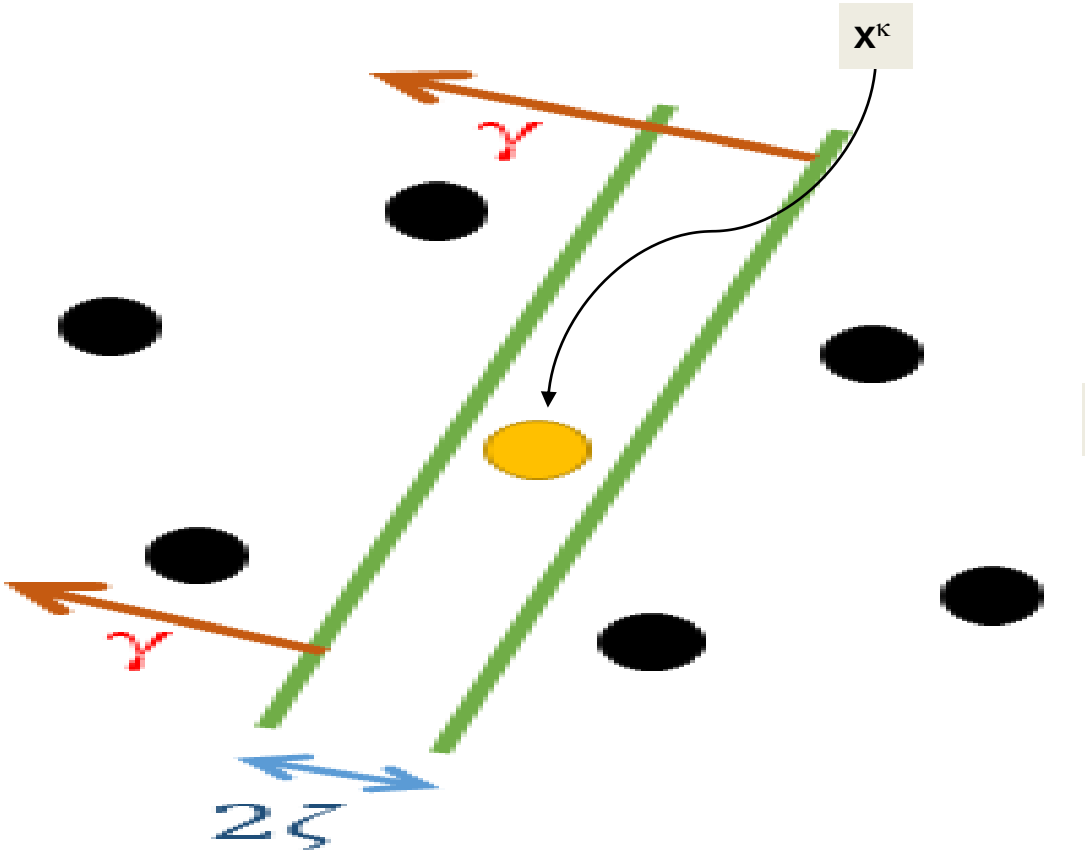
(Cramming (education) - Wikipedia)

- ✓ In education, **cramming** (also known as **mugging** or **swotting**, from *swot*, akin to "sweat", meaning "to study with determination") is the practice of working intensively to absorb large volumes of information in short amounts of time.
- ✓ It is often done by students in preparation for upcoming exams, especially just before they are due.
- ✓ Usually the student's priority is to obtain shallow recall suited to a superficial examination protocol, rather than to internalize the deep structure of the subject matter.
- ✓ Cramming is often discouraged by educators because the hurried coverage of material tends to result in poor long-term retention of material.

The cramming mechanism

Idea/concept:

- Assumption: Only the output of κ^{th} case is unacceptable regarding the learning goal, while outputs of other cases are acceptable regarding the learning goal.
- Method: With recruiting extra hidden nodes, the κ^{th} input is isolated from all other inputs so that it's output can be changed into the right value while outputs of other inputs are still the same.



A rule-based mechanism

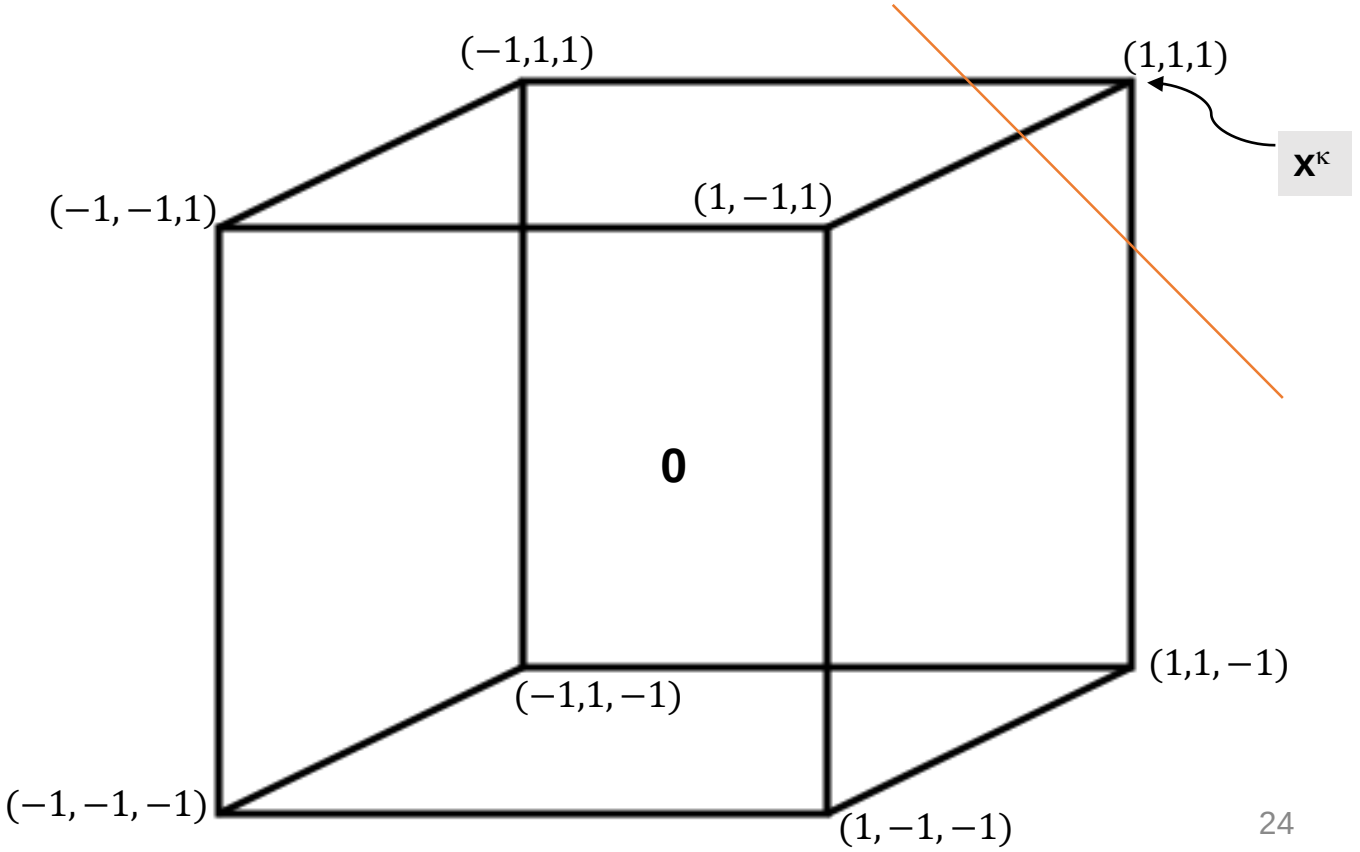
Binary-number inputs

The isolation module with ReLU in the $\{-1,1\}^m$ space

Idea/concept:

- Assumption: Only the output of κ^{th} case is unacceptable regarding the learning goal, while outputs of other cases are acceptable regarding the learning goal.
- Method: With recruiting an extra hidden node, the κ^{th} input is isolated from all other inputs so that it's output can be changed into the right value while outputs of other inputs are still the same.

The $\{-1,1\}^3$ space

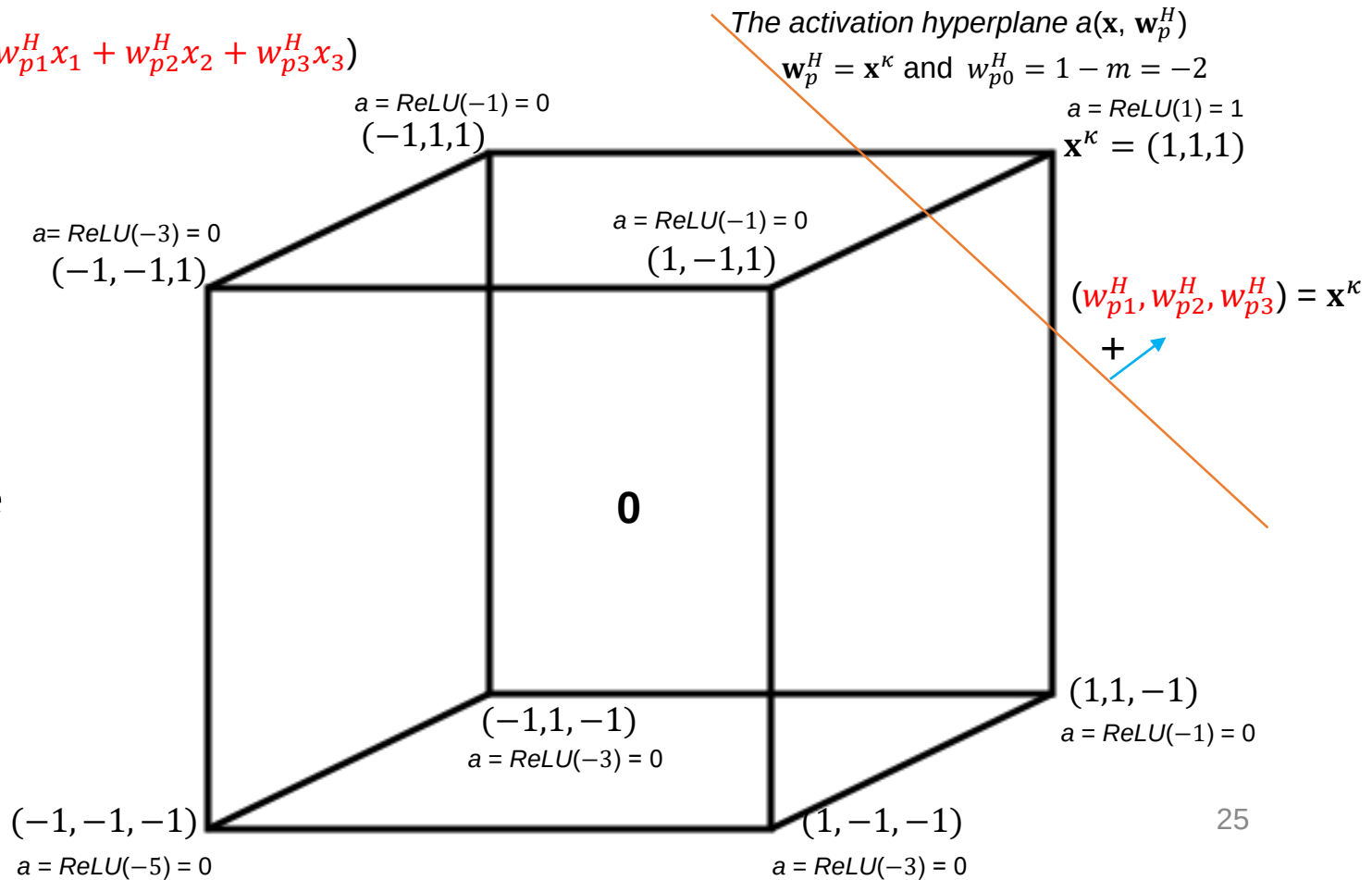


The isolation module with ReLU in the $\{-1,1\}^m$ space

Let $p + 1 \rightarrow p$, add the new p^{th} hidden node with $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \; \forall \; j$) and $w_{p0}^H = 1 - m$

Because of $\mathbf{x} \in \{-1, 1\}^m$, this isolation module renders only $a(\mathbf{x}^\kappa, \mathbf{w}_p^H)$, which equals *ReLU*(1), be 1 and the other $a(\mathbf{x}^c, \mathbf{w}_p^H)$s, which equal *ReLU*(-1), *ReLU*(-3), *ReLU*(-5), …, be 0.

$a(\mathbf{x}, \mathbf{w}_p^H) = ReLU(w_{p0}^H + w_{p1}^H x_1 + w_{p2}^H x_2 + w_{p3}^H x_3)$



The $\{-1,1\}^3$ space

Binary-number inputs

Classification applications

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

- Two-class classification problems with $\mathbf{I} \equiv \mathbf{I}_1 \cup \mathbf{I}_2$, where $\mathbf{I}_1$ and $\mathbf{I}_2$ are the sets of indices of given cases in classes 1 and 2. Furthermore, y^c is the target of the c^{th} case, with 1 and 0 being the targets of classes 1 and 2
- When the SLFN with only one output node whose output value is real number, the stopping criteria may be as follows:

1. $|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \ \forall\, c$
2. $f(\mathbf{x}^c, \mathbf{w}) > \nu \ \forall\, c \in \mathbf{I}_1$ and $f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \ \forall\, c \in \mathbf{I}_2$, with $1 > \nu > 0$
3. $\alpha \equiv \min_{c \epsilon \mathbf{I}_1} f(\mathbf{x}^c, \mathbf{w}) > \beta \equiv \max_{c \epsilon \mathbf{I}_2} f(\mathbf{x}^c, \mathbf{w})$
   (Linearly seperating condition, LSC)

Different stopping criterion results in different length of training time and different model.

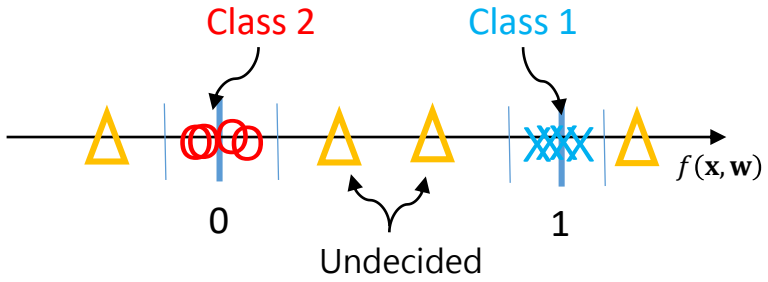
Stopping criteria (also the learning goals) for the SLFN with each output node whose output values are real numbers for the two-class classification application

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

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

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

ε Is a hyperparameter

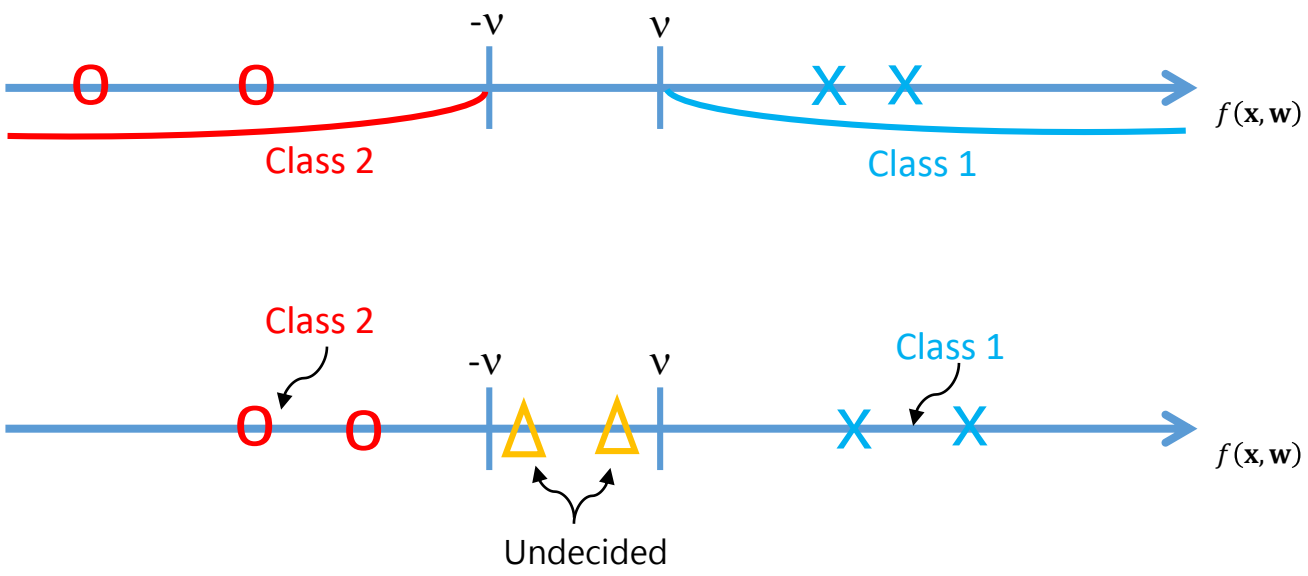


The inferencing mechanism

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 : $f(\mathbf{x}^c, \mathbf{w}), \; \forall \; c \in \mathbf{I}_2$

- $f(\mathbf{x}^c, \mathbf{w}) \geq \nu \; \forall \; c \in \mathbf{I}_1$ and $f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \; \forall \; c \in \mathbf{I}_2$ with $1 > \nu > 0$.



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

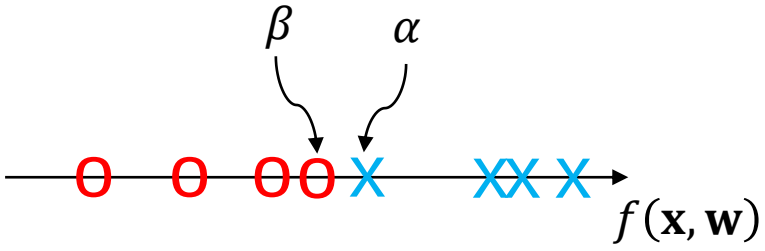
ν Is a hyperparameter

The inferencing mechanism

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

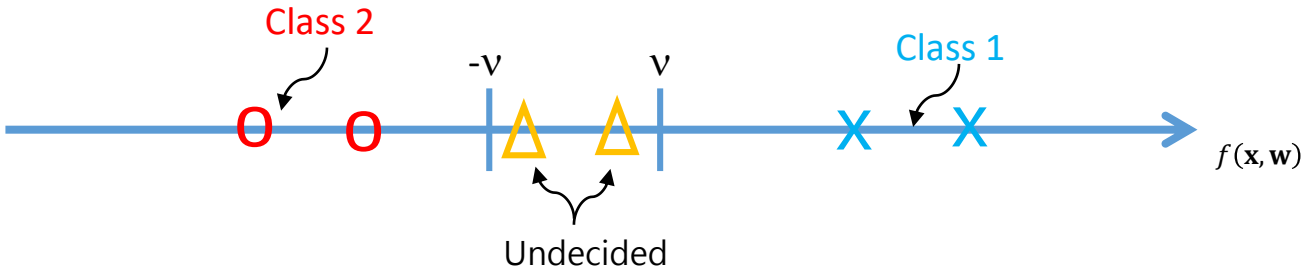
$y^c = 1 \ \forall\ c \in \mathbf{I}_1; y^c = 0 \ \forall\ c \in \mathbf{I}_2$ X : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_1$ O : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_2$

- The LSC ($\alpha > \beta$) (Tsaih, 1993)



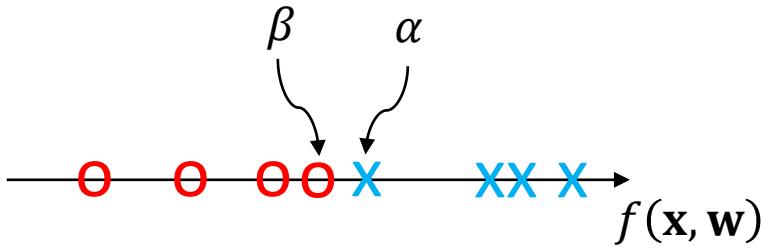
$$\alpha \equiv \min_{c \epsilon \mathbf{I}_1} f(\mathbf{x}^c, \mathbf{w}) ;\ \beta \equiv \max_{c \epsilon \mathbf{I}_2} f(\mathbf{x}^c, \mathbf{w})$$

learning goal type 3: LSC



The inferencing mechanism

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

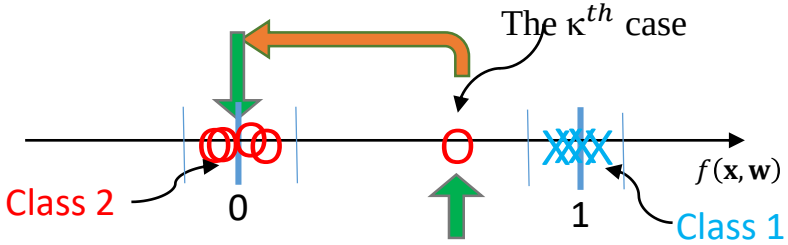
The threshold of the output node

The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

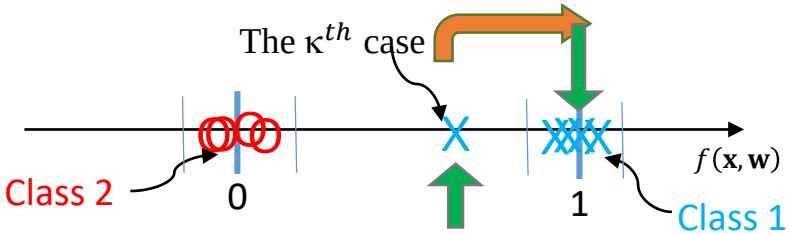
classification problems with LGT1

- $\mathbf{x} \in \{-1, 1\}^m$; $y^c = 1$ if $c \in \mathbf{I}_1$; $y^c = 0$ if $c \in \mathbf{I}_2$
- Assume the current SLFN makes LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\text{-}\{\kappa\}\}$ true, but LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ is false.
- The cramming module wants to recruit merely one extra hidden node to make LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ true.
- Method: With recruiting an extra hidden node, the κ^{th} input is isolated from all other inputs so that it's output can be changed into the right value while outputs of other inputs are still the same.

$\kappa \in \mathbf{I}_2$



$\kappa \in \mathbf{I}_1$



The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

classification problems with LGT1

Let $p + 1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \; \forall \; j$) and $w_{p0}^H = 1 - m$

Because of $\mathbf{x} \in \{-1, 1\}^m$, this isolation module renders only $a(\mathbf{x}^\kappa, \mathbf{w}_p^H)$, which equals *ReLU*(1), be 1 and the other $a(\mathbf{x}^c, \mathbf{w}_p^H)$s, which equal *ReLU*(-1), *ReLU*(-3), *ReLU*(-5), …, be 0.

The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

classification problems with LGT1

Let $\rho + 1 \rightarrow \rho$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \; \forall \; j$) and $w_{p0}^H = 1 - m$

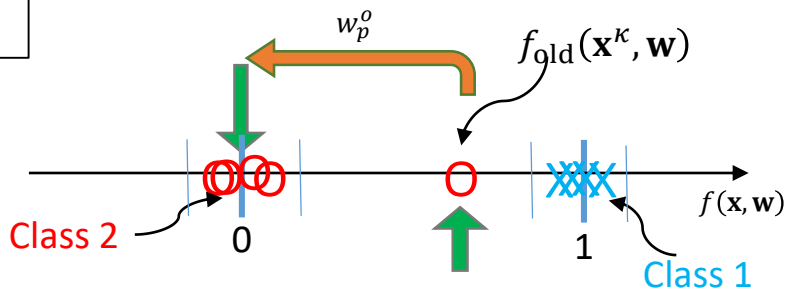
2) $w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

$f_{\text{old}}(\mathbf{x}^\kappa, \mathbf{w}) = w_0^o + \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

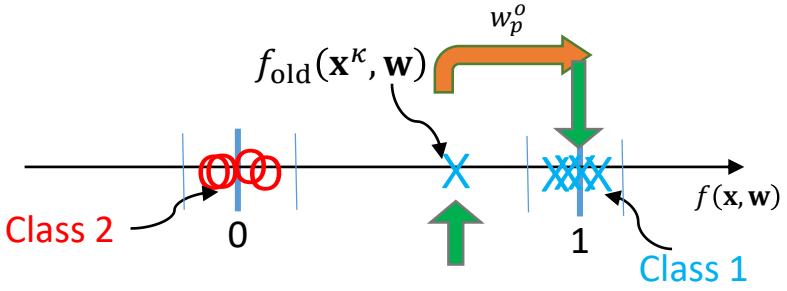
$f(\mathbf{x}^c, \mathbf{w}) = f_{\text{old}}(\mathbf{x}^c, \mathbf{w}) + w_p^o a_p^c$

Only the output of κ^{th} input is changed into the right value, while outputs of other inputs are still the same.

$\kappa \in \mathbf{I}_2$



$\kappa \in \mathbf{I}_1$



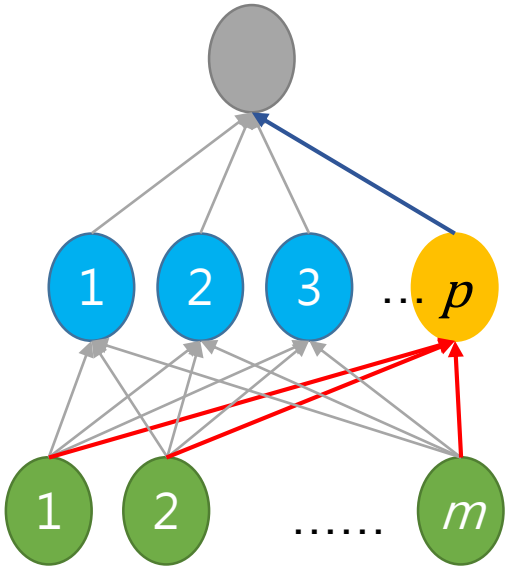
The cramming module_ReLU_BI_LGT1

Let $p+1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LGT1 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \; \forall \; j$) and $w_{p0}^H = 1 - m$

2) $w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

A rule-based mechanism



$$w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$$

$$w_{p0}^H = 1 - m$$

$$w_{pj}^H = x_j^\kappa \; \forall \; j$$

The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

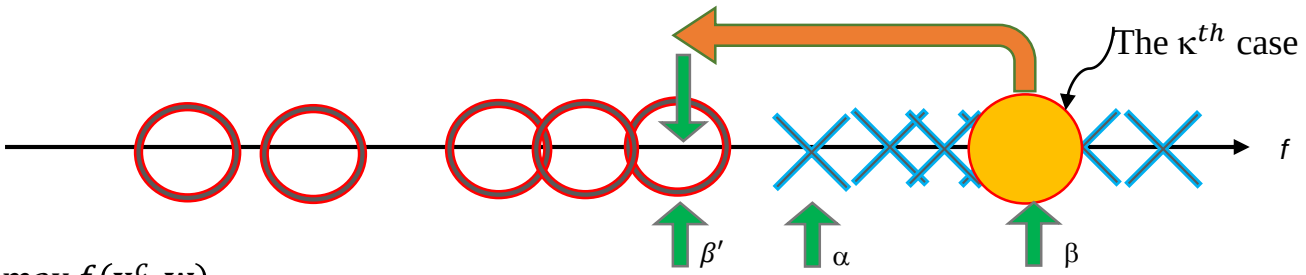
classification problems with LGT3

- Assume $\mathbf{x} \in \{-1, 1\}^m$; $y^c = 1$ if $c \in \mathbf{I}_1$; $y^c = 0$ if $c \in \mathbf{I}_2$
- Assume the current SLFN makes LGT3 (i.e., LSC) regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\text{-}\{\kappa\}\}$ true, but LGT3 regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ false.
- Regarding the case of binary-number inputs, binary-number desired output and ReLU, the cramming module recruits an extra hidden node to make LSC regarding $\{f(\mathbf{x}^c, \mathbf{w}) \; \forall \; c \in \mathbf{I}\}$ true.
- Method: With recruiting an extra hidden node, the κ^{th} input is isolated from all other inputs so that it's output can be changed into the right value while outputs of other inputs are still the same.

$\kappa \in \mathbf{I}_2$

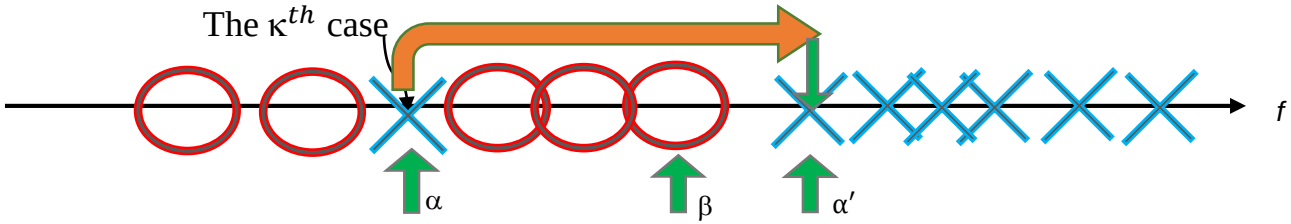
$\beta' = \max_{c \in \mathbf{I}_2 - \{\kappa\}} f(\mathbf{x}^c, \mathbf{w})$

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



$\kappa \in \mathbf{I}_1$

$\alpha' = \min_{c \in \mathbf{I}_1 - \{\kappa\}} f(\mathbf{x}^c, \mathbf{w})$



The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

classification problems with LGT3

Let $\rho + 1 \rightarrow \rho$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LSC regarding $\{f(\mathbf{x}^c, \mathbf{w}) \ \forall\, c \in \mathbf{I}\}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall\, j$) and $w_{p0}^H = 1 - m$

Because of $\mathbf{x} \in \{-1, 1\}^m$, this isolation module renders only $a(\mathbf{x}^\kappa, \mathbf{w}_p^H)$, which equals *ReLU*(1), be 1 and the other $a(\mathbf{x}^c, \mathbf{w}_p^H)$s, which equal *ReLU*(-1), *ReLU*(-3), *ReLU*(-5), …, be 0.

The cramming module – the case of binary-number inputs, binary-number desired output & ReLU

classification problems with LGT3

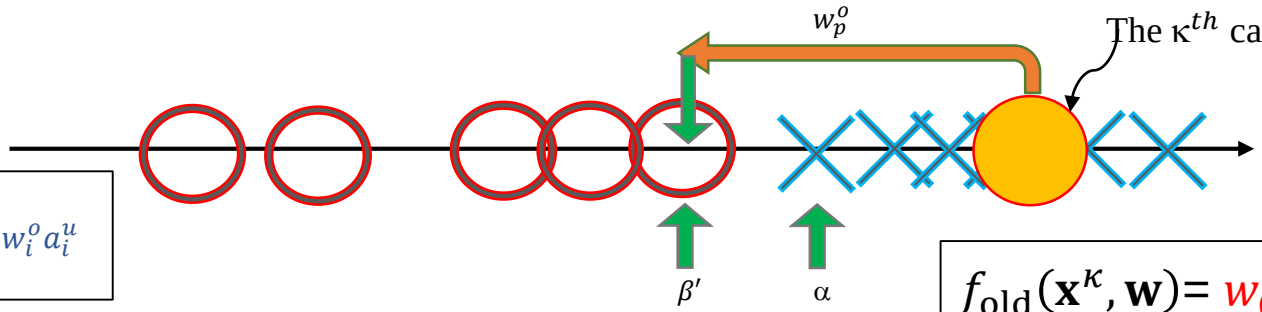
Let $p+1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LSC regarding $\{f(\mathbf{x}^c, \mathbf{w}) \ \forall\, c \in \mathbf{I}\}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall\, j$) and $w_{p0}^H = 1 - m$

2) $$w_p^o = \begin{cases} \max\limits_{u \in \mathbf{I}_2 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^u - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_2 \\ \min\limits_{v \in \mathbf{I}_1 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^v - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_1 \end{cases}$$

$\kappa \in \mathbf{I}_2$

$\beta' = w_0^o + \max\limits_{u \in \mathbf{I}_2 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^u$

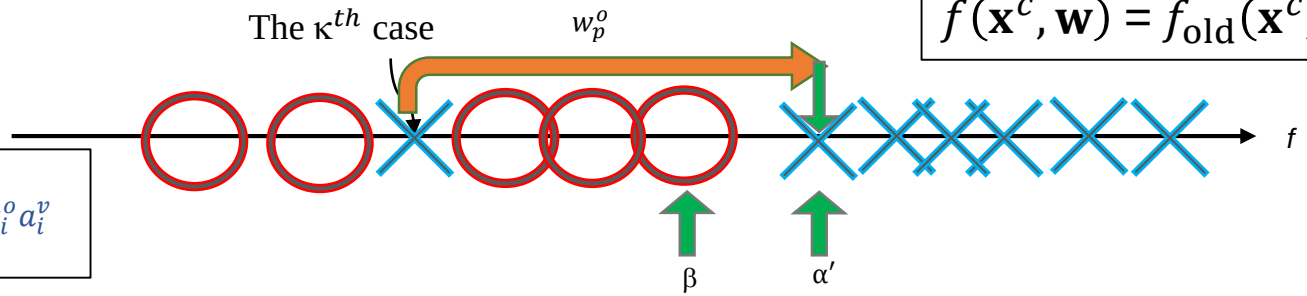


$f_{\text{old}}(\mathbf{x}^\kappa, \mathbf{w}) = w_0^o + \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

$f(\mathbf{x}^c, \mathbf{w}) = f_{\text{old}}(\mathbf{x}^c, \mathbf{w}) + w_p^o a_p^c$

$\kappa \in \mathbf{I}_1$

$\alpha' = w_0^o + \min\limits_{v \in \mathbf{I}_1 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^v$



The cramming module_ReLU_BI_LGT3

classification problems with LGT3

Let $p+1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make LSC regarding $\{f(\mathbf{x}^c, \mathbf{w}) \ \forall\ c \in \mathbf{I}\}$ true:

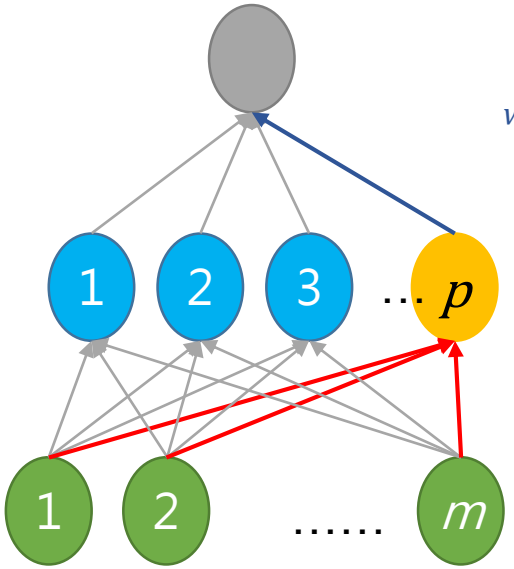
1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall\ j$) and $w_{p0}^H = 1 - m$

2) $$w_p^o = \begin{cases} \max\limits_{u \in \mathbf{I}_2 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^u - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_2 \\ \min\limits_{v \in \mathbf{I}_1 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^v - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_1 \end{cases}$$

$$\beta' = w_0^o + \max_{u \in \mathbf{I}_2 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^u$$

$$\alpha' = w_0^o + \min_{v \in \mathbf{I}_1 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^v$$

A rule-based mechanism



$$w_p^o = \begin{cases} \max\limits_{u \in \mathbf{I}_2 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^u - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_2 \\ \min\limits_{v \in \mathbf{I}_1 - \{\kappa\}} \sum_{i=1}^{p-1} w_i^o a_i^v - \sum_{i=1}^{p-1} w_i^o a_i^\kappa & if\ \kappa \in \mathbf{I}_1 \end{cases}$$

$$w_{p0}^H = 1 - m$$

$$w_{pj}^H = x_j^\kappa \ \forall\ j$$

Binary-number inputs

Regression applications

Stopping criteria (also the learning goals) for regression applications

The learning process should stop when

one output node

~~1. $L_N(\mathbf{w}) = 0$~~

2. a tiny $L_N(\mathbf{w})$ value

$$L_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c=1}^{N}(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2$$

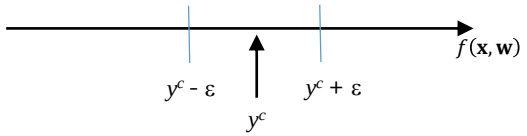
3. $|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \;\; \forall \; c \;\; with \; \varepsilon \; being \; tiny$

- Each reasonable learning goal can be used as a stopping criterion.
- Different stopping criterion results in different length of training time and different model.

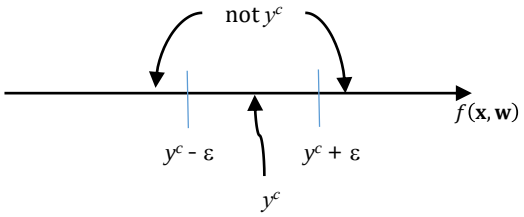
The regression applications

The learning goal

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



The inferencing mechanism

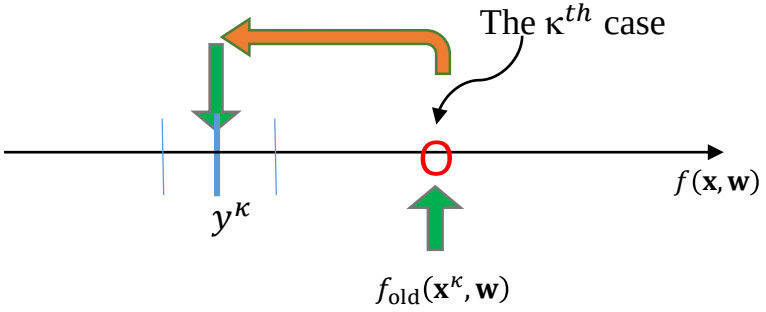


This learning goal and the associated inferencing mechanism is similar to LGT1 (also inferencing goal): $|f(\mathbf{x}^c, \mathbf{w})\text{-}1| \leq \varepsilon \;\forall\; c \in \mathbf{I}_1$ and $|f(\mathbf{x}^c, \mathbf{w})| \leq \varepsilon \;\forall\; c \in \mathbf{I}_2$

The cramming module – the case of binary-number inputs, real-number desired output & ReLU

regression problems

- $\mathbf{x} \in \{-1, 1\}^m$.
- Assume the current SLFN makes $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}\text{-}\{\kappa\}$ true, but $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}$ is false.
- Regarding the case of binary-number inputs, real-number desired output and ReLU, the cramming module recruits an extra hidden node to make $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}$ true.
- Method: With recruiting an extra hidden node, the κ^{th} input is isolated from all other inputs so that it's output can be changed into the right value while outputs of other inputs are still the same.



The cramming module – the case of binary-number inputs, real-number desired output & ReLU

regression problems

Let $\rho + 1 \rightarrow \rho$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall \ j$) and $w_{p0}^H = 1 - m$

Because of $\mathbf{x} \in \{-1, 1\}^m$, this isolation module renders only $a(\mathbf{x}^\kappa, \mathbf{w}_p^H)$, which equals *ReLU*(1), be 1 and the other $a(\mathbf{x}^c, \mathbf{w}_p^H)$s, which equal *ReLU*(-1), *ReLU*(-3), *ReLU*(-5), …, be 0.

The cramming module – the case of binary-number inputs, real-number desired output & ReLU

regression problems

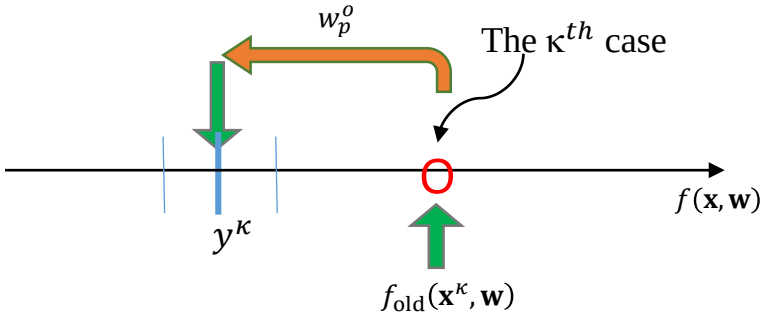
Let $p + 1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall \ j$) and $w_{p0}^H = 1 - m$

2) $w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

$f_{\text{old}}(\mathbf{x}^\kappa, \mathbf{w}) = w_0^o + \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

$f(\mathbf{x}^c, \mathbf{w}) = f_{\text{old}}(\mathbf{x}^c, \mathbf{w}) + w_p^o a_p^c$



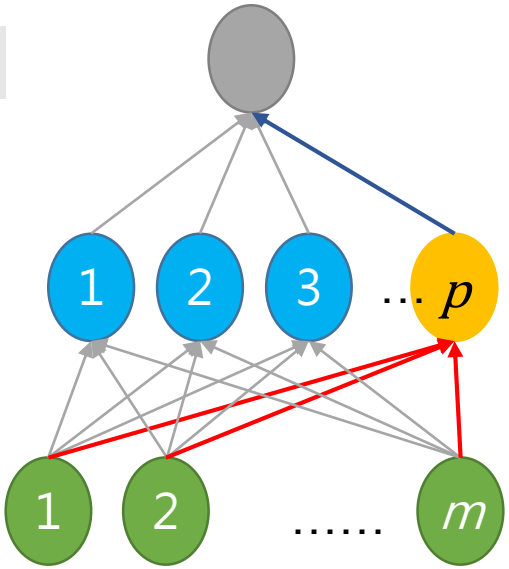
The cramming module_ReLU_BI_RE

Let $p+1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way to make $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2 \ \forall \ c \in \mathbf{I}$ true:

1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa \ \forall \ j$) and $w_{p0}^H = 1 - m$

2) $w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$

A rule-based mechanism



$$w_p^o = y^\kappa - \sum_{i=1}^{p-1} w_i^o a_i^\kappa$$

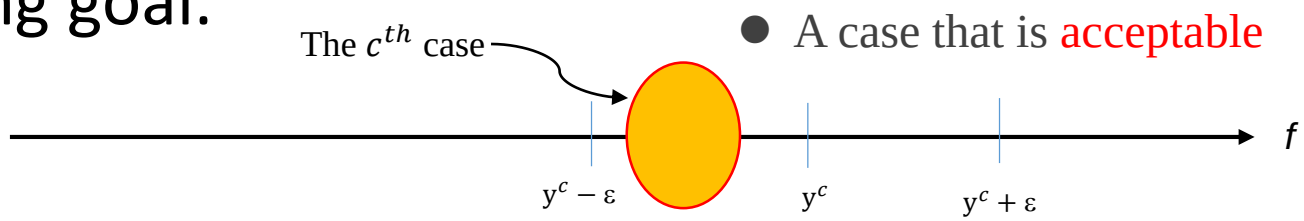
$$w_{p0}^H = 1 - m$$

$$w_{pj}^H = x_j^\kappa \ \forall \ j$$

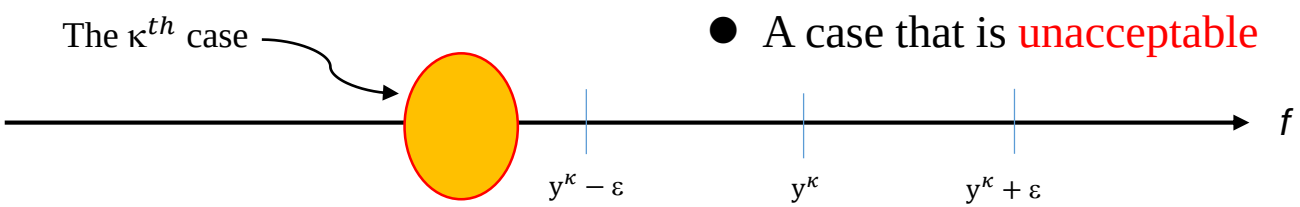
Note that the cramming module_ReLU_BI_RE is the same as the cramming module_ReLU_BI_LGT1.

Multiple output nodes and Multiple unacceptable cases

- Can the cramming module work for multiple output node? ← Yes
- Can the cramming module work for multiple unacceptable cases? ← Yes
- The acceptability of each case is related with the learning goal.



- The acceptance is based on the learning goal $(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2 \leq \varepsilon^2$



The cramming module_ReLU_BI_RE for multiple output nodes

regression problems

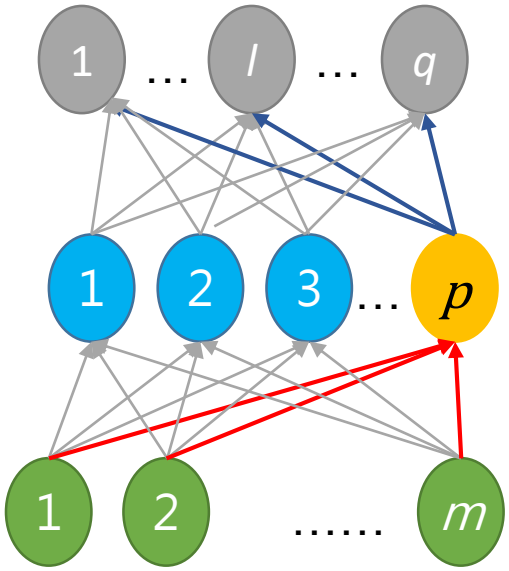
Regarding every l^{th} output node, in which $(e_l^c)^2 \le \varepsilon^2 \ \forall\ c \in \mathbf{I}\text{-}\{\kappa\}$ true, but $(e_l^c)^2 \le \varepsilon^2 \ \forall\ c \in \mathbf{I}$ is false:

1. Let $p + 1 \rightarrow p$ and add the new p^{th} hidden node to the existing SLFN.
2. Assign $\mathbf{w}_p^H$, w_{p0}^H, w_{lp}^o and $w_{kp}^o\ \forall\ k \neq l$ in the following way to make $(e_l^c)^2 \le \varepsilon^2 \ \forall\ c \in \mathbf{I}$ true:

   1) $\mathbf{w}_p^H = \mathbf{x}^\kappa$ (i.e., $w_{pj}^H = x_j^\kappa\ \forall\ j$) and $w_{p0}^H = 1 - m$

   2) $w_{lp}^o = y_l^\kappa - w_{l0}^o - \sum_{i=1}^{p-1} w_{li}^o a_i^\kappa$ and $w_{kp}^o = 0\ \forall\ k \neq l$

$e_l^c \equiv f_l(\mathbf{x}^c, \mathbf{w}) - y_l^c$



$w_{lp}^o = y_l^\kappa - w_{l0}^o - \sum_{i=1}^{p-1} w_{li}^o a_i^\kappa$ & $w_{kp}^o = 0\ \forall\ k \neq l$

$w_{p0}^H = 1 - m$

$w_{pj}^H = x_j^\kappa\ \forall\ j$

The cramming module_ReLU_BI_RE for multiple unacceptable cases

For each unacceptable case ($\mathbf{x}^{\kappa}$, y^{κ}):

Let $p + 1 \rightarrow p$, add the new p^{th} hidden node to the existing SLFN, and then assign $\mathbf{w}_p^H$, w_{p0}^H and w_p^o in the following way:

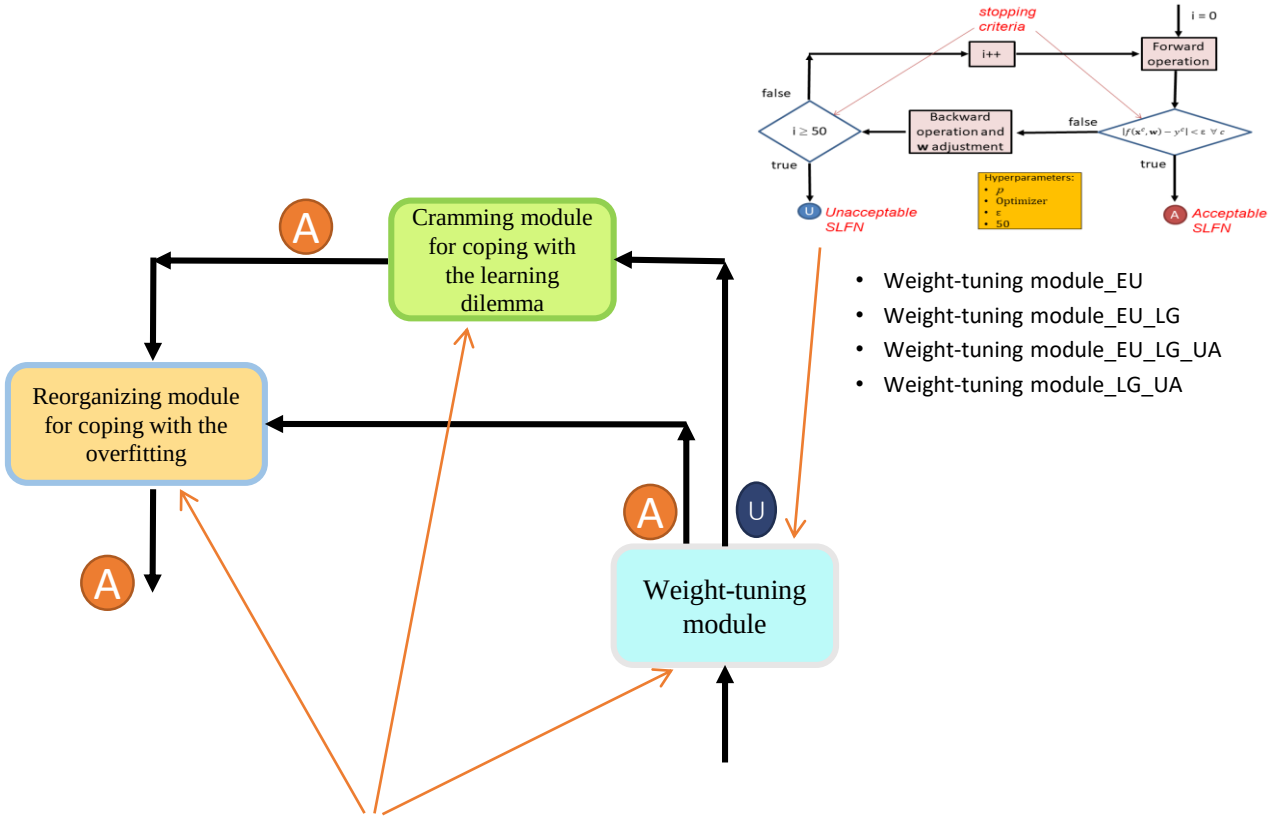
1) $\mathbf{w}_p^H = \mathbf{x}^{\kappa}$ (i.e., $w_{pj}^H = x_j^{\kappa}\ \forall\ j$) and $w_{p0}^H = 1 - m$

2) $w_p^o = y^{\kappa} - \sum_{i=1}^{p-1} w_i^o a_i^{\kappa}$

Homework #4

- Make the coding of AI system stated in page 51.
- Pick up one of the following cramming modules:
  - ✓ the cramming module_ReLU_BI_LGT1 stated in page 35
  - ✓ the cramming module_ReLU_BI_LGT3 stated in page 39
- Note that the learning goals used in the weight-tuning module, the cramming module, the regularizing module, and the reorganizing module should be the same.

The weight-tuning, cramming and reorganizing modules



- Weight-tuning module_EU
- Weight-tuning module_EU_LG
- Weight-tuning module_EU_LG_UA
- Weight-tuning module_LG_UA

Learning is the process of acquiring new, or modifying existing, knowledge, behaviors, skills, values, or preferences. -- Richard Gross, Psychology: The Science of Mind and Behaviour 6E, Hachette UK, ISBN 978-1-4441-6436-7

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

Program testing -- New learning mechanism & Coding

- Not merely double check the correctness of codes
- New learning mechanism → new learning process ← Double check the learning process (ALWAYS!!!)
- Simple checks:

  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 $f(\mathbf{x}^c, \mathbf{w})$ values all c is reasonable?
- Complicated checks:

  whether the learning process is reasonable?

Not the performance, which is related with the inferencing.