Coping with the overfitting issue: regularizing and pruning irrelevant hidden nodes

國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

AI applications

- Training phase: (training) data + AI model + algorithm & code + setting of network & hyperparameters → AI model/AI system
- Inferencing phase: performance is obtained from model((test) data)
- Goals of training are reasonable inferencing

Where we are now...

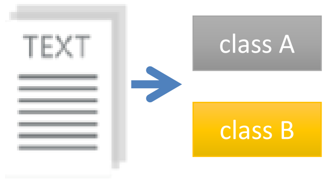
ideas/concepts →
modules →
learning algorithm →
codes →
intelligent systems

Types of Learning

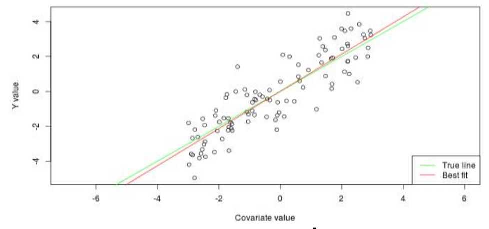
Supervised: Learning with a **labeled training set**
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled data**
Example: *cluster* similar documents based on text

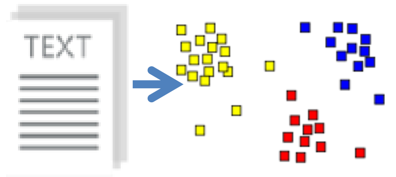
Reinforcement learning: learn to **act** based on **feedback/reward**
Example: learn to *play* Go, reward: *win or lose*



Classification



Regression



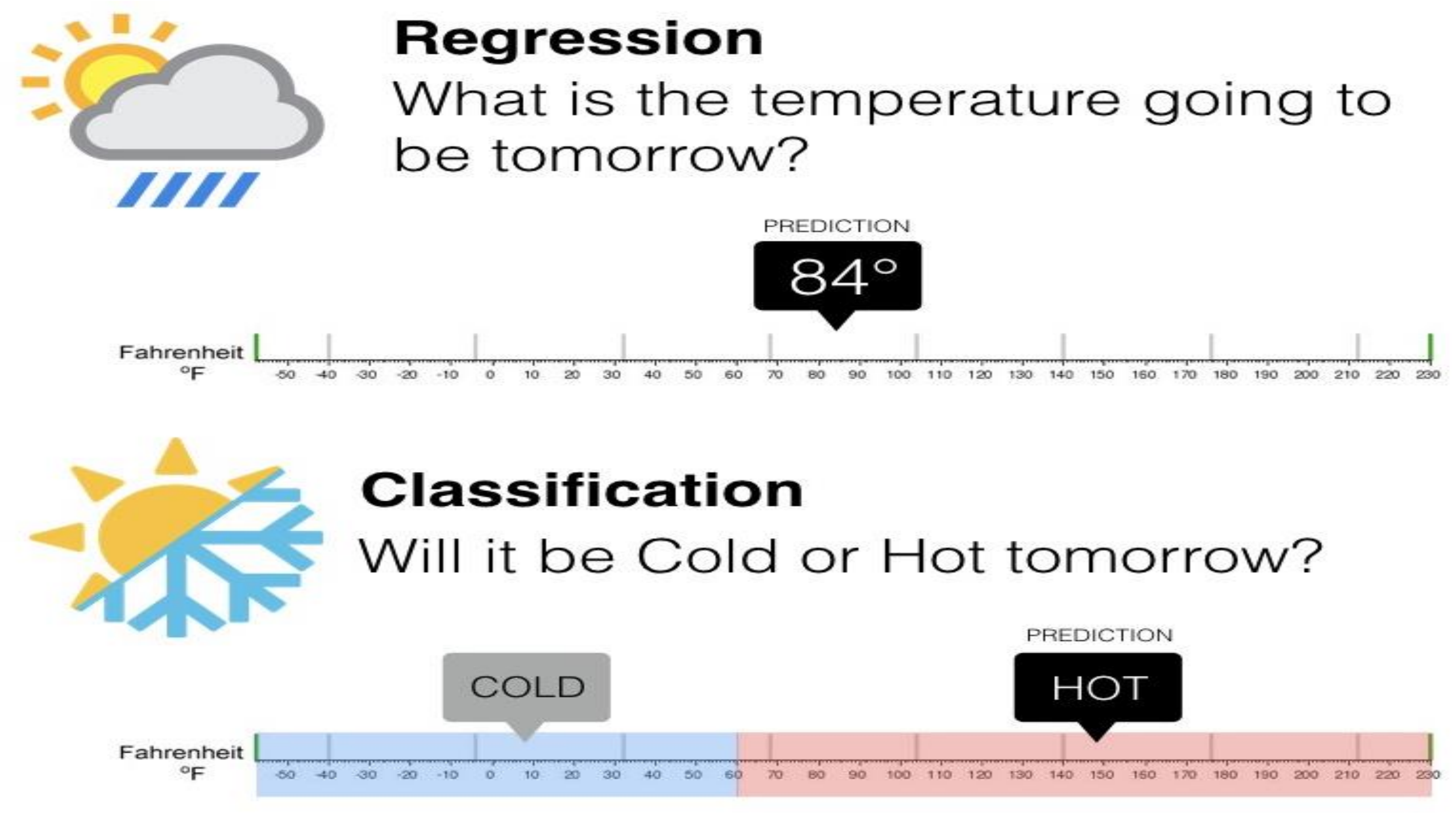
Clustering

Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

Where we are now...

The supervised learning problems: Regression and Classification



Regression
What is the temperature going to be tomorrow?

Classification
Will it be Cold or Hot tomorrow?

Stopping criteria (also the learning goals) for regression applications

one output node

The learning process should stop when

1. ~~$L_N(\mathbf{w}) = 0$~~
2. a tiny $L_N(\mathbf{w})$ value
3. $|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \;\forall\, c$ *with* ε *being tiny*

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

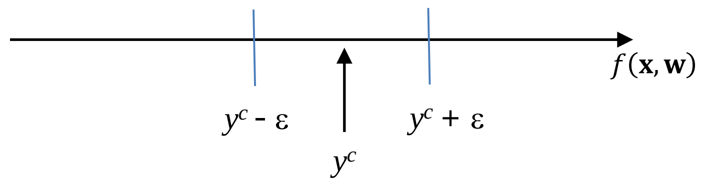
- Each reasonable learning goal can be used as a stopping criterion.
- Different stopping criterion results in different length of training time and different model.

Where we are now...

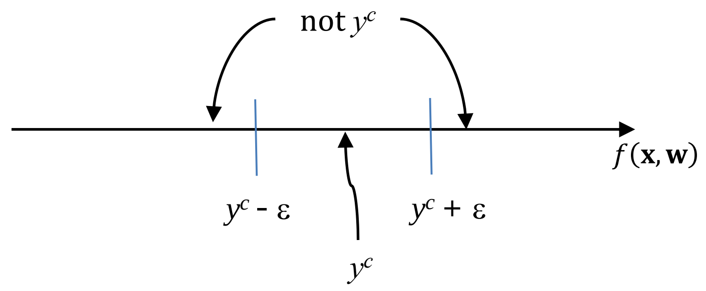
The regression applications

The learning goal

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

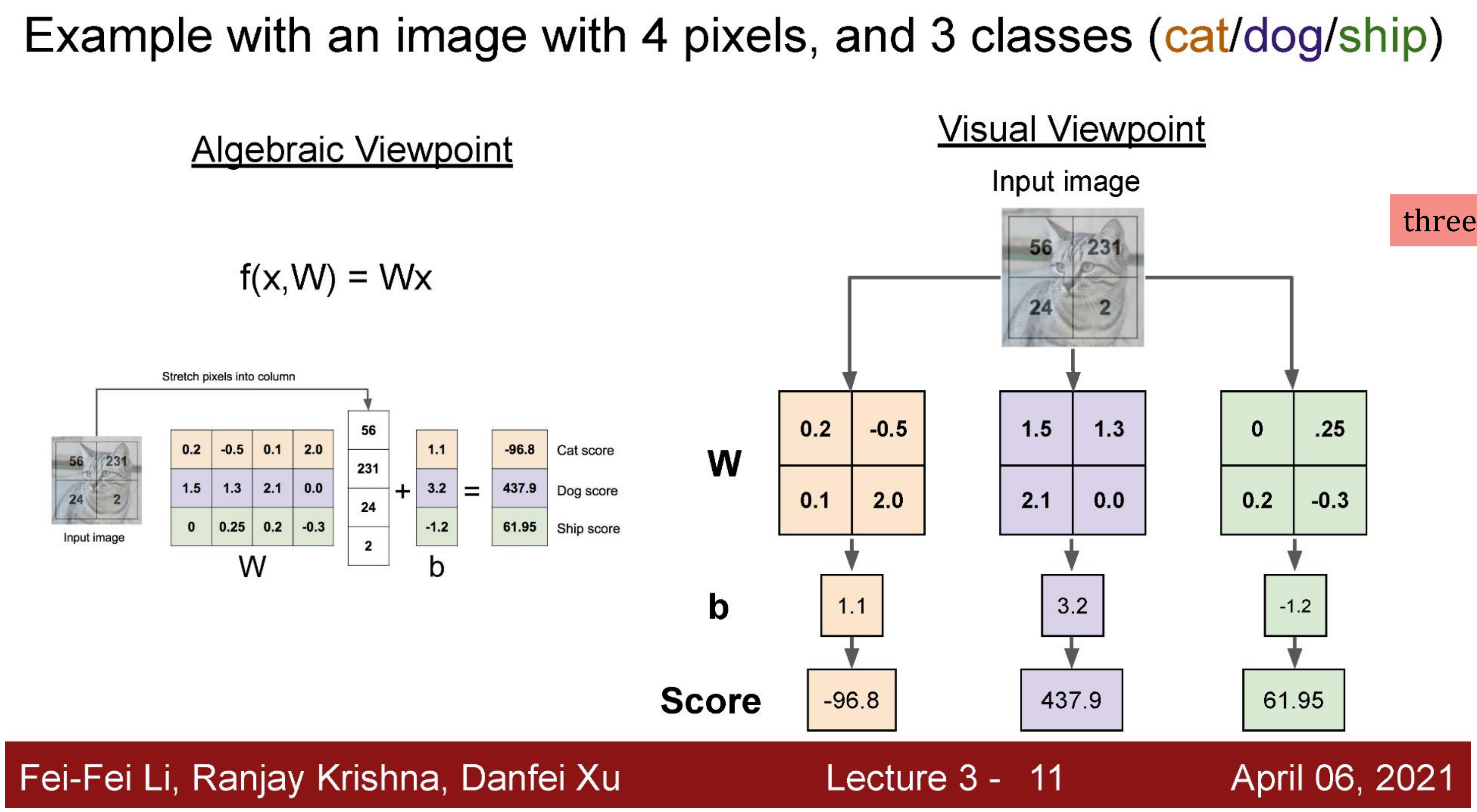


The inferencing mechanism



Where we are now...

The three-class classification applications


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

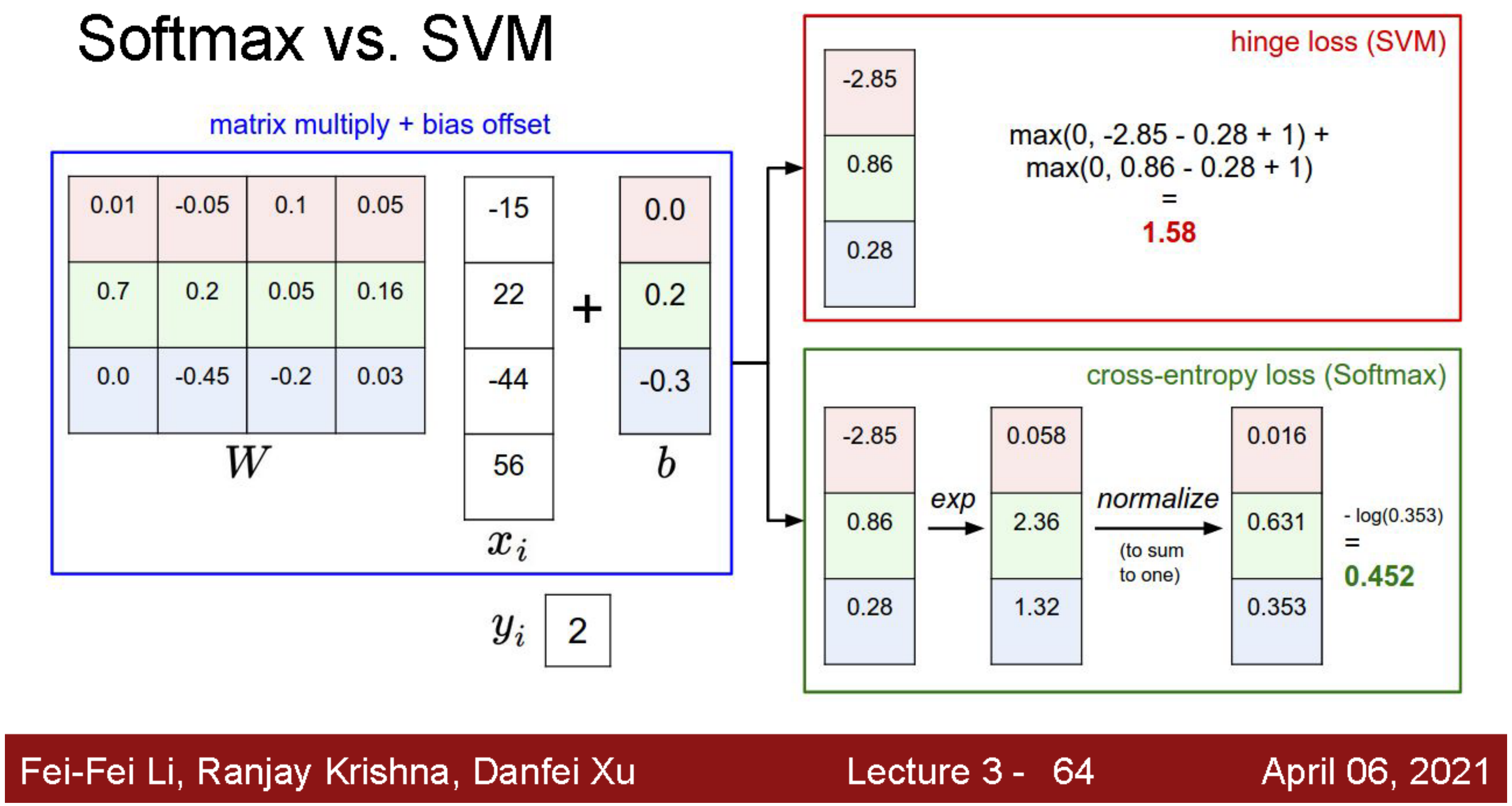
$$\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 3 - 11 April 06, 2021

three output nodes

Where we are now...

The three-class classification applications



Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 3 - 64 April 06, 2021

three output nodes

Classification Applications Design (**x** attributes & *y* label)

X:

- ✓ 年齡
- ✓ 性別
- ✓ 診斷
- ✓ 分期
- ✓ 診斷癌症期間
- ✓ 復發與否
- ✓ Kps (身體功能)
- ✓ gs1-gs22 (22項)症狀 (0:無; 1:有)

y: 疲倦，個案總計 686位，過去一周平均疲倦程度 (f3) (0-10分，real-value variable) → 過去一周疲倦分組 (Gf3) (分成三組，binary variable) :

- ✓ 無: 46位
- ✓ 輕度:346 位
- ✓ 中至重度:294位

Where we are now...

Classification Applications Design (*y* label)

Output value: real number

SLFN with one output node and linear (output) function

疲倦
- ✓ 無: 46位
- ✓ 輕度:346 位
- ✓ 中至重度:294位

Learning phase:
y (i.e., target output):
- ✓ 無: 0
- ✓ 輕度: 5
- ✓ 中至重度: 10

Inferencing phase:
f (i.e., actual output):
- ✓ [-2.5, 2.5) → 無
- ✓ [2.5, 7.5) → 輕度
- ✓ [7.5, 12.5) →中至重度
- ✓ (-∞, -2.5) OR [12.5, ∞) → unknown

Output value: binary number

SLFN with three output nodes and softmax arrangement

疲倦
- ✓ 無: 46位
- ✓ 輕度:346 位
- ✓ 中至重度:294位

Learning phase:
y (i.e., target output):
- ✓ 無: (1, 0, 0)
- ✓ 輕度: (0, 1, 0)
- ✓ 中至重度: (0, 0, 1)

Inferencing phase:
f (i.e., actual output):
- ✓ (1, 0, 0) → 無
- ✓ (0, 1, 0) → 輕度
- ✓ (0, 0, 1) →中至重度

Where we are now...

Another classification application
Design (**x** attributes & y label)

- Data

x attributes	
x1	性別
x2	年齡
x3	國籍
x4	婚姻狀態
x5	直系親屬數
x6	最高學歷
x7	來台時長
x8	平均月收入
x9	剩餘居留時間

x attributes	
x10	借款時長
x11	借款金額
x12	用途
x13	工作性質
x14	工作地點
x15	雇主資訊
x16	薪資如期撥入
x17	薪資撥付方式
x18	薪資結匯方式

y label	
y (target output, real number)	信用評級 有5個等級

y	信用評級
1	E最差
2	D
3	C
4	B
5	A最好

Another classification application Design (*y* label)

one output node

- y (target output) $\in \{1, 2, 3, 4, 5\}$
- At the learning phase, let $\varepsilon = 0.2$. Then the learning goal is to make f (actual output) $\in \{[0.8, 1.2], [1.8, 2.2], [2.8, 3.2], [3.8, 4.2], [4.8, 5.2]\}$.
- At the inferencing phase, $y = 1$ if $f \in [0.5, 1.5)$; $y = 2$ if $f \in [1.5, 2.5)$; $y = 3$ if $f \in [2.5, 3.5)$; $y = 4$ if $f \in [3.5, 4.5)$; $y = 5$ if $f \in [4.5, 5.5)$
- y is unknown if $f < 0.5$ OR $f \geq 5.5$.

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:

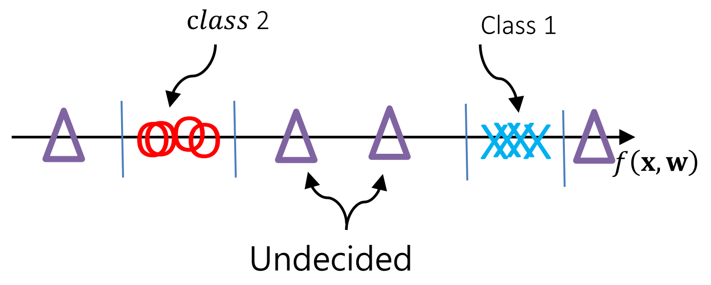
1. $|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \ \forall\, c$
2. $f(\mathbf{x}^c, \mathbf{w}) > \nu \ \forall\, c \in \mathbf{I}_1$ and $f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \ \forall\, c \in \mathbf{I}_2$, with $1 > \nu > 0$
3. $\alpha \equiv \min_{c \epsilon \mathbf{I}_1} f(\mathbf{x}^c, \mathbf{w})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

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

Where we are now...

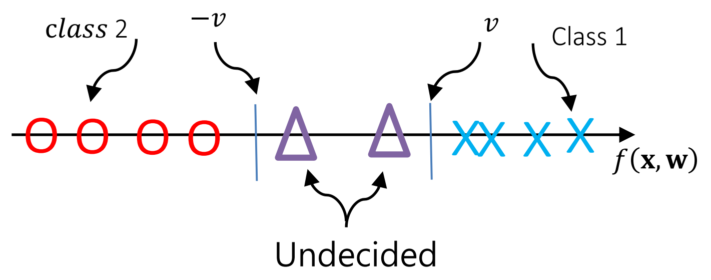
Stopping criteria (also the learning goals) for the SLFN with each output node whose output values are real numbers for the two-class classification application

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



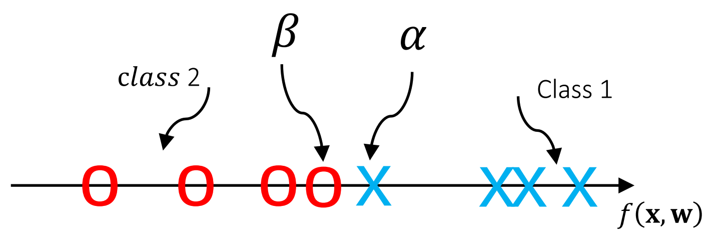
learning goal type 1
(also inferencing goal):
$|f(\mathbf{x}^c, \mathbf{w}) - y^c| \le \varepsilon \ \forall\ c \in \mathbf{I}_1;$
$|f(\mathbf{x}^c, \mathbf{w}) + y^c| \le \varepsilon \ \forall\ c \in \mathbf{I}_2$

ε Is a hyperparameter regarding the learning!



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

ν Is a hyperparameter regarding the inferencing!



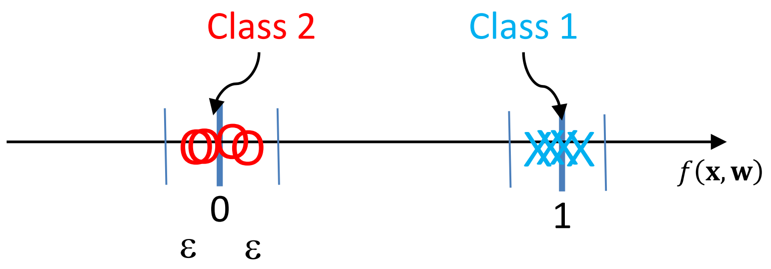
learning goal type 3: LSC

Where we are now...

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

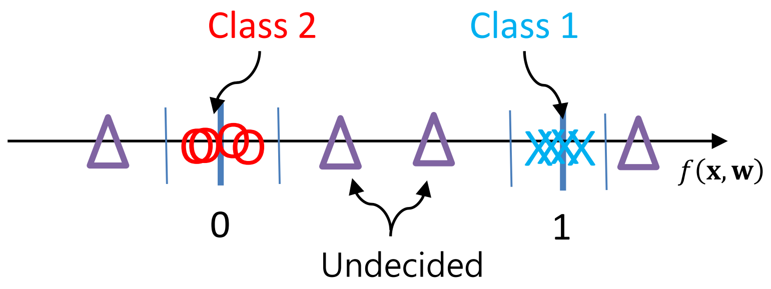
$y^c = 1 \ \forall\ c \in \mathbf{I}_1; y^c = 0 \ \forall\ c \in \mathbf{I}_2$ X : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_1$ O : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_2$

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



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

ε Is a hyperparameter



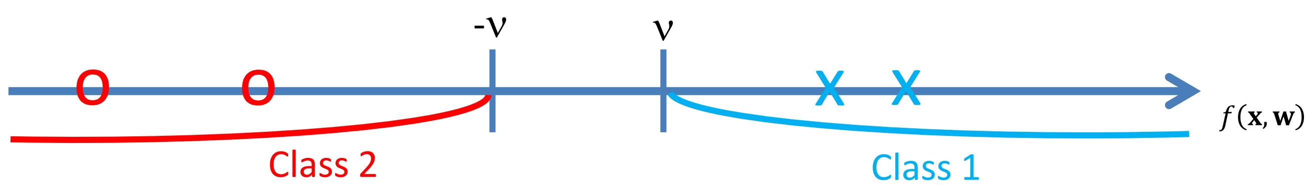
The inferencing mechanism

Where we are now...

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

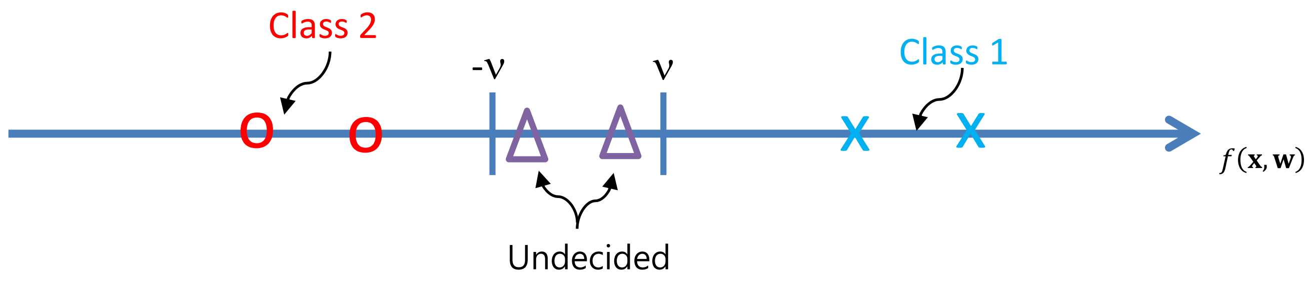
$y^c = 1 \ \forall\ c \in \mathbf{I}_1; y^c = 0 \ \forall\ c \in \mathbf{I}_2$ X : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_1$ O : $f(\mathbf{x}^c, \mathbf{w}),\ \forall\ c \in \mathbf{I}_2$

- $f(\mathbf{x}^c, \mathbf{w}) \geq \nu \ \ \forall\ c \in \mathbf{I}_1$ 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

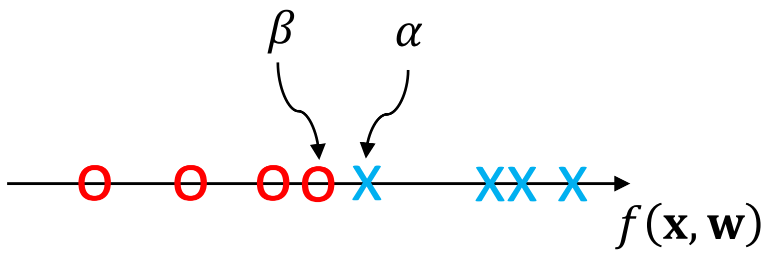


The inferencing mechanism

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

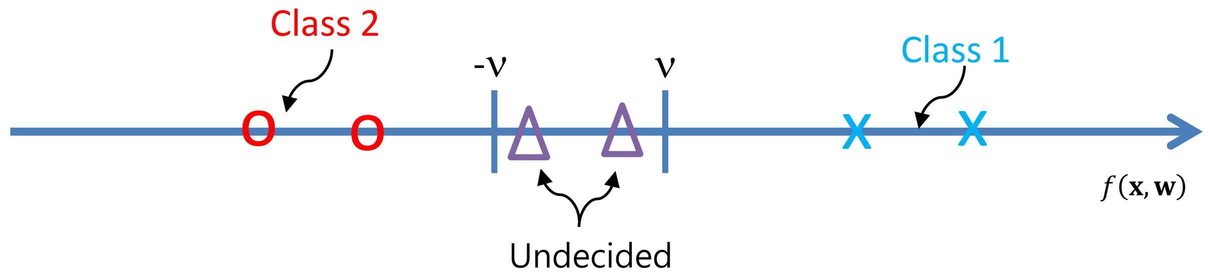
$y^c = 1 \;\forall\, c \in \mathbf{I}_1;\, y^c = 0 \;\forall\, c \in \mathbf{I}_2$ X : $f(\mathbf{x}^c, \mathbf{w}),\; \forall\, c \in \mathbf{I}_1$ O : $f(\mathbf{x}^c, \mathbf{w}),\; \forall\, c \in \mathbf{I}_2$

- The LSC (Tsaih, 1993)



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

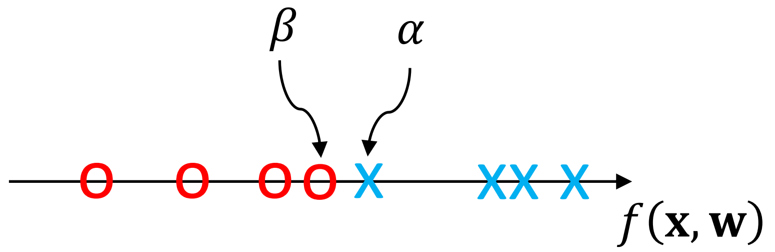
learning goal type 3: LSC



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

Where we are now...

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:

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

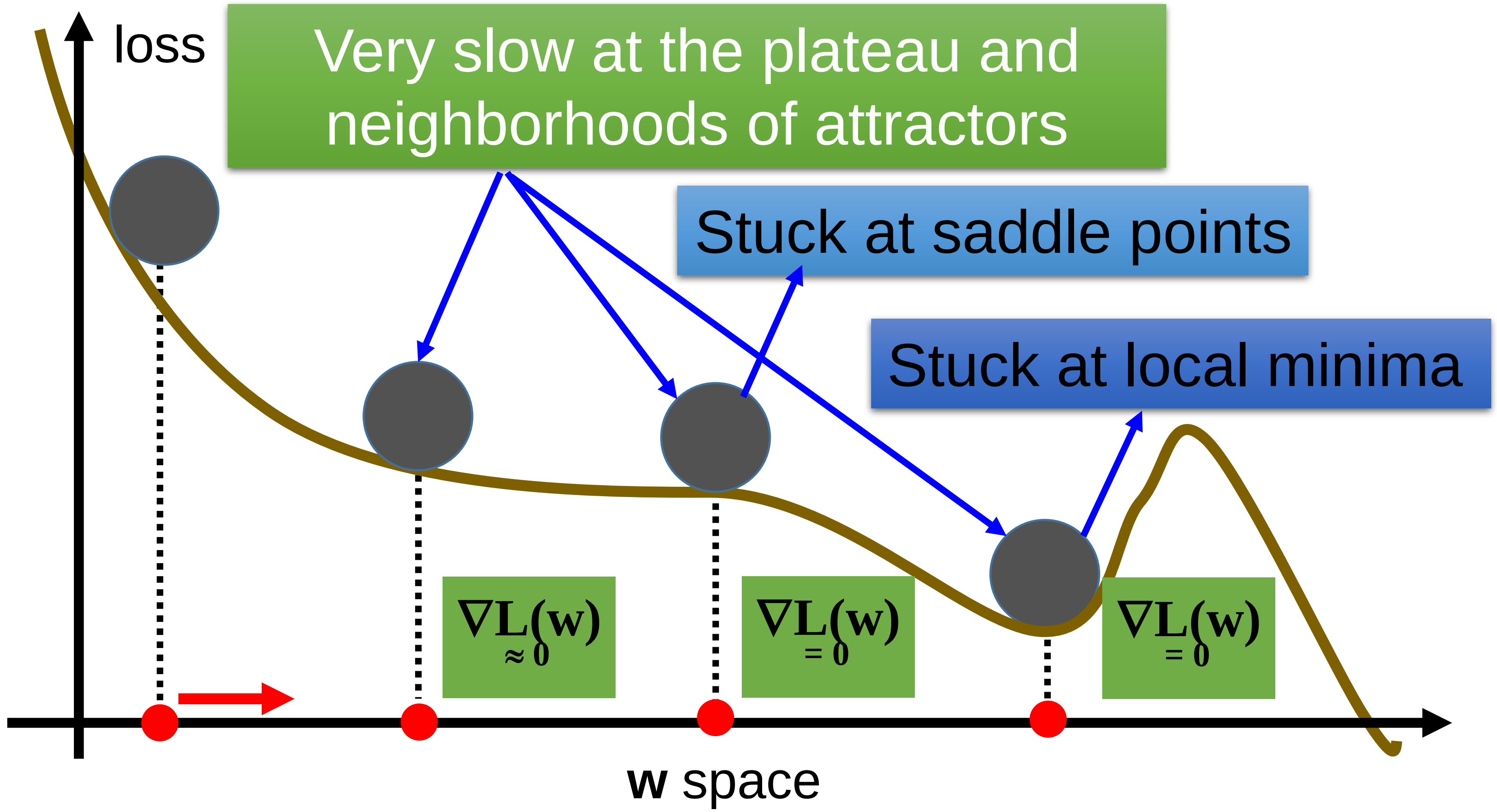
The weight vector between the hidden layer and the output node

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

Where we are now...

Learning dilemma of gradient-descent-based learning



Extra stopping criteria for the learning (not the learning goals)

$||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})||$ is the length of $\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})$.

1. ~~The learning process should stop when $||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})|| = 0$ but a tiny $\mathrm{L}_N(\mathbf{w})$ value cannot be accomplished.~~
2. The learning process should stop when $||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})||$ is tiny but a tiny $\mathrm{L}_N(\mathbf{w})$ value cannot be accomplished.
3. The learning process should stop when η (the learning rate) is tiny but a tiny $\mathrm{L}_N(\mathbf{w})$ value cannot be accomplished.

The neighborhood of undesired attractors, where $||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})|| \approx 0$ but a tiny $\mathrm{L}_N(\mathbf{w})$ value cannot be accomplished:

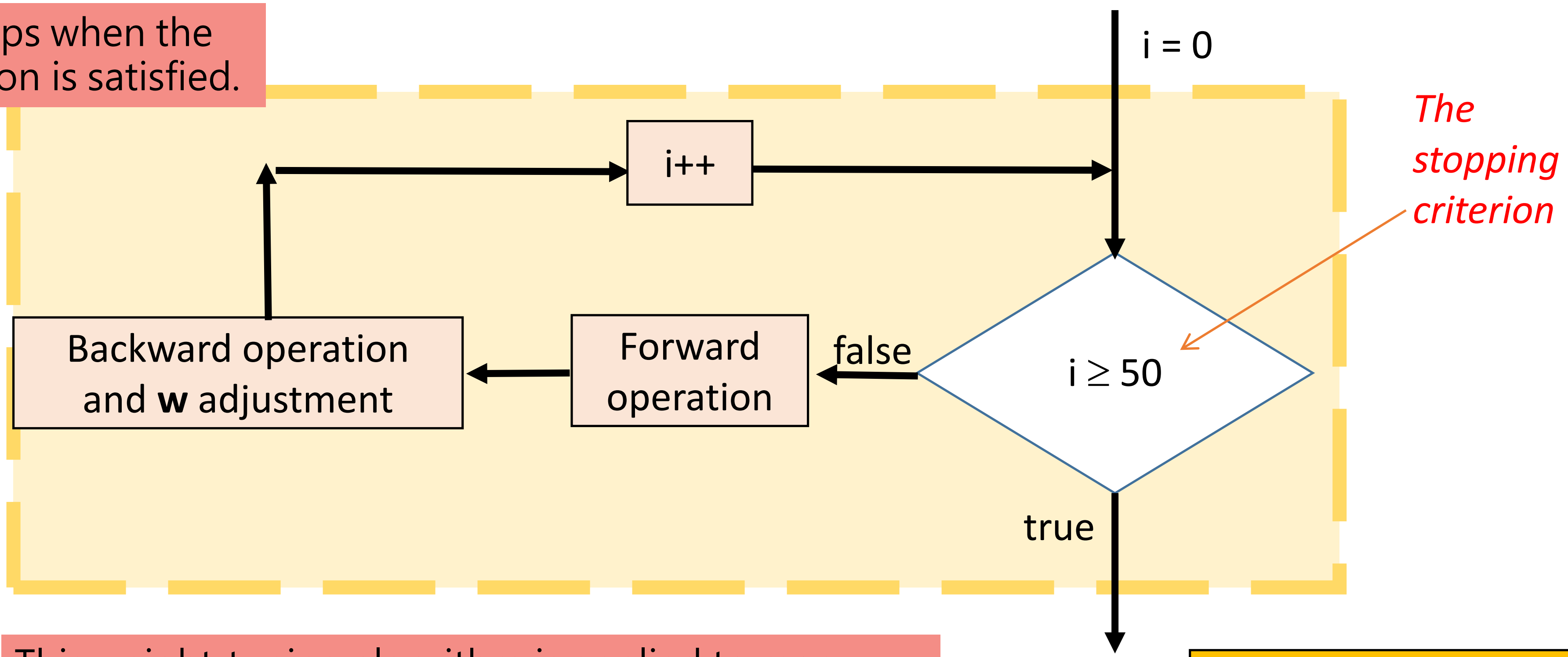
a) the local minimum, the saddle point, or the plateau.

b) the global minimum of the defective network architecture.

Where we are now...

The flowchart of weight-tuning algorithms for 2-layer neural networks in CS231n

The process stops when the stopping criterion is satisfied.

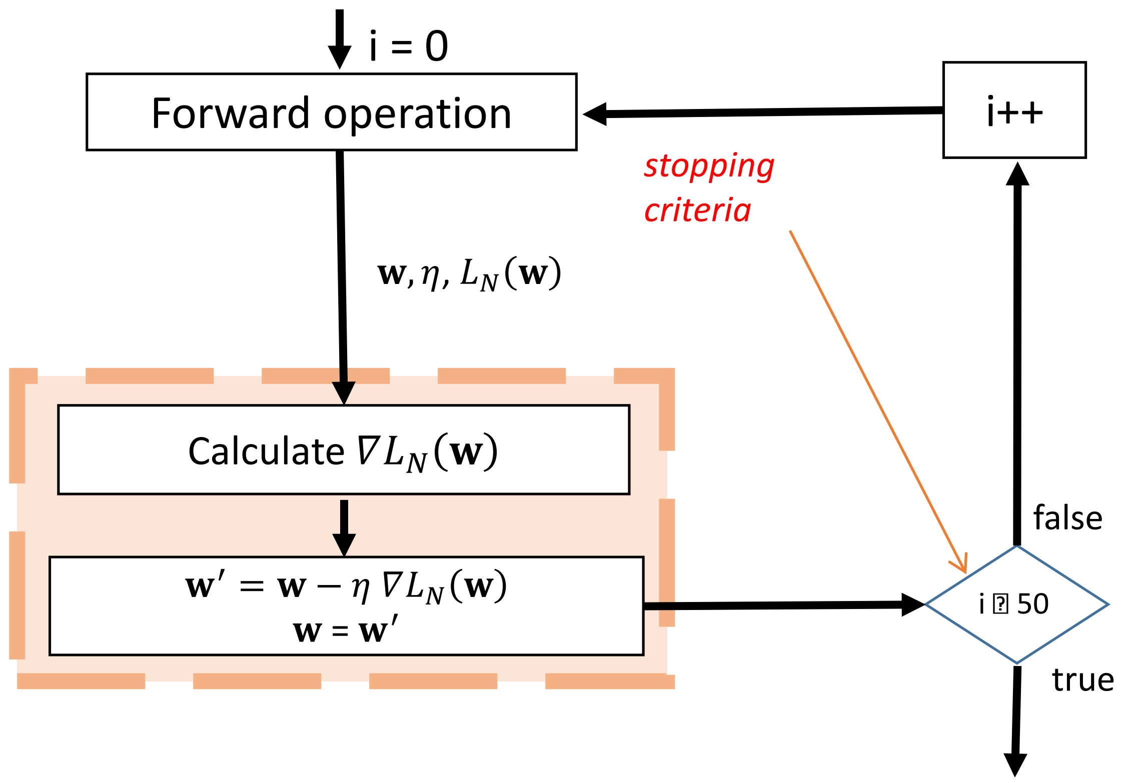


This weight-tuning algorithm is applied to many kinds of Neural Networks, including 2-layer neural networks, CNN, RNN, reinforcement learning, GAN, BERT, and so on.

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

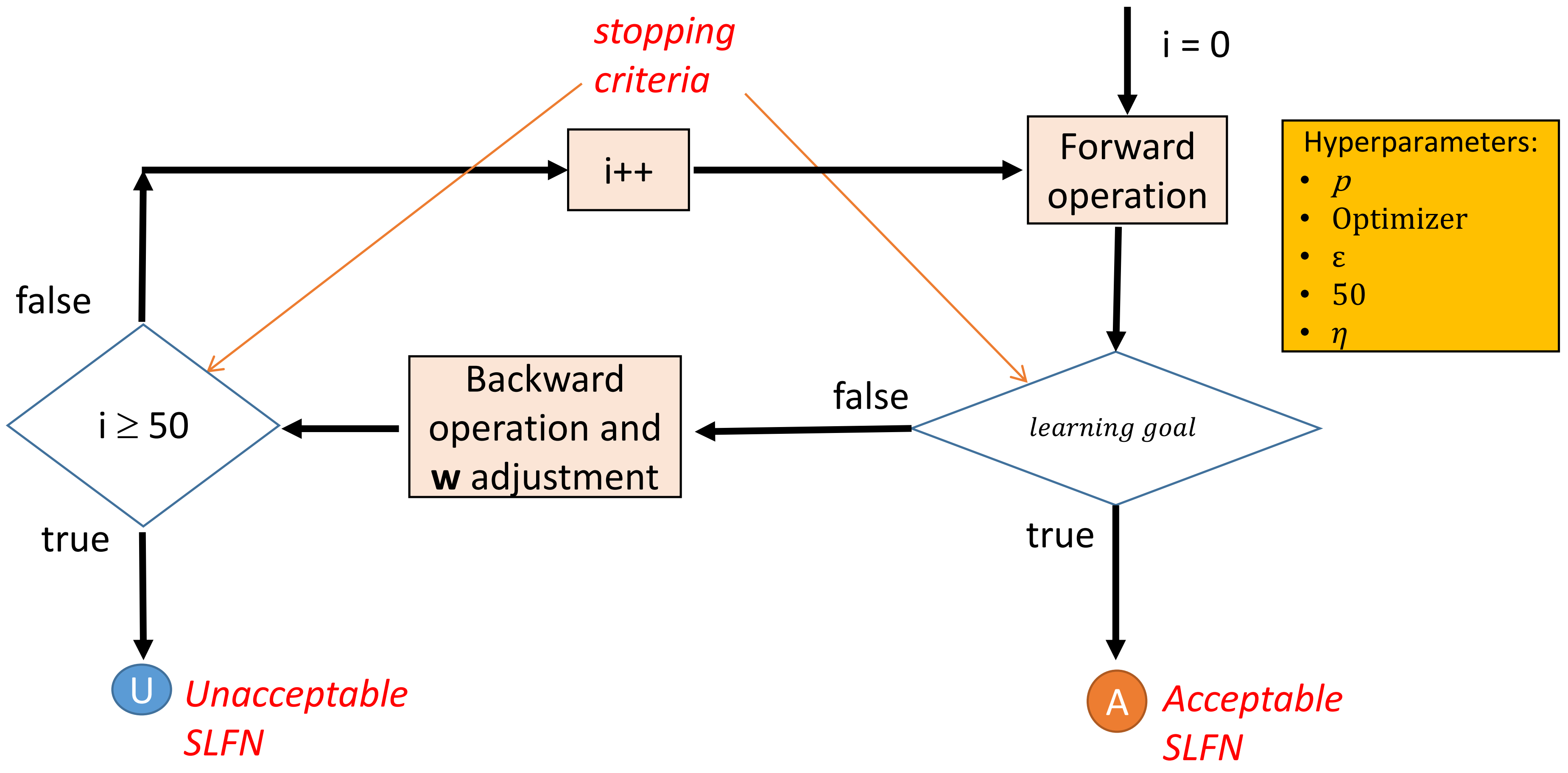
Where we are now...

The flowchart of weight-tuning module_EU


i = 0
Forward operation
i++
stopping criteria
$\mathbf{w}, \eta, L_N(\mathbf{w})$
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta \nabla L_N(\mathbf{w})$
$\mathbf{w} = \mathbf{w}'$
false
i ⍰ 50
true

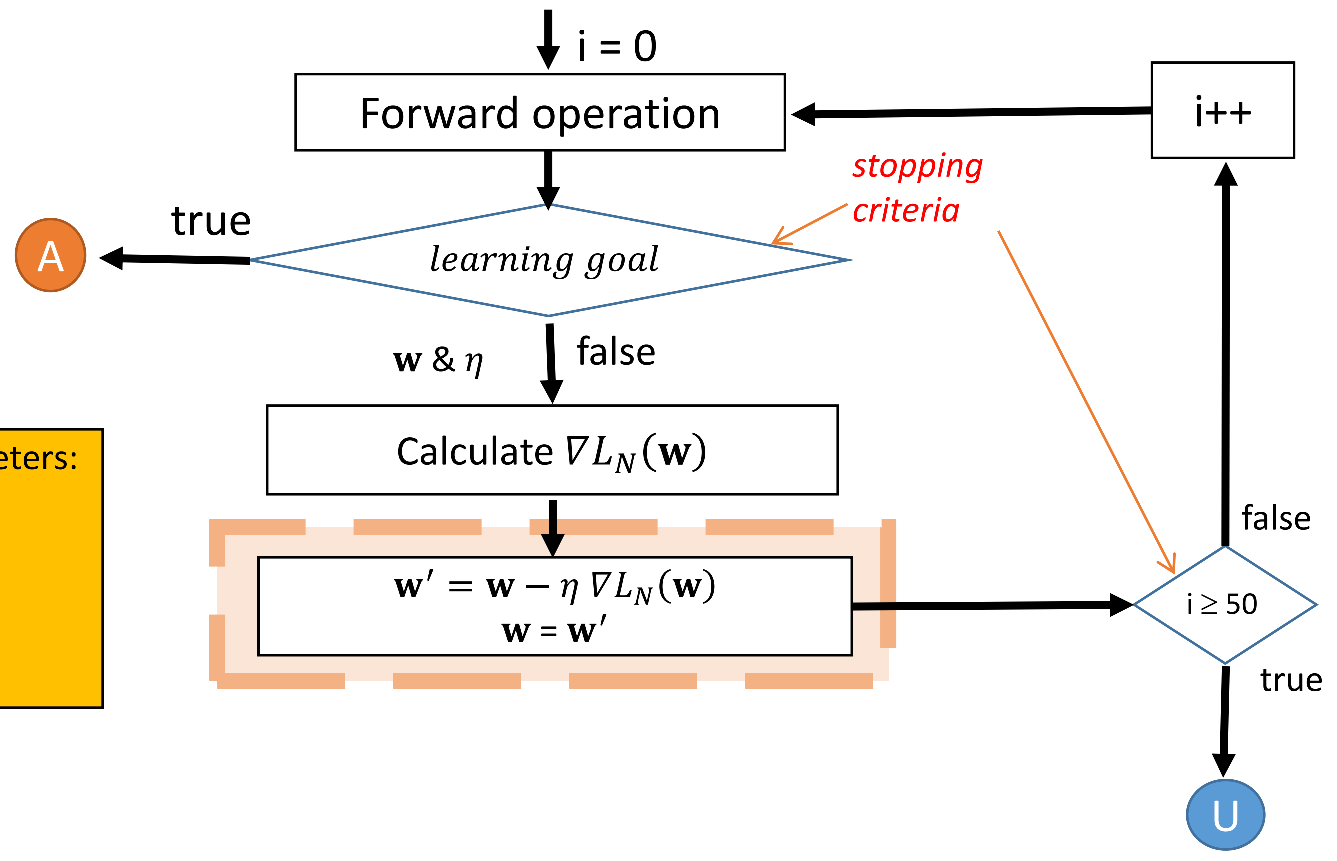
Hyperparameters:
- p
- Optimizer
- ε
- 50
- η

The flowchart of weight-tuning module_EU_LG that indicate either an unacceptable SLFN or an acceptable SLFN



Where we are now...

The flowchart of weight-tuning module_EU_LG

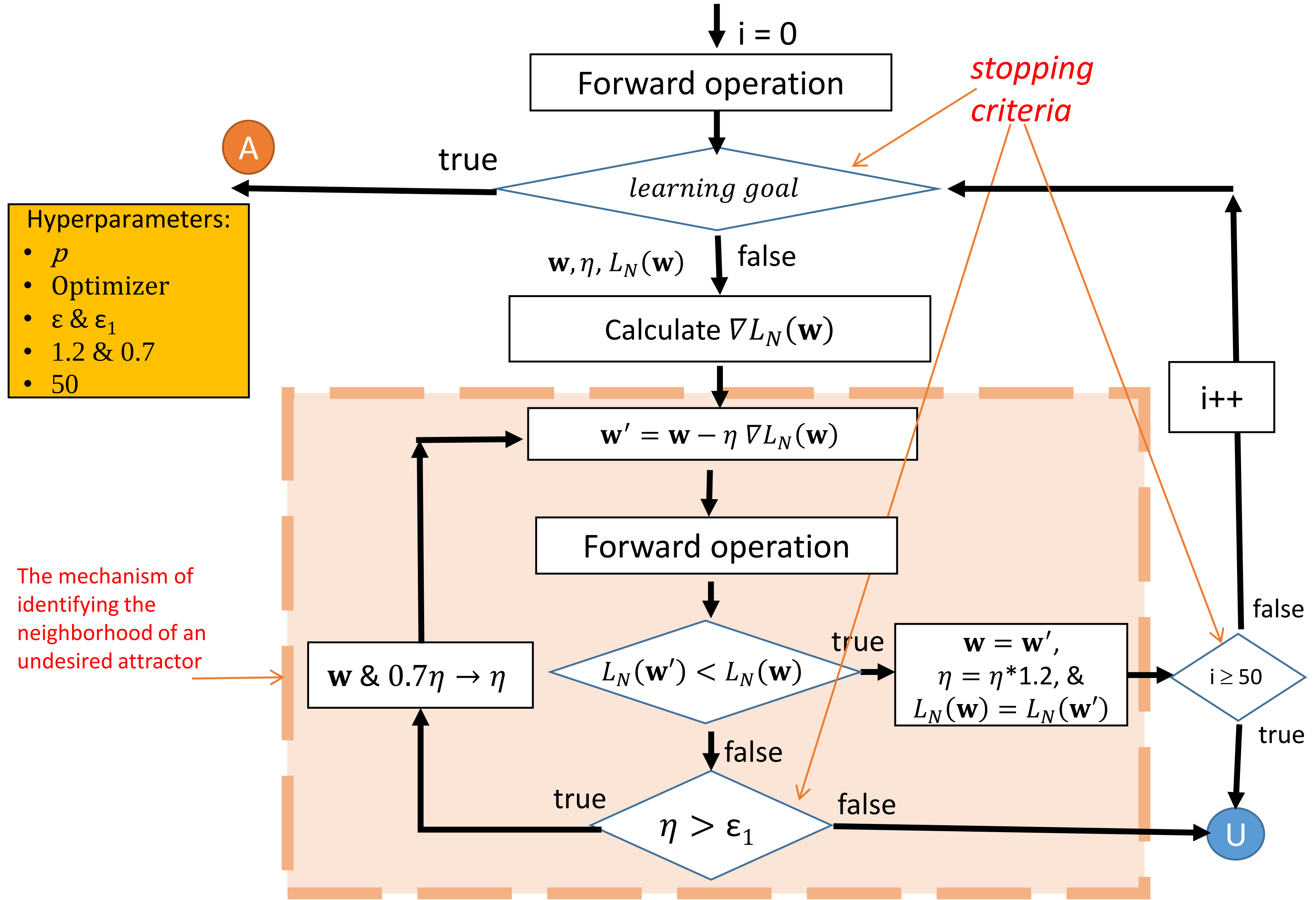


i = 0
Forward operation
i++
stopping criteria
true
A
$learning\ goal$
$\mathbf{w}$ & η
false
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta\, \nabla L_N(\mathbf{w})$
$\mathbf{w} = \mathbf{w}'$
false
$i \geq 50$
true
U

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

Where we are now...

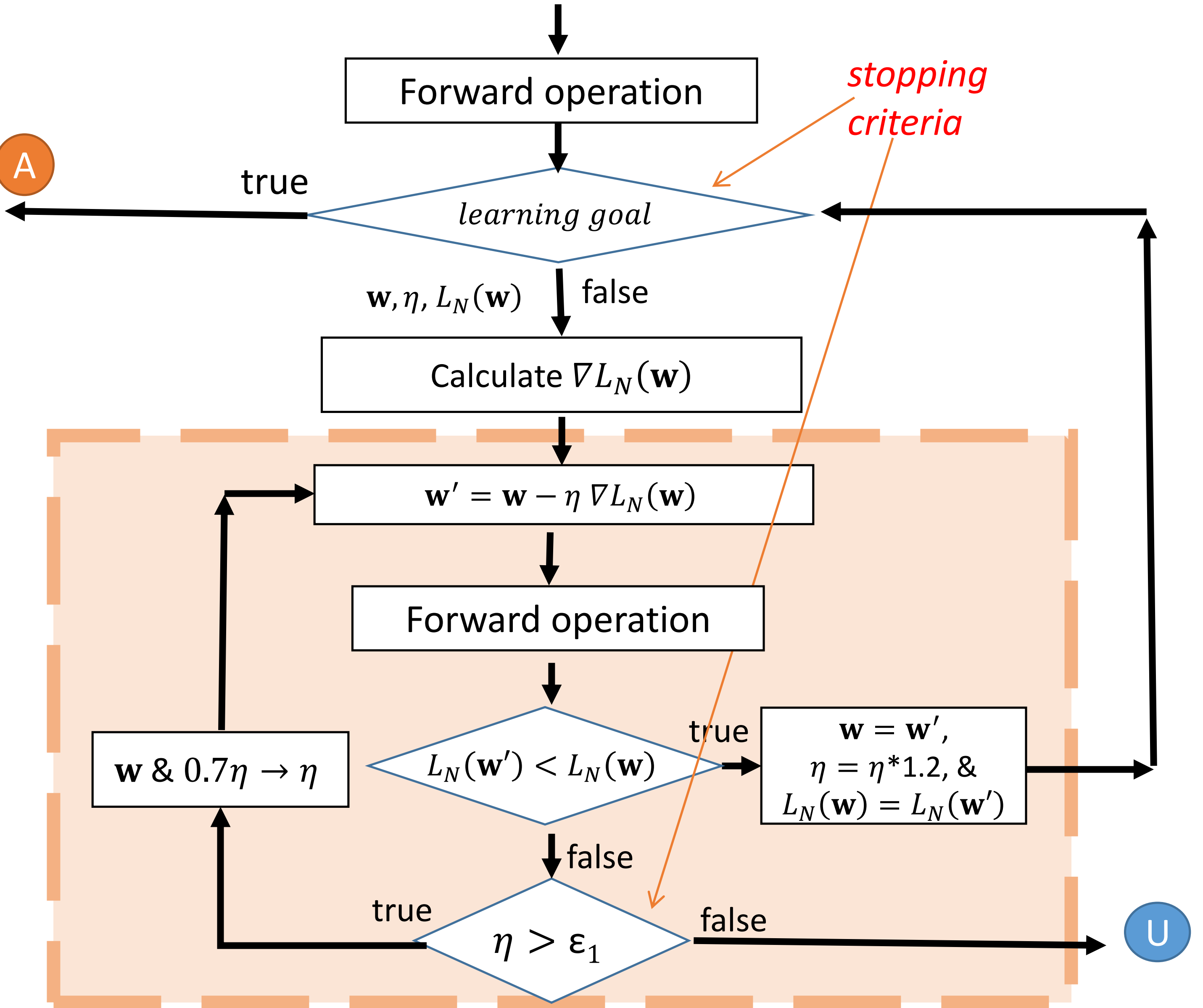
The flowchart of weight-tuning module_EU_LG_UA



i = 0
Forward operation
stopping criteria
A
true
learning goal
Hyperparameters:
• p
• Optimizer
• ε & ε_1
• 1.2 & 0.7
• 50
$\mathbf{w}, \eta, L_N(\mathbf{w})$
false
Calculate $\nabla L_N(\mathbf{w})$
i++
$\mathbf{w}' = \mathbf{w} - \eta \nabla L_N(\mathbf{w})$
Forward operation
The mechanism of identifying the neighborhood of an undesired attractor
false
$\mathbf{w}$ & $0.7\eta \to \eta$
$L_N(\mathbf{w}') < L_N(\mathbf{w})$
true
$\mathbf{w} = \mathbf{w}'$, $\eta = \eta$*1.2, & $L_N(\mathbf{w}) = L_N(\mathbf{w}')$
i ≥ 50
true
false
true
$\eta > \varepsilon_1$
false
U

Where we are now...

The flowchart of weight-tuning module_LG_UA



Forward operation
stopping criteria
A
true
learning goal
$\mathbf{w}, \eta, L_N(\mathbf{w})$
false
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta \nabla L_N(\mathbf{w})$
Forward operation
true
$\mathbf{w} = \mathbf{w}'$, $\eta = \eta$*1.2, & $L_N(\mathbf{w}) = L_N(\mathbf{w}')$
$\mathbf{w}$ & $0.7\eta \rightarrow \eta$
$L_N(\mathbf{w}') < L_N(\mathbf{w})$
false
true
$\eta > \varepsilon_1$
false
U

Hyperparameters:

- p
- Optimizer
- ε & ε_1
- 1.2 & 0.7

Performance differences amongst weight-tuning modules

- 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

Algorithm 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 ← learning-based prediction problem
  - ✓ Development of a model ← 2-layer net, 4-layer net, or deep neural networks
  - ✓ Specification of the algorithm ← The learning algorithm
  - ✓ Designing an algorithm ← The gradient-descent-based learning algorithm
  - ✓ Checking the correctness of the algorithm ← The math proof of the proposed learning algorithm
  - ✓ Analysis of algorithm ← The amount of parameters, the (learning and inferencing) time scale, …
  - ✓ Implementation of algorithm ← The coding
  - ✓ Program testing
  - ✓ Documentation preparation

Program testing --
Performance of AI Applications

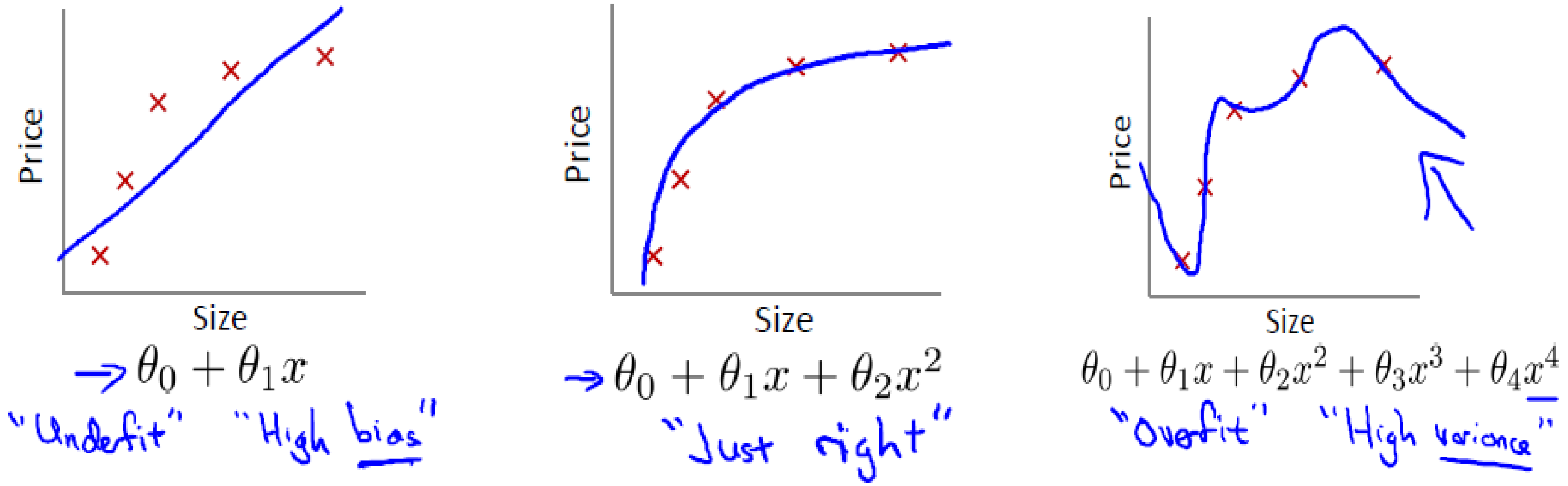
- How do AI professionals evaluate the performance of the AI applications?

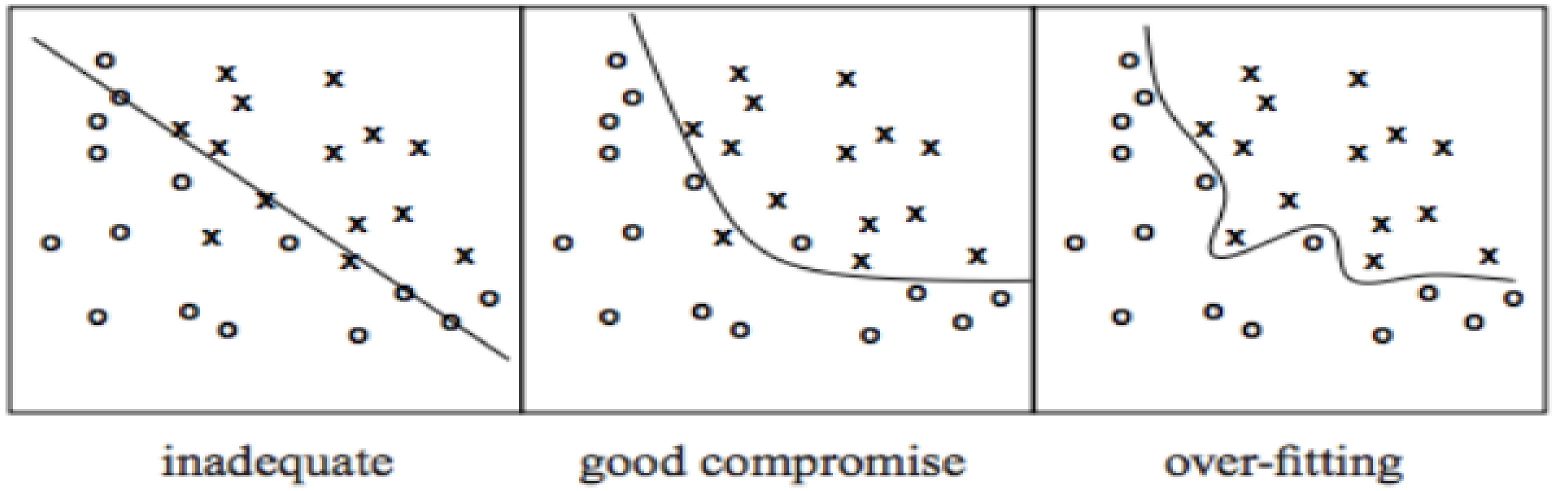
  ← effectiveness & efficiency

- However, there are learning dilemma and overfitting when evaluating the effectiveness & efficiency.
- You need to deal with learning dilemma and overfitting, not only for the purposes of learning, but also of inferencing.

Where we are now...

overfitting





http://wiki.bethanycrane.com/overfitting-of-data

Generalization

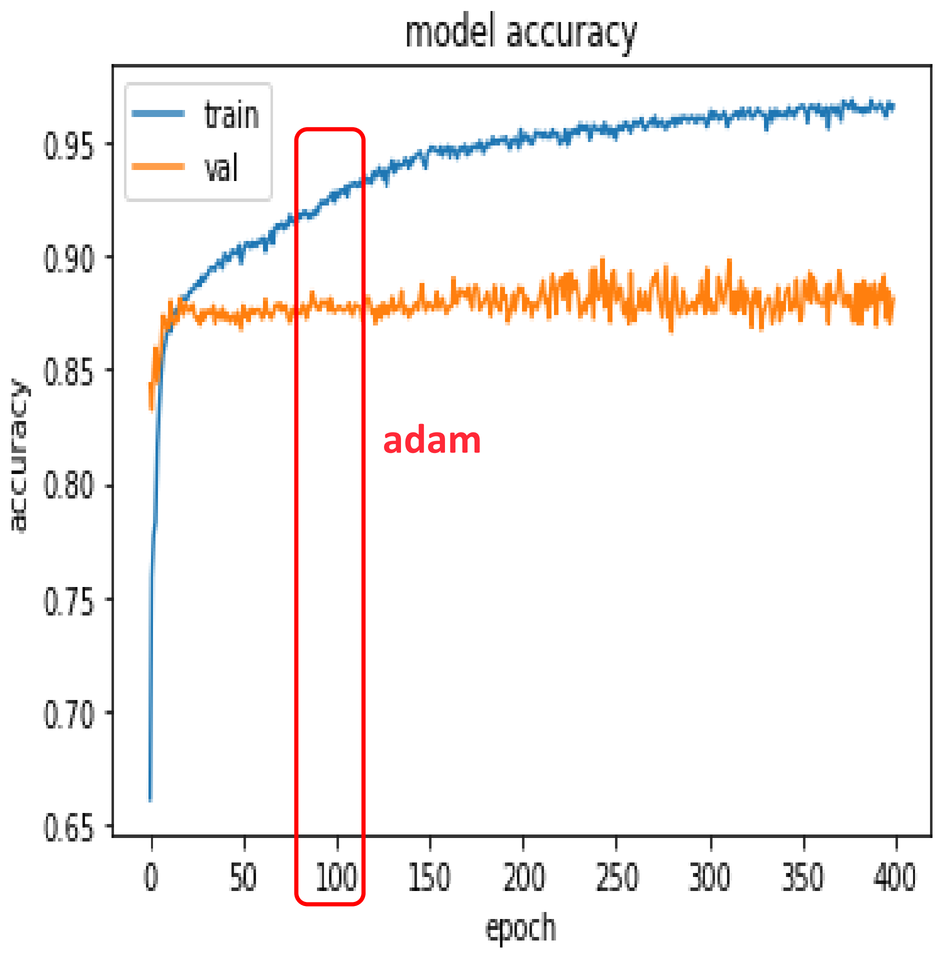
- Learned hypothesis may fit the training data very well, even noises (or outliers) in the training data, but fail to generalize to new examples (test data)
- In machine learning and statistical regression, the generalization error (also known as the out-of-sample error) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.

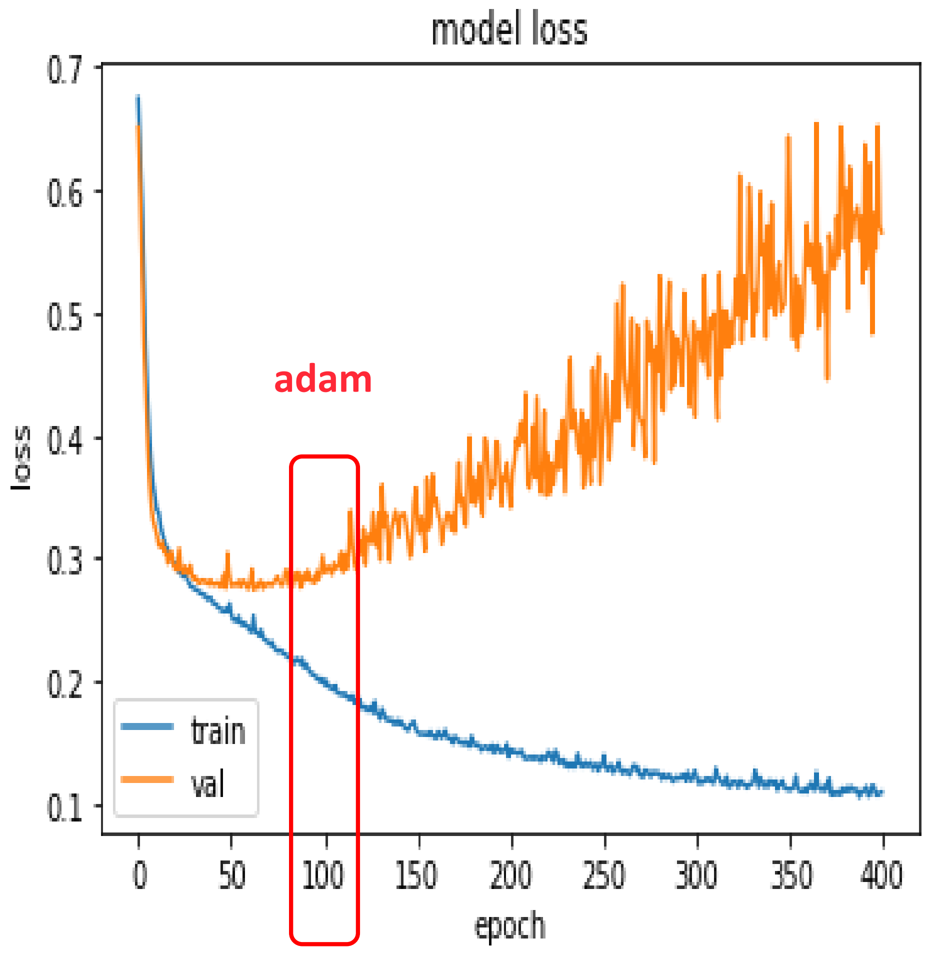
Learning curves

- Because learning algorithms are evaluated on finite samples, the evaluation of a learning algorithm may be sensitive to sampling error.
- As a result, measurements of prediction error on the current data may not provide much information about predictive ability on new data.
- The performance of a learning algorithm is measured by plots of the generalization error values through the learning process, which are called learning curves.
- Generalization error can be minimized by avoiding overfitting in the learning process.

Where we are now...

Learning curve and overfitting

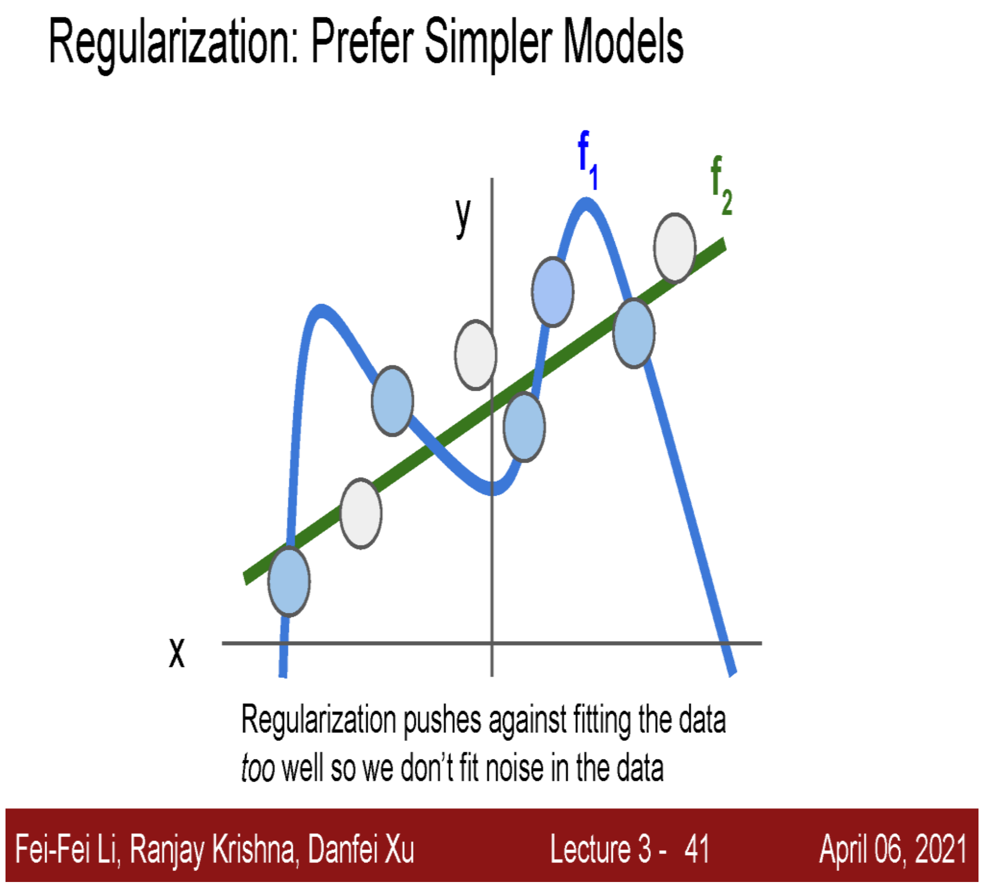




Early stop!

Overfitting

In statistics, **overfitting** is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably."



Regularization: Prefer Simpler Models

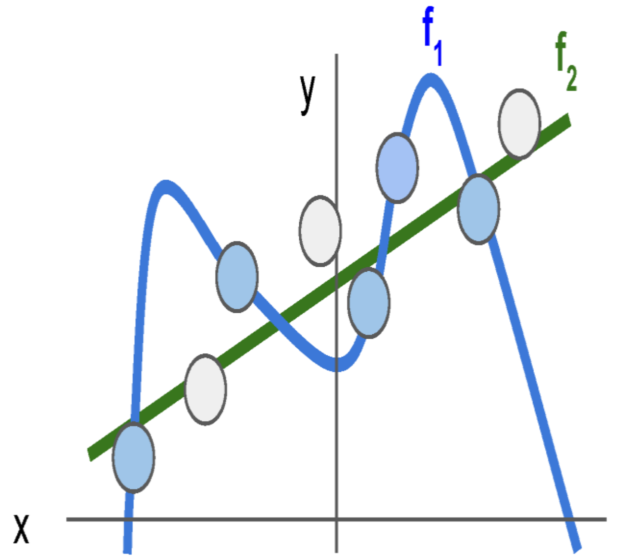
Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 3 - 41 April 06, 2021

Overfitting

An **over-fitted model** is a model that contains more parameters than can be justified by the data.

Regularization: Prefer Simpler Models



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 3 - 41 April 06, 2021

Regularization

λ = regularization strength (hyperparameter)

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Why regularize?
- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Regularization - In practice

Training: Add random noise
Testing: Marginalize over the noise

Examples:
Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop
Mixup

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets

Summary: the overfitting may be due to big weights

Regularization

λ = regularization strength (hyperparameter)

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

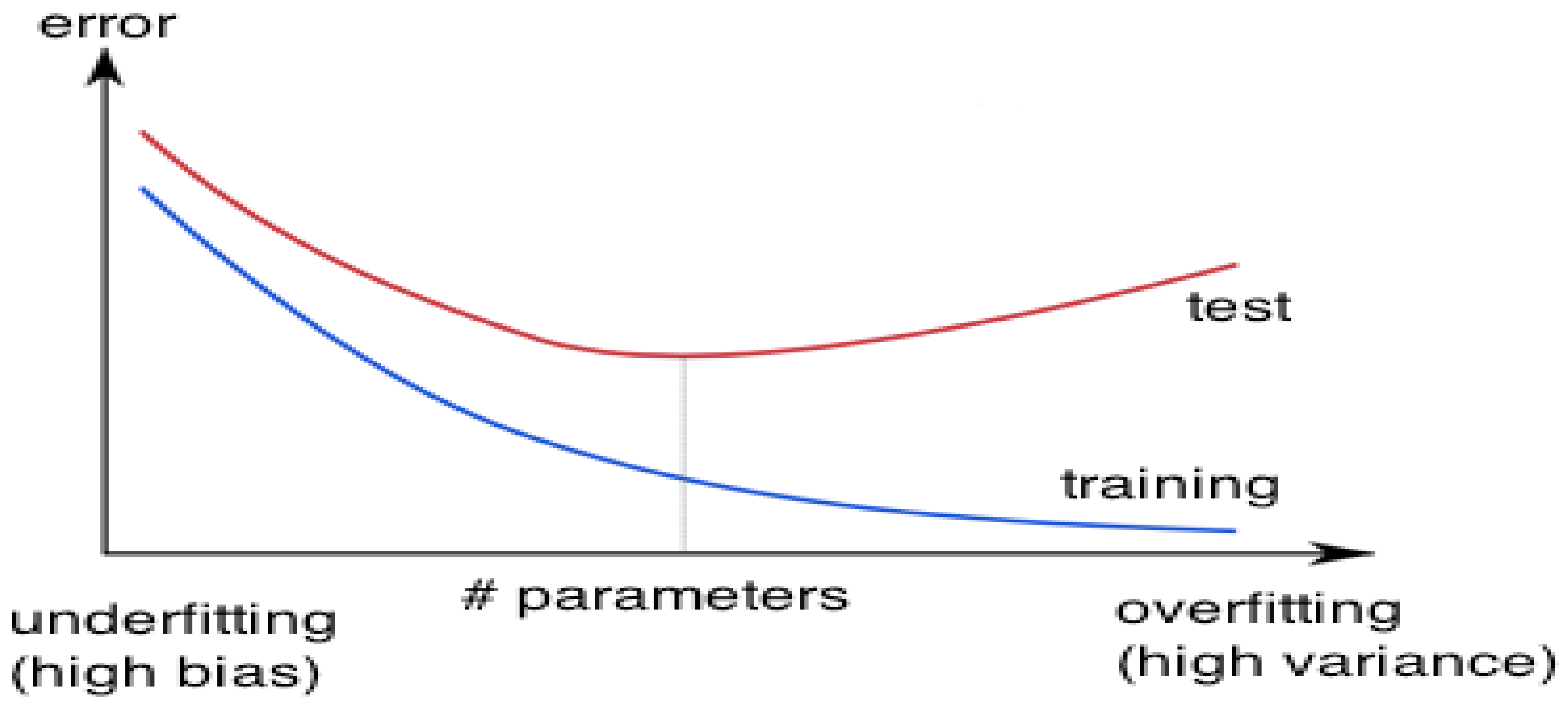
More complex:

Dropout

Batch normalization

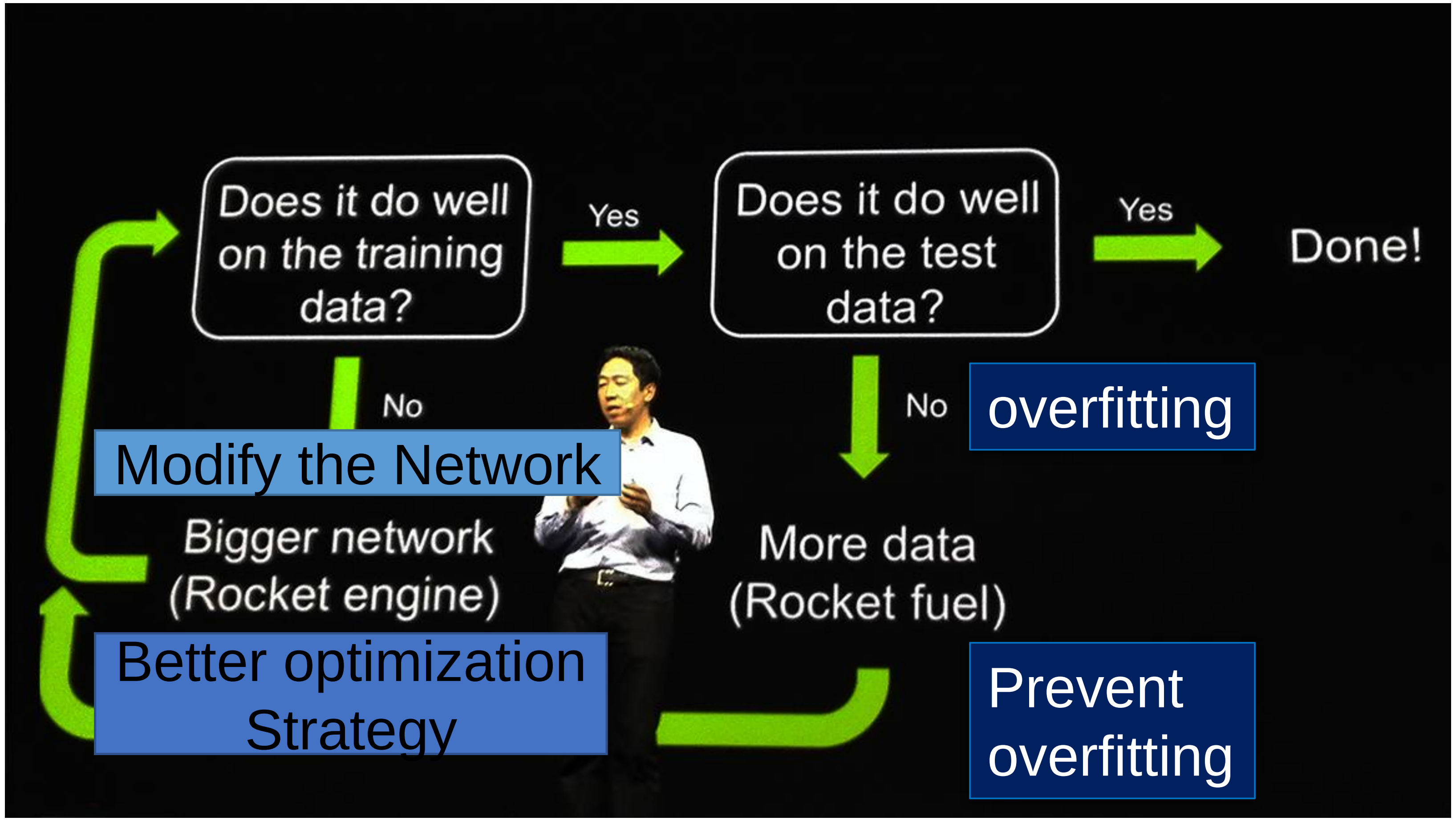
Stochastic depth, fractional pooling, etc

Summary: the overfitting may be due to too many hidden nodes



https://www.neuraldesigner.com/images/learning/selection_error.svg

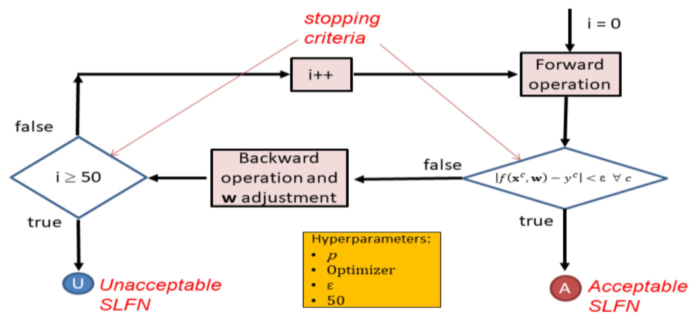
Recipe for Deep Learning

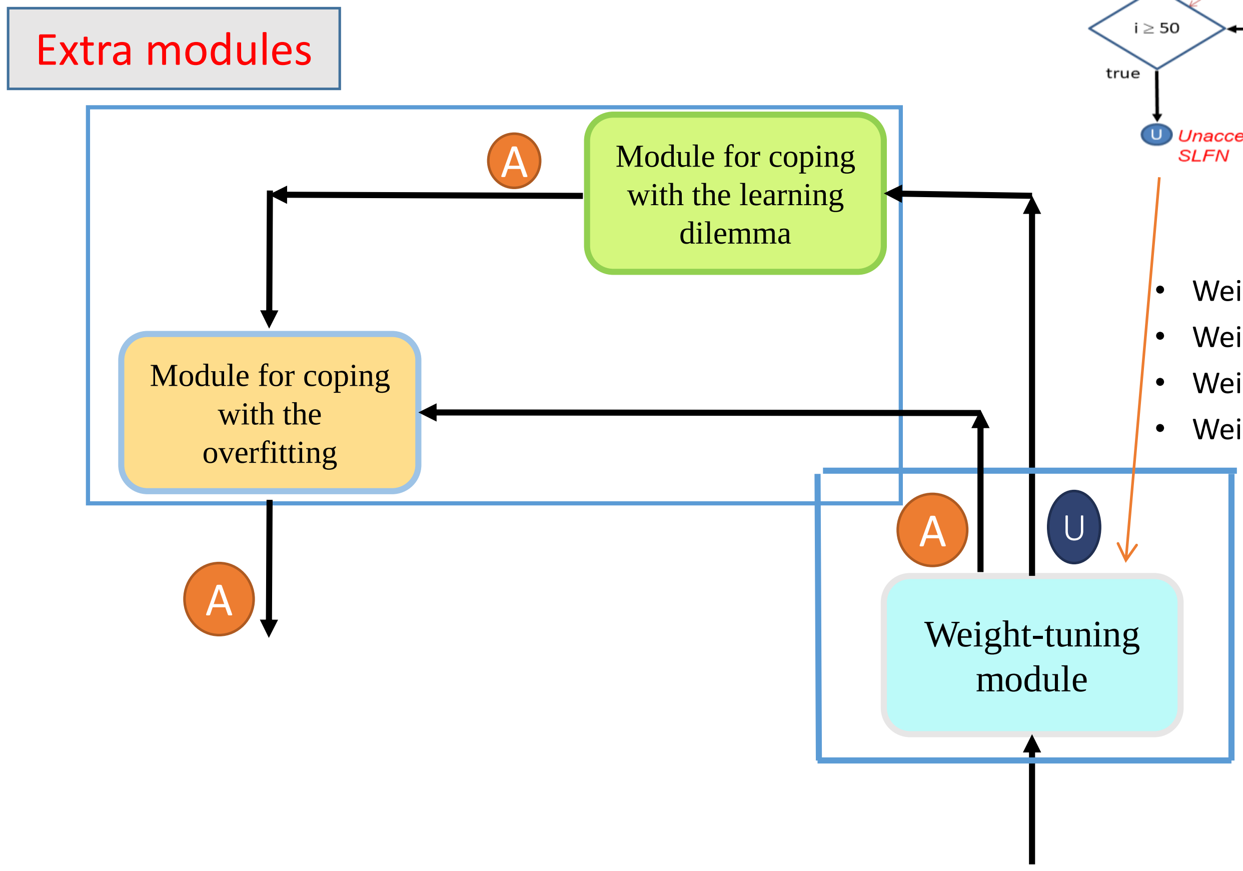


Does it do well on the training data?
Yes
Does it do well on the test data?
Yes
Done!
No
No
overfitting
Modify the Network
Bigger network (Rocket engine)
More data (Rocket fuel)
Better optimization Strategy
Prevent overfitting

http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

Inferencing Issues

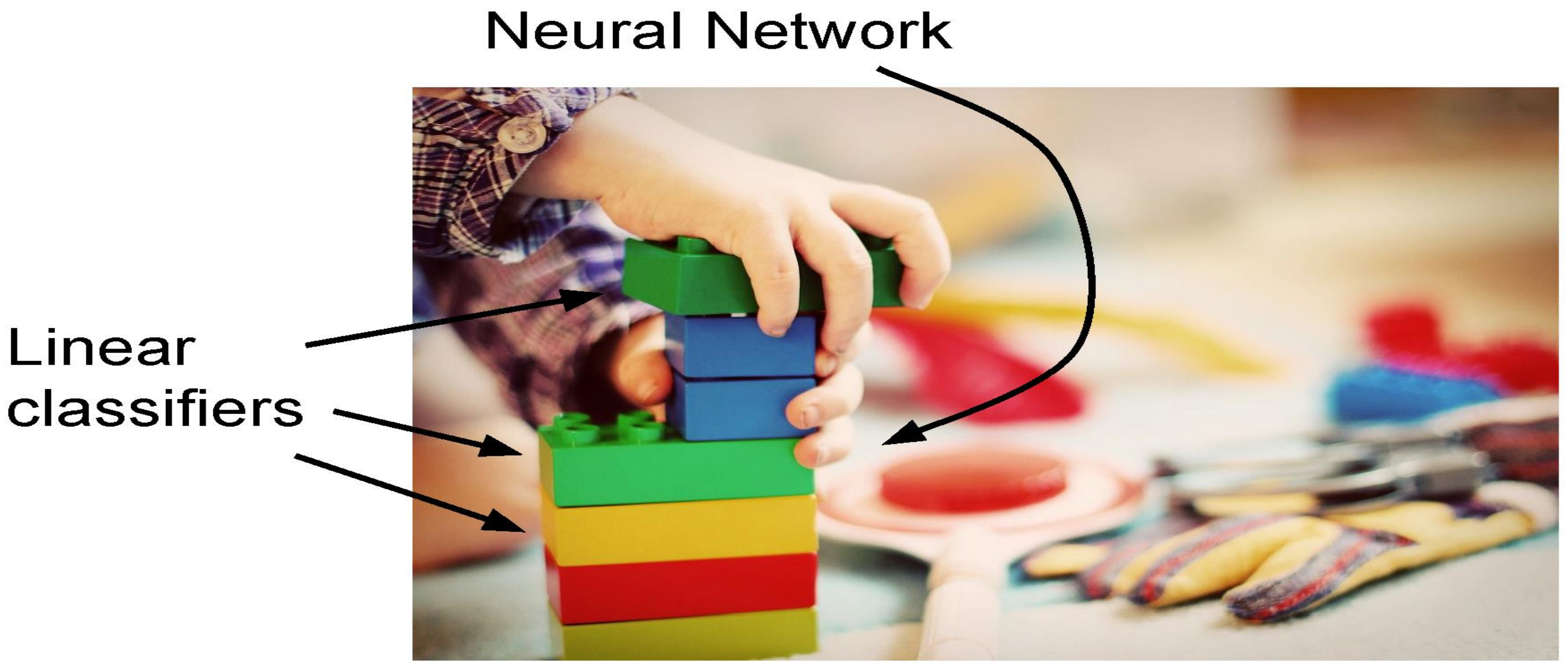



Extra modules
A
Module for coping with the learning dilemma
Module for coping with the overfitting
A
A
U
Weight-tuning module

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

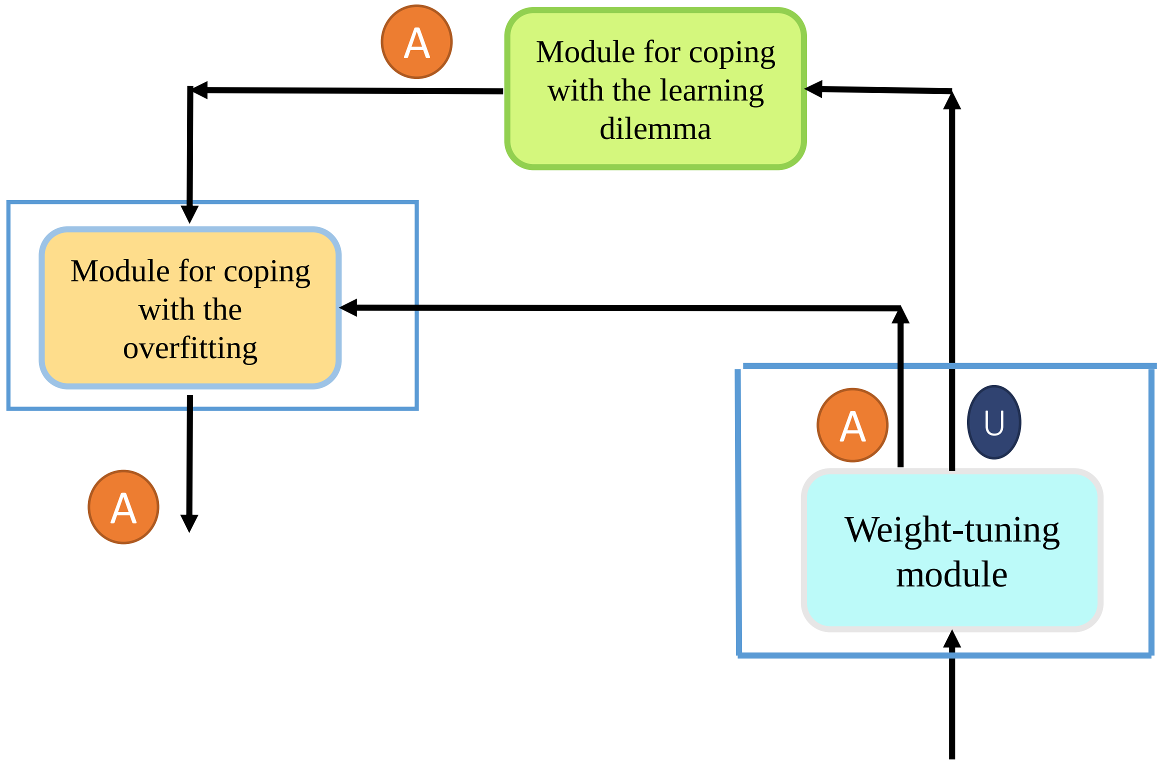
Where we are now...

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



Dealing with the overfitting due to big weights – the regularizing module

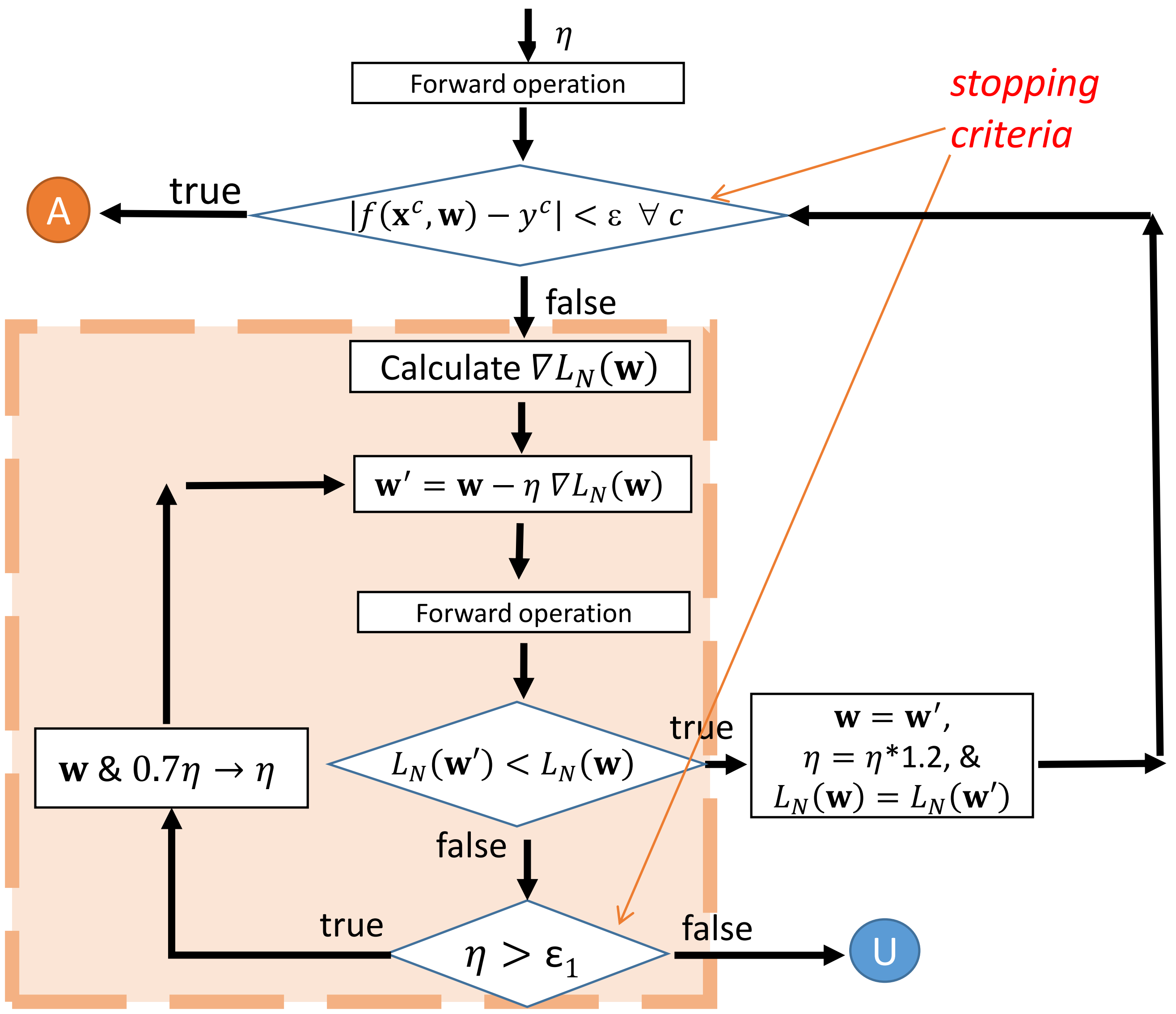
$$L_N(\mathbf{w}) \equiv \frac{\sum_c (f(\mathbf{x}^c, \mathbf{w}) - y^c)^2}{N} + \lambda\left(\sum_{i=0}^{p} (w_i^o)^2 + \sum_{i=1}^{p}\sum_{j=0}^{m} (w_{ij}^H)^2\right)$$


A
Module for coping with the learning dilemma
Module for coping with the overfitting
A
A
U
Weight-tuning module

Where we are now...

The flowchart of weight-tuning module_LG_UA

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

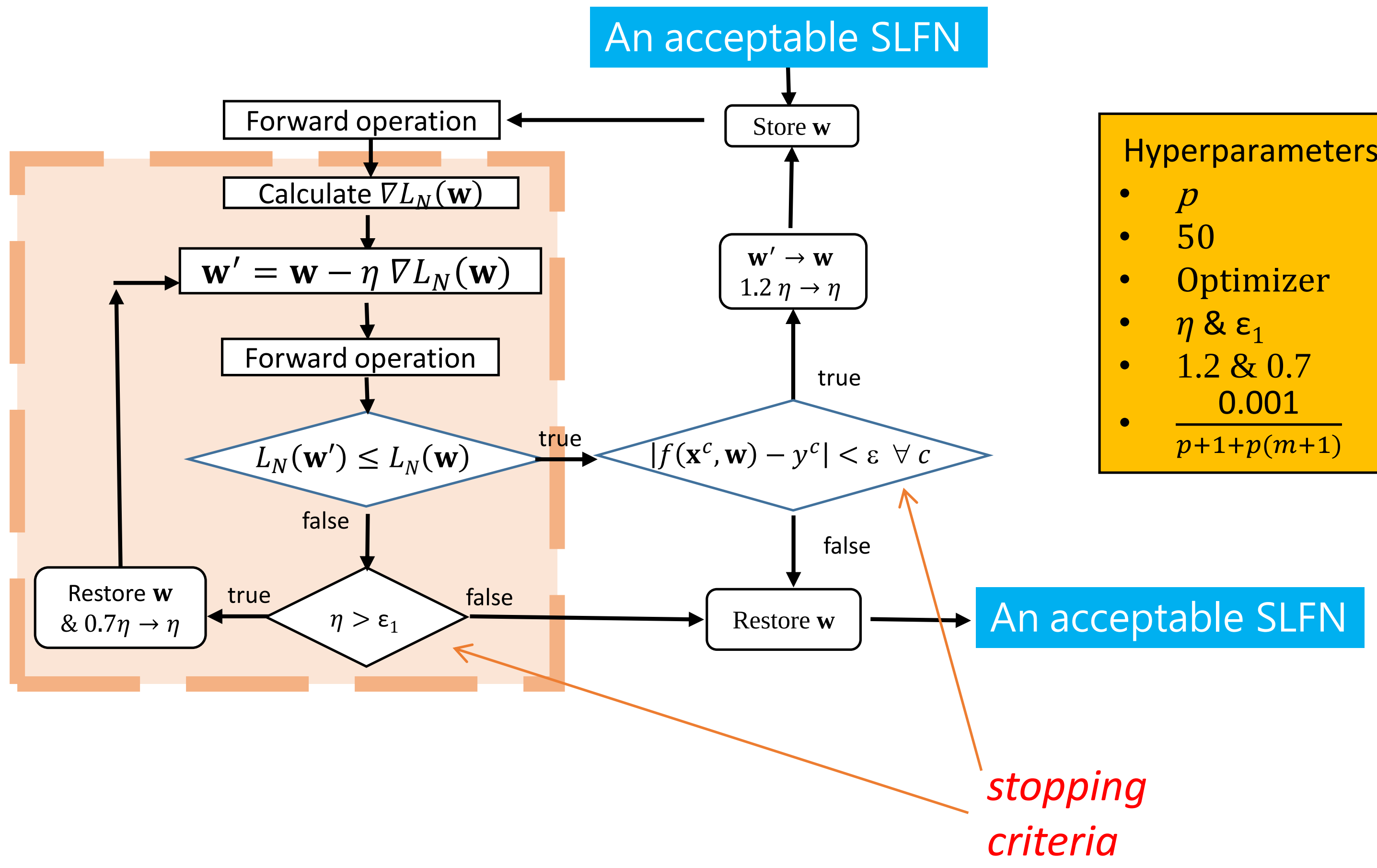


η
Forward operation
true
A
$|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \ \forall\, c$
stopping criteria
false
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta\, \nabla L_N(\mathbf{w})$
Forward operation
$\mathbf{w}$ & $0.7\eta \rightarrow \eta$
$L_N(\mathbf{w}') < L_N(\mathbf{w})$
true
$\mathbf{w} = \mathbf{w}'$, $\eta = \eta$*1.2, & $L_N(\mathbf{w}) = L_N(\mathbf{w}')$
false
true
$\eta > \varepsilon_1$
false
U

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

The flowchart of regularizing module_LG_UA that tries to further regularize weights of an acceptable SLFN

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



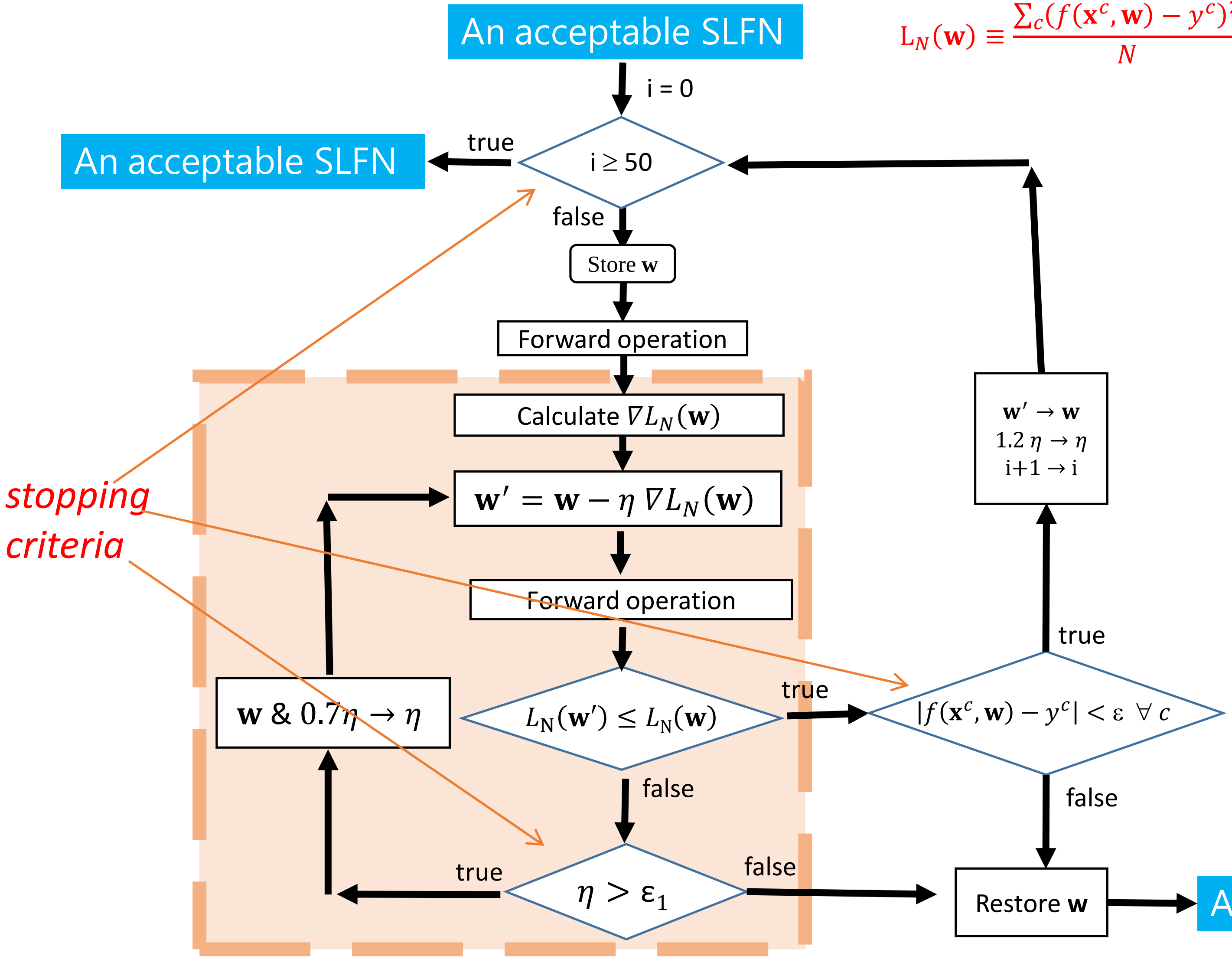
An acceptable SLFN
Store $\mathbf{w}$
Forward operation
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta \nabla L_N(\mathbf{w})$
Forward operation
$L_N(\mathbf{w}') \le L_N(\mathbf{w})$
true
false
$|f(\mathbf{x}^c,\mathbf{w}) - y^c| < \varepsilon \ \forall c$
true
$\mathbf{w}' \to \mathbf{w}$, $1.2\,\eta \to \eta$
false
$\eta > \varepsilon_1$
true
Restore $\mathbf{w}$ & $0.7\eta \to \eta$
false
Restore $\mathbf{w}$
An acceptable SLFN
stopping criteria

Hyperparameters:

- p
- 50
- Optimizer
- η & ε_1
- 1.2 & 0.7
- $\frac{0.001}{p+1+p(m+1)}$

The flowchart of regularizing module_EU_LG_UA

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



An acceptable SLFN
i = 0
$i \geq 50$
true
An acceptable SLFN
false
Store $\mathbf{w}$
Forward operation
Calculate $\nabla L_N(\mathbf{w})$
$\mathbf{w}' = \mathbf{w} - \eta \nabla L_N(\mathbf{w})$
Forward operation
$L_N(\mathbf{w}') \leq L_N(\mathbf{w})$
true
$|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \ \forall c$
true
$\mathbf{w}' \to \mathbf{w}$
$1.2\eta \to \eta$
$i+1 \to i$
false
Restore $\mathbf{w}$
An acceptable SLFN
false
$\eta > \varepsilon_1$
true
false
$\mathbf{w}$ & $0.7\eta \to \eta$
stopping criteria

Hyperparameters:

- p
- 50
- Optimizer
- η & ε_1
- 1.2 & 0.7
- $\frac{0.001}{p+1+p(m+1)}$

The regularizing module

- The weight-tuning module helps tune up the weights to decrease the data error to obtain an acceptable SLFN.
- 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.

Q: Which optimizer does better in the regularizing module?

A: Need to conduct experiments to get the better optimizer.

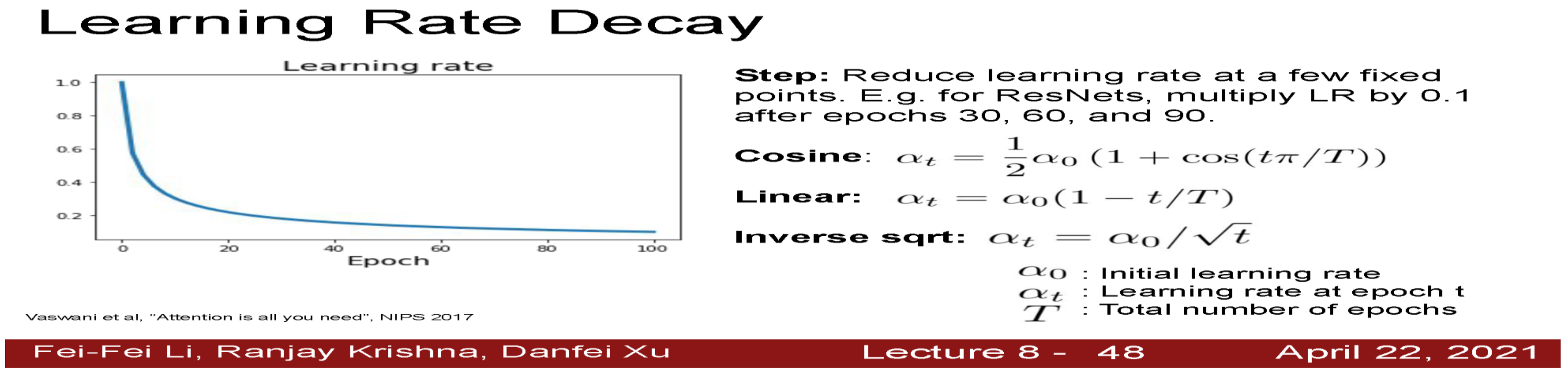
The overfitting due to big weights

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

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

- 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
- Maybe there should be a RS scheduling like the LR scheduling. The RS should be enlarged from a tiny value.



Learning Rate Decay

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

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

Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 8 - 48 April 22, 2021

Performance differences amongst regularizing modules

- There are two regularizing modules
  - ✓ the regularizing module_EU_LG_UA
  - ✓ the regularizing module_LG_UA
  - ✓ the regularizing module_EU

- What are the differences amongst these regularizing modules?

Performance differences amongst regularizing modules

- There are two regularizing modules
  - ✓ the regularizing module_EU_LG_UA
    The regularizing time length is expected
  - ✓ the regularizing module_LG_UA
    The regularizing time length may be much longer
  - ✓ the regularizing module_EU
    The simplest and the regularizing time length is expected

Regularization - In practice

Training: Add random noise

Testing: Marginalize over the noise

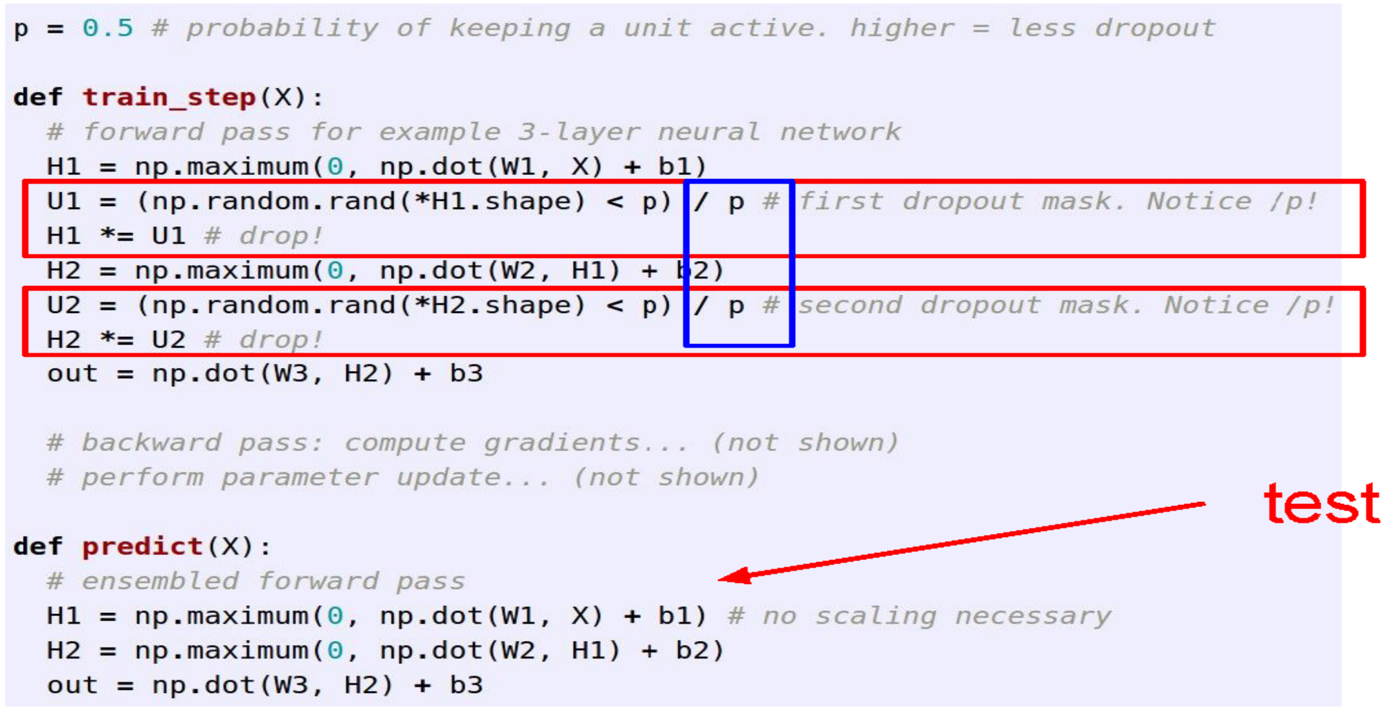
Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop
Mixup

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets

The regularizing module_DO

More common: “Inverted dropout”



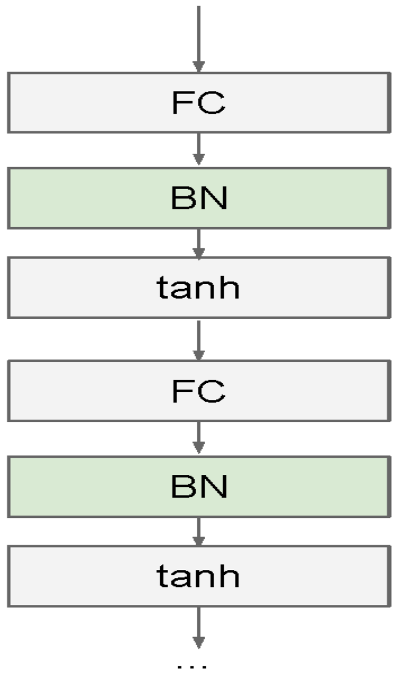
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

The regularizing module_BN

Batch Normalization

[Ioffe and Szegedy, 2015]

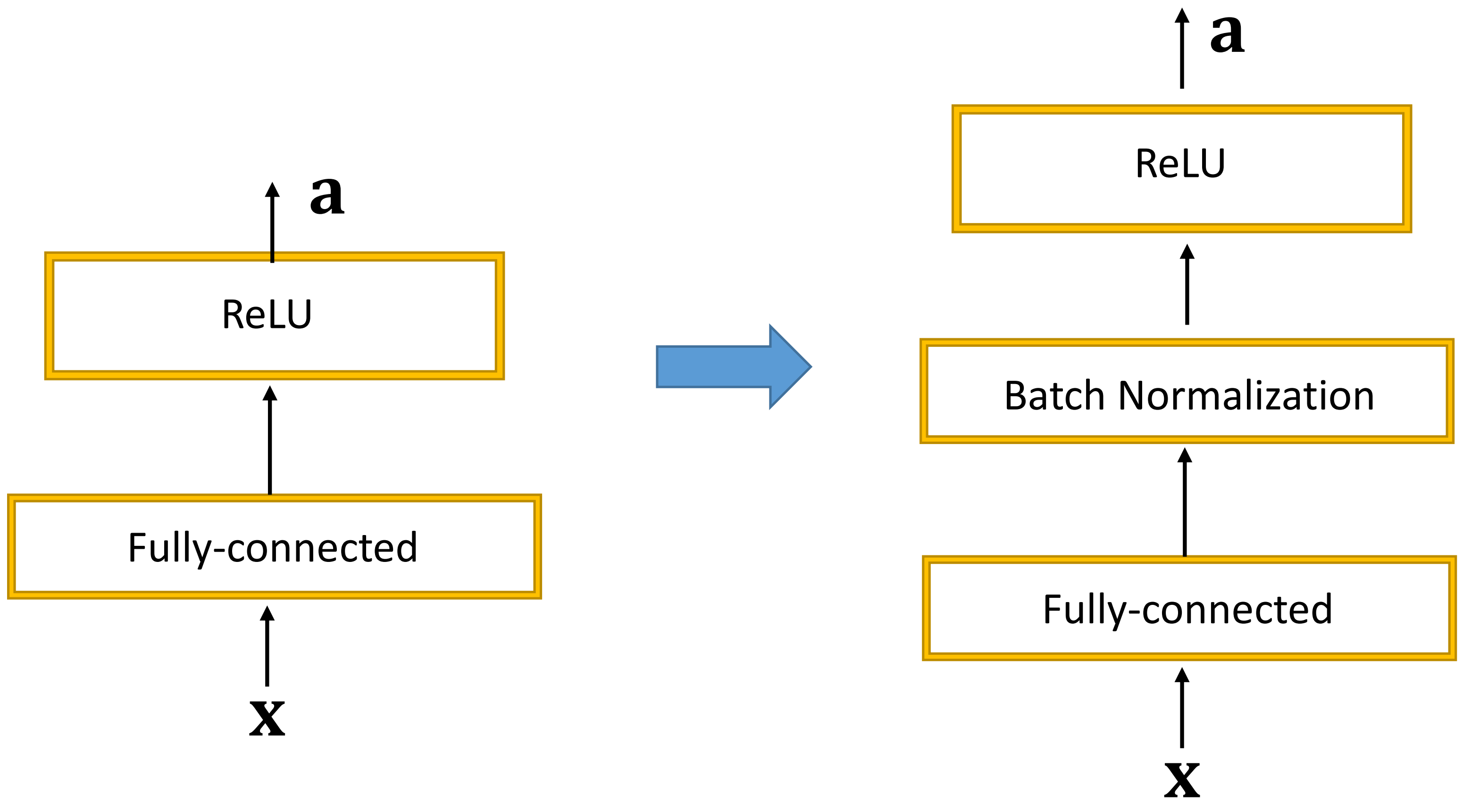


- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Fei-Fei Li, Ranjay Krishna, Danfei Xu Lecture 7 - 90 April 20, 2021

The regularizing module_BN

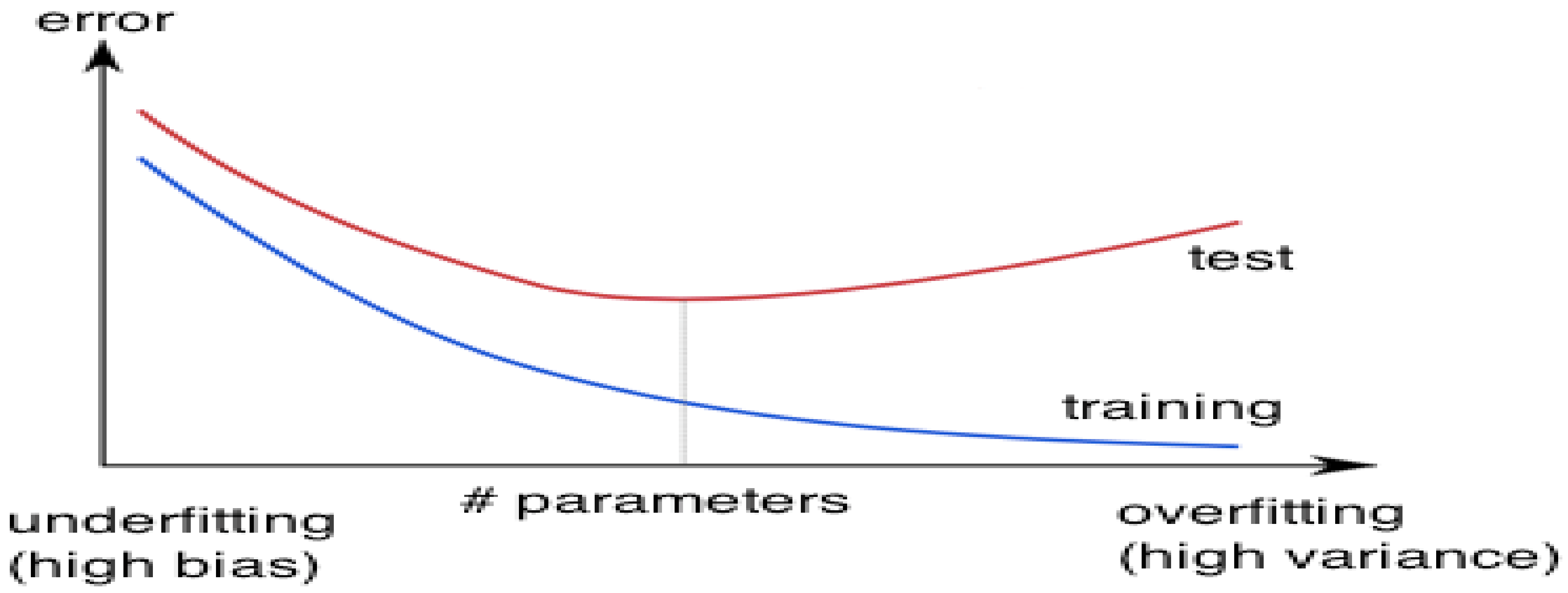
In the regularizing module_BN, the batch normalization operation is inserted after the FC layer and before the nonlinearity layer.



Homework 3_1

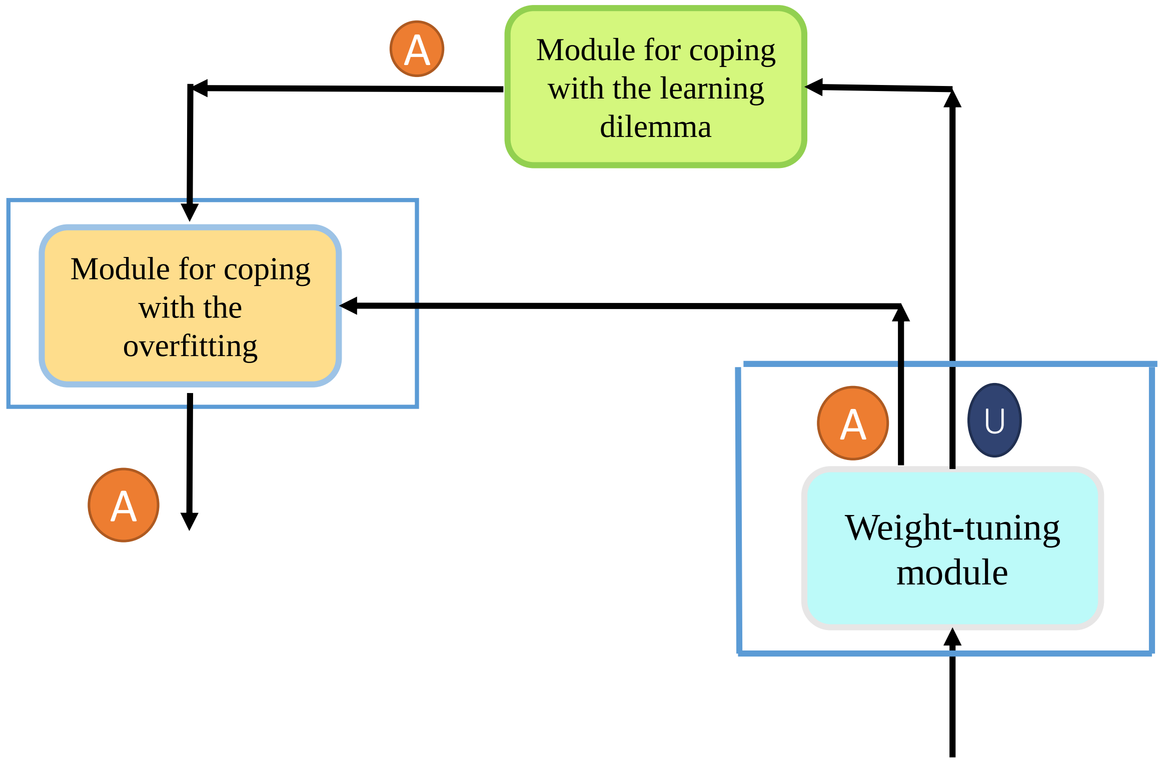
- Rewrite the code of weight-tuning module_LG_UA stated in page 46 to make the coding of regularizing module_LG_UA stated in page 47.
- Rewrite the code of regularizing module_LG_UA stated in page 47 to make the coding of regularizing module_EU_LG_UA stated in page 48.
- Make the coding of regularizing module_DO stated in page 54.
- Make the coding of regularizing module_BN stated in page 56.

Overfitting due to too many hidden nodes



https://www.neuraldesigner.com/images/learning/selection_error.svg

Dealing with the overfitting due to big weights and too many hidden nodes – the reorganizing module


A
Module for coping with the learning dilemma
Module for coping with the overfitting
A
U
Weight-tuning module
A

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

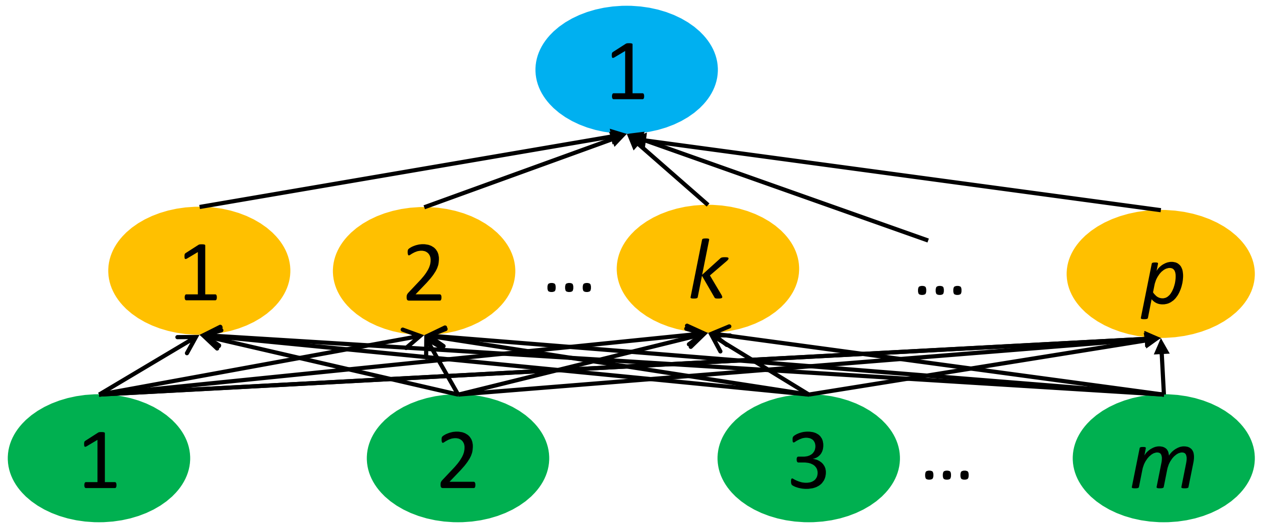
Occam's Razar: Among multiple competing hypotheses, the simplest is the best,
William of Ockham 1285-1347

irrelevant hidden nodes & potentially irrelevant hidden nodes

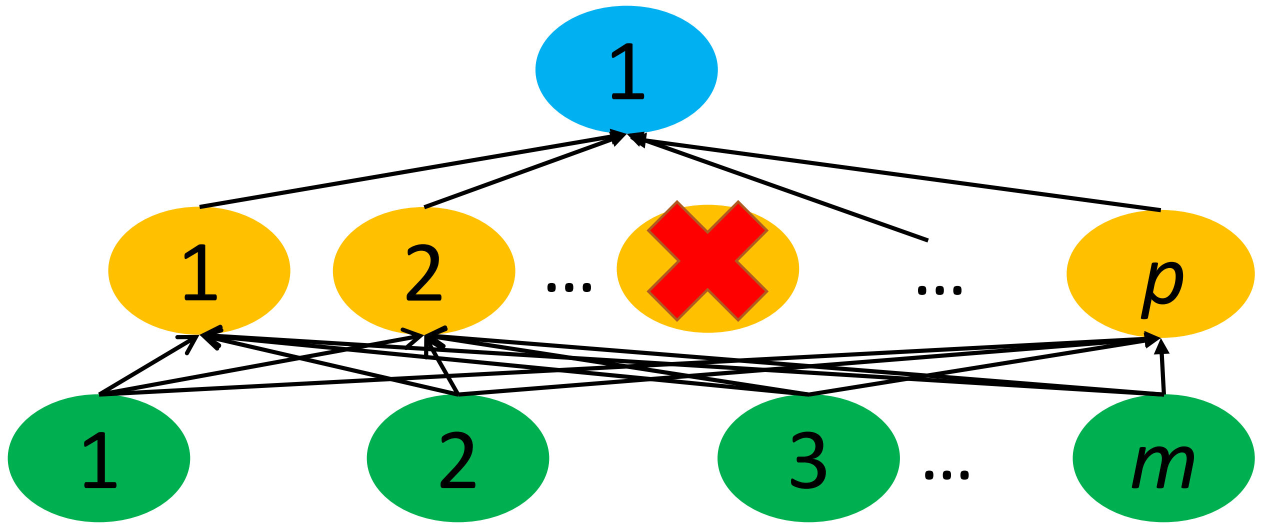
- Develop the reorganizing module that helps regularize weights of an acceptable SLFN while keeping the learning goal satisfied as well as identify and remove the *potentially irrelevant hidden node*.

- The hidden node that can be pruned without making the learning goal unsatisfied is an *irrelevant hidden node*. (Tsaih, 1993)
- For the SLFN with the $\mathbf{w}$, the k^{th} hidden node is *potentially irrelevant* if the learning goal can be accomplished via minimizing $\mathrm{L}_N(\mathbf{w}'_k)$, where $\mathbf{w}'_k \equiv \mathbf{w} - \{w^o_k, w^H_{k0}, \mathbf{w}^H_k\}$ and $f(\mathbf{x}^c, \mathbf{w}'_k) \equiv w^o_0 + \sum_{i \neq k} w^o_i a^c_i \; \forall \; c$. (Tsaih, 1993)

$\mathbf{w}$



$$\mathbf{w}'_k \equiv \mathbf{w} - \left\{w_k^o, w_{k0}^H, \mathbf{w}_k^H\right\}$$



The reorganizing module that helps regularize weights of an acceptable SLFN and examines its hidden nodes one by one

Note that there are two optimizers:

One for the regularizing purpose

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

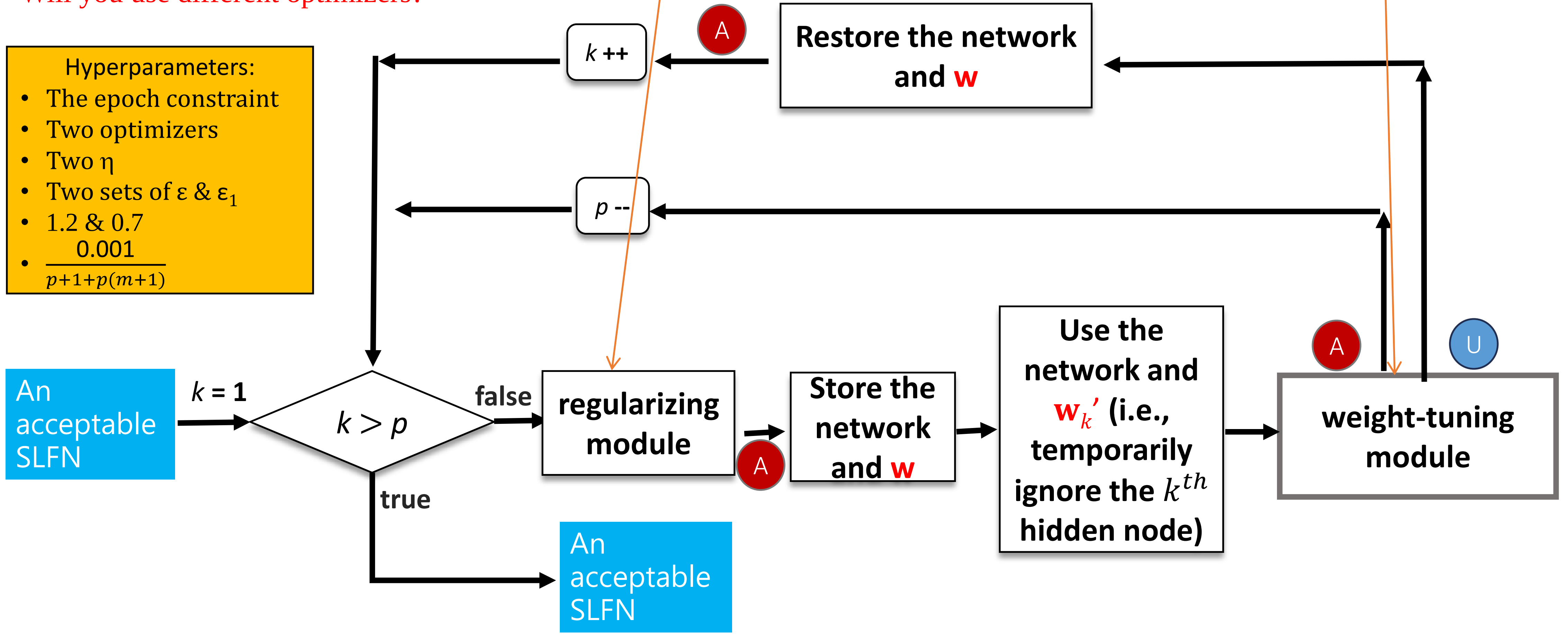
Another for the pruning purpose

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

Will you use different optimizers?

Hyperparameters:

- The epoch constraint
- Two optimizers
- Two η
- Two sets of ε & ε_1
- 1.2 & 0.7
- $\frac{0.001}{p+1+p(m+1)}$



An acceptable SLFN
k = 1
$k > p$
false
true
regularizing module
A
Store the network and **w**
Use the network and $\mathbf{w}_k{}'$ (i.e., temporarily ignore the k^{th} hidden node)
weight-tuning module
A
U
p --
Restore the network and **w**
A
k ++
An acceptable SLFN

The reorganizing module_ALL_r_EU_LG_UA_w_EU_LG_UA

Note that there are two optimizers:

One for the regularizing purpose

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

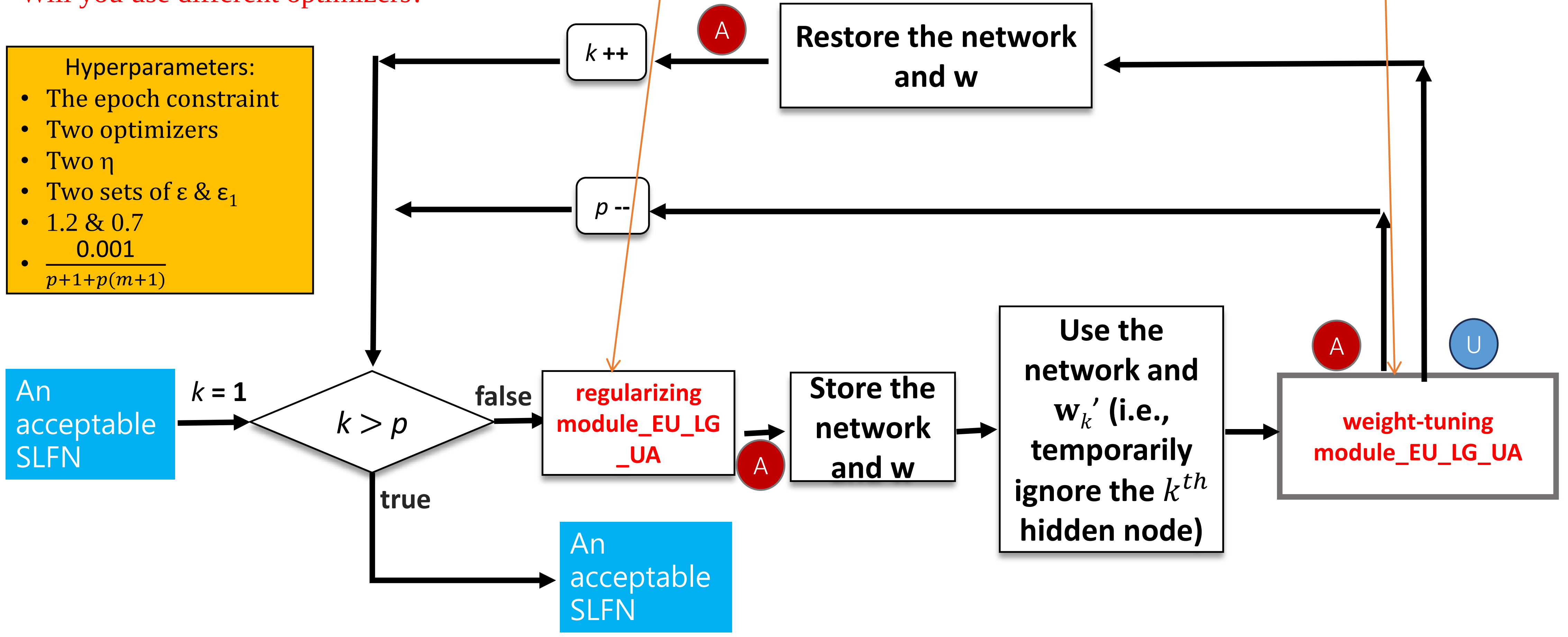
Another for the pruning purpose

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

Will you use different optimizers?

Hyperparameters:

- The epoch constraint
- Two optimizers
- Two η
- Two sets of ε & ε_1
- 1.2 & 0.7
- $\frac{0.001}{p+1+p(m+1)}$


An acceptable SLFN
$k = 1$
$k > p$
false
true
An acceptable SLFN
regularizing module_EU_LG_UA
A
Store the network and w
Use the network and $\mathbf{w}_k{}'$ (i.e., temporarily ignore the k^{th} hidden node)
weight-tuning module_EU_LG_UA
A
U
Restore the network and w
A
k ++
p --

More ideas for the reorganizing module that examines merely some hidden nodes one by one

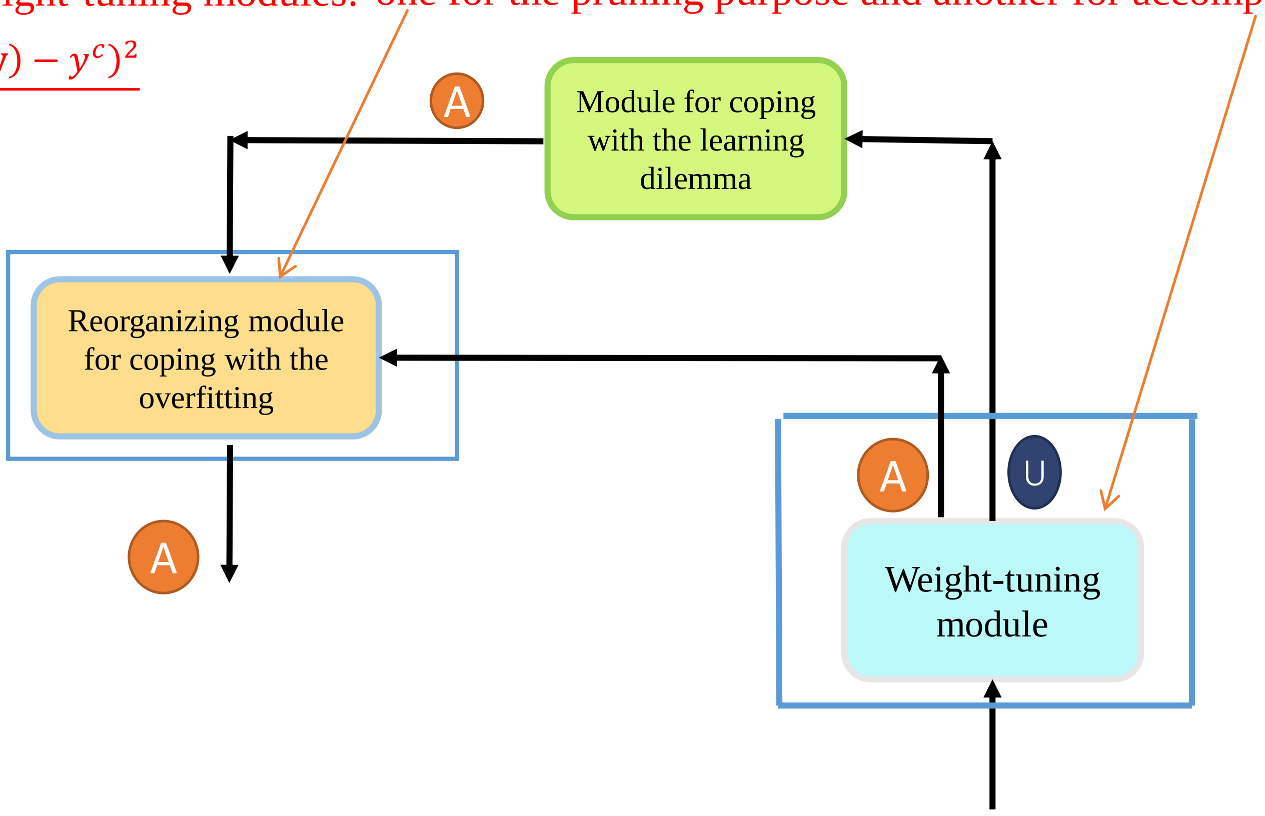
1. The reorganizing module_R3_r_EU_LG_UA_w_EU_LG_UA that randomly picks up 3 hidden nodes and examines whether they are potentially irrelevant. Remove potentially irrelevant hidden nodes identified within the process.
2. The reorganizing module_MAW_r_EU_LG_UA_w_EU_LG_UA that uses $k = \arg\min_i |w_i^o|$ 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.
3. The reorganizing module_PCA_r_EU_LG_UA_w_EU_LG_UA that 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.
4. The reorganizing module_ETP_r_EU_LG_UA_w_EU_LG_UA that 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.

   In information theory, the **entropy** of a random variable is the average level of "information", "surprise", or "uncertainty" inherent in the variable's possible outcomes. Given a discrete random variable **X**, with possible outcomes $x_1,\ldots,x_n$, which occur with probability $P(x_1), \ldots, P(x_n)$, the entropy of **X** is formally defined as: $H(\mathbf{X}) = -\sum_{i=1}^{n} P(x_i)\log(P(x_i))$. (Entropy - Wikipedia)
5. Your idea?

The weight-tuning module and the reorganizing module

Note that there are two weight-tuning modules: one for the pruning purpose and another for accomplishing the learning goal.

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


A
Module for coping with the learning dilemma
Reorganizing module for coping with the overfitting
A
U
Weight-tuning module
A

Will you use different weight-tuning modules for the pruning purpose and for accomplishing the learning goal?

Homework 3

- Make the coding of AI system stated in page 67 without the module for coping with the learning dilemma.
- Use the reorganizing module_ALL_r_EU_LG_UA_w_EU_LG_UA stated in page 65.
- You may pick up one of the following weight-tuning modules:
  - ✓ the weight-tuning module_ EU
  - ✓ the weight-tuning module_EU_LG
  - ✓ the weight-tuning module_ EU_LG_UA
  - ✓ the weight-tuning module_ LG_UA
- Note that the learning goals used in the weight-tuning module, the regularizing module, and the reorganizing module should be the same.