Inferencing Issues:
Overfitting and Learning Dilemma

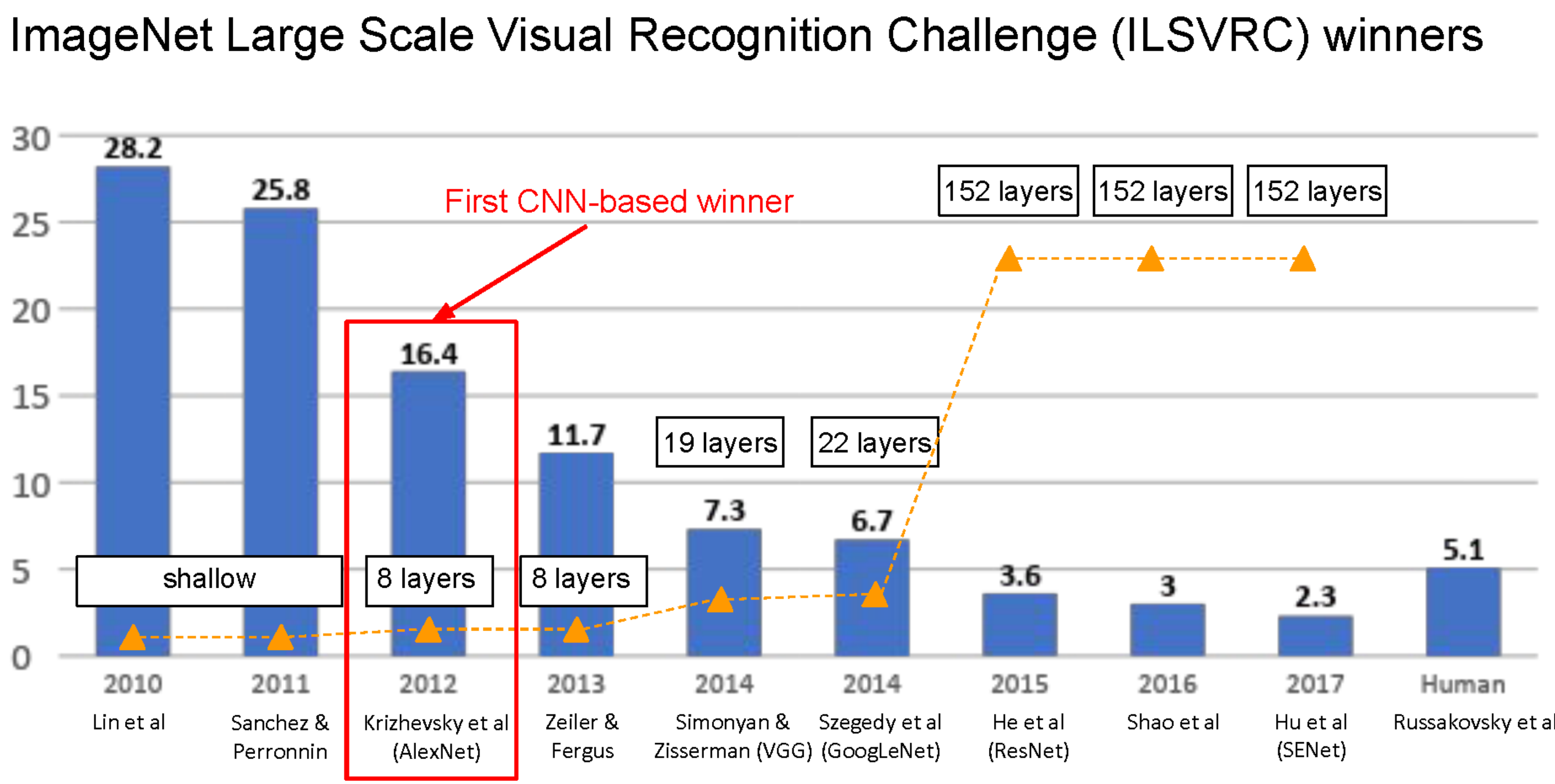
國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

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...

CNN models



idea/concept of learning →

a learning algorithm →

codes →

an AI model/system

Algorithm

(Algorithm - Wikipedia)

- In mathematics and computer science, an **algorithm** is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.

✓ Initial state/system
✓ Finite steps/blocks/modules
✓ Sequential order (→)
✓ Loop (for 迴圈； iteration/epoch)
✓ Goal
✓ Stopping criteria

Where we are now...

TensorFlow: Loss

Use predefined loss functions

The programming language form of algorithm

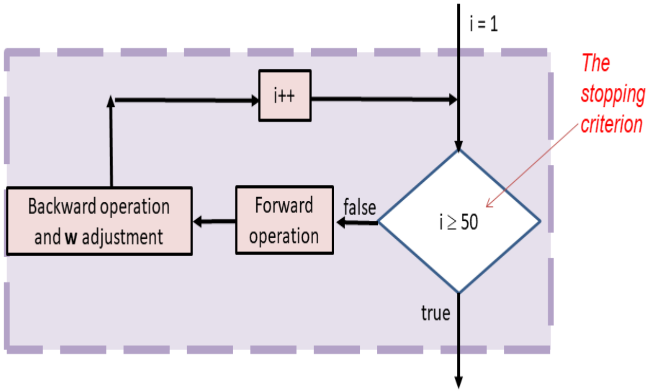
```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

optimizer = tf.optimizers.SGD(1e-6)

for t in range(50):
  with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.losses.MeanSquaredError()(y_pred, y)
  gradients = tape.gradient(loss, [w1, w2])
  optimizer.apply_gradients(zip(gradients, [w1, w2]))
```

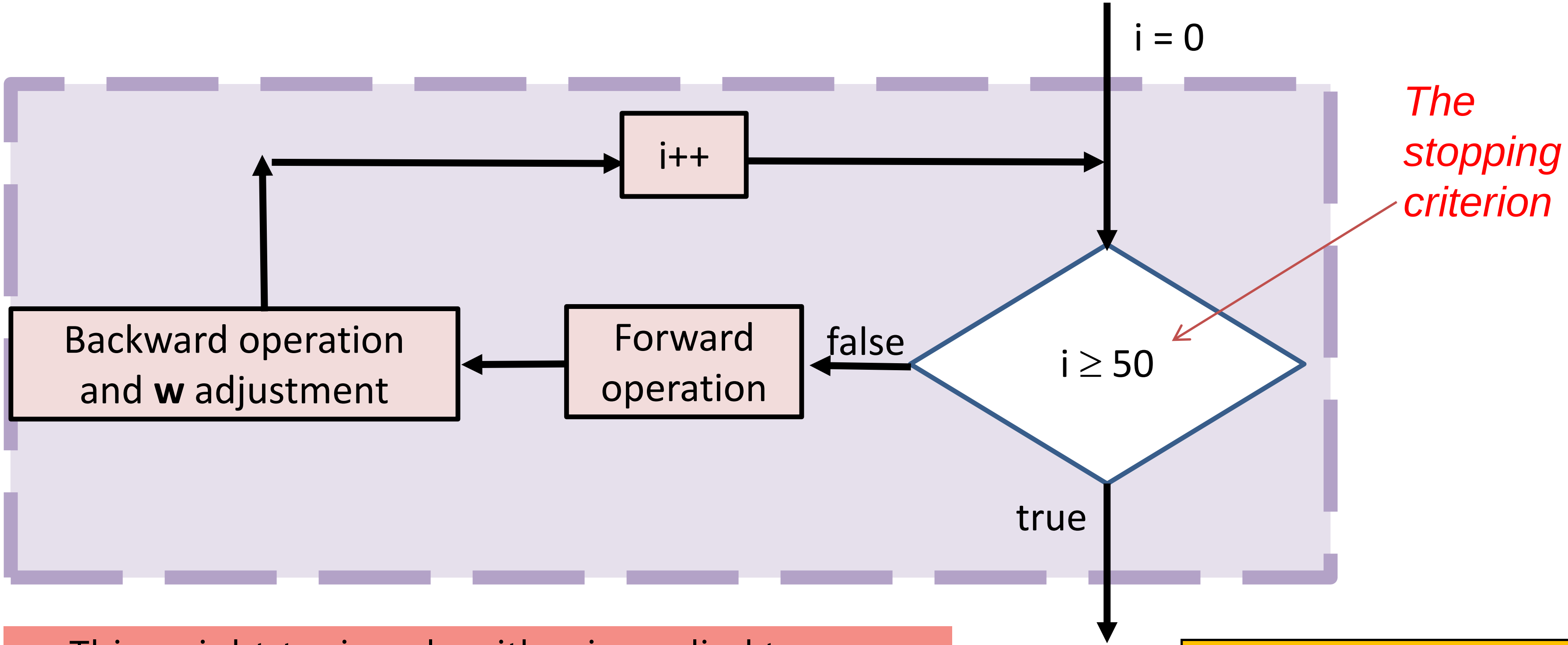
The flowchart form of algorithm



Where we are now...

Weight-tuning module_EU

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

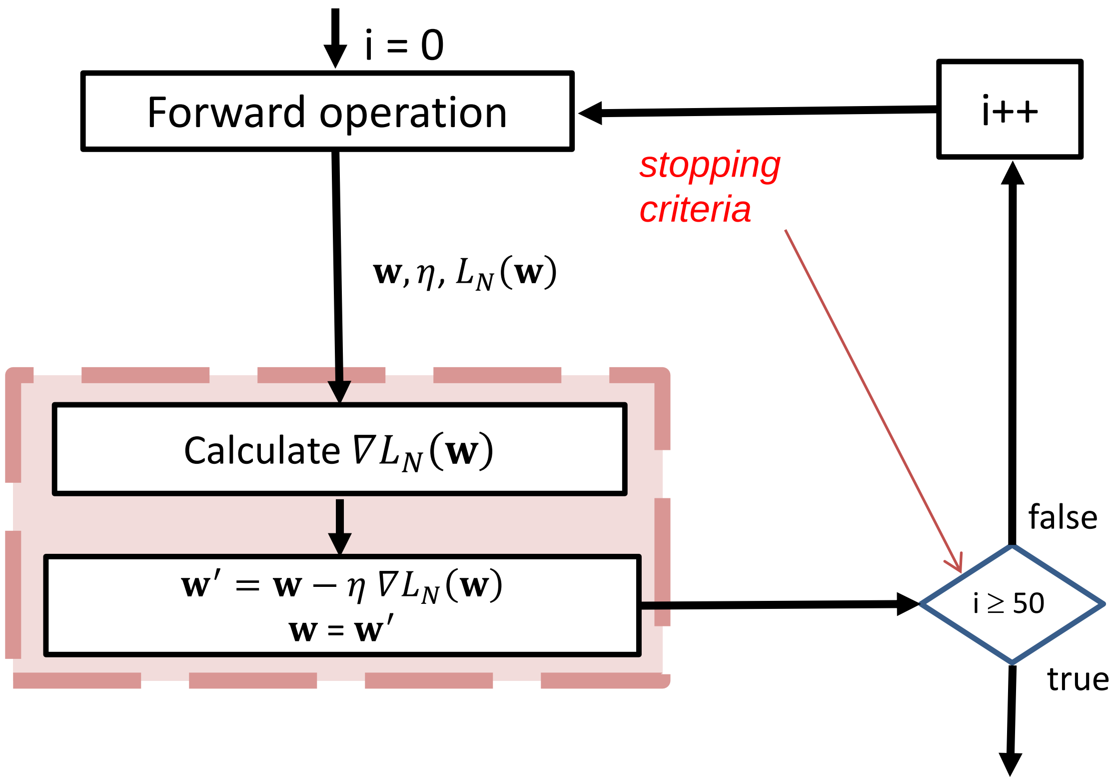


- This weight-tuning algorithm is applied to many kinds of Neural Networks, including 2-layer neural networks, CNN, RNN, reinforcement learning, GAN, BERT, and so on.
- The weight-tuning process stops when the stopping criterion is satisfied.

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

Where we are now...

The flowchart of weight-tuning module_EU



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

The learning goals
(also the stopping criteria for the learning)

The learning process should stop when

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

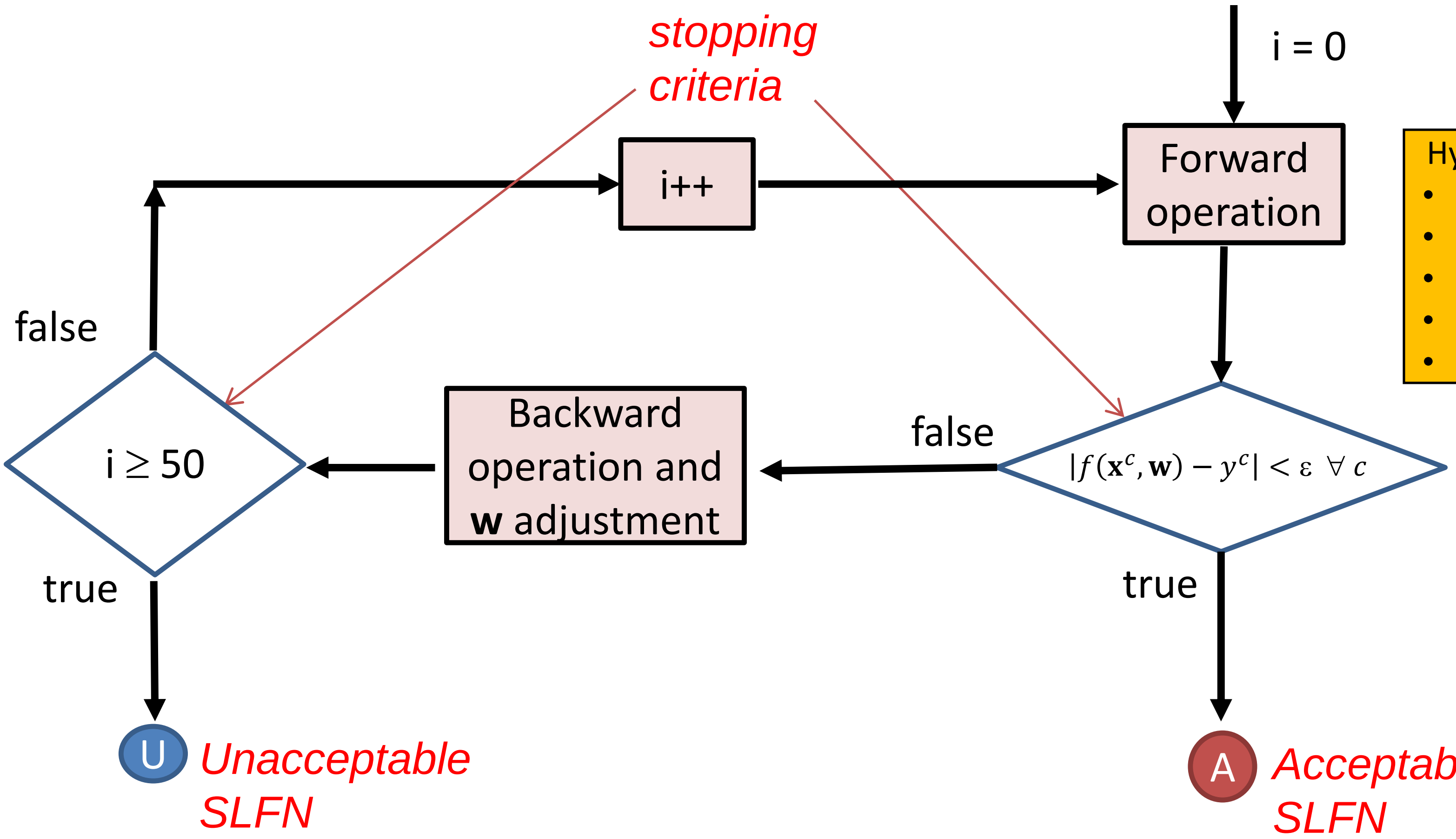
$$\boxed{\mathrm{L}_N(\mathbf{w}) \equiv \frac{1}{N}\sum_{c=1}^{N}(f(\mathbf{x}^c, \mathbf{w}) - y^c)^2}$$

- 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...

Weight-tuning module_EU_LG

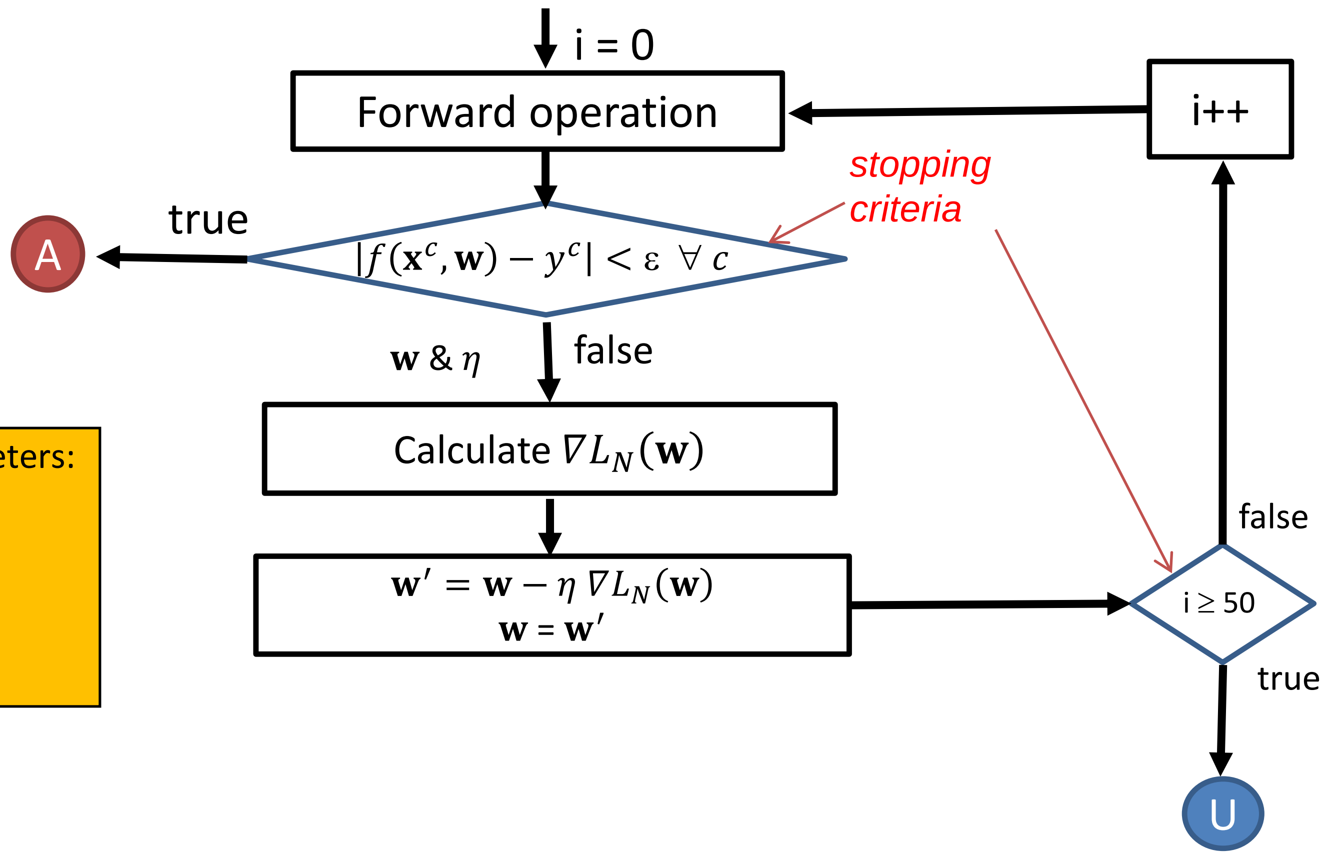
The flowchart of weight-tuning algorithm including two stopping criteria that indicate either an unacceptable SLFN or an acceptable SLFN



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

Where we are now...

The flowchart of weight-tuning module_EU_LG

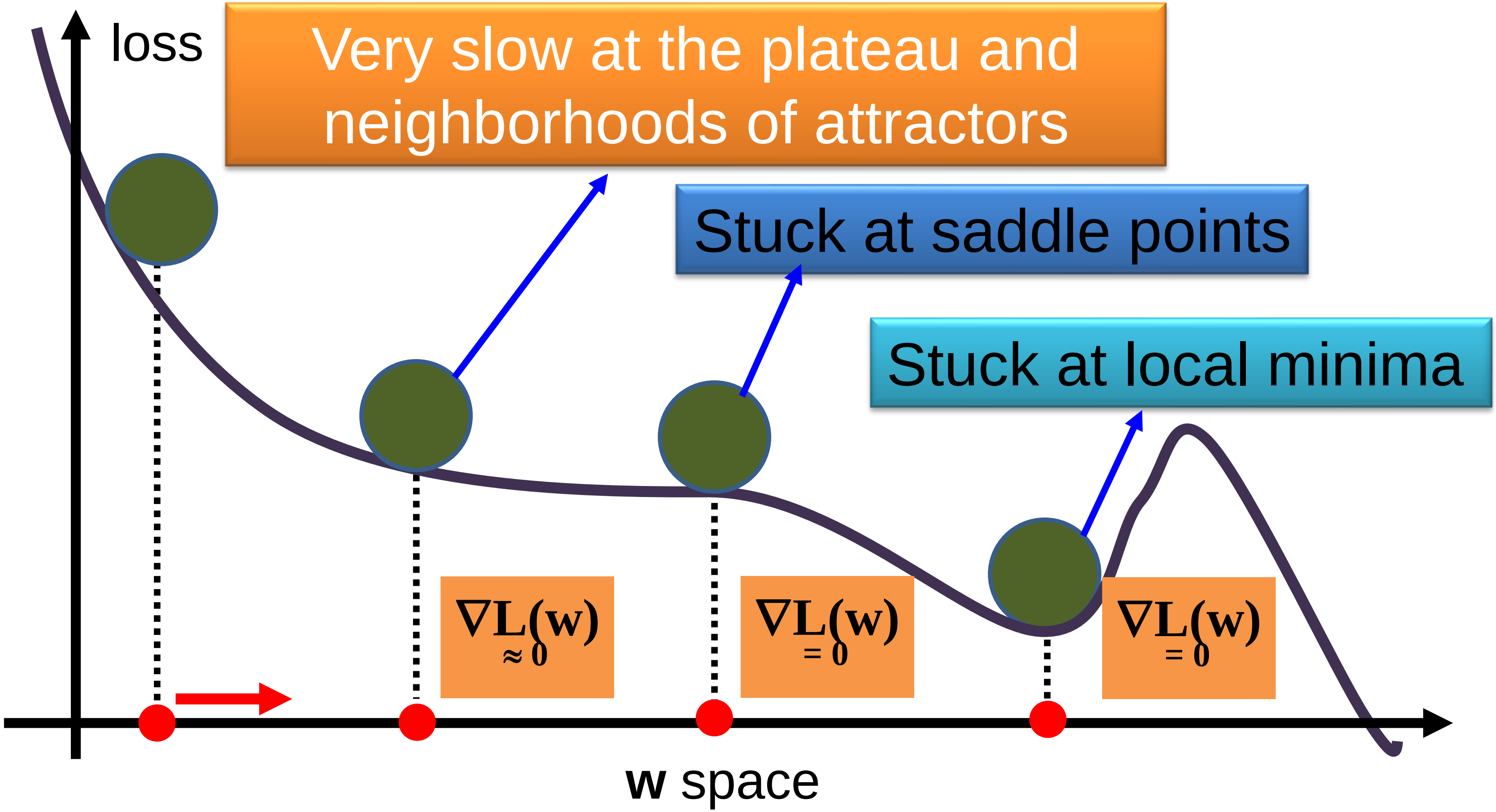


Hyperparameters:

- p
- Optimizer
- ε
- 50
- η

Where we are now...

Learning dilemma of gradient-descent-based learning



Where we are now...

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

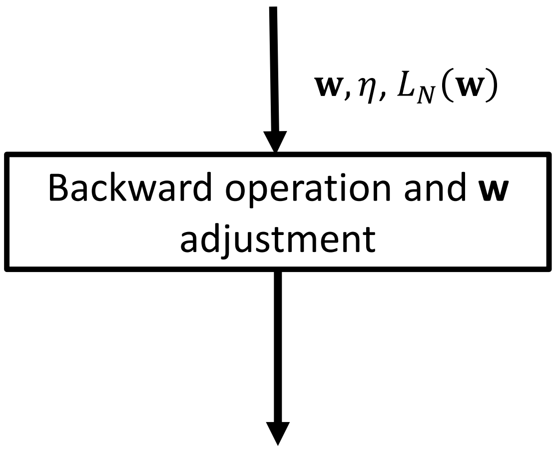
$||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})||$ is the length of $\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})$.

1. ~~The learning process should stop when~~ $\cancel{||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})|| = 0}$ ~~but a tiny~~ $\cancel{\mathrm{L}_N(\mathbf{w})}$ ~~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.

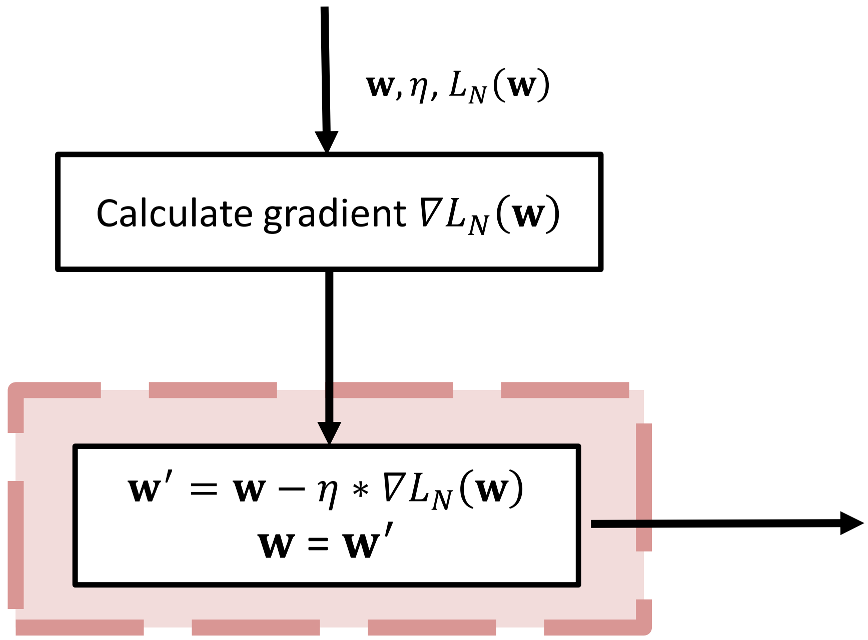
The neighborhood of undesired attractors, where $||\nabla_{\mathbf{w}} \mathrm{L}_N(\mathbf{w})|| \approx 0$ but a tiny $\mathrm{L}_N(\mathbf{w})$ value cannot be accomplished:

a) the local minimum, the saddle point, or the plateau.
b) the global minimum of the defective network architecture.

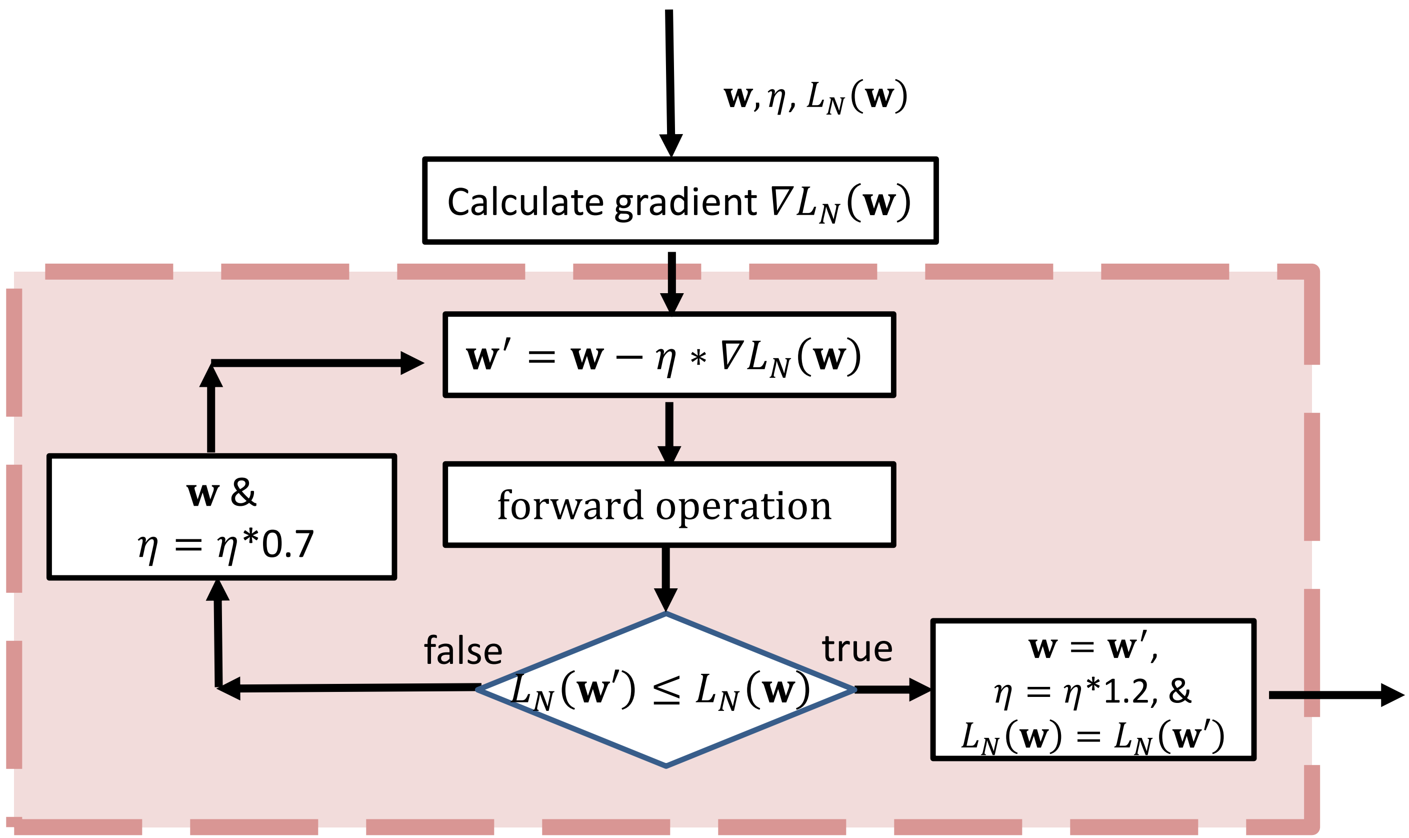
Module of backward operation and **w** adjustment



Module of backward operation and **w** adjustment



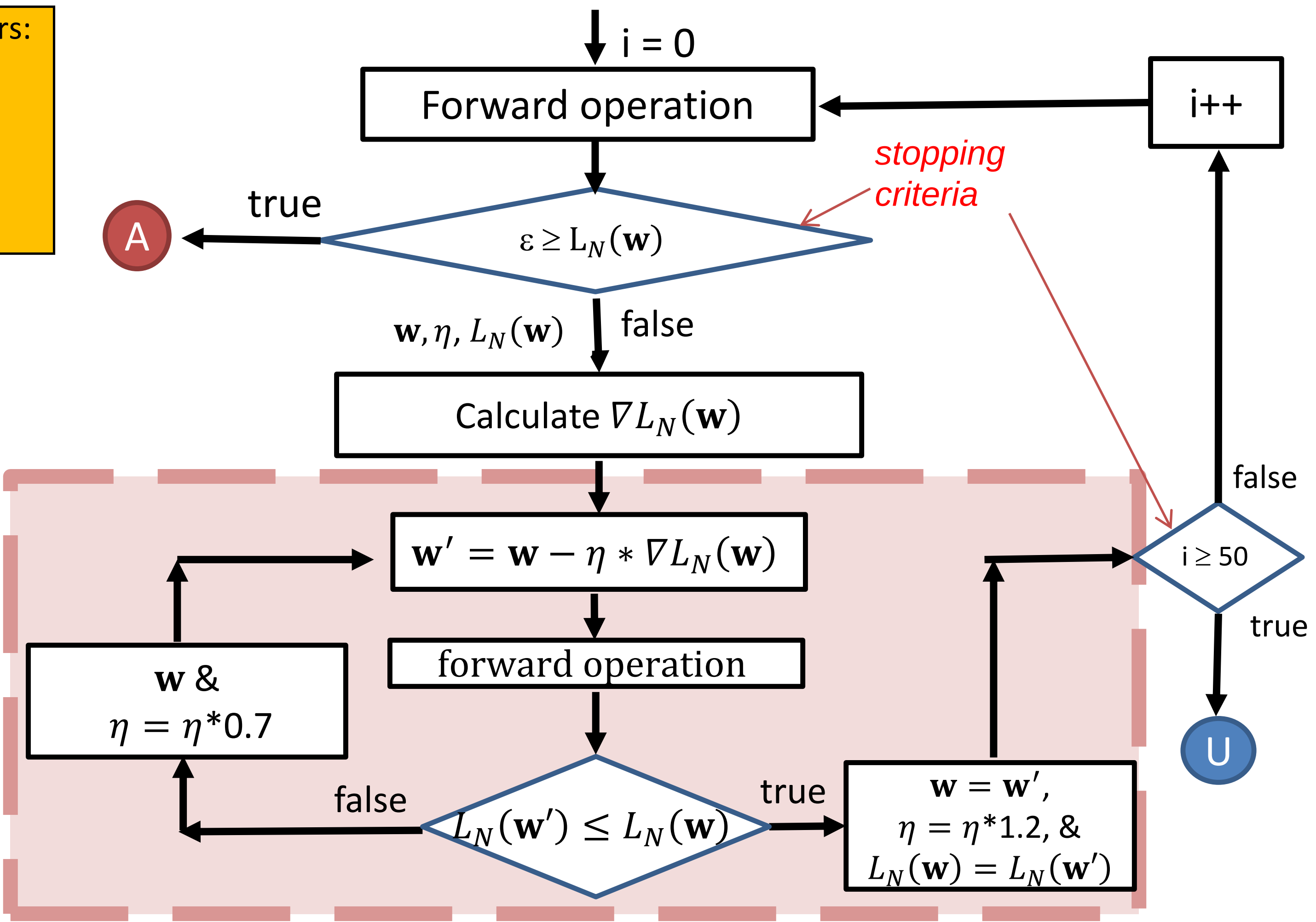
Module of backward operation and **w** adjustment with the adaptable learning rate arrangement



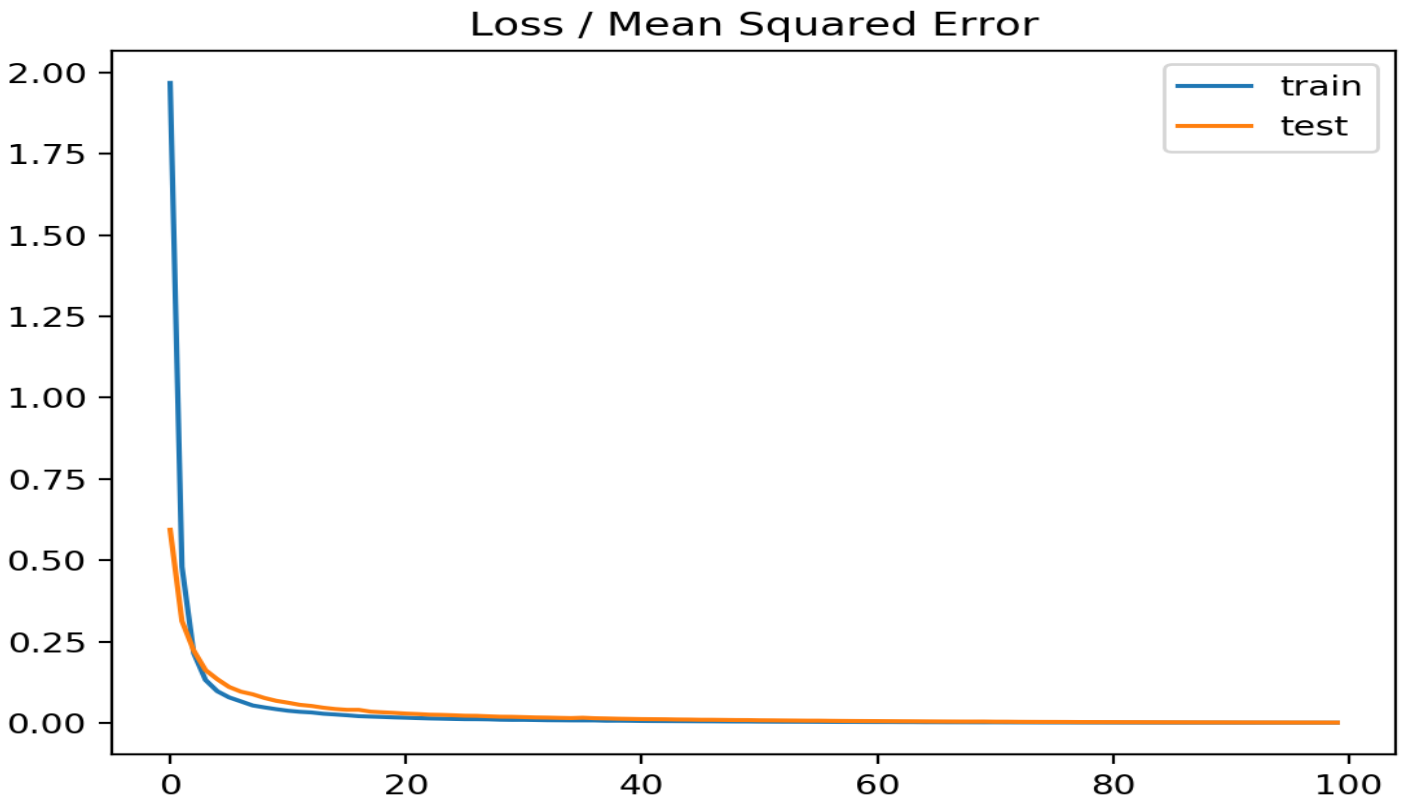
Where we are now...

The flowchart of weight-tuning module_EU_LG

Hyperparameters:
- p
- Optimizer
- ε
- 50
- 1.2 & 0.7

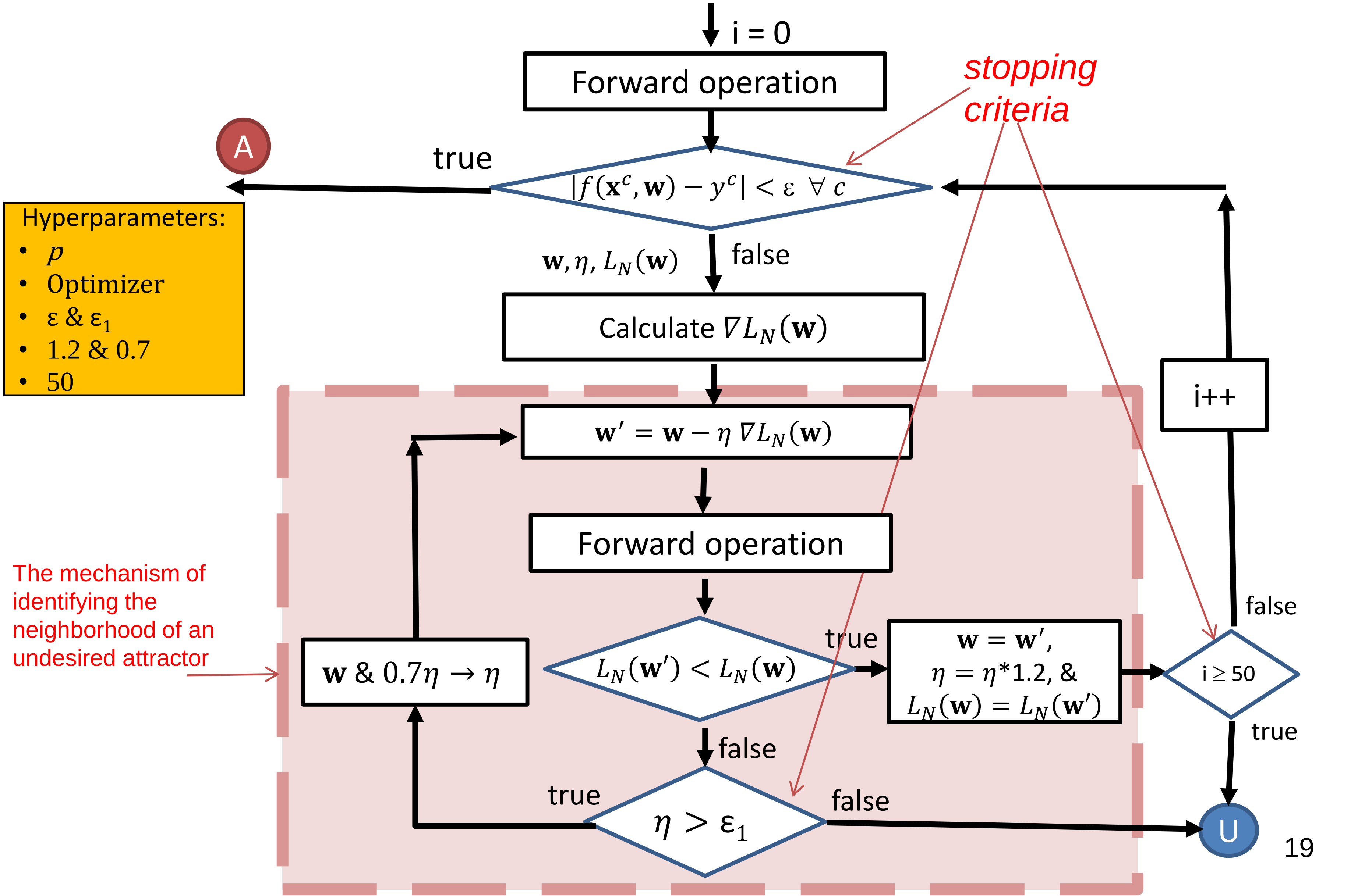


The effect of adaptable learning rate

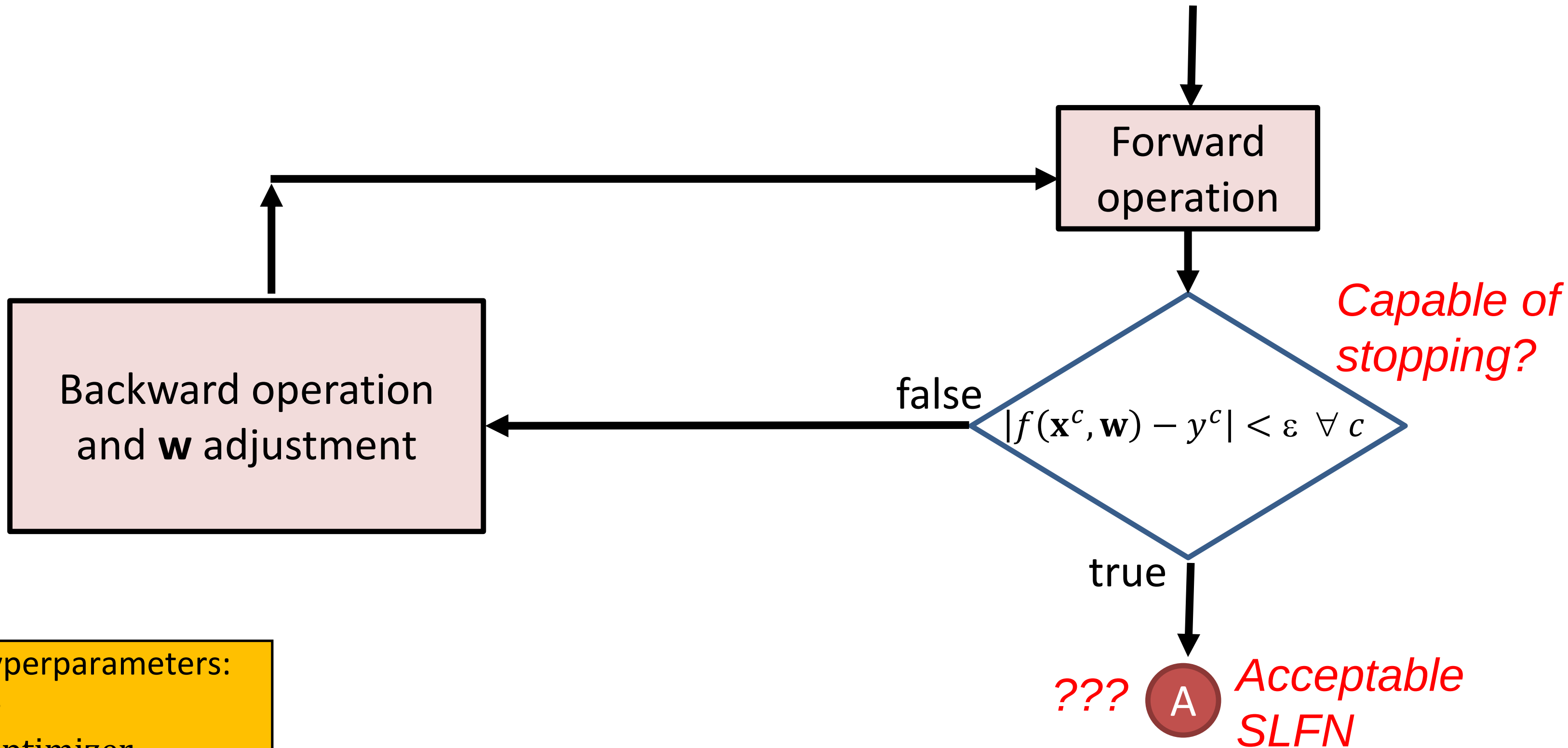


Where we are now...

The flowchart of weight-tuning module_EU_LG_UA



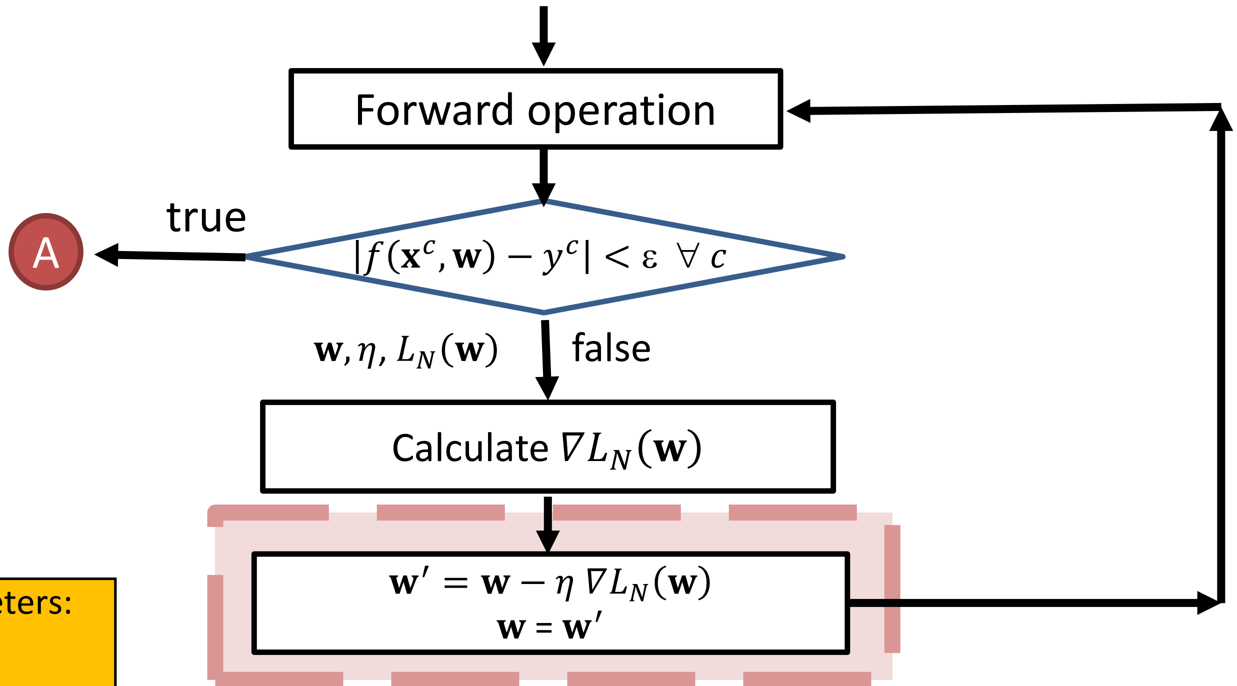
The flowchart of BP learning algorithm



Hyperparameters:
- p
- Optimizer
- ε
- η

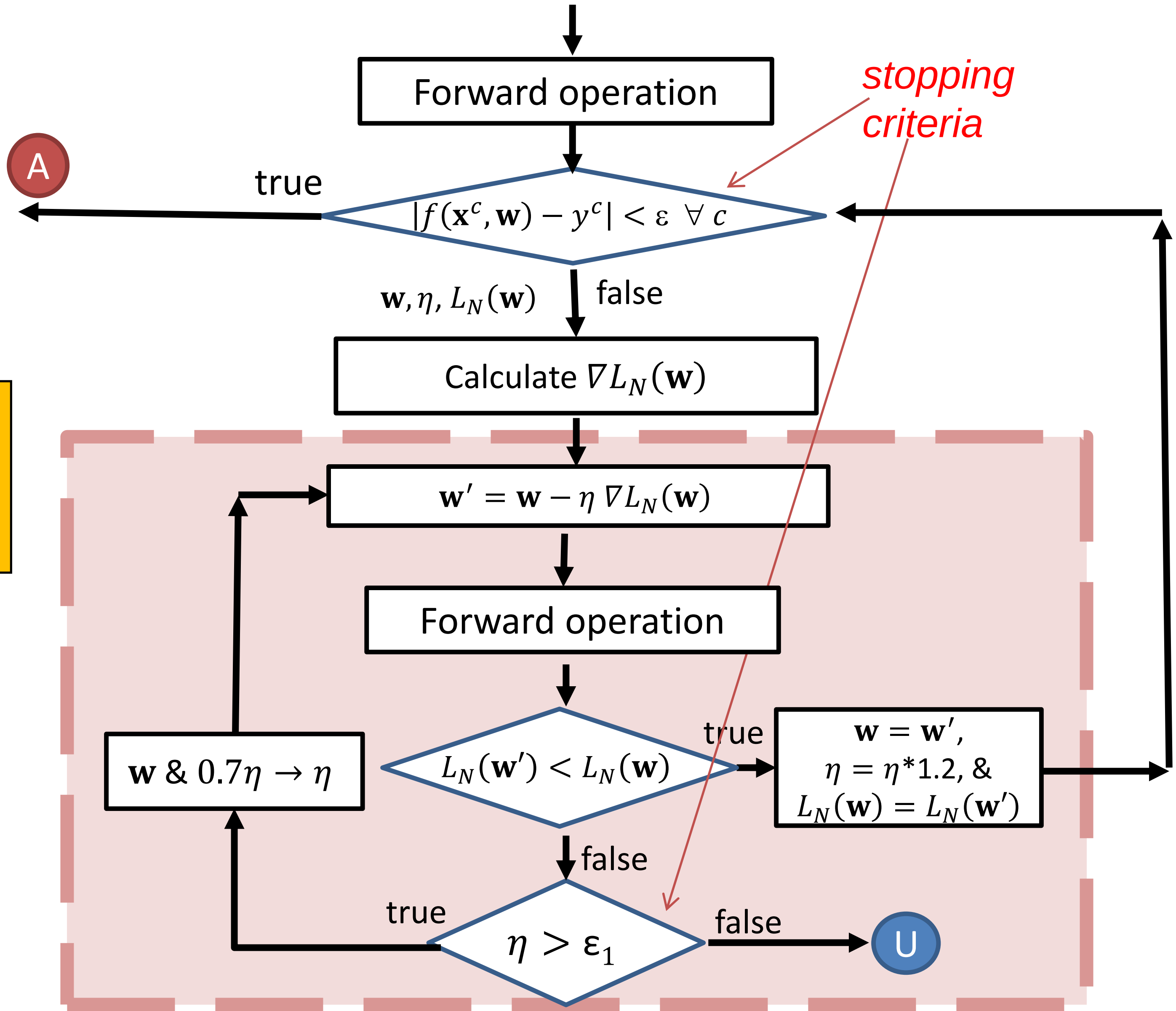
Q: Can this the learning process stop through satisfying the stopping criterion?
A: May not be! So, this stopping criterion is not good and thus this algorithm is not good.

The flowchart of BP learning algorithm



Hyperparameters:
- p
- Optimizer
- ε
- η

The flowchart of weight-tuning module_LG_UA



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 weight-tuning module_EU_LG
  - ✓ the weight-tuning module_EU_LG_UA
  - ✓ the weight-tuning module_LG_UA
- What are the performance differences amongst these weight-tuning modules?

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 the issue

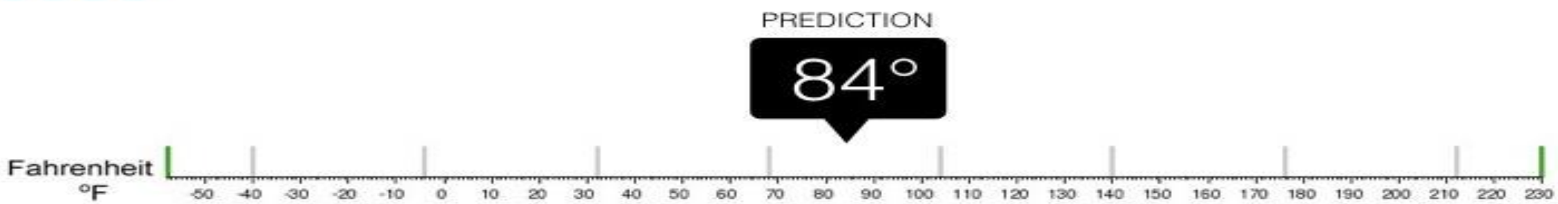
Homework #2

- Rewrite the code you have for HW #1 (the weight-tuning module_EU referring to page 8) into the code of the weight-tuning module_EU_LG referring to page 11.
- Rewrite the code you have for HW #1 (the weight-tuning module_EU) into the code of the weight-tuning module_EU_LG_UA referring to page 19.
- Rewrite the code you have for HW #1 (the weight-tuning module_EU) into the code of the weight-tuning module_LG_UA referring to page 22.
- Once you have the code (regardless of which framework you choose above), you will apply the code to learn the train_all_0.csv dataset given in the LINE group.
- The training and test dataset is 80%/20%.
- The performance comparison benchmark is the code of the weight-tuning module_EU.
- Note that the output nodes of your SLFN may be one or two.

The supervised learning problems: Regression and Classification

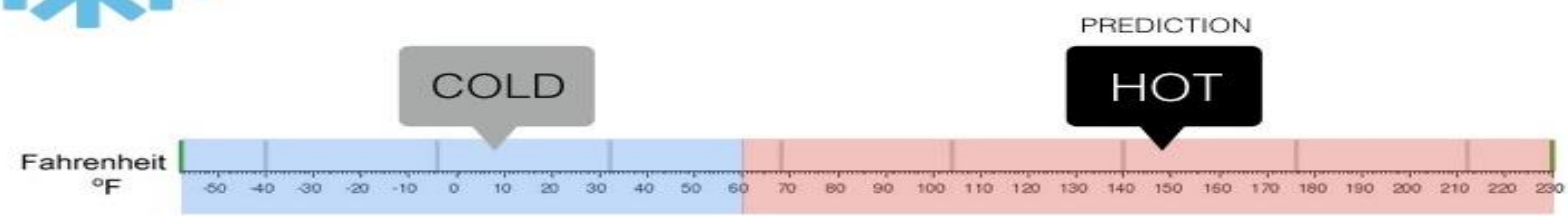
Regression

What is the temperature going to be tomorrow?



Classification

Will it be Cold or Hot tomorrow?



Stopping criteria (also the learning goals) for regression problems

one output node

The learning process should stop when

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

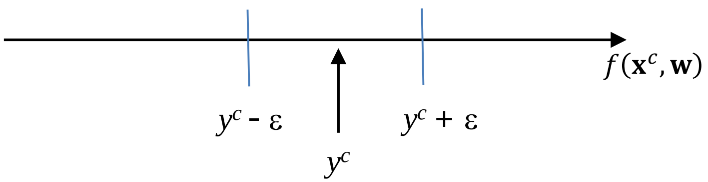
$$\mathrm{L}_N(\mathbf{w}) \equiv \frac{1}{N} \sum_{c=1}^{N} (f(\mathbf{x}^c, \mathbf{w}) - y^c)^2$$

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

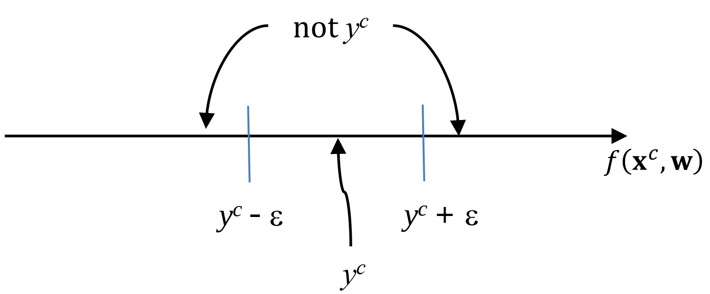
The regression applications

The learning mechanism

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



The inferencing mechanism



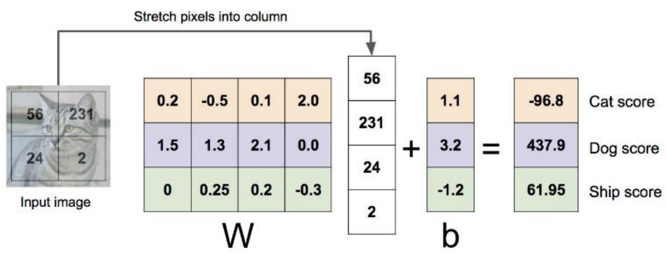
The three-class classification problems

three output nodes

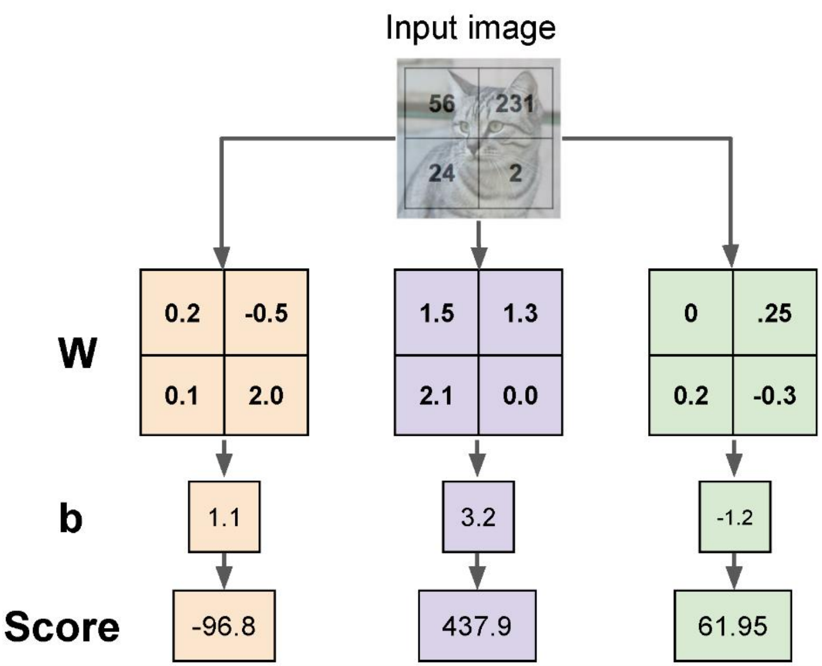
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic Viewpoint

$f(x,W) = Wx$

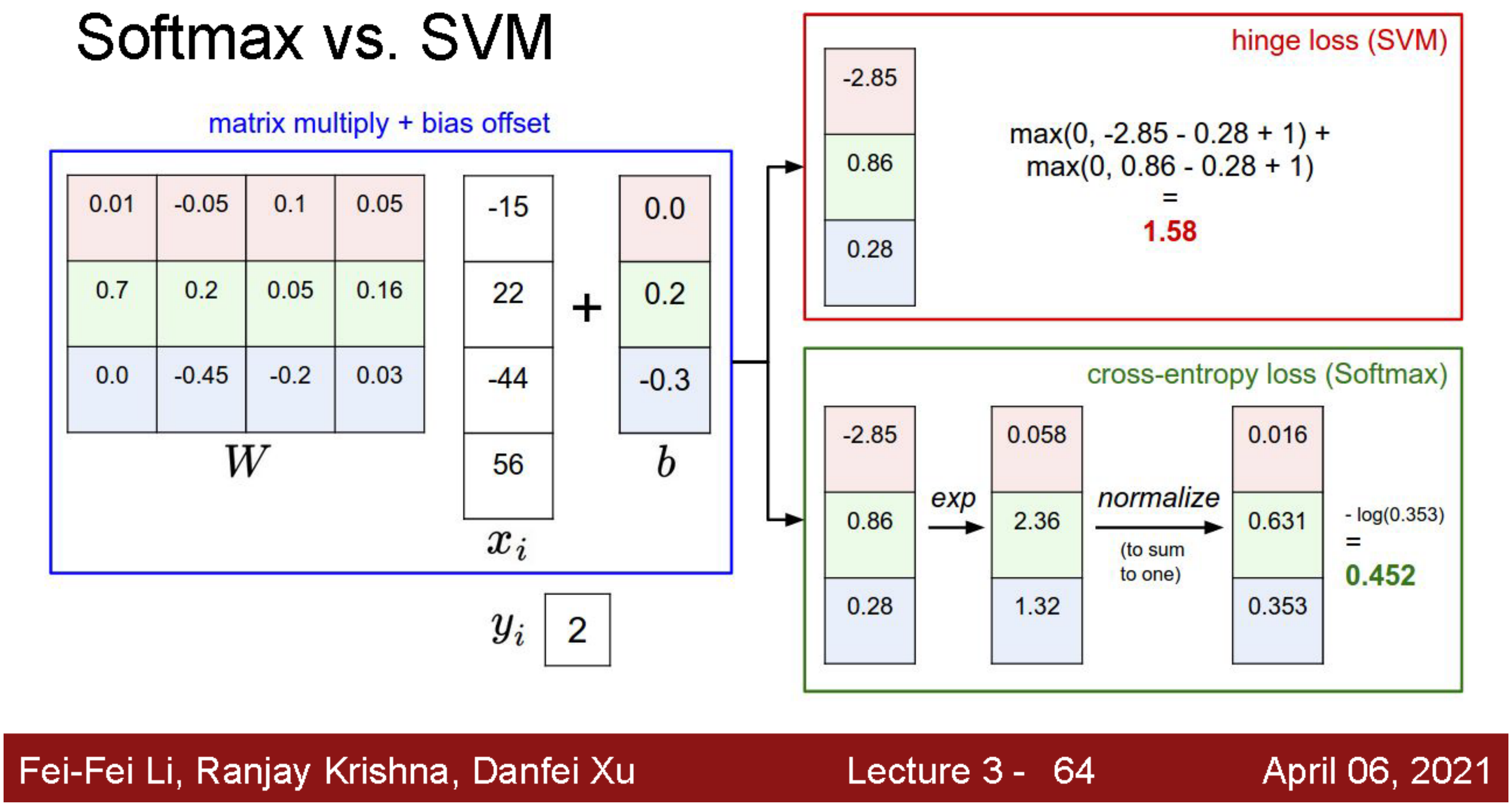


Visual Viewpoint



The three-class classification problems

three output nodes



Classification Applications Design (y label)

Output value: real number

$SLFN\ with$ one output node and linear arrangement

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

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

- 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

one output node

- When the SLFN with only one output node whose output value is real number, the stopping criteria may be as follows:

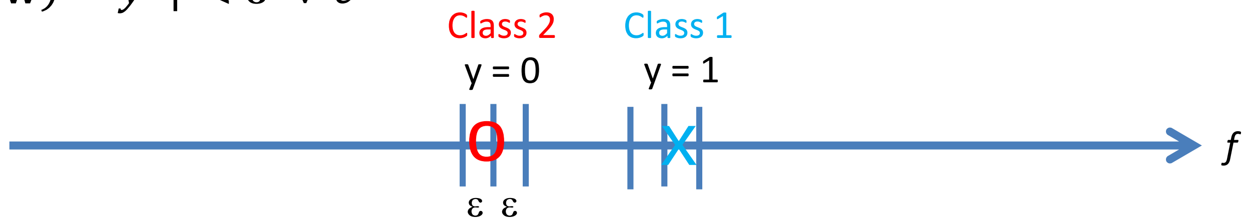
1. $|f(\mathbf{x}^c, \mathbf{w}) - y^c| < \varepsilon \ \forall\, c$
2. $f(\mathbf{x}^c, \mathbf{w}) > \nu \ \forall\, c \in \mathbf{I}_1$ and $f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \ \forall\, c \in \mathbf{I}_2$, with $1 > \nu > 0$
3. $\alpha \equiv \min\limits_{c \epsilon \mathbf{I}_1} f(\mathbf{x}^c,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

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

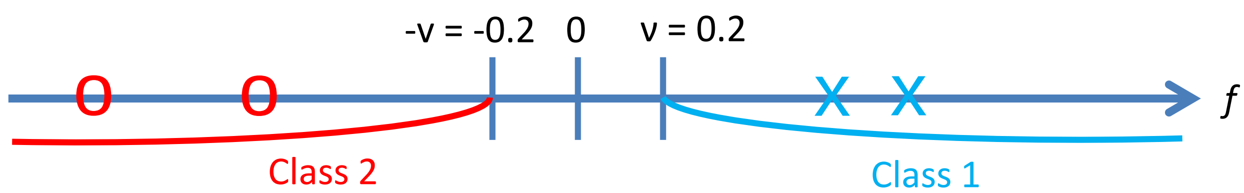
Stopping criteria (also the learning goals) for two-class classification problems dealt with the SLFN with one output node whose output values are real numbers

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

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



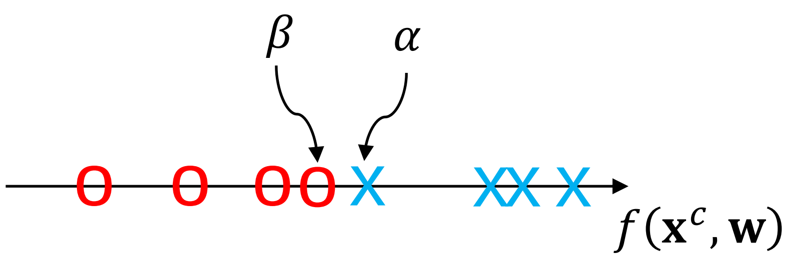
- $f(\mathbf{x}^c, \mathbf{w}) \geq \nu \ \forall\, c \in \mathbf{I}_1$ and $f(\mathbf{x}^c, \mathbf{w}) \leq -\nu \ \forall\, c \in \mathbf{I}_2$ with $1 > \nu > 0$.

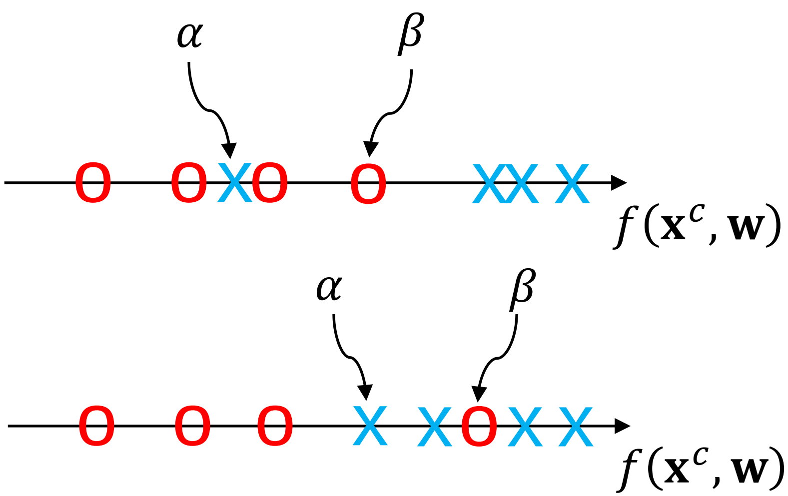


Stopping criteria (also the learning goals) for two-class classification problems dealt with the SLFN with one output node whose output values are real numbers

- The LSC regarding $\{f(\mathbf{x}^c, \mathbf{w})\ \forall\, c \in \mathbf{I}\}$ (Tsaih, 1993)

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



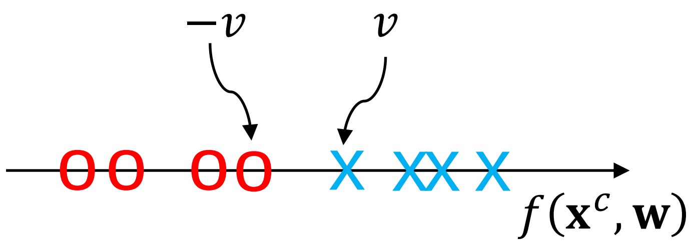


$\alpha > \beta$
LSC : True

$\alpha < \beta$
LSC: False

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

the classification inferencing mechanism



When LSC $(\alpha > \beta)$ is satisfied, the classification 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 $v - \min_{c \in \mathbf{I}_1} \sum_{i=1}^{p} w_i^o a_i^c \rightarrow w_0^o$

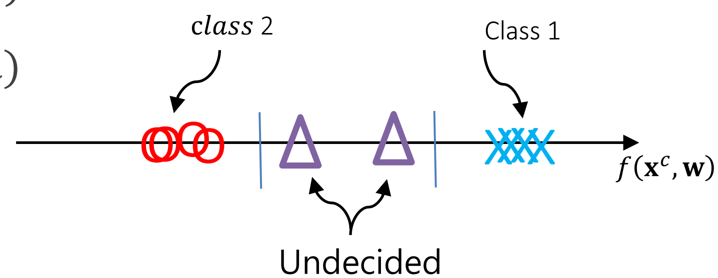
The threshold of the output node

The classification applications

y^c = class 1 $\forall\, c \in \mathbf{I}_1(n)$; y^c = class 2 $\forall\, c \in \mathbf{I}_2(n)$; $\mathbf{I}(n) \equiv \mathbf{I}_1(n) \cup \mathbf{I}_2(n)$

X: $f(\mathbf{x}^c, \mathbf{w})\ \forall\, c \in \mathbf{I}_1(n)$

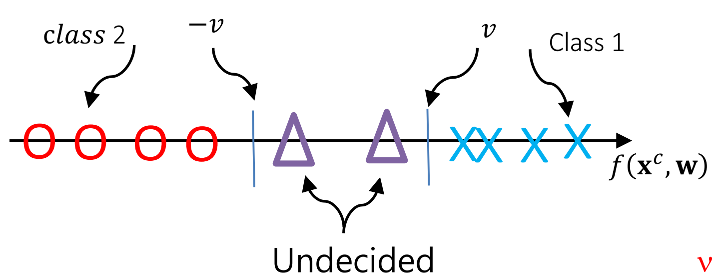
O: $f(\mathbf{x}^c, \mathbf{w})\ \forall\, c \in \mathbf{I}_2(n)$



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

ε Is a hyperparameter regarding the learning!

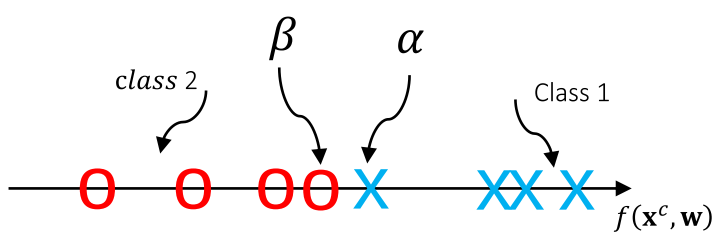
The inferencing mechanism



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

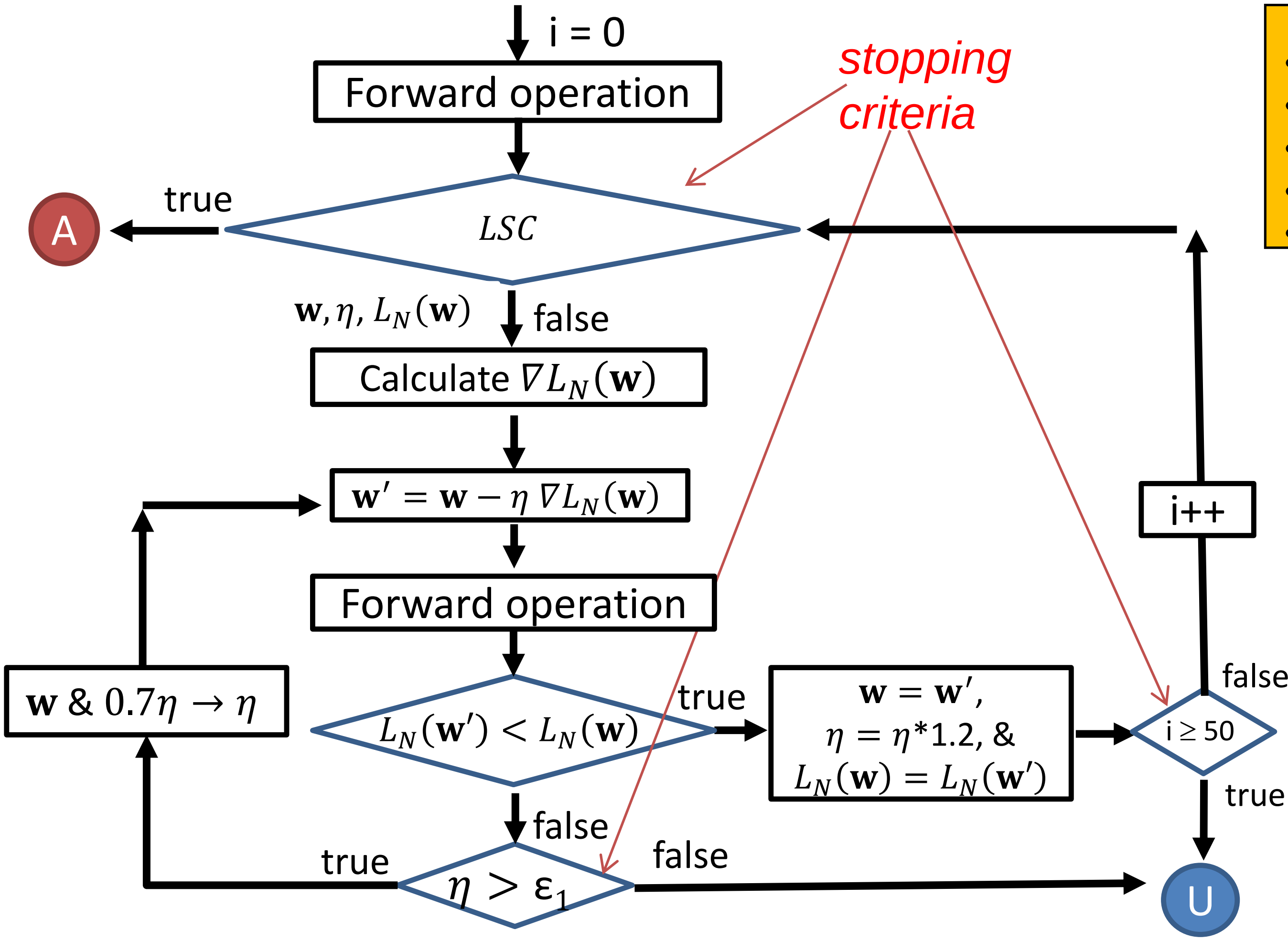
ν Is a hyperparameter regarding the inferencing!

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



learning goal type III: LSC

The flowchart of weight-tuning module_EU_LG_UA



Hyperparameters:

- p
- Optimizer
- ε & ε_1
- 1.2 & 0.7
- 50

Where we are now...

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:

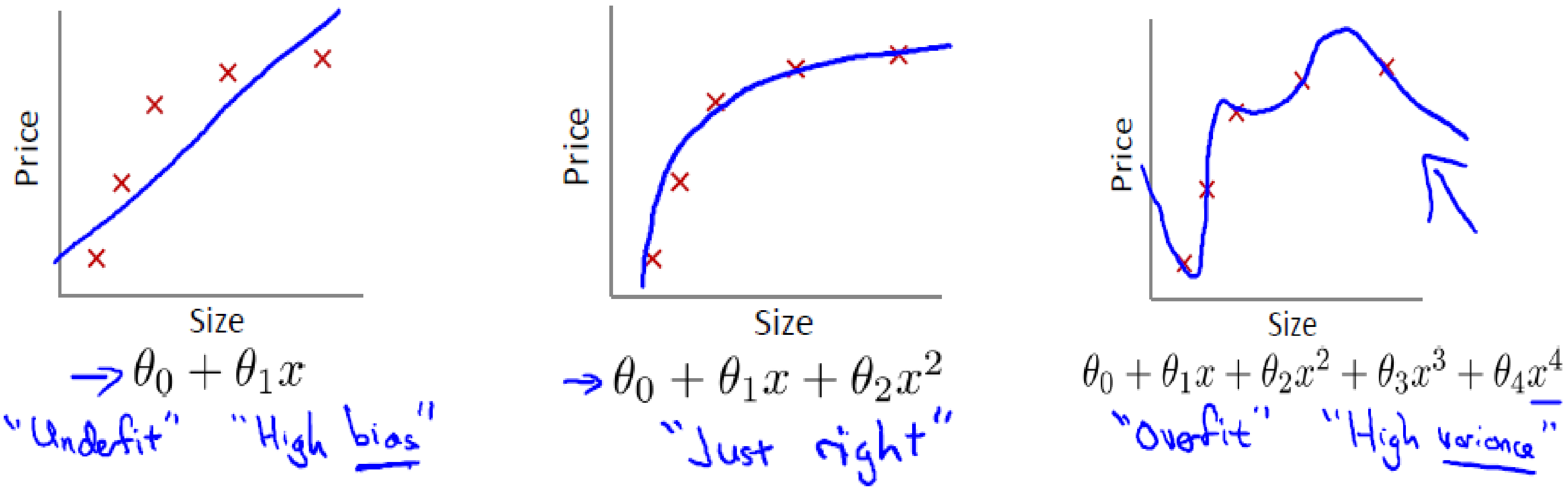
✓ Problem definition ← learning-based prediction problem
✓ Development of a model ← 2-layer net, 4-layer net, or deep neural networks
✓ Specification of the algorithm ← The learning algorithm
✓ Designing an algorithm ← The gradient-descent-based learning algorithm
✓ Checking the correctness of the algorithm ← 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

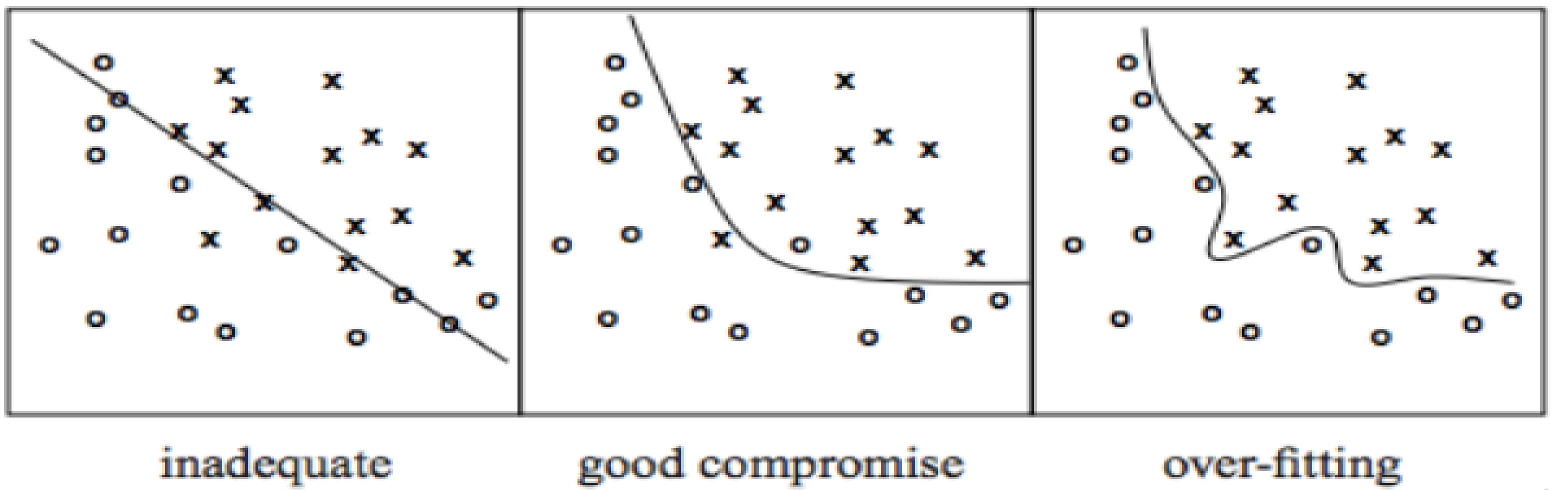
Program testing -- Performance of AI Applications

- 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.

overfitting





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

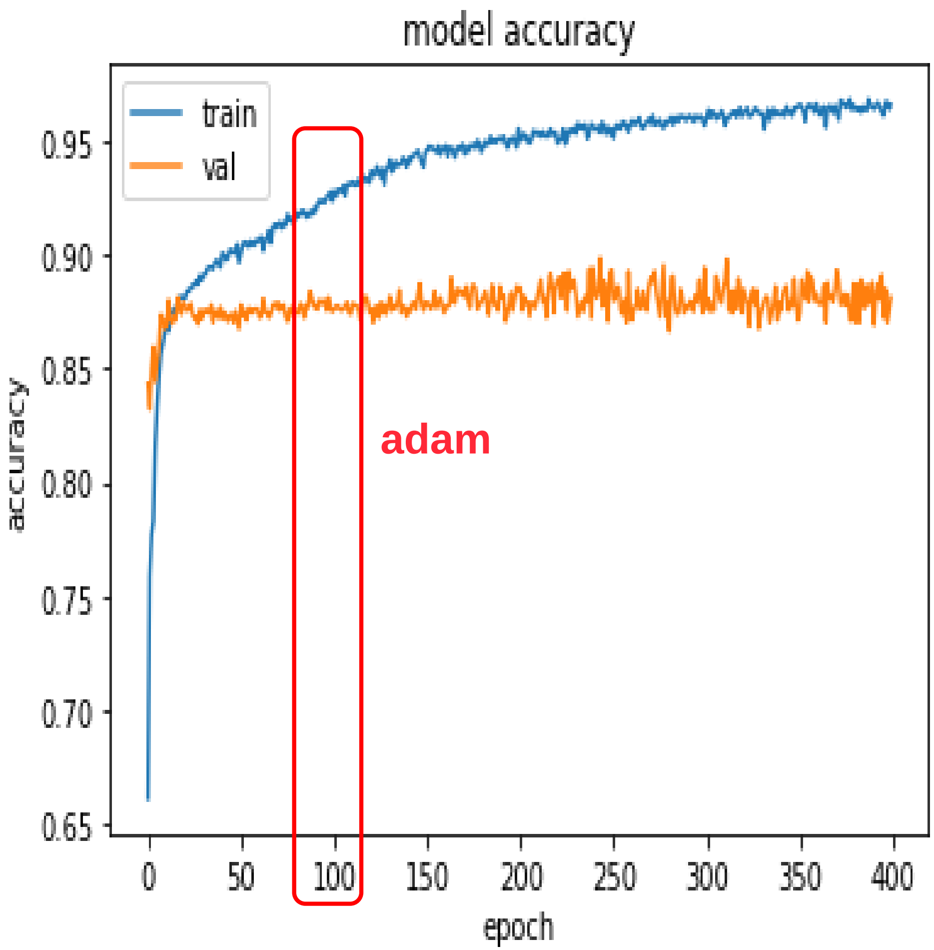
Generalization

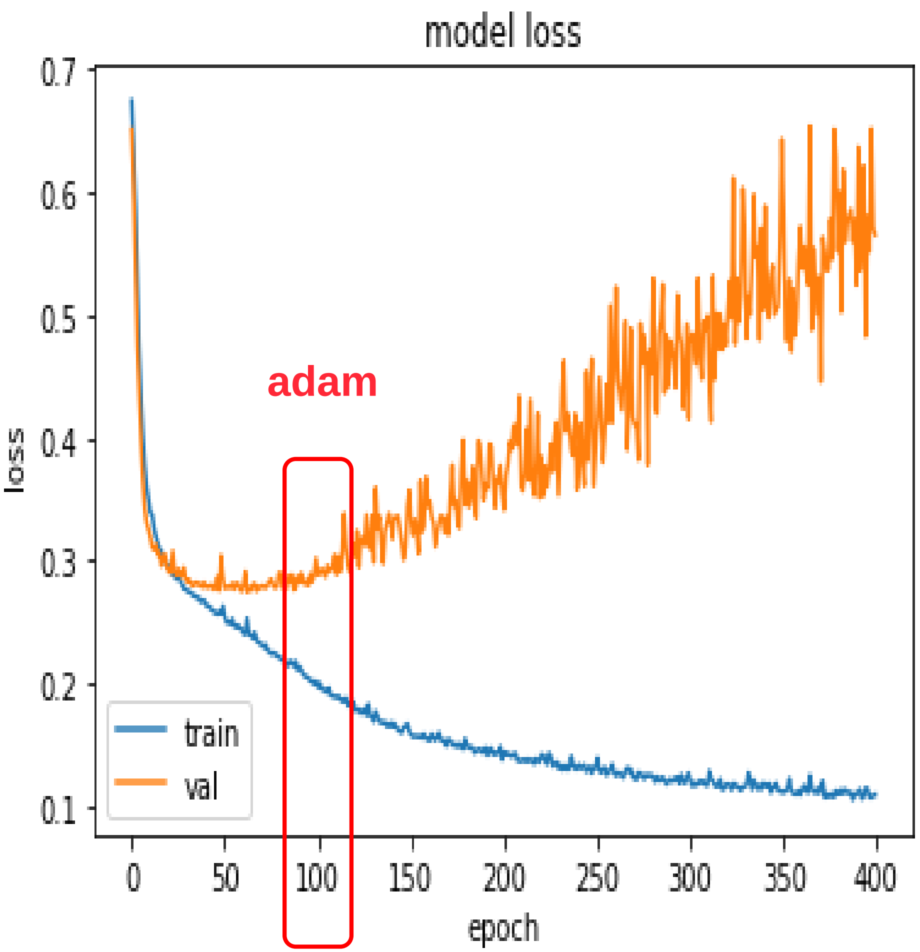
- Learned hypothesis may fit the training data very well, even noises (or outliers) in the training data, but fail to generalize to new examples (test data)
- In machine learning and statistical learning, the generalization error (also known as the out-of-sample error) is a measure of how accurately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

Learning curve and overfitting

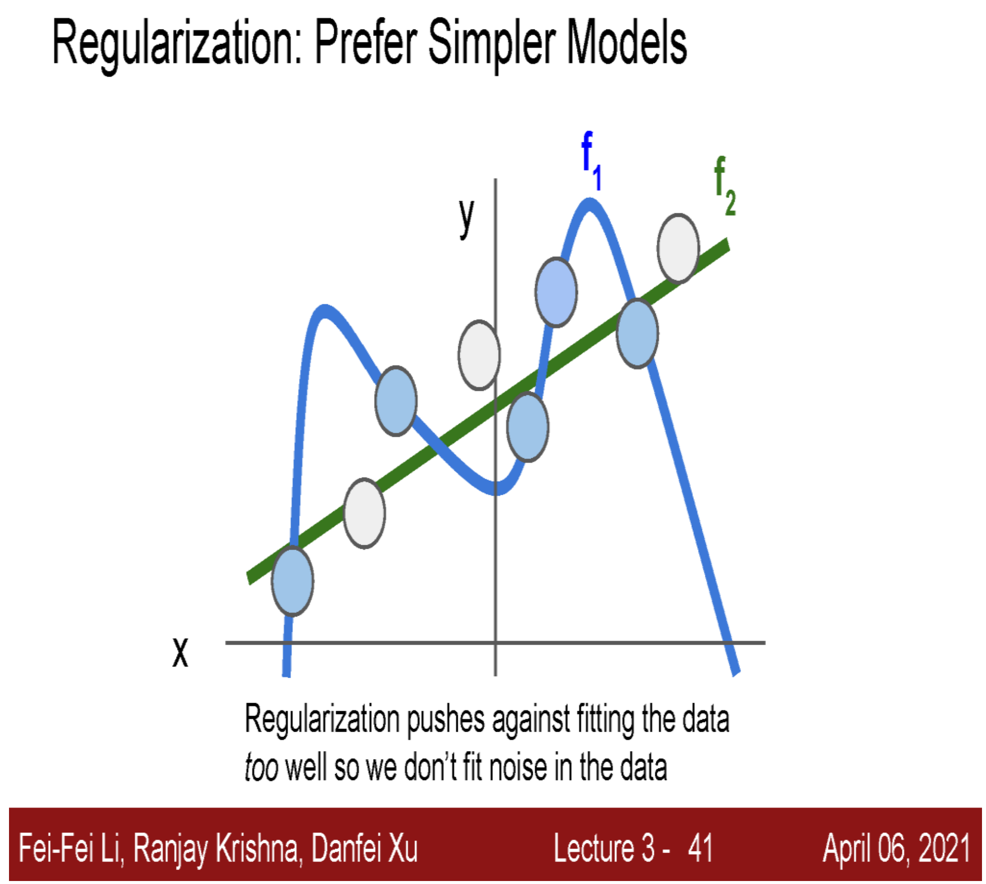




Early stop!

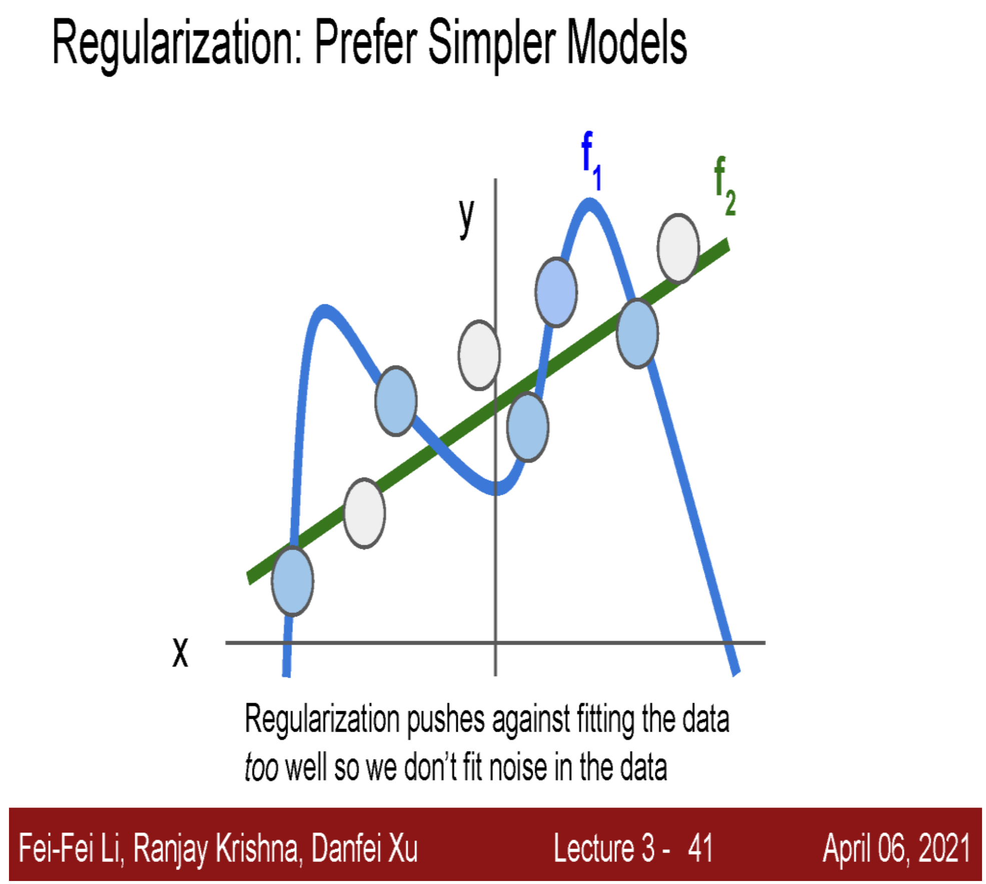
Overfitting

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."



Overfitting

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



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

Regularization

λ = regularization strength (hyperparameter)

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

Data loss: Model predictions should match training data

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

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

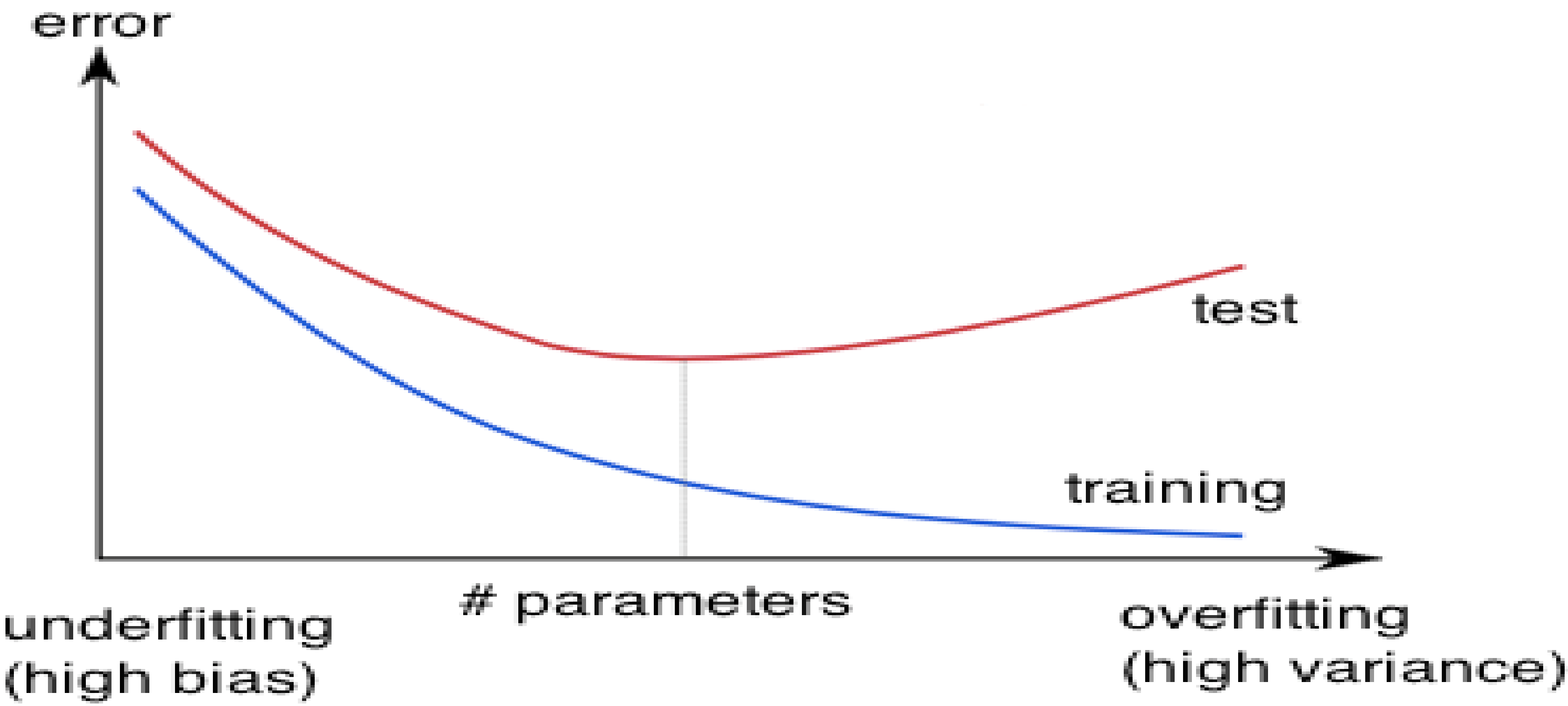
- Adopt a regularization term in the loss function to penalize big weights:

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

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

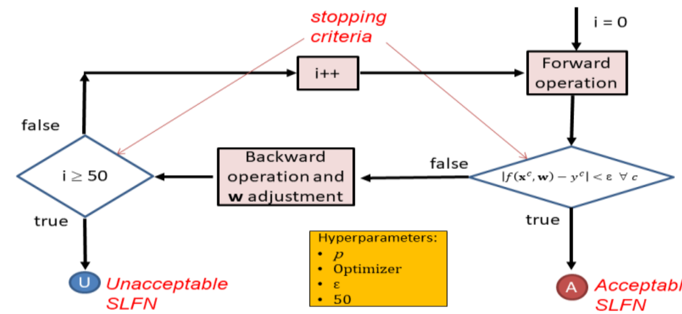
- The regularization coefficient λ determines how dominant the regularization is during gradient computation
- Big regularization coefficient → big penalty for big weights
- The above is the L2 regularization
- L1 regularization: $\lambda|\mathbf{w}|$
- Elastic net regularization: L1 + L2

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

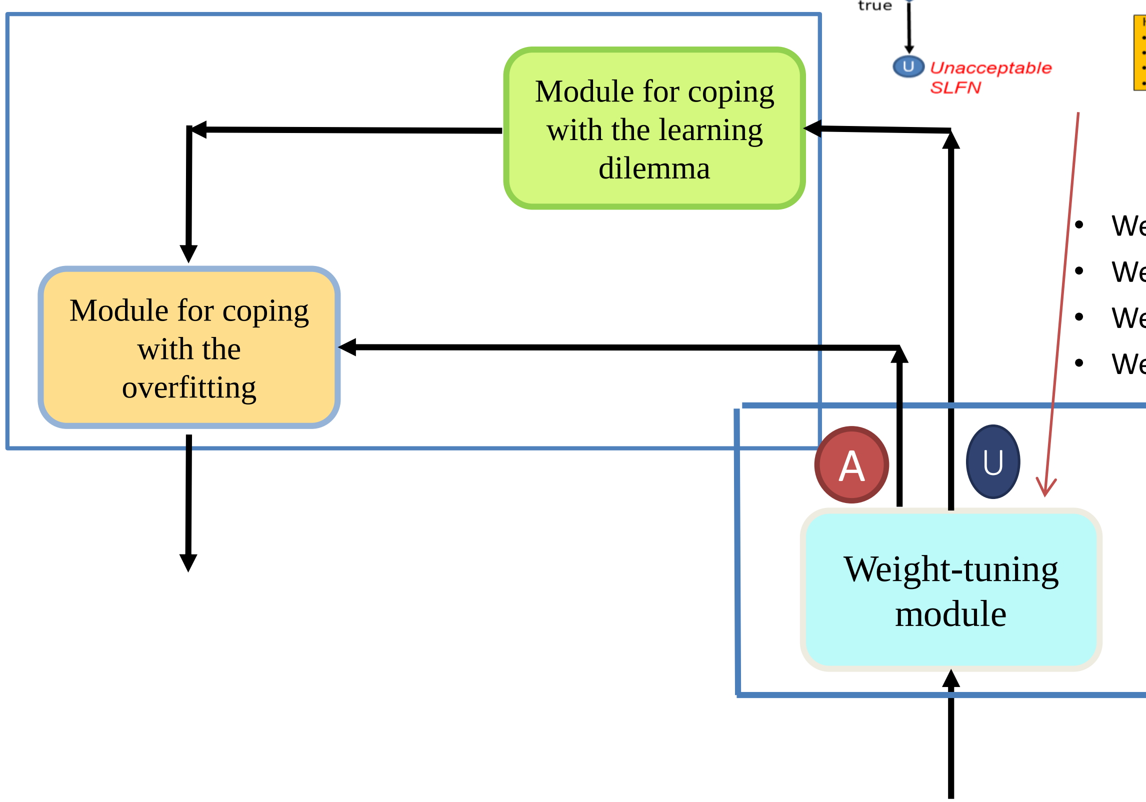


https://www.neuraldesigner.com/images/learning/selection_error.svg

Inferencing Issues

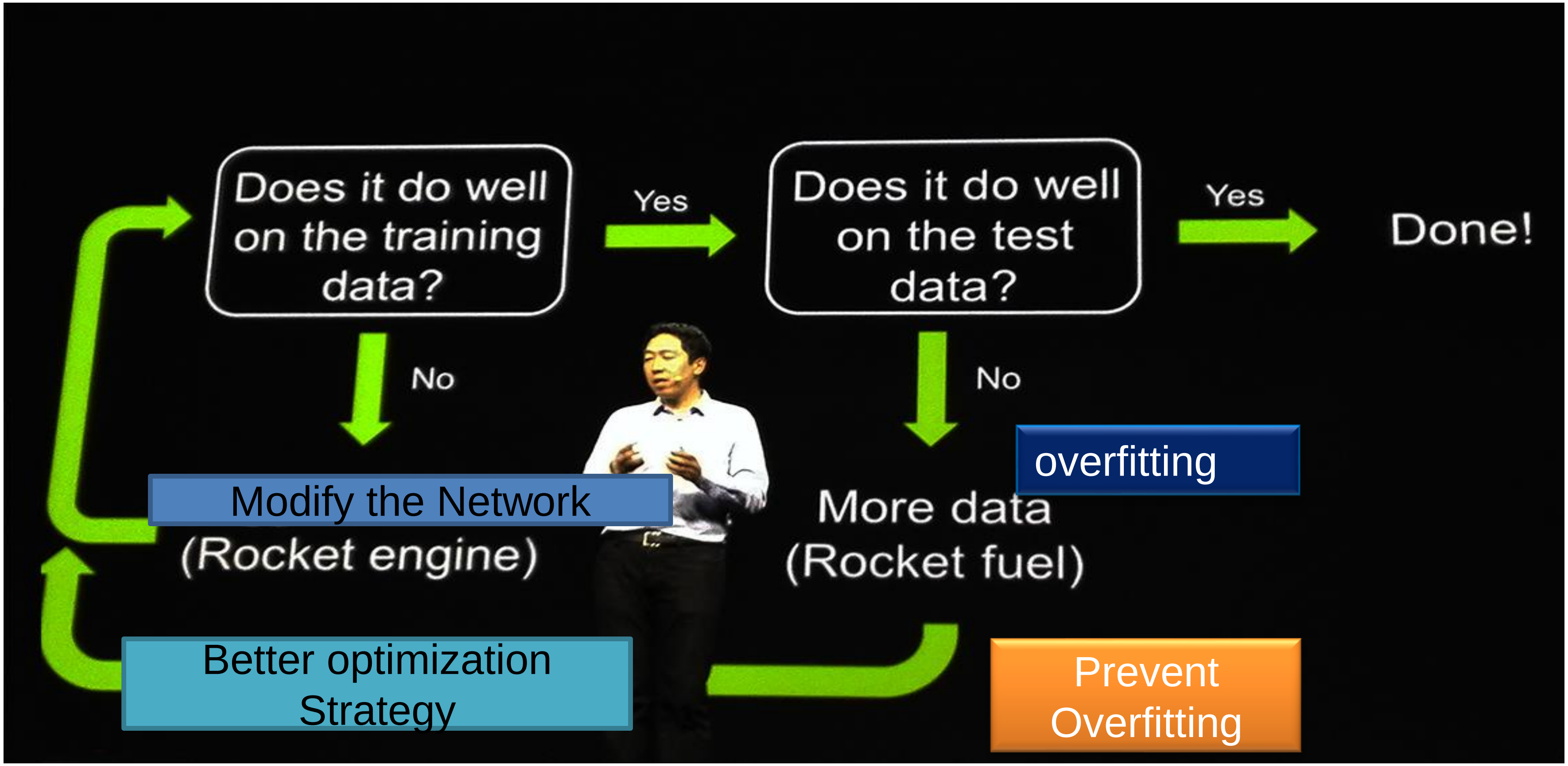


Extra modules



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

Recipe for Deep Learning



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

Next ...

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

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