Inferencing Issues:
prune irrelevant hidden nodes

國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

The point of deep learning frameworks

(1) Quick to develop and test new ideas
(2) Automatically compute gradients
(3) Run it all efficiently on GPU (wrap cuDNN, cuBLAS, OpenCL, etc)

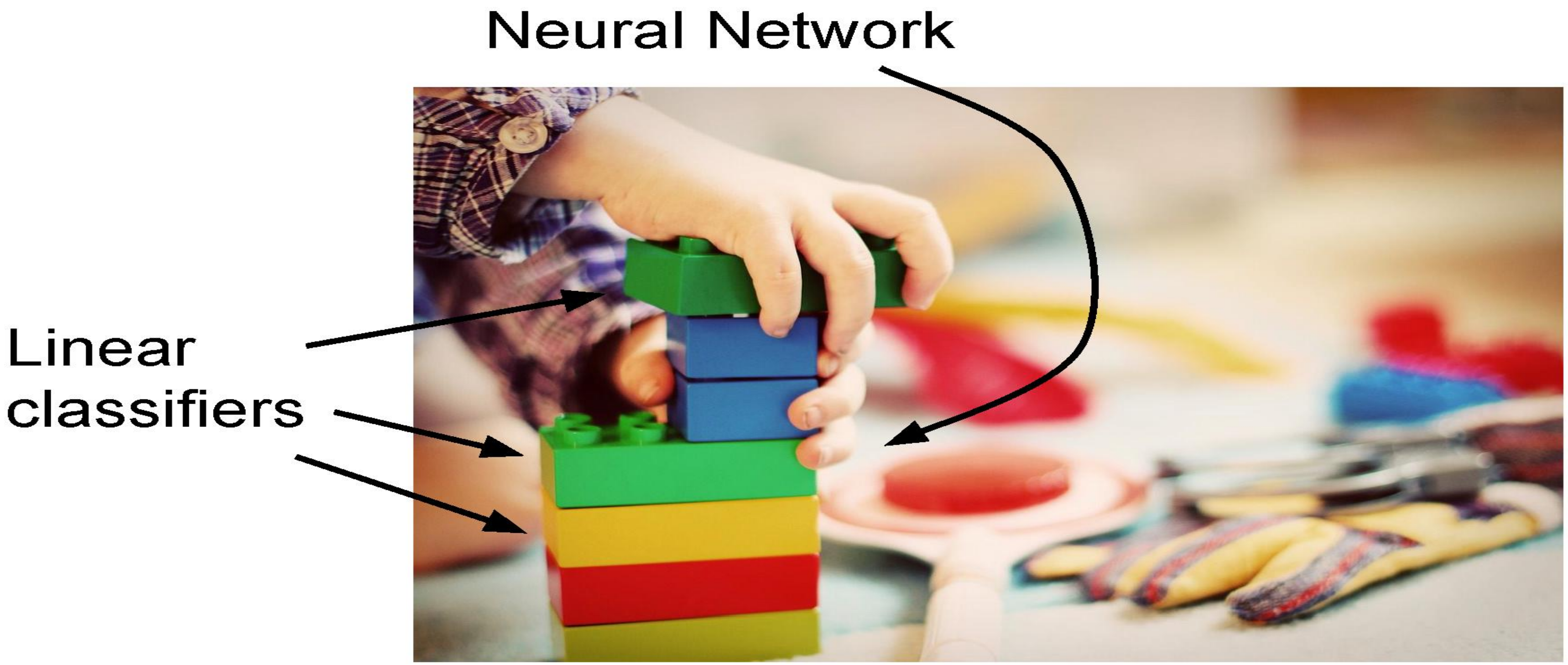
PyTorch: Fundamental Concepts

Tensor: Like a numpy array, but can run on GPU

Autograd: Package for building computational graphs out of Tensors, and automatically computing gradients

Module: A neural network layer; may store state or learnable weights

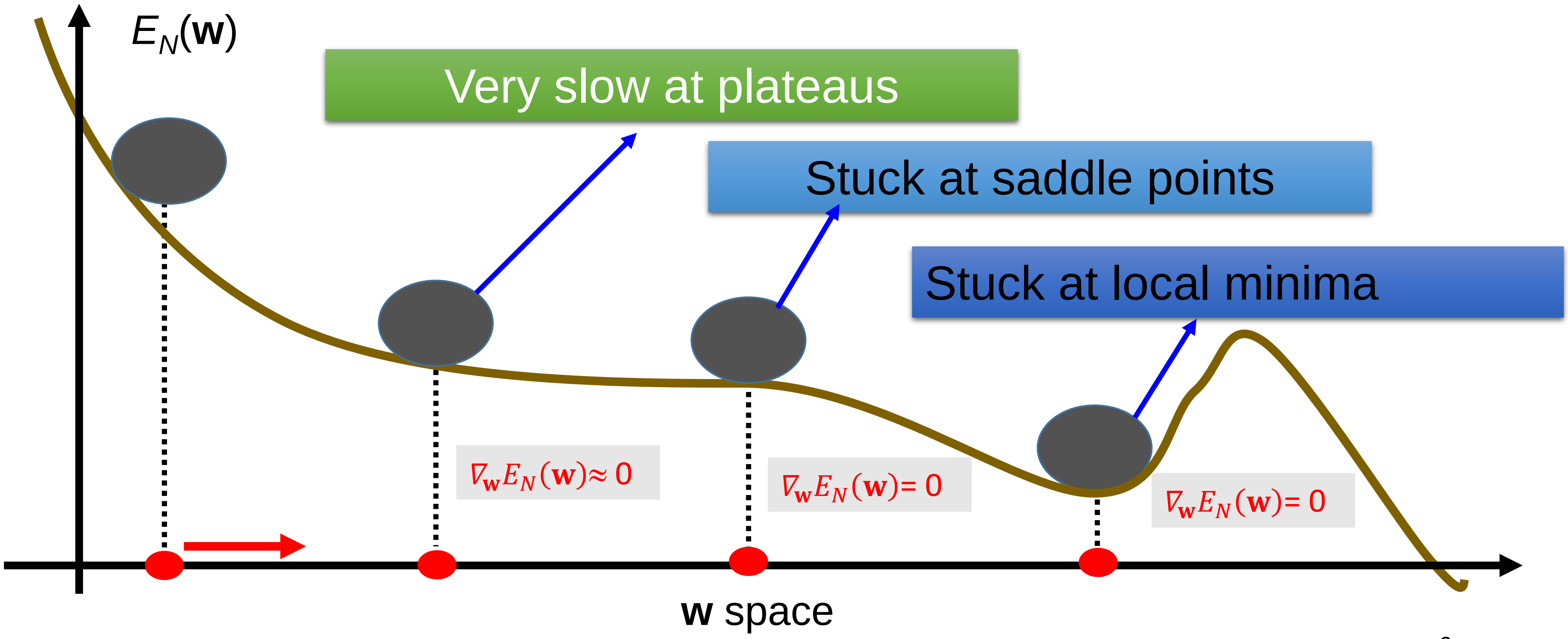
Developing a new AI system is like playing with Lego – lots of (pre-built or self-built) modules



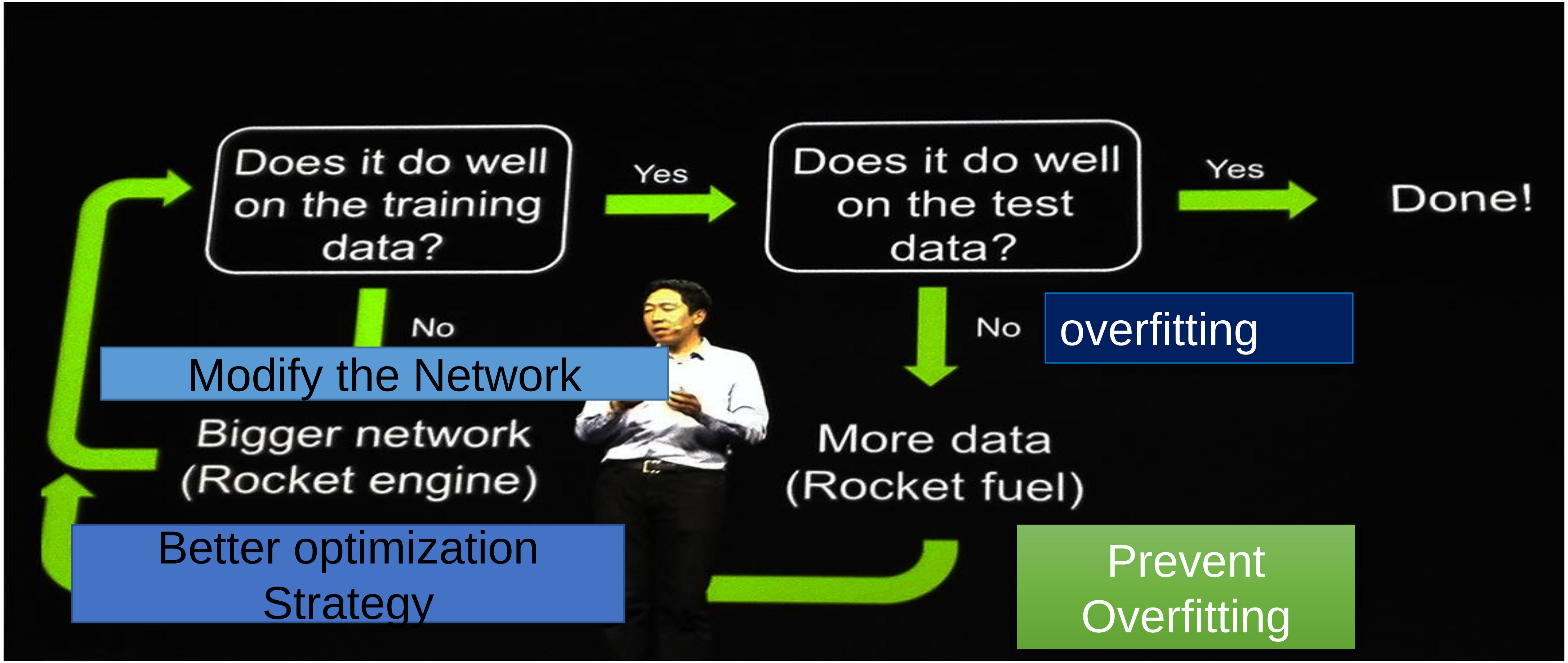
Where we are now...

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

You need to deal with undesired attractors. Not only for the purpose of learning, but of inferencing.



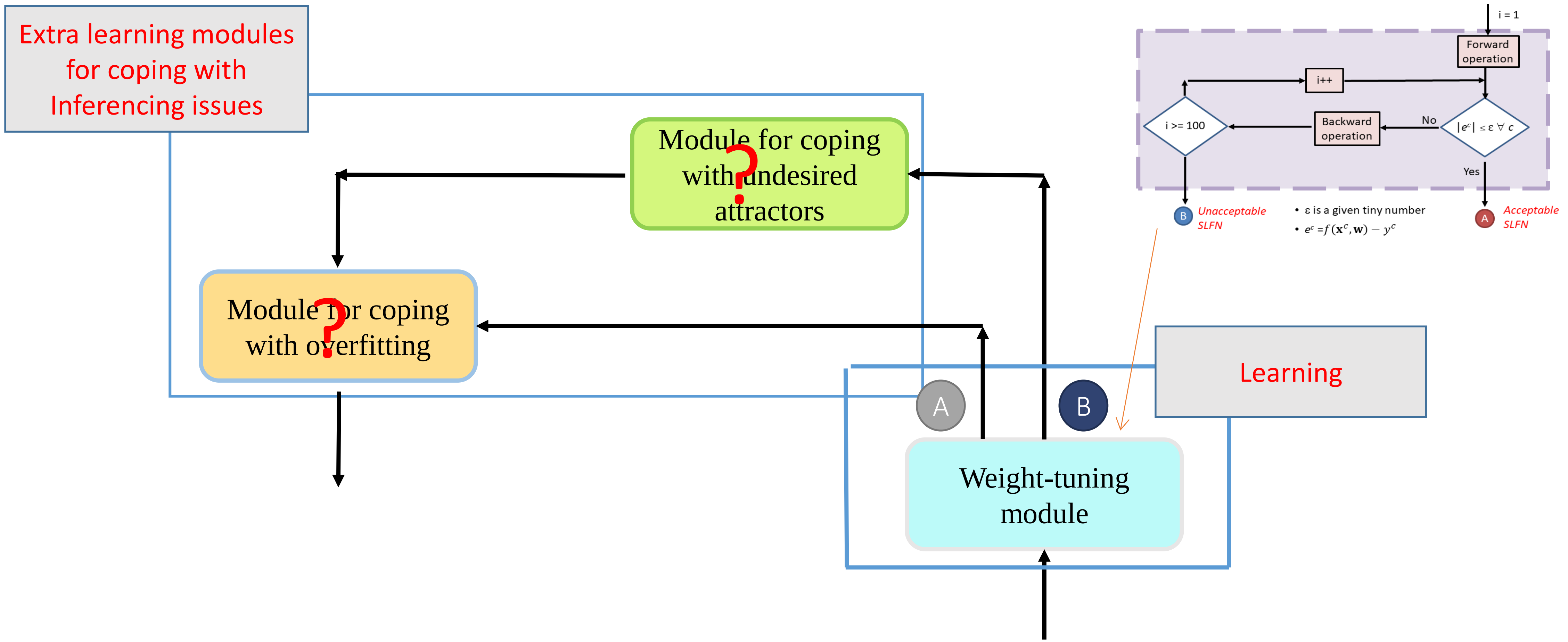
Recipe for Deep Learning



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

Where we are now...

Inferencing Issues



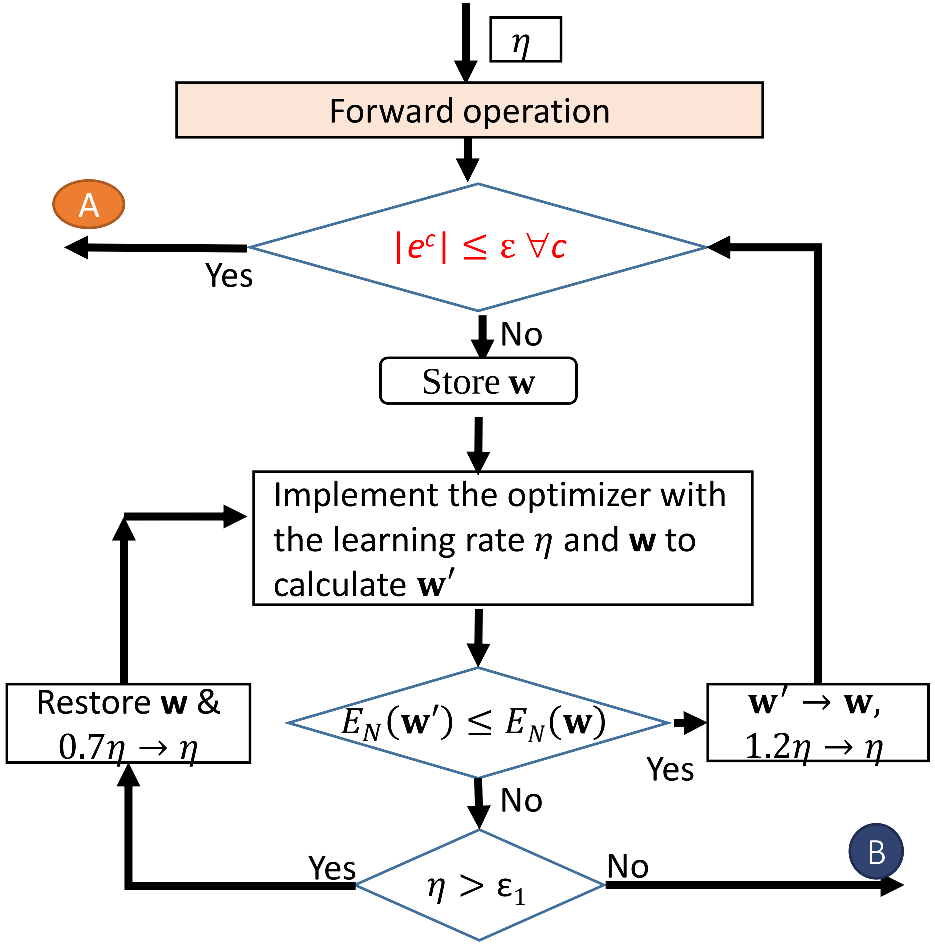
Where we are now...

The weight-tuning module

- helps tune weights to obtain an acceptable SLFN.

Hyperparameters:

- Optimizer
- ε & ε_1
- 1.2 & 0.7



Overfitting due to big weights

- To penalize big weights, there is a regularization term in the loss function:

- decay term: tiny λ
- Regularization term: arbitrary λ

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

- The weight decay coefficient λ determines how dominant the regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

- The above is the L2 regularization term
- L1 regularization: $\lambda|\mathbf{w}|$
- Elastic net (L1 + L2)

Where we are now...

Regularization

λ = regularization strength (hyperparameter)

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

With this $L(\mathbf{w})$, the learning process tries to make sure (1) $|e^c| \leq \varepsilon\ \forall c$ and (2) a smaller $\mathbf{w}$

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

Where we are now...

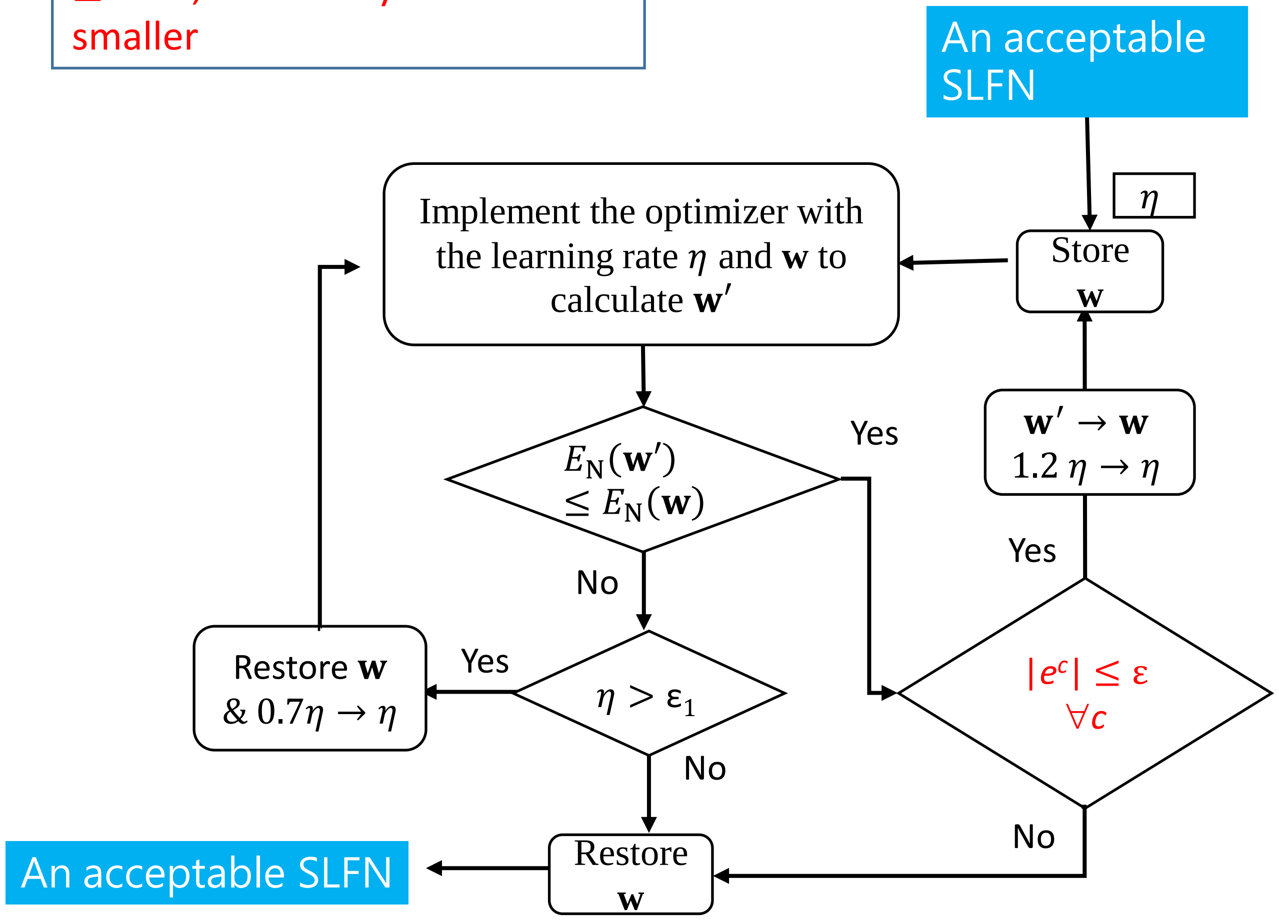
For an acceptable SLFN, $|e^c| \leq \varepsilon\ \forall c$, but **w** may not be smaller

The regularizing module

- helps further regularize weights after obtaining an acceptable SLFN.

Hyperparameters:

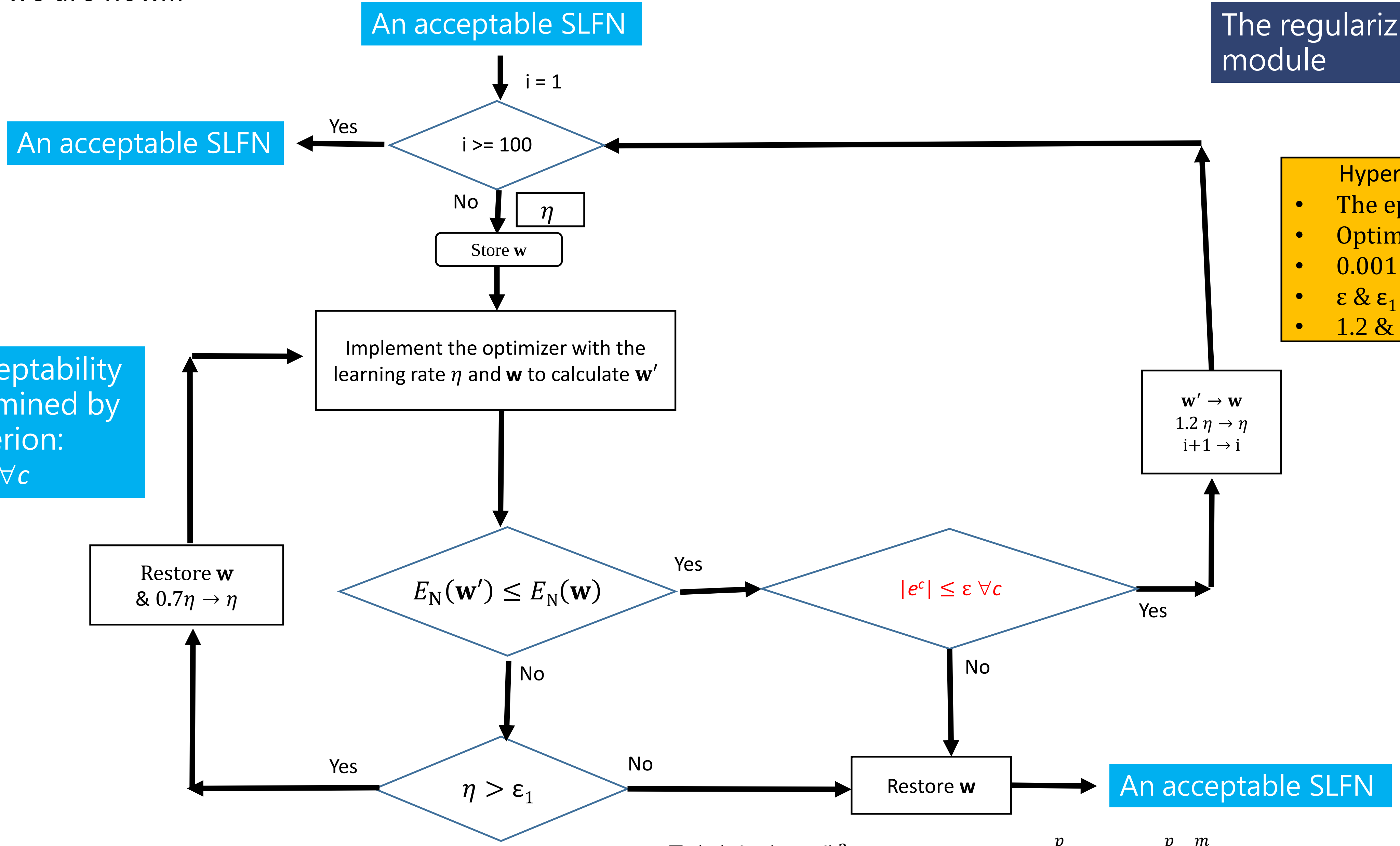
- Optimizer
- 0.001
- ε & ε_1
- 1.2 & 0.7



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

Where we are now...

The regularizing module



Hyperparameters:

- The epoch constraint
- Optimizer
- 0.001
- ε & ε_1
- 1.2 & 0.7

The acceptability is determined by the criterion: $|e^c| \leq \varepsilon\ \forall c$

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

In the learning process

- **The weight-tuning module helps tune weights to decrease the data error to obtain an acceptable SLFN.**
- **After obtaining an acceptable SLFN, the regularizing module with the regularization term helps further regularize weights while keeping the data error within the tolerance.**
- **A well-regularized SLFN can reduce the overfitting tendency.**

In practice:

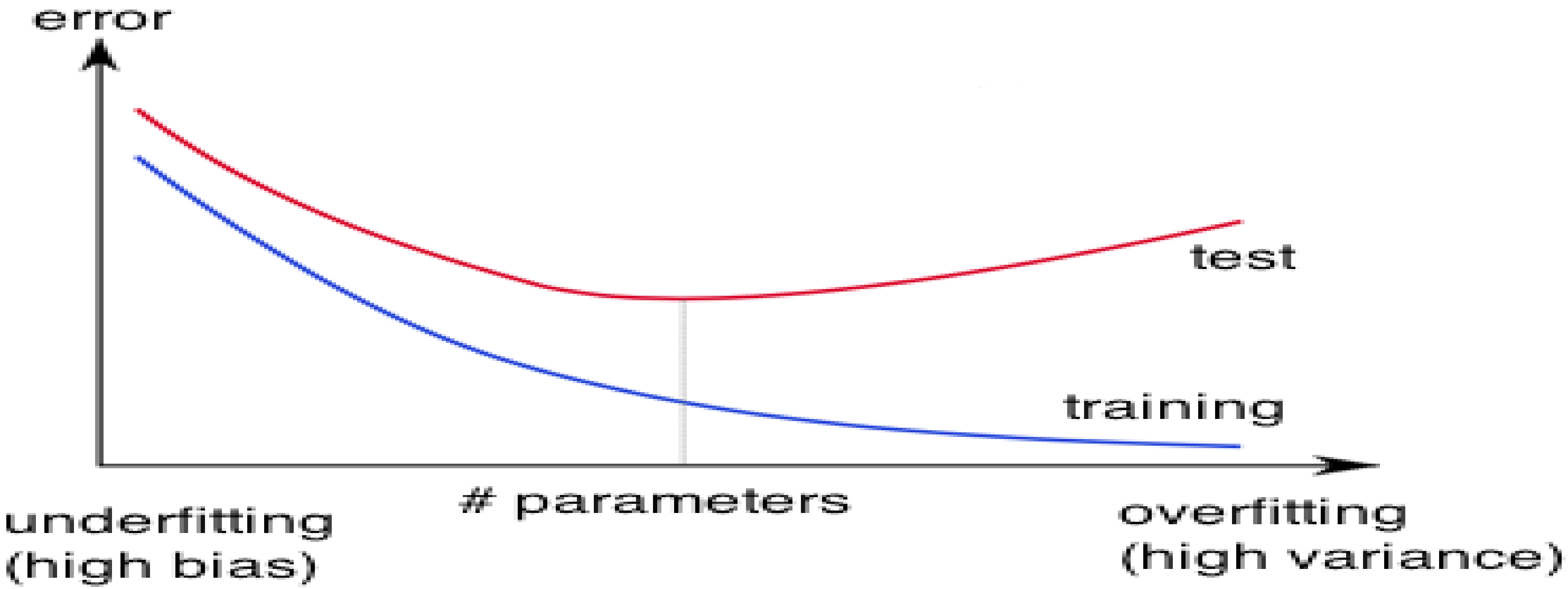
- **Adam** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule
  - Try cosine schedule, very few hyperparameters!

- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

In the learning process

Q: Which optimizer does better in the regularizing module?

Overfitting due to too many hidden nodes



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

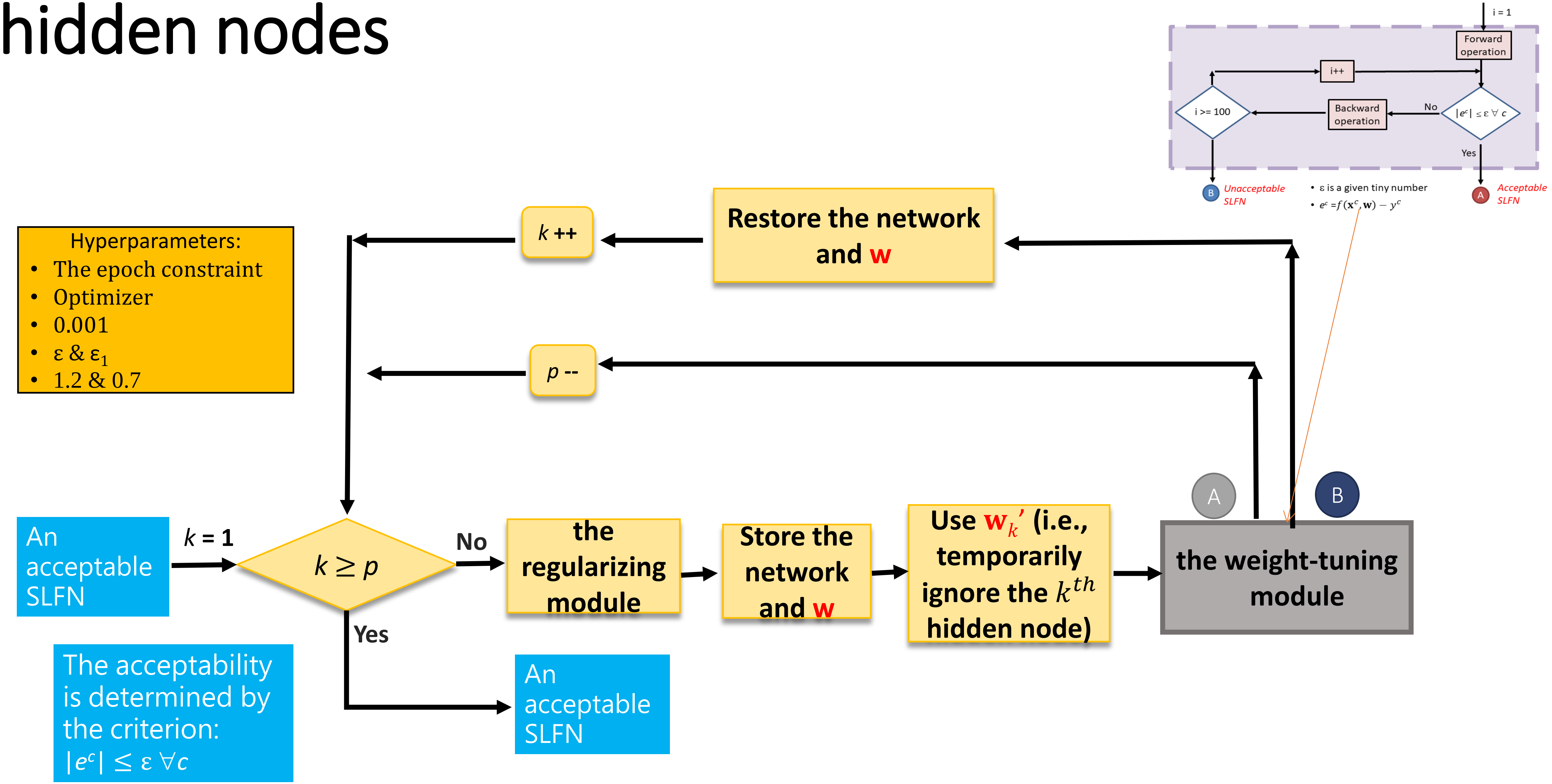
irrelevant hidden nodes & potentially irrelevant hidden nodes

- 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 i^{th} hidden node is *potentially irrelevant* if the learning goal can be accomplished via minimizing $E_N(\mathbf{w}_i')$, where $\mathbf{w}_i' \equiv \mathbf{w}-\{w_i^o, w_{i0}^H, \mathbf{w}_i^H\}$ and $f(\mathbf{x}^c, \mathbf{w}_i') \equiv \sum_{k\neq i} w_k^o a_k^c \ \forall\ c$. (Tsaih, 1993)

- Develop the reorganizing module that helps identify and remove the ***potentially irrelevant hidden node***.

The reorganizing module that one by one examines all hidden nodes



Indexes and Parameters

m	單筆輸入資料中共有m個變數，即SLFN模型中共有m個輸入節點
p	SLFN模型共有p個隱藏節點
w_i^o	第i個隱藏節點與輸出節點之間的激發值之權重，上標o表示該變數與輸出層相關
$\mathbf{w}^o = (w_1^o, w_2^o, \ldots, w_p^o)^{\mathrm{T}}$	所有隱藏節點與輸出節點之間的激發值之權重的向量，($(\cdot)^{\mathrm{T}}$ 為矩陣$(\cdot)$的轉置矩陣)
w_0^o	為輸出節點之閾值
w_{ij}^H	為第j個輸入節點與第i個隱藏節點之間的權重，上標H表示該變數與隱藏層相關
$\mathbf{w}_i^H = (w_{i1}^H, w_{i2}^H, \ldots, w_{im}^H)^{\mathrm{T}}$	第i個隱藏節點與所有輸入節點即輸入層之間的權重之向量
$\mathbf{W}^H = (\mathbf{w}_1^H, \mathbf{w}_2^H, \ldots, \mathbf{w}_p^H)^{\mathrm{T}}$	所有隱藏節點的權重的矩陣，即隱藏層與輸入層之間的權重的矩陣
w_{i0}^H	第i個隱藏節點之閾值
$\mathbf{w}_0^H = (w_{1,0}^H, w_{2,0}^H, \ldots, w_{p0}^H)^{\mathrm{T}}$	所有隱藏節點的閾值之向量
$\mathbf{x}^c \equiv (x_1^c, x_2^c, \ldots, x_m^c)^{\mathrm{T}}$	the input vector of the c^{th} case
$\boldsymbol{a}^c \equiv (a_1^c, a_2^c, \ldots, a_m^c)^{\mathrm{T}}$	the hidden activation vector of the c^{th} case
y^c	the desired output associated with $\mathbf{x}^c$

irrelevant hidden nodes & potentially irrelevant hidden nodes

- 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 **w**, the i^{th} hidden node is *potentially irrelevant* if the learning goal can be accomplished via minimizing $E_N(\mathbf{w}_i')$, where $\mathbf{w}_i' \equiv \mathbf{w} - \{w_i^o, w_{i0}^H, \mathbf{w}_i^H\}$ and $f(\mathbf{x}^c, \mathbf{w}_i') \equiv \sum_{k \neq i} w_k^o a_k^c \ \forall\ c$. (Tsaih, 1993)

- Use the principal component analysis (PCA) to help identify the ***potentially irrelevant hidden node***.

Principal Component Analysis (Wold, Esbensen, & Geladi, 1987; Jolliffe, 2020)

$$(\mathbf{M} - \lambda\mathbf{I})\boldsymbol{\alpha} = \mathbf{0}$$

$$\lambda = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_i \end{bmatrix} \ (Score\ Vector) \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha'_{11} & \dots & \alpha'_{1m} \\ \vdots & \ddots & \vdots \\ \alpha'_{i1} & \dots & \alpha'_{im} \end{bmatrix} (Loading\ Vector)$$

$$pc_i = \alpha'_{i1}x_1 + \alpha'_{i2}x_2 + \cdots + \alpha'_{im}x_m = \sum_j^m \alpha'_{ij}x_j \ \forall\ 1 \le i \le h, 1 \le j \le m$$

There are function calls related with PCA in PyTorch and TensorFlow.

Principal Component Analysis, PCA

1. Standardize the data set $\{\mathbf{x}^c\}$ with mean = 0 and variance = 1 to get the data set $\{\hat{\mathbf{x}}^c\}$.

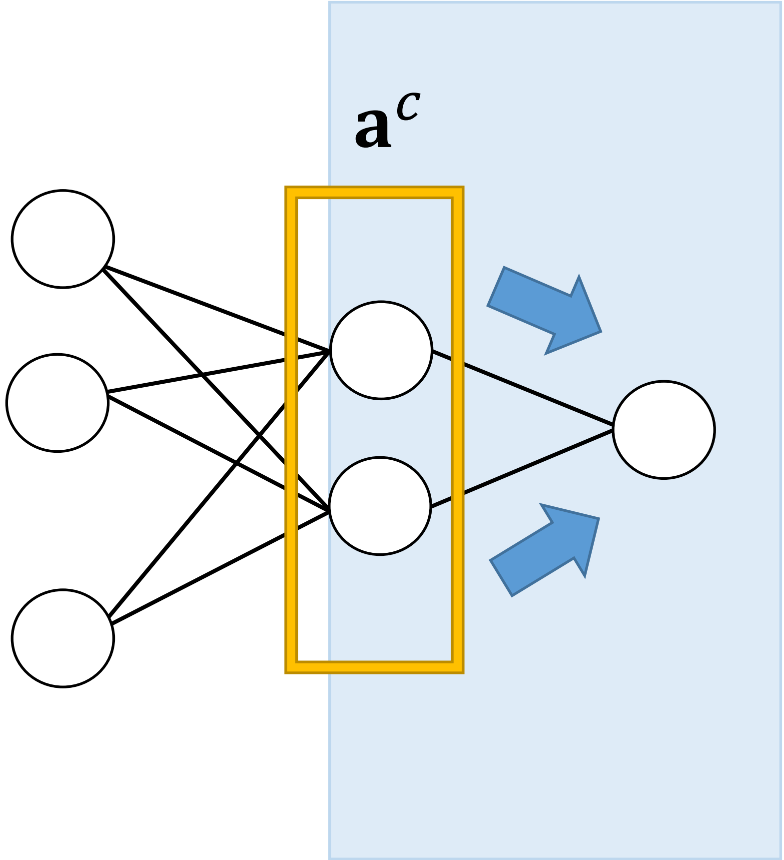
   May not be necessary

2. Compute the covariance matrix **M** of the data set $\hat{\mathbf{x}}^c$.
3. Obtain the eigenvectors and eigenvalues from the covariance matrix **M**.
4. Sort eigenvalues in descending order and choose the top h eigenvectors that correspond to the h largest eigenvalues.
5. Construct the projection matrix $\boldsymbol{\alpha}$ from the selected h eigenvectors.
6. Transform the data set $\hat{\mathbf{x}}^c$ via $\boldsymbol{\alpha}$ to obtain the new h-dimensional feature subspace.

There are function calls related with PCA in PyTorch and TensorFlow.

The PCA application to SLFN

- Standardize the data set $\{\mathbf{a}^c\}$ with mean = 0 and variance = 1 to get the data set $\{\hat{\mathbf{a}}^c\}$. May not be necessary
- Apply PCA to $\{\hat{\mathbf{a}}^c\}$ to generate principal components (PCs) denoted as pc_i.
- Select the h top significant pc_i at the criterion of 85% of total explanation ability satisfied.



PCA

- $pc_1 = \boxed{\alpha'_{11}\hat{a}_1 + \cdots + \alpha'_{1p}\hat{a}_p}$
- $pc_2 = \boxed{\alpha'_{21}\hat{a}_1 + \cdots + \alpha'_{2p}\hat{a}_p}$
- $pc_3 = \boxed{\alpha'_{31}\hat{a}_1 + \cdots + \alpha'_{3p}\hat{a}_p}$
- $pc_4 = \boxed{\alpha'_{41}\hat{a}_1 + \cdots + \alpha'_{4p}\hat{a}_p}$
- $pc_5 = \boxed{\alpha'_{51}\hat{a}_1 + \cdots + \alpha'_{5p}\hat{a}_p}$
- $pc_6 = \boxed{\alpha'_{61}\hat{a}_1 + \cdots + \alpha'_{6p}\hat{a}_p}$

$\boldsymbol{f}' = \beta_1 \times$ $+ \beta_2 \times$ 85% of total explanation ability $+ \beta_3 \times$

Linear Regression

PCA

$$\boldsymbol{f}' = \beta_1 \times \boxed{\alpha'_{11}\hat{a}_1 + \cdots + \alpha'_{1p}\hat{a}_p} + \beta_2 \times \boxed{\alpha'_{21}\hat{a}_1 + \cdots + \alpha'_{2p}\hat{a}_p} + \beta_3 \times \boxed{\alpha'_{31}\hat{a}_1 + \cdots + \alpha'_{3p}\hat{a}_p}$$

$$\boldsymbol{f}' = \boxed{\beta_1\alpha'_{11} + \beta_2\alpha'_{21} + \beta_3\alpha'_{31}} \times \hat{a}_1 + \boxed{\beta_1\alpha'_{12} + \beta_2\alpha'_{22} + \beta_3\alpha'_{32}} \times \hat{a}_2 + \cdots + \boxed{\beta_1\alpha'_{1p} + \beta_2\alpha'_{2p} + \beta_3\alpha'_{3p}} \times \hat{a}_p$$

PCA

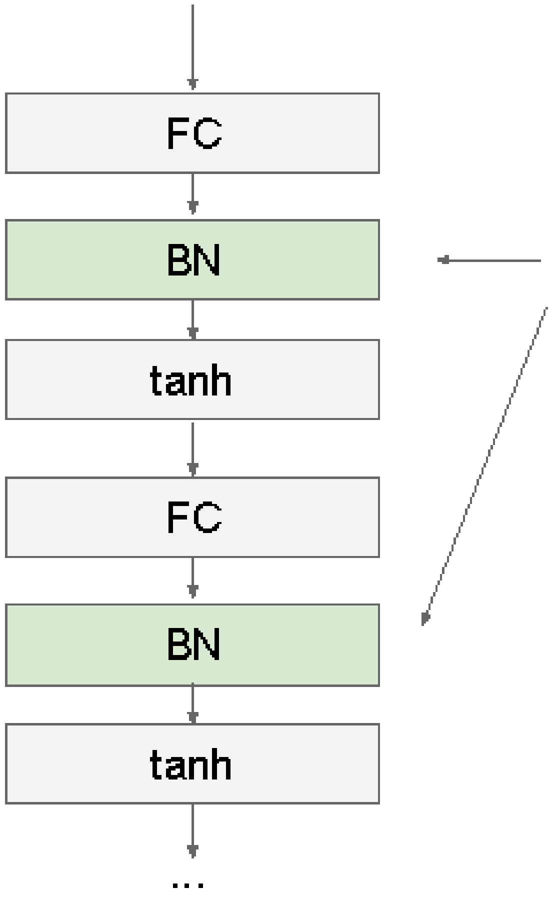
$$\boldsymbol{f}' = \beta_1 \times \boxed{\alpha'_{11}\hat{a}_1 + \cdots + \alpha'_{1p}\hat{a}_p} + \beta_2 \times \boxed{\alpha'_{21}\hat{a}_1 + \cdots + \alpha'_{2p}\hat{a}_p} + \beta_3 \times \boxed{\alpha'_{31}\hat{a}_1 + \cdots + \alpha'_{3p}\hat{a}_p}$$

- Let $f' = \omega_1\hat{a}_1 + \omega_2\hat{a}_2 + \cdots + \omega_p\hat{a}_p$
- Let $k = \mathrm{argmin}_i|\omega_i|$.
- Obtain $\mathbf{w}'_k = \mathbf{w} - \{w^o_k, w^{\mathrm{H}}_{k0}, \mathbf{w}^H_k\}$

$$\boldsymbol{f}' = \boxed{\beta_1\alpha'_{11} + \beta_2\alpha'_{21} + \beta_3\alpha'_{31}} \times \hat{a}_1 + \boxed{\beta_1\alpha'_{12} + \beta_2\alpha'_{22} + \beta_3\alpha'_{32}} \times \hat{a}_2 + \cdots + \boxed{\beta_1\alpha'_{1p} + \beta_2\alpha'_{2p} + \beta_3\alpha'_{3p}} \times \hat{a}_p$$

Batch Normalization

[Ioffe and Szegedy, 2015]



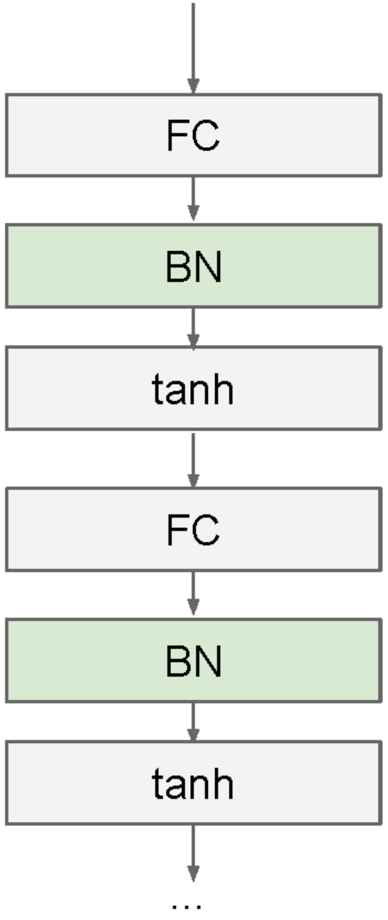
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Where we are now...

Batch Normalization

[Ioffe and Szegedy, 2015]

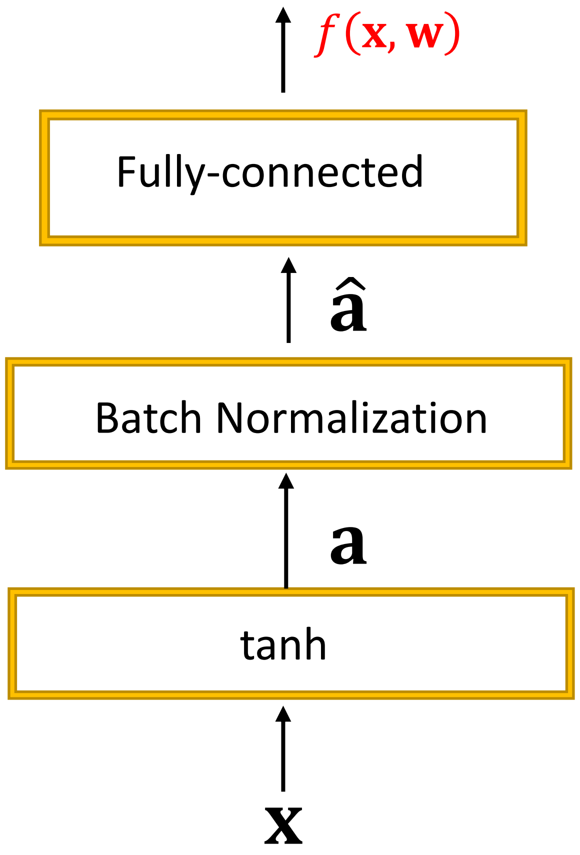


- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training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

The SLFN with BN for PCA

In the reorganizing module, Batch Normalization is inserted after the nonlinearity layer and before the FC layer.

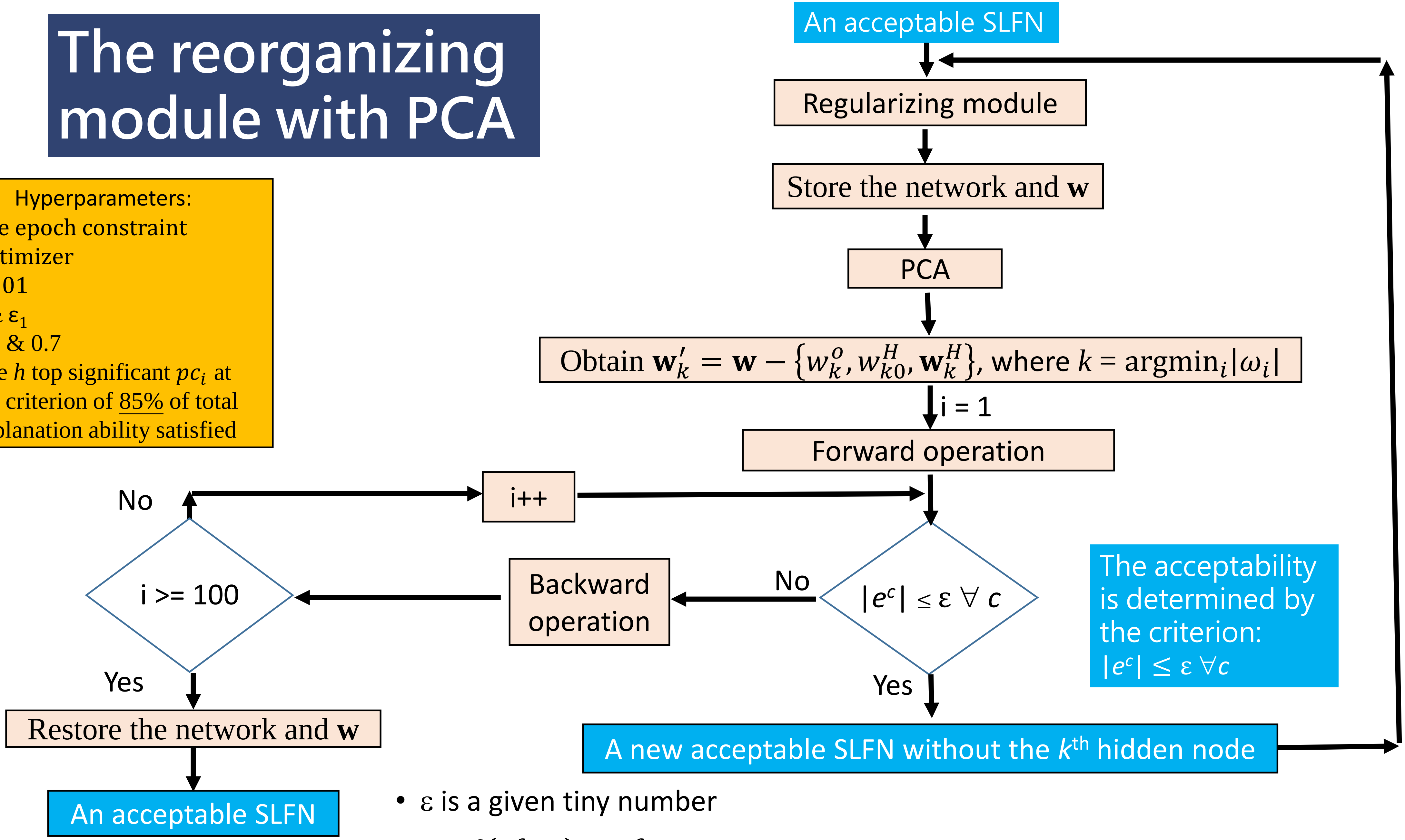
This may not be necessary.



The reorganizing module with PCA

Hyperparameters:

- The epoch constraint
- Optimizer
- 0.001
- ε & ε_1
- 1.2 & 0.7
- The h top significant pc_i at the criterion of <u>85%</u> of total explanation ability satisfied



- ε is a given tiny number
- $e^c = f(\mathbf{x}^c, \mathbf{w}) - y^c$

Homework #5

Write down the code of the reorganizing module that helps identify and remove the potential irrelevant hidden node.