The learning algorithm: Back Propagation

國立政治大學 資訊管理學系
蔡瑞煌 特聘教授

Where we are now...

TensorFlow

https://www.tensorflow.org/

- TensorFlow 是 Google 開發的開源機器學習工具
- 透過使用Tensor, Computational graph, and GPU，來進行數值演算
- 支援程式語言：python、C++
- 系統需求：
  - 作業系統必須為Mac、Linux或Windows
  - Upgraded Python 2.7 或 3.3 （含以上）

Tensor 張量

- n維度的陣列資料
  可為純量、向量或矩陣

An n-dimensional array

0-d tensor : scalar (number)

1-d tensor : vector

2-d tensor : matrix

and so on

- **rank** 表示張量的維度

```
[1. ,2., 3.] # a rank 1 tensor; this is a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
```

Where we are now...

機器學習Library (ex, scikit-learn)

Frameworks: PyTorch / TensorFlow

從頭開始寫

把資料整理好後，剩下的就直接呼叫API

自行定義 forward operations及 loss function，交由 framework 來運算Computational Graph and gradients

自己推導微分公式，自己寫整個流程

低 技術門檻 高

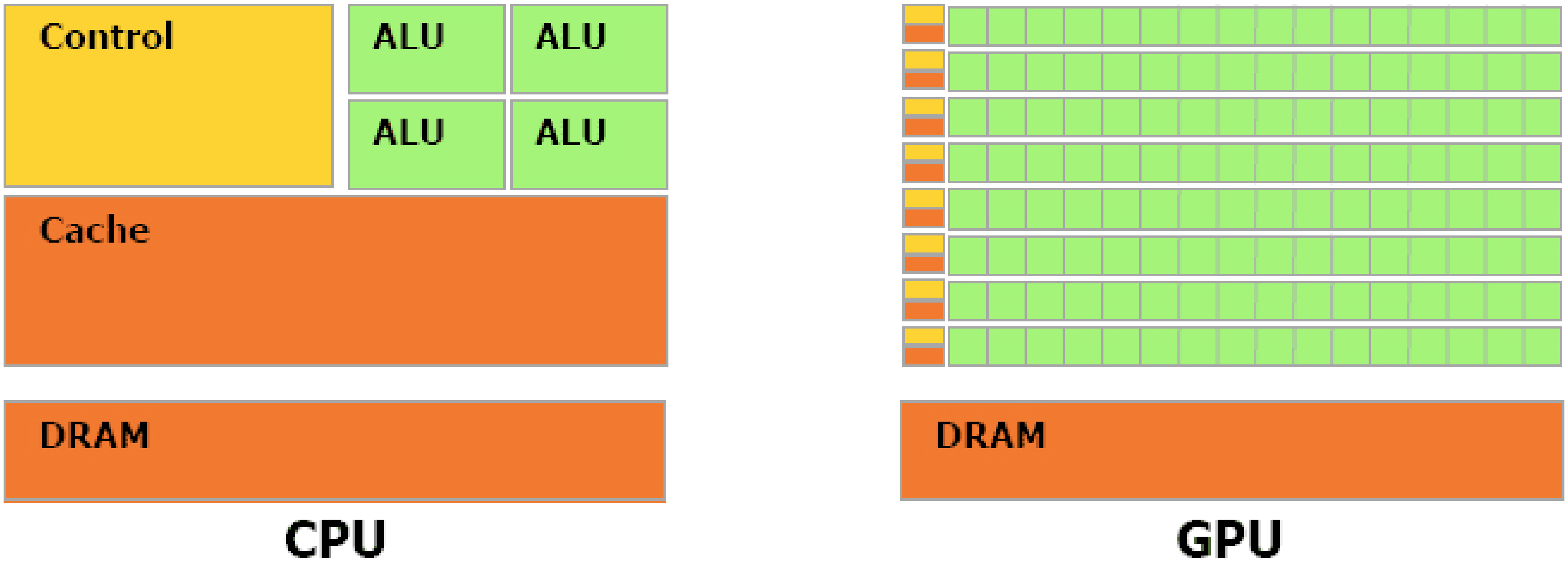
低 彈性 高

Where we are now...

- 彈性
  - 只要是可以用Computational Graph來表達的運算，都可以用PyTorch/TensorFlow來coding
- 自動微分
  - 自動計算Computational Graph及微分後的結果
- 平台相容性
  - 同樣的程式碼可用CPU執行，亦可用GPU執行

Where we are now...

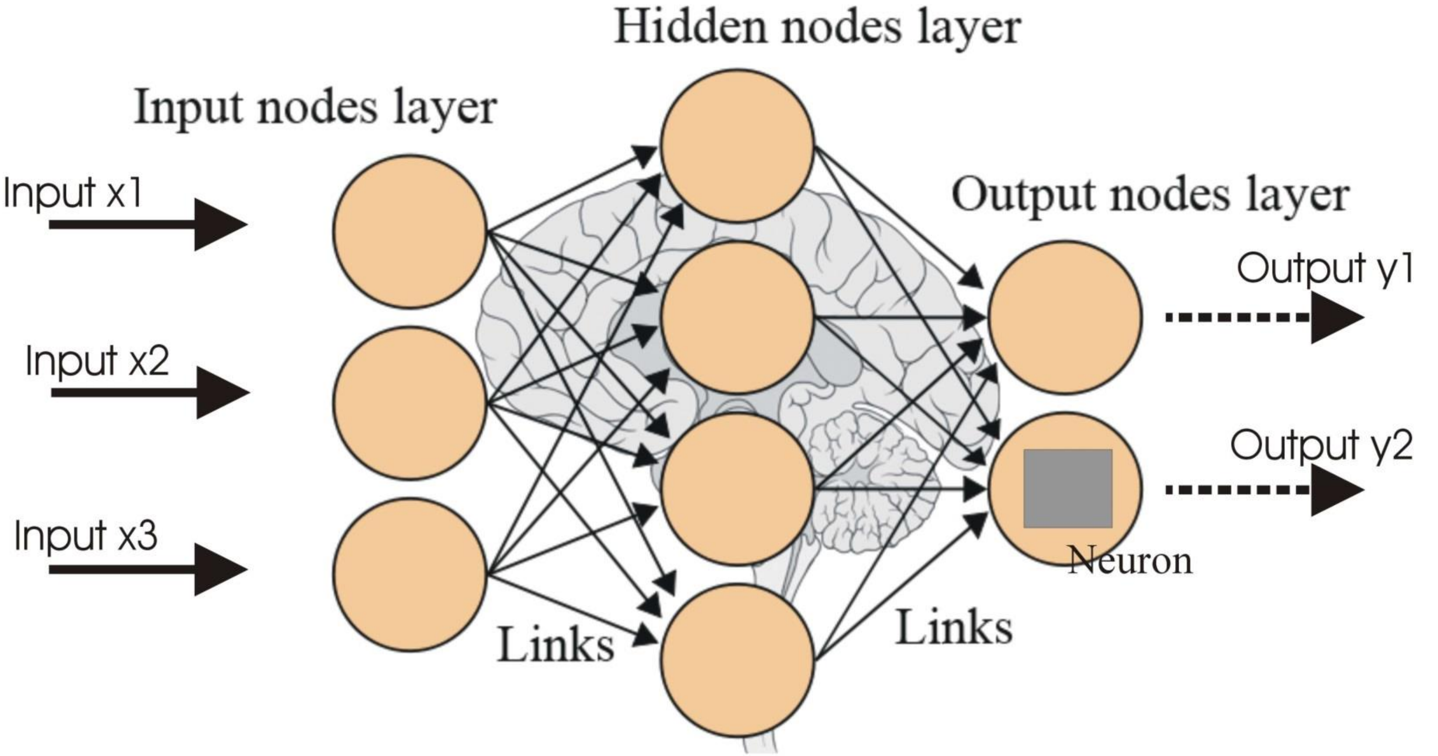
CPU V.S. GPU



http://allegroviva.com/gpu-computing/difference-between-gpu-and-cpu/

Where we are now...

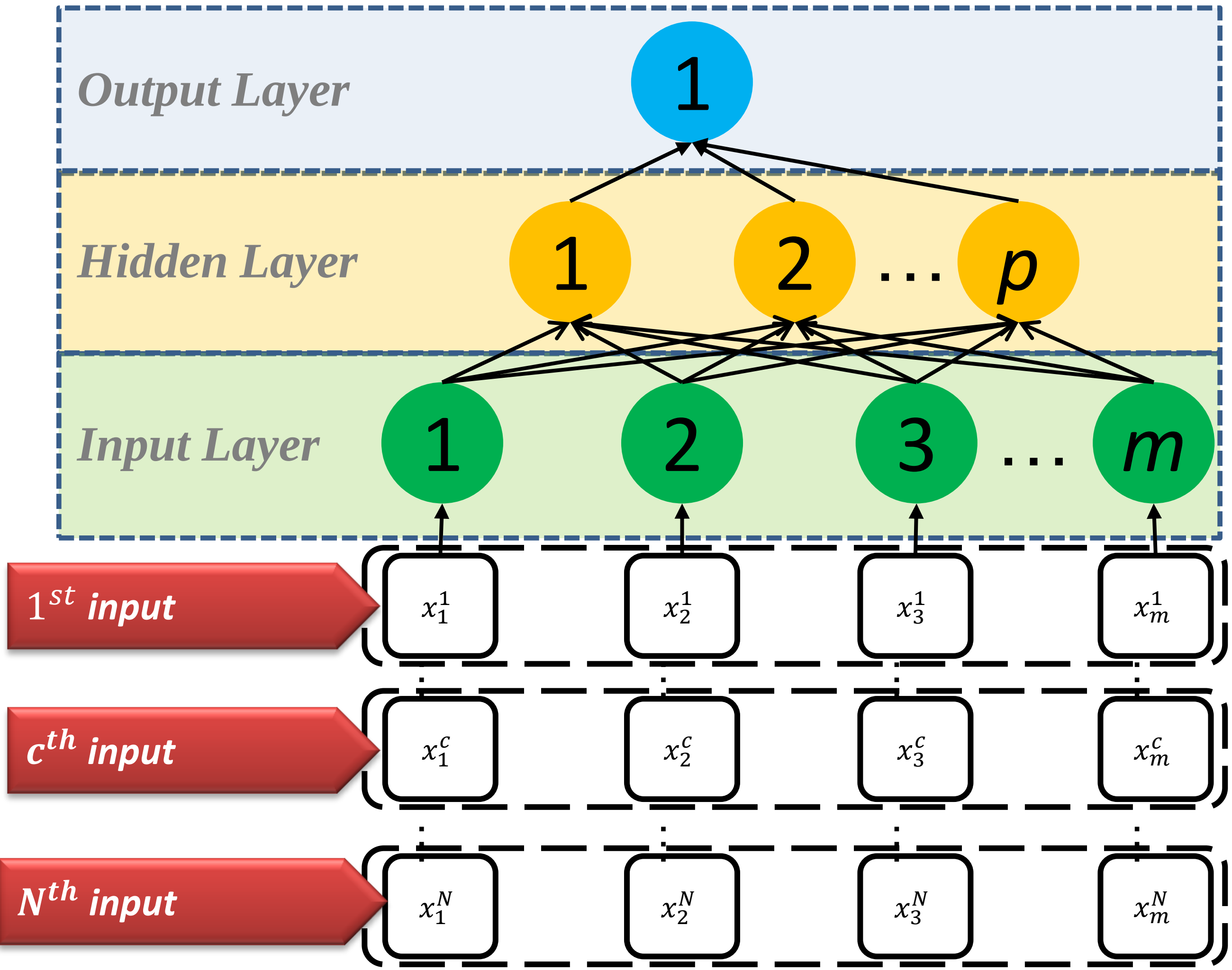
2-Layer Neural Networks; Single-hidden Layer Feed-forward Neural Networks (SLFN)



Where we are now...

The Network Structure of the SLFN with one output node

2-layer Neural Networks



- ***1*** output node
- ***p*** hidden nodes
- ***m*** input nodes for inputs

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 $\mathbf{x}^c$

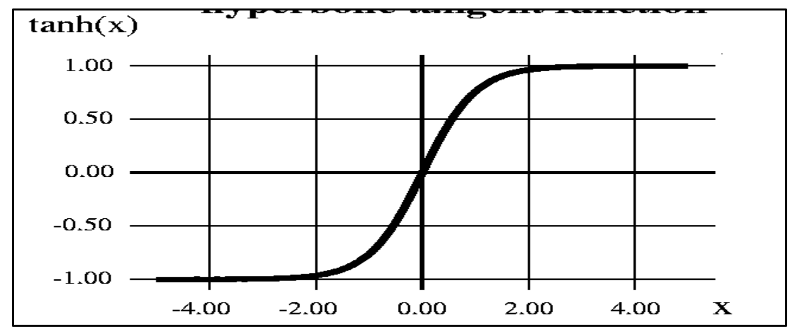
Where we are now...

Popular Activation Functions

- *tanh*, the hyperbolic tangent activation function

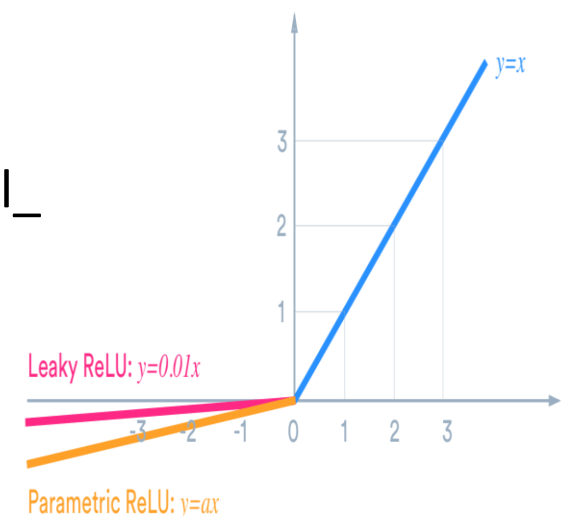
$$tanh: R \rightarrow [-1.0, 1.0],$$

$$tanh(x) \equiv \frac{1.0 - exp(-2x)}{1.0 + exp(-2x)}$$



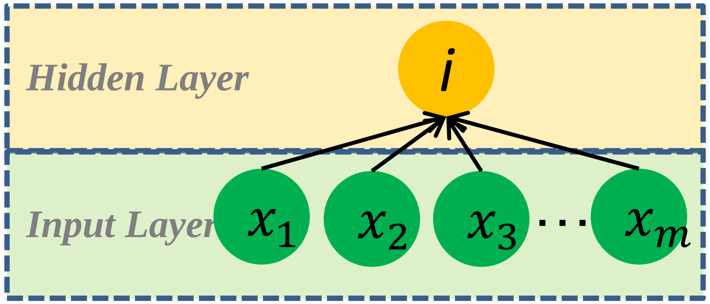
- the *ReLU* activation function and its variants

https://en.wikipedia.org/wiki/Rectifier_(neural_networks)



Where we are now...

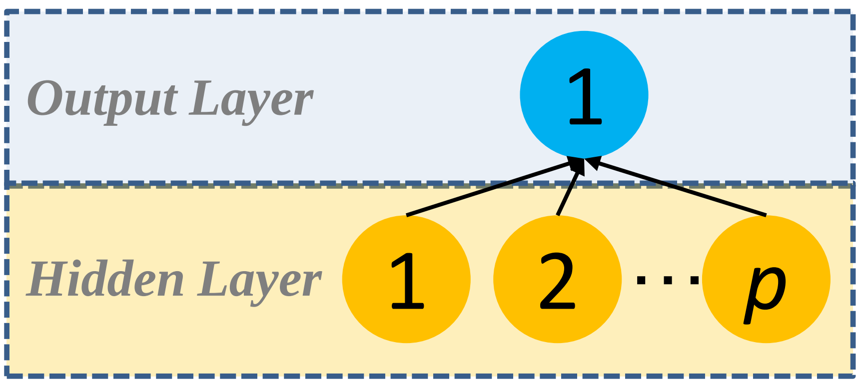
Activation Values of Nodes



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

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

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



The activation value of the output node:

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

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

The loss function and the learning goal

- In the learning stage, a set of N training cases $\{(\mathbf{x}^1, y^1),(\mathbf{x}^2, y^2), . . . ,(\mathbf{x}^N, y^N)\}$ is given.
- The loss function is defined as:

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

- The learning becomes an optimization problem: $\min_{\mathbf{w}} E_N(\mathbf{w})$
- Regarding all training cases, the learning goal is to seek a $\mathbf{w}$ where, for all c (training cases) and a tiny ε, $|f(\mathbf{x}^c,\mathbf{w}) - y^c| < \varepsilon$.

Back Propagation learning algorithm

Step 0.1: Input all training data $\{(\mathbf{x}^1, y^1),(\mathbf{x}^2, y^2), . . . ,(\mathbf{x}^N, y^N)\}$.

Step 0.2: Generate the initial values of $\mathbf{w}$.

Step 1: Execute the forward operation of SLFN regarding all training data.

Step 2: Based upon $f(\mathbf{x}^c, \mathbf{w})$ and y^c values, calculate the $E_N(\mathbf{w})$ value and store it.

Step 3: If $E_N(\mathbf{w})$is less than the predetermined value (says, ε), then STOP.

Step 4: Calculate the gradient vector $\nabla_{\mathbf{w}} E_N(\mathbf{w})$ of SLFN.

Step 5: With the gradient vector $\nabla_{\mathbf{w}} E_N(\mathbf{w})$ obtained in *Step* 4 and the learning rate η, update the values of $\mathbf{w}$ $(\text{i.e.}, \mathbf{w} \leftarrow \mathbf{w} - \eta * \nabla_{\mathbf{w}} E_N(\mathbf{w}))$.

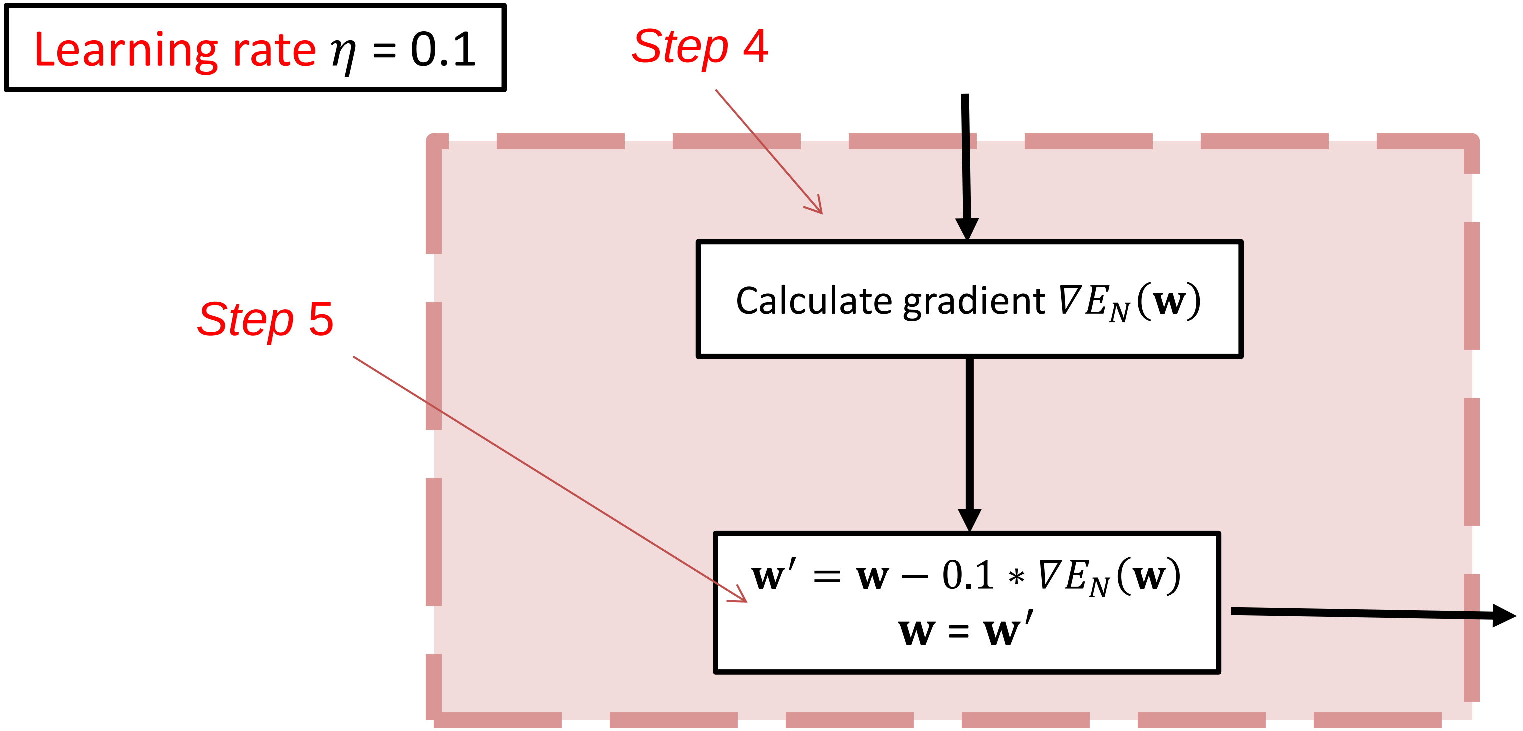
Step 6: go to Step 1.

a loop

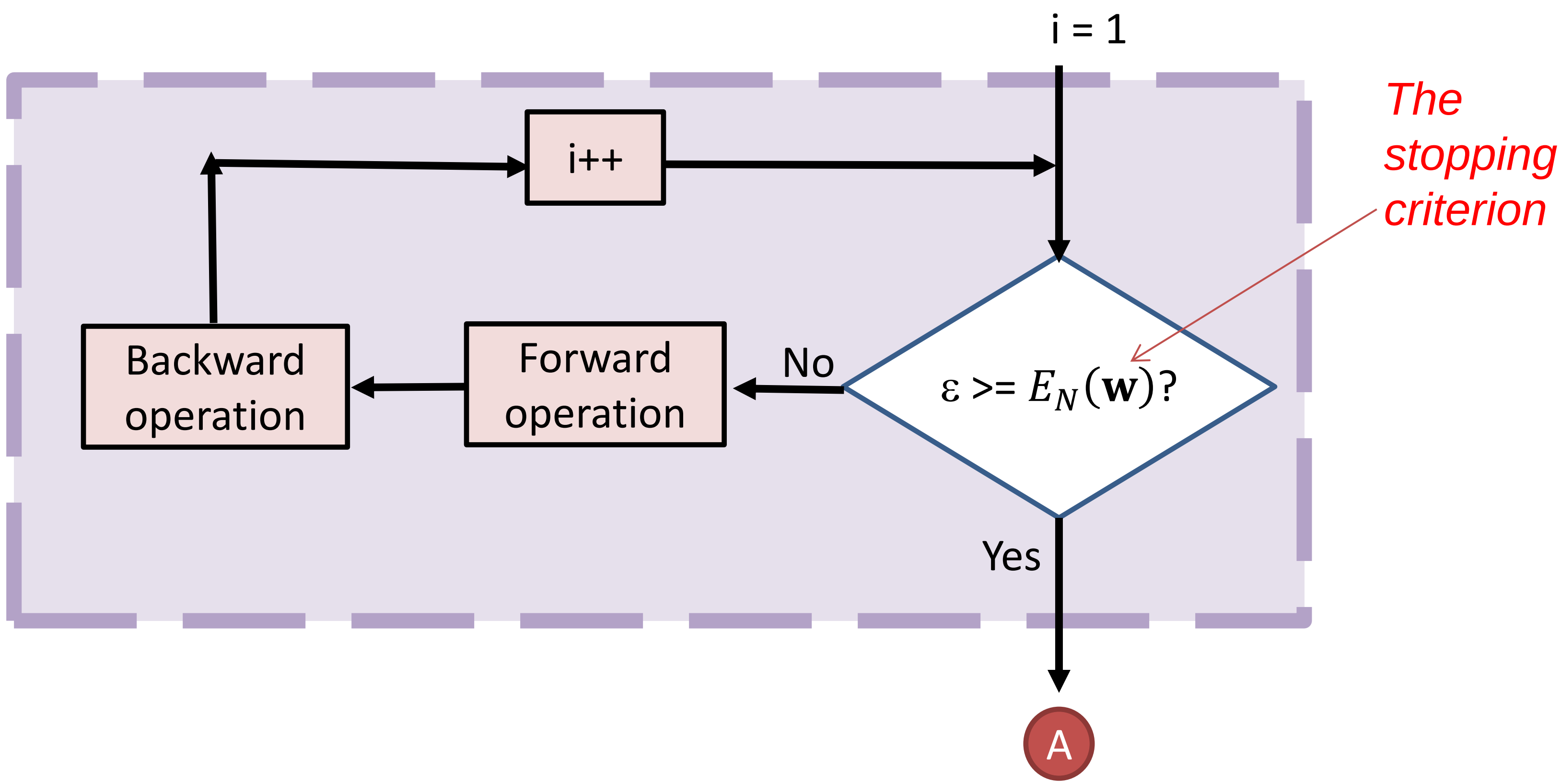
Most learning algorithms has similar structure. When you change some parts, you get a different algorithm.

Backward operation

- Calculate the gradient and the adjustment of **w**



The program for learning



The Backward Operation

Step 4: Based upon values of a_i^c all *i* all *c* and $f(\mathbf{x}^c, \mathbf{w})$ all *c*, execute the following backward operations:

Step 4.1: calculate the values of $\frac{\partial E(\mathbf{w})}{\partial w_0^o} \equiv \frac{2}{N} \sum_{c=1}^{N} (f(\mathbf{x}^c, \mathbf{w}) - y^c)$ and store them.

Step 4.2: calculate the values of $\frac{\partial E(\mathbf{w})}{\partial w_i^o} \equiv \frac{2}{N} \sum_{c=1}^{N} (f(\mathbf{x}^c, \mathbf{w}) - y^c) a_i^c$ all *i* and store them.

Step 4.3: calculate the values of $\frac{\partial E(\mathbf{w})}{\partial w_{i0}^H} \equiv \frac{2}{N} \sum_{c=1}^{N} (f(\mathbf{x}^c, \mathbf{w}) - y^c)\,(1 - (a_i^c)^2)\, w_{li}^o$ all *i* and store them.

Step 4.4: calculate the values of $\frac{\partial E(\mathbf{w})}{\partial w_{ij}^H} \equiv \frac{2}{N} \sum_{c=1}^{N} (f(\mathbf{x}^c, \mathbf{w}) - y^c)\,(1 - (a_i^c)^2)\, w_{li}^o x_{cj}$ all *i* all *j* and store them.

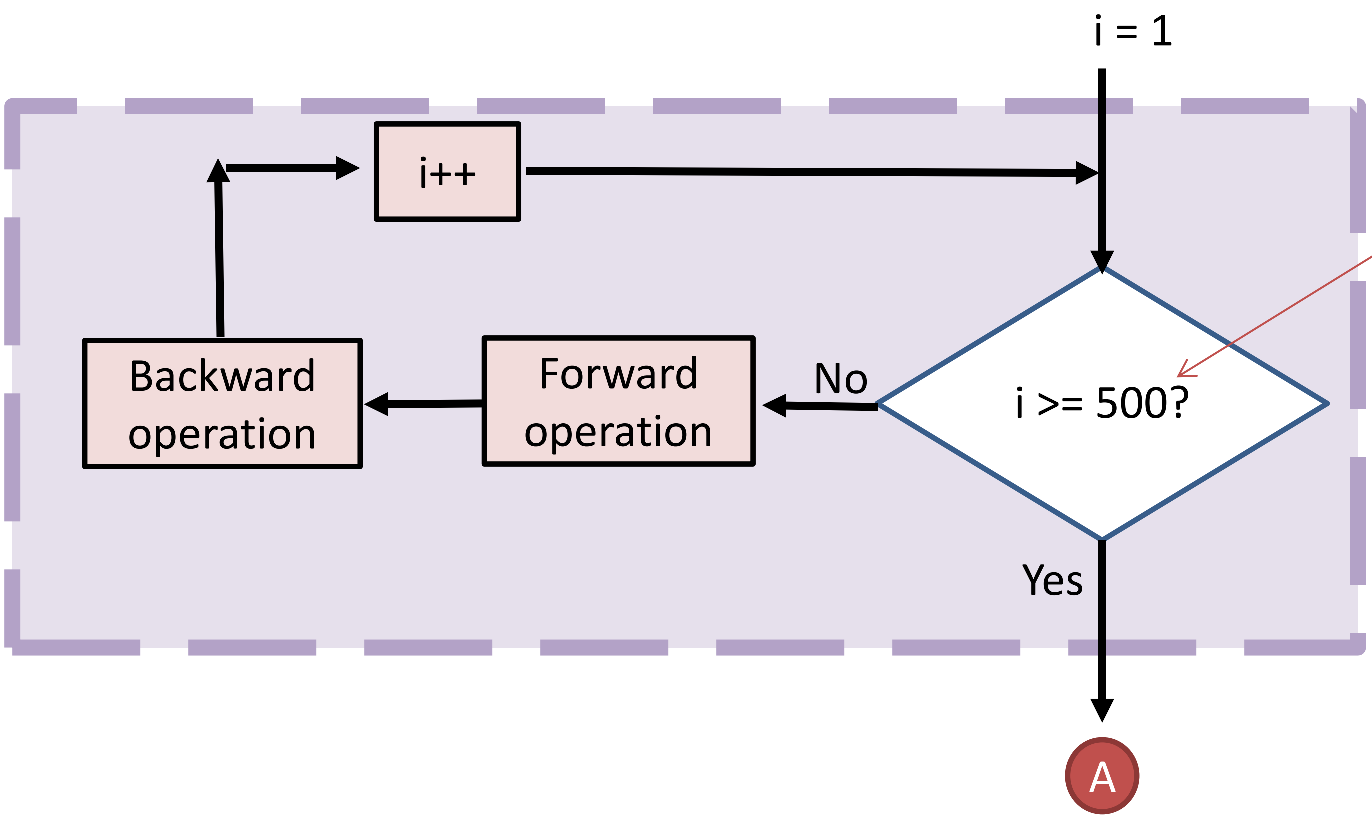
Backpropagation

- A neural network can have millions of parameters.
  - Gradient descent method is the way to compute the gradients efficiently
- Many frameworks (e.g., TensorFlow and PyTorch) can compute the gradients automatically based upon the obtained computational graph

Codes for SLFN

- PyTorch: cs231n 2020 Lecture 6-57
- PyTorch: cs231n 2020 Lecture 6-65
- TensorFlow 2.0+ vs. pre-2.0: cs231n 2020 Lecture 6-91
- TensorFlow: cs231n 2020 Lecture 6-101
- TensorFlow with optimizer: cs231n 2020 Lecture 6-102
- TensorFlow with optimizer & predefined loss: cs231n 2020 Lecture 6-103
- Keras: cs231n 2020 Lecture 6-104
- Keras: cs231n 2020 Lecture 6-106 (help handle the training loop)

The program for learning



This stopping criterion is different with Step 3 of BP learning algorithm