Deep Learning

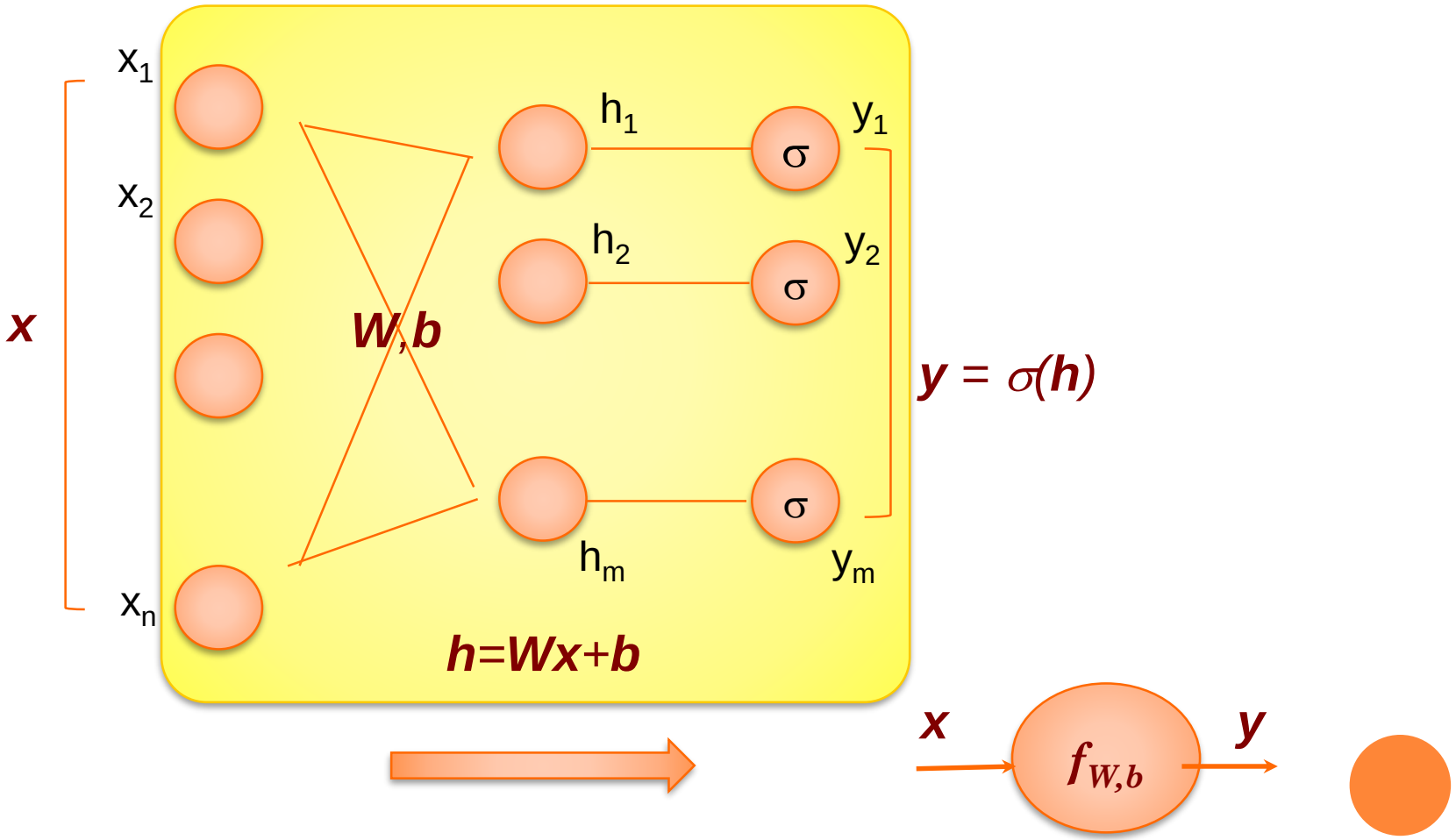
FNN, RNN and CNN

林伯慎 Bor-shen Lin
bslin@cs.ntust.edu.tw
http:www.cs.ntust.edu.tw/~bslin

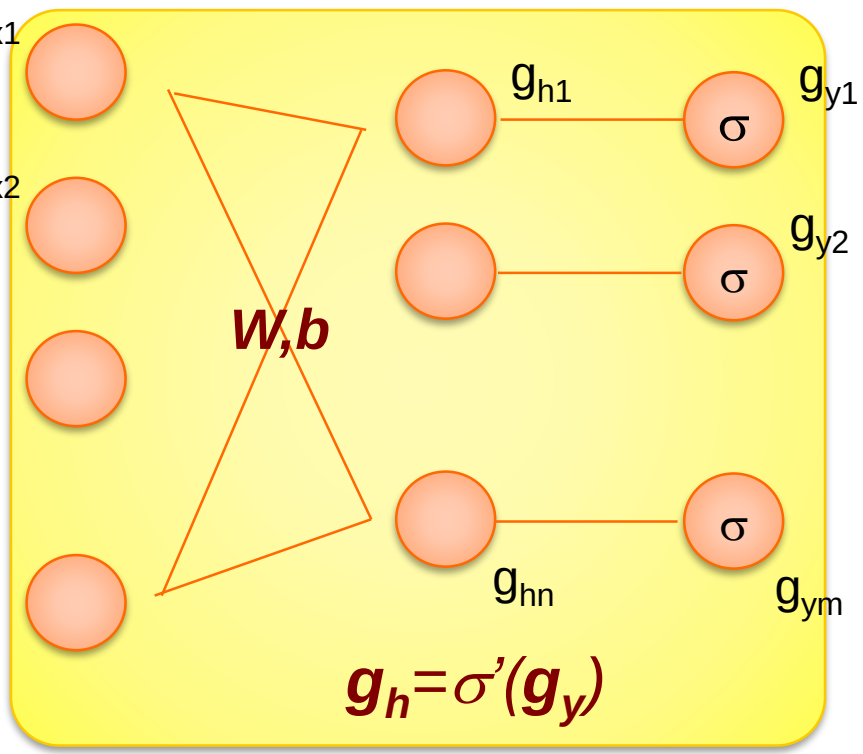
Agenda

- Feedforward Neural Network
- Recurrent Neural Network
- Convolutional Neural Network

Feed Forward for MLP (Layer view)



Learn Backward for MLP



$g_x=W^t g_h$

$\nabla_W L = g_h \cdot x^t$

g_y: *gradients*

$\Delta W = \varepsilon \cdot g_h \cdot x^t$

$\Delta b = \varepsilon \cdot g_h$

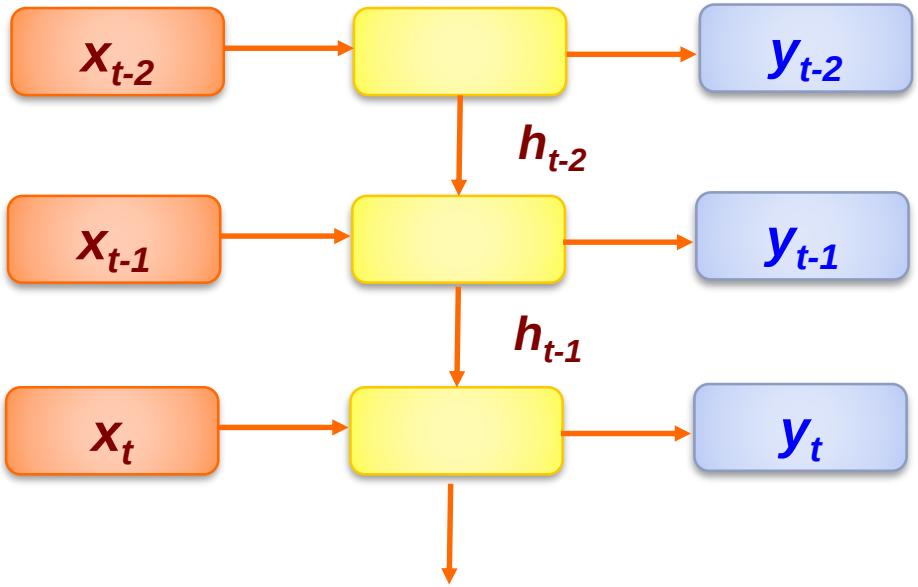
RNN: with Memories

Independent of the history

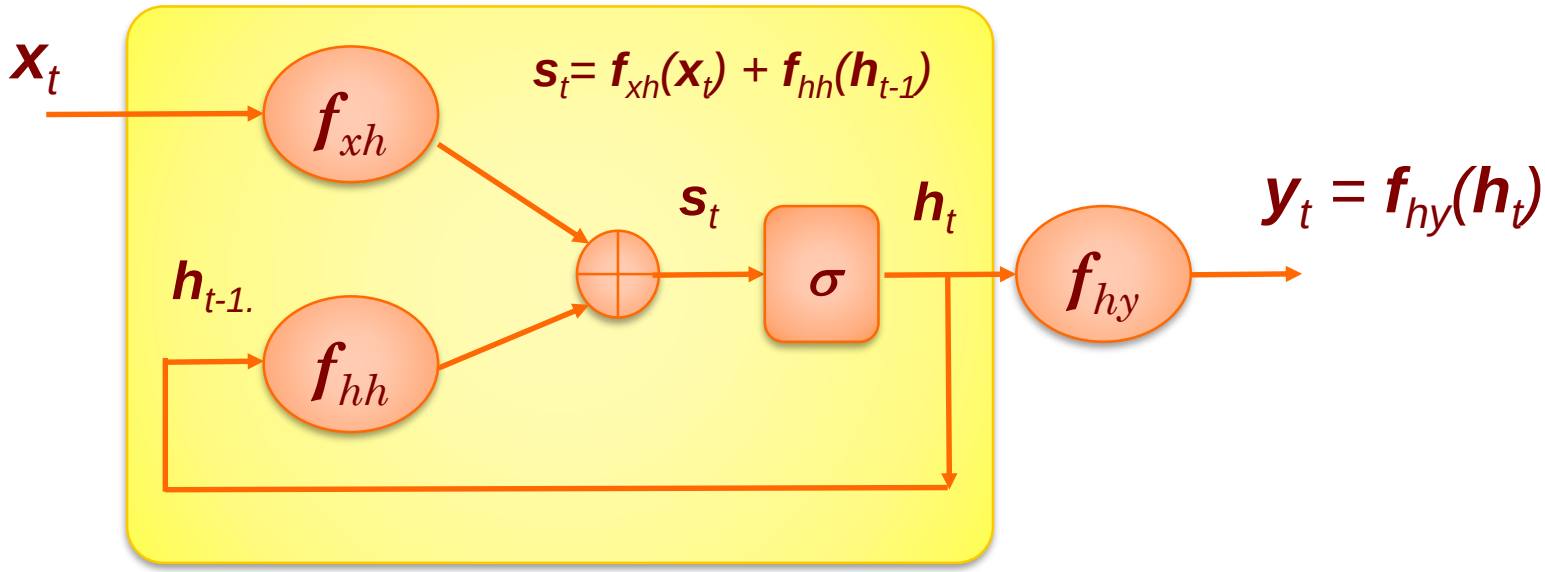
MLP

dependent of the history

RNN



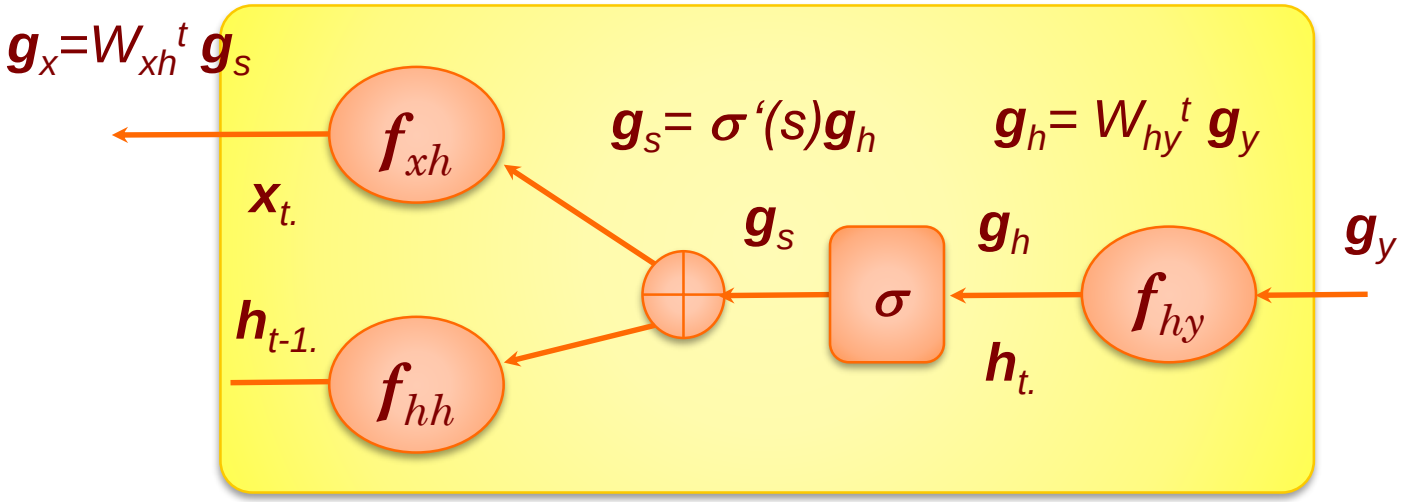
Single RNN Cell (Closed Loop)



- h_t internal state (hidden and invisible)
- f_{xh}, f_{hh} and f_{hy} are the function $f(x) = Wx+b$ *with corresponding* W & b

Gradient Propagation on RNN

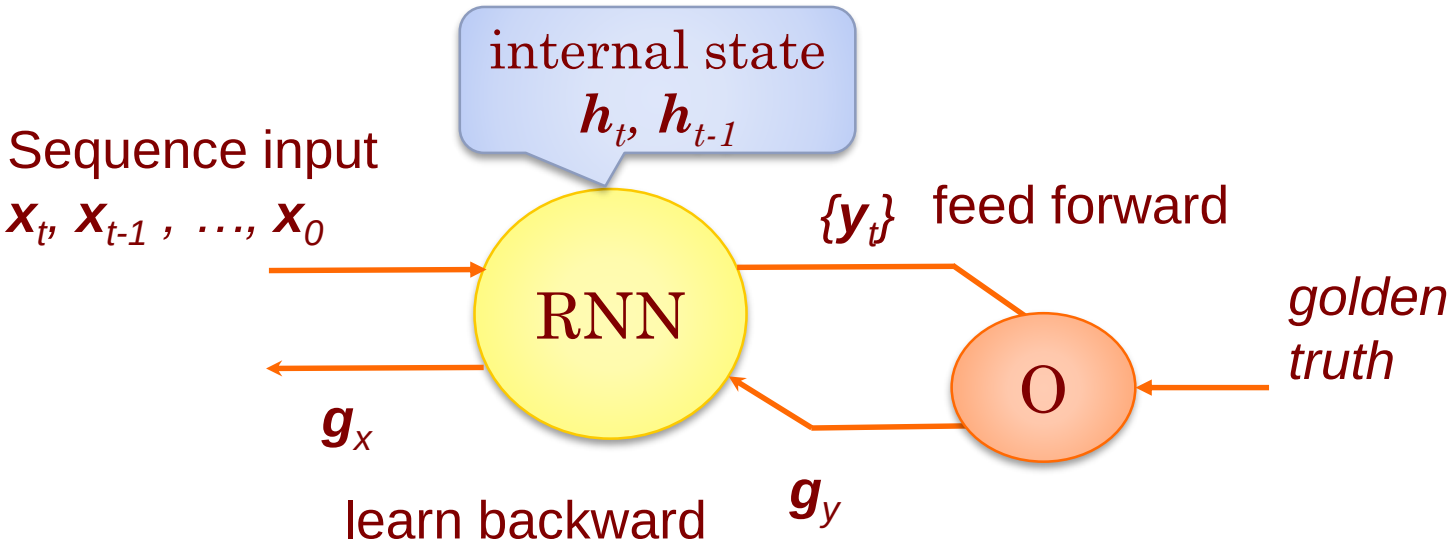
$$\Delta W_{xh} = \varepsilon \cdot g_s \cdot x_t^{\,t}$$
$$\Delta b_{xh} = \varepsilon \cdot g_s$$



$$\Delta W_{hh} = \varepsilon \cdot g_s\, h_{t-1.}^{\,t}$$
$$\Delta b_{hh} = \varepsilon \cdot g_s$$

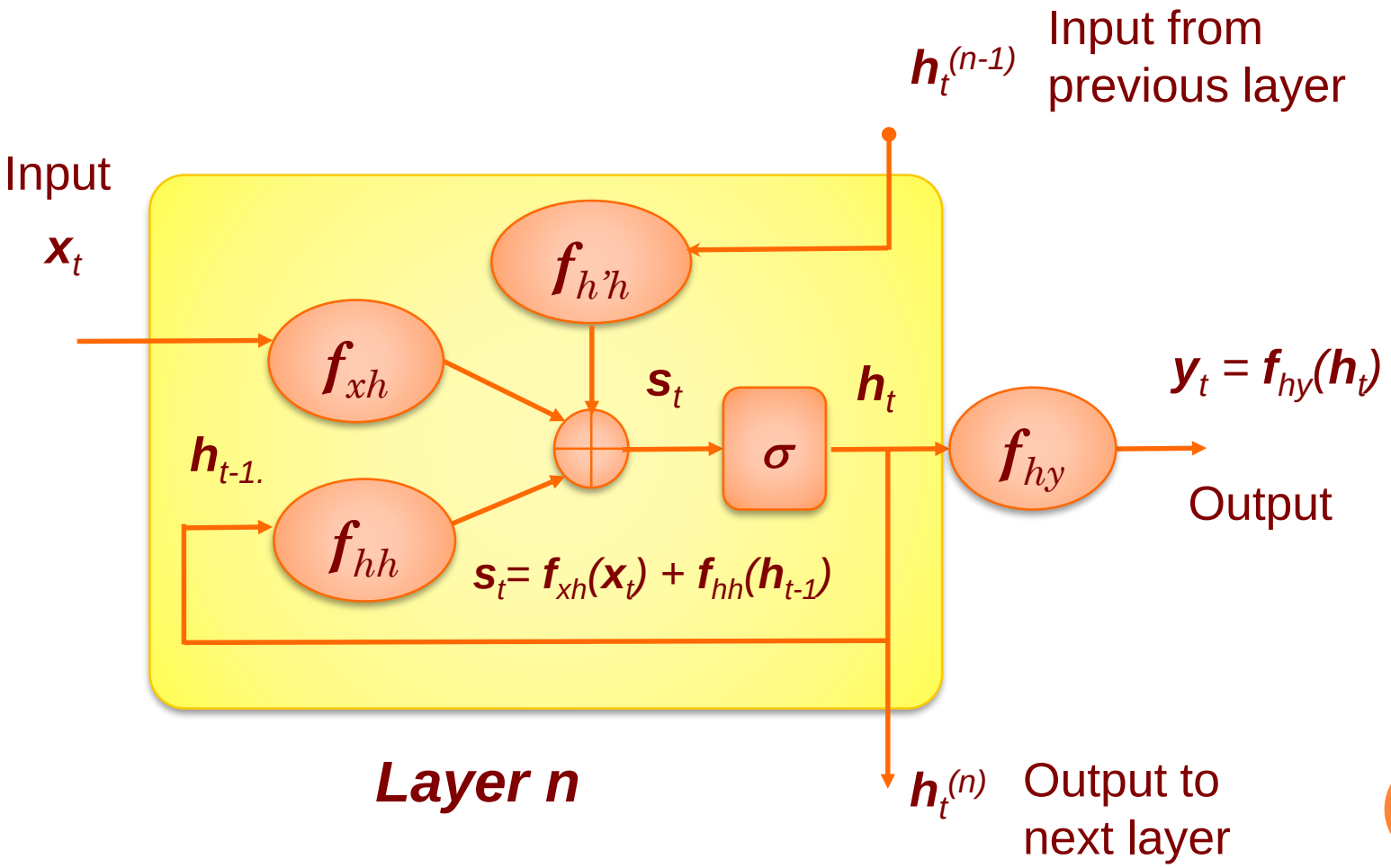
$$\Delta W_{hy} = \varepsilon\, g_y\, h_{t.}^{\,t}$$
$$\Delta b_{hy} = \varepsilon \cdot g_y$$

Single Layer RNN



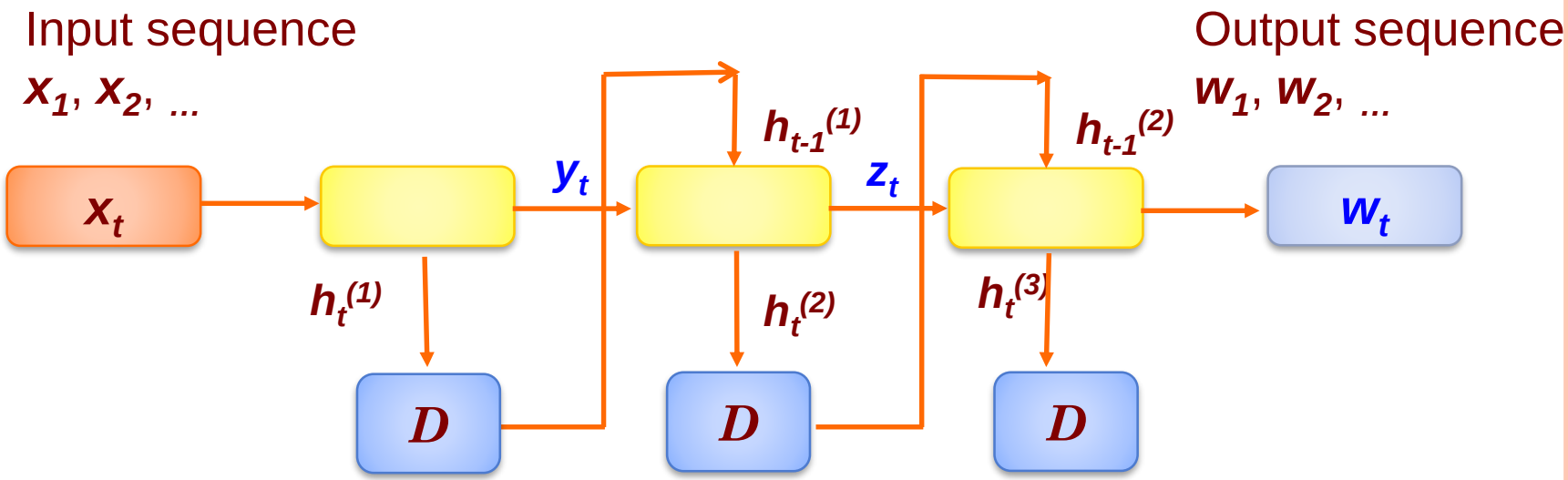
- **RNN**: An RNN Layer with 3 kernel functions: f_{xh}, f_{xh} and f_{hy}
- **O**: Objective function
  - Gradients propagated backward on *the golden truth*
  - *Softmax/square-error* for classification task
  - *square_error* for regression task

RNN Cell for Multiple Layers

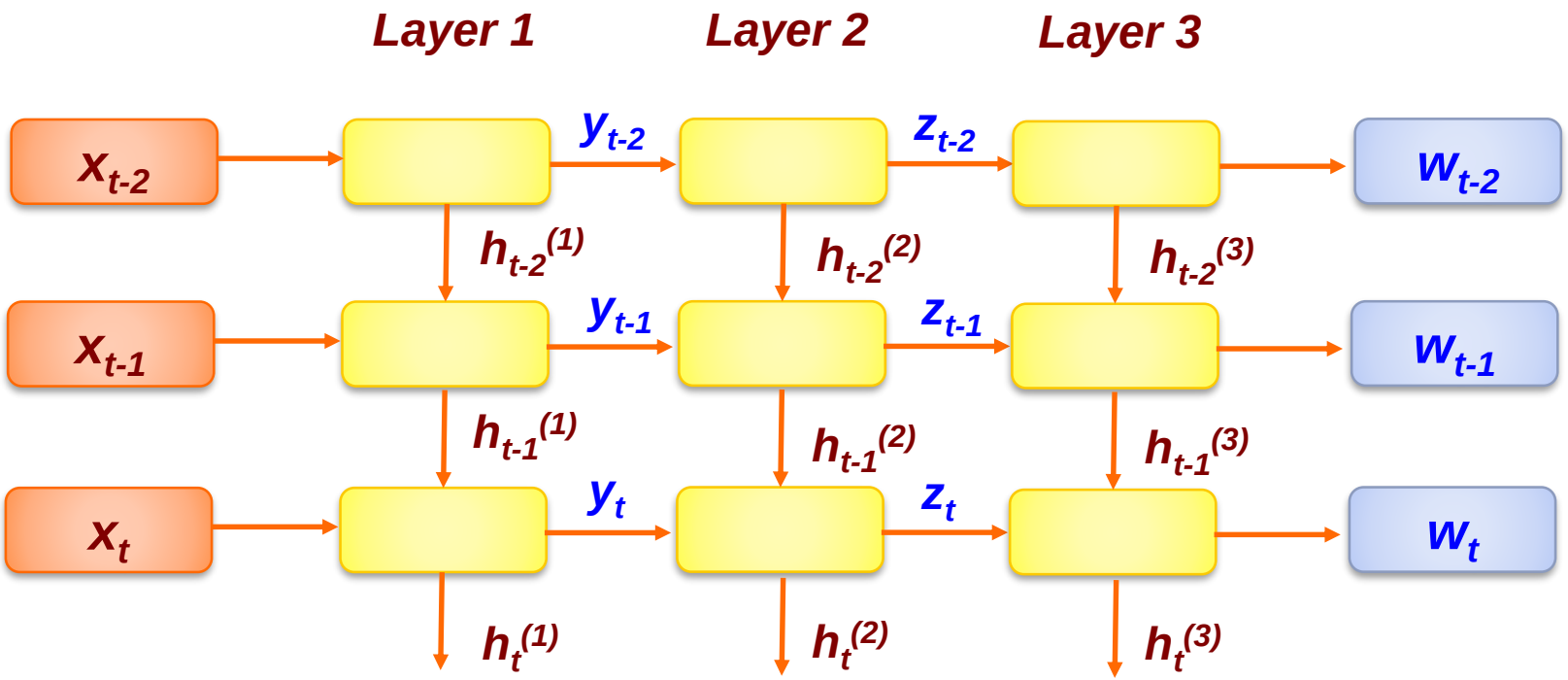


- Internal state is delivered across layers

Multiple RNN Layers (Implementation)

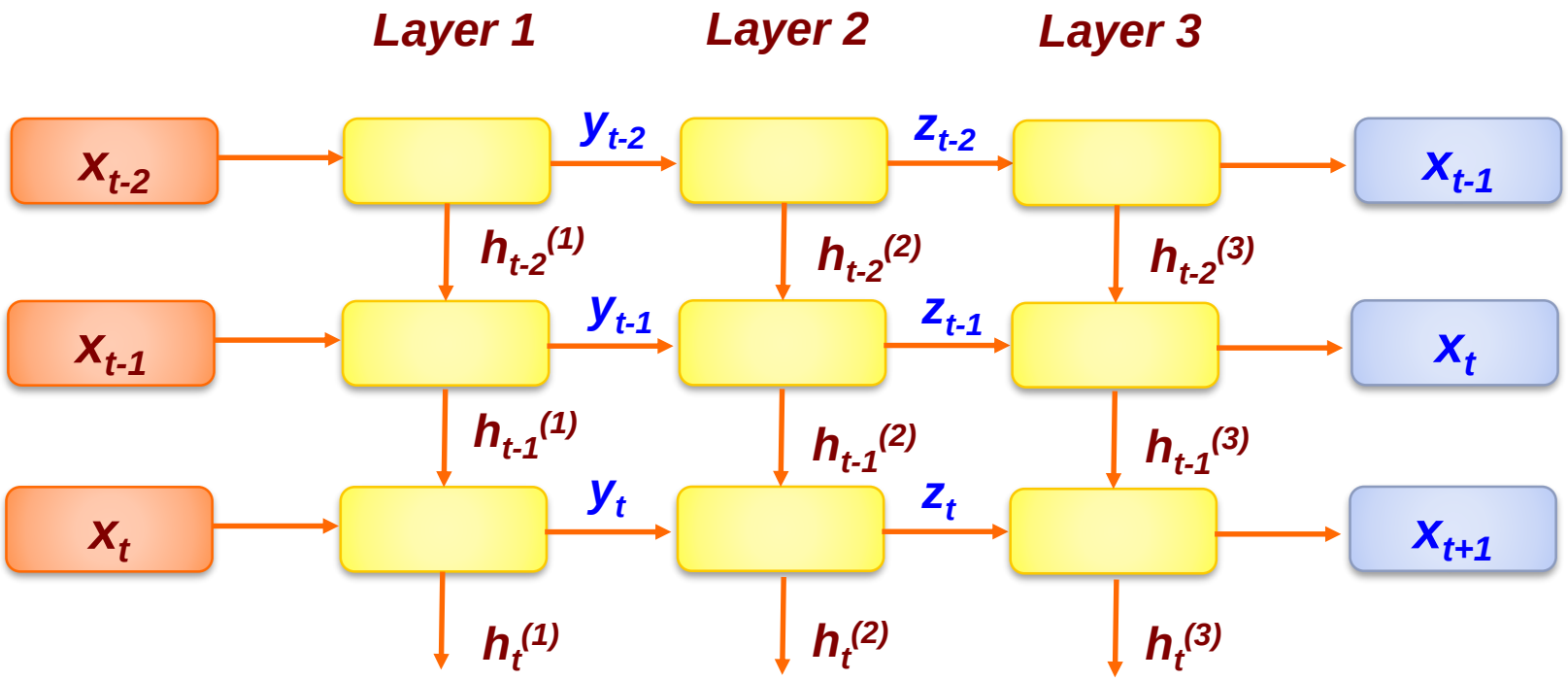


Multiple RNN Layers



- Links of Internal states $h_t^{(i)}$ across layers are abbreviated for simplification

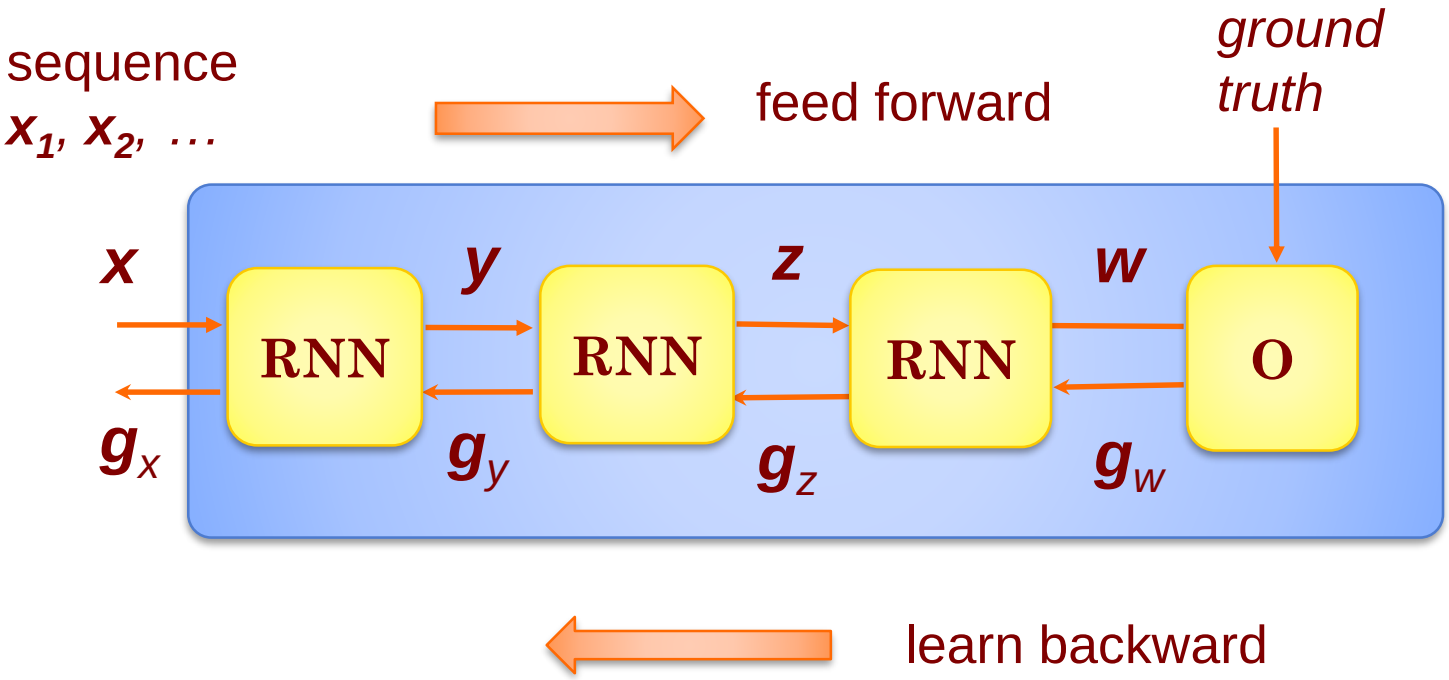
RNN for Sequence Prediction



Self prediction for sequence $x_1, x_2, \ldots$

- A random process for generating the sequences
- c.f. to hidden Markov Model

Network of Stacked RNNs



- Regression task: ground truth is desjred vector for w_T (use square_error as objective function)
- Classification task: ground truth is desired *class label* (use entropy or square_error as objective function)

Example: Character Language Model

- Character prediction or sentence synthesis
- Character sequence "hello\s"

  input: $\{X_t\}$ = [h, e, l, l, o]

  output: $\{Y_t\}$ = [e, l, l, **o**, \s]
- Input symbol: *one-hot encoding*

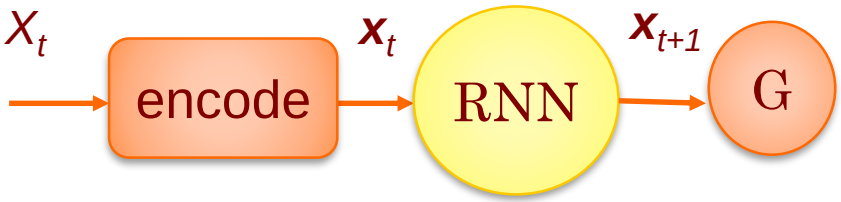
  h → $\mathbf{x}_0$=[1, 0, 0, 0, 0]

  e → $\mathbf{x}_1$= [0, 1, 0, 0, 0]

  l → $\mathbf{x}_2$= [0, 0, 1, 0, 0]

  o → $\mathbf{x}_3$= [0, 0, 0, 1, 0]

  \s → $\mathbf{x}_4$= [0, 0, 0, 0, 1]

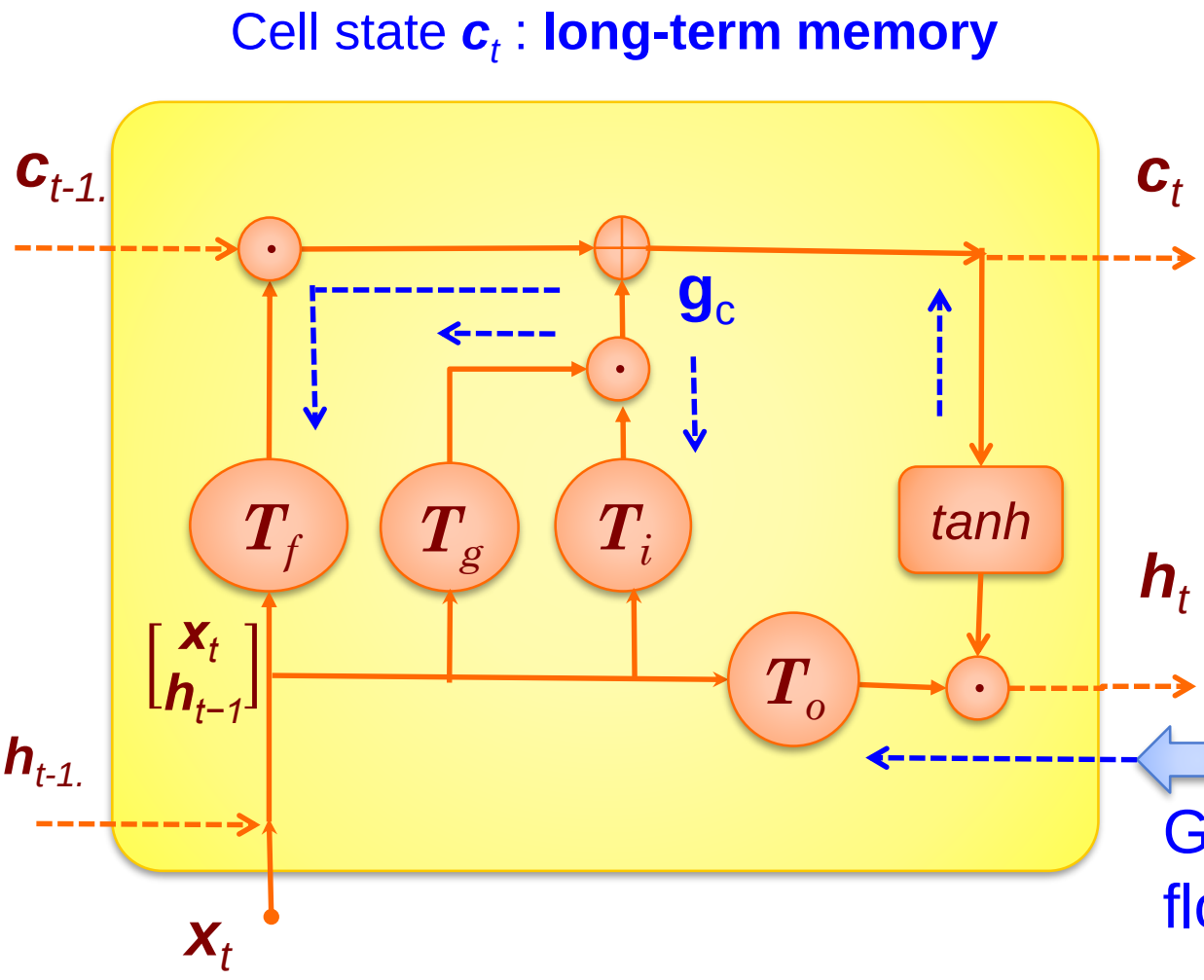


- Output symbols: encoded as class index

Synthesis Example (1-Layer RNN)

{an province in southeastern China, back in 2003. at ease showing me whatever they have," he said. "They'd first feel wary about my request, but when I showed the samples they thought it was interesting." Although China now is the second largest economy after the United States, it is also home to the second largest number of poor people in the world, according to the World Bank. Almost 100 million people lived below the national poverty line of $1 a day in 2012.
Humble endurance
In his book, which is due to be published later in May, Ma gives the back stories of the families he's photographed. The deepest impression the people he profiled has left upon him is their attitude. Powerless, they never demand anything, Ma said. "Are they satisfied? No. But they have no choice but to accept whatever happens to them," he added. "But they try to live with however little they possess." Although humbling, it also frustrated him. "They suffer from silence." Ma recently revisited the first …}

Long Short Term Memory (by Google)



T: contain *W*, *b* and sigmoid/tanh

T_f: forget (*sigmoid*)

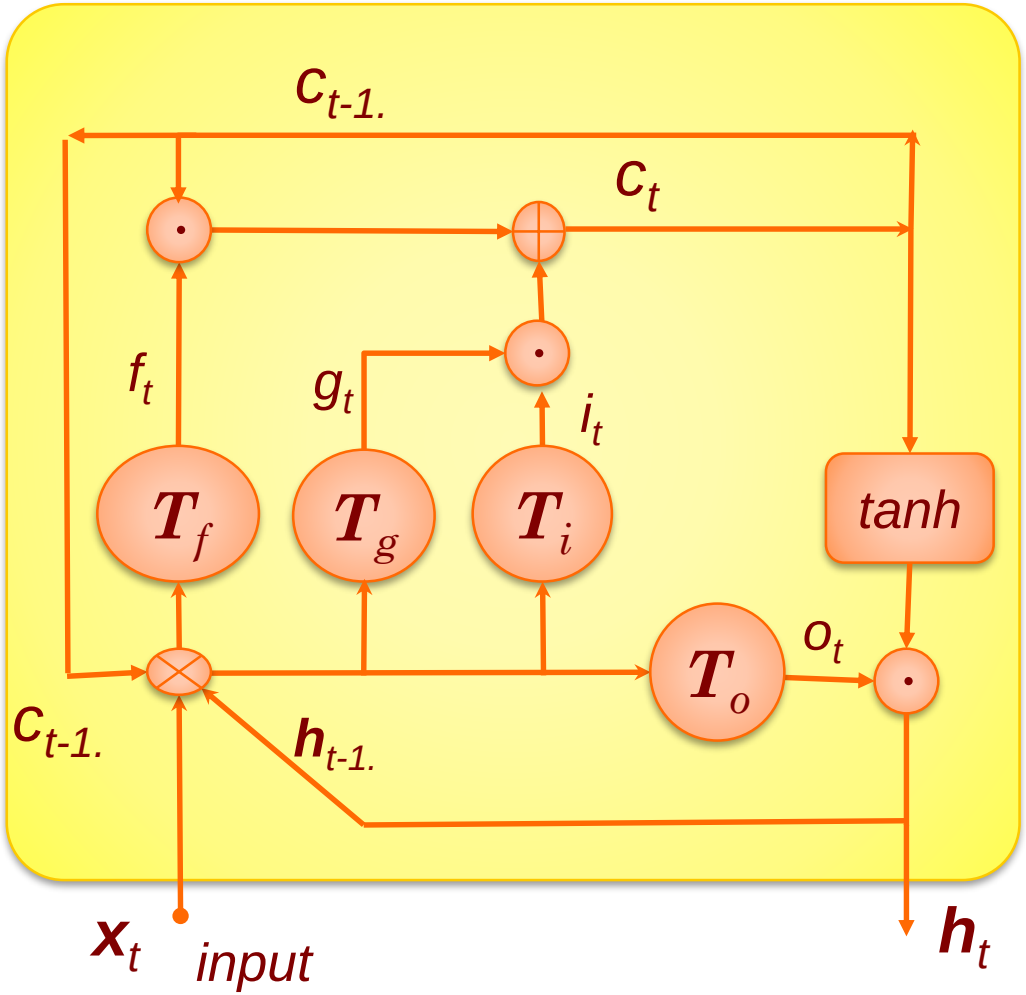
T_i : input (***tanh***)

T_g : gate (*sigmoid*)

T_o : output (*sigmoid*)

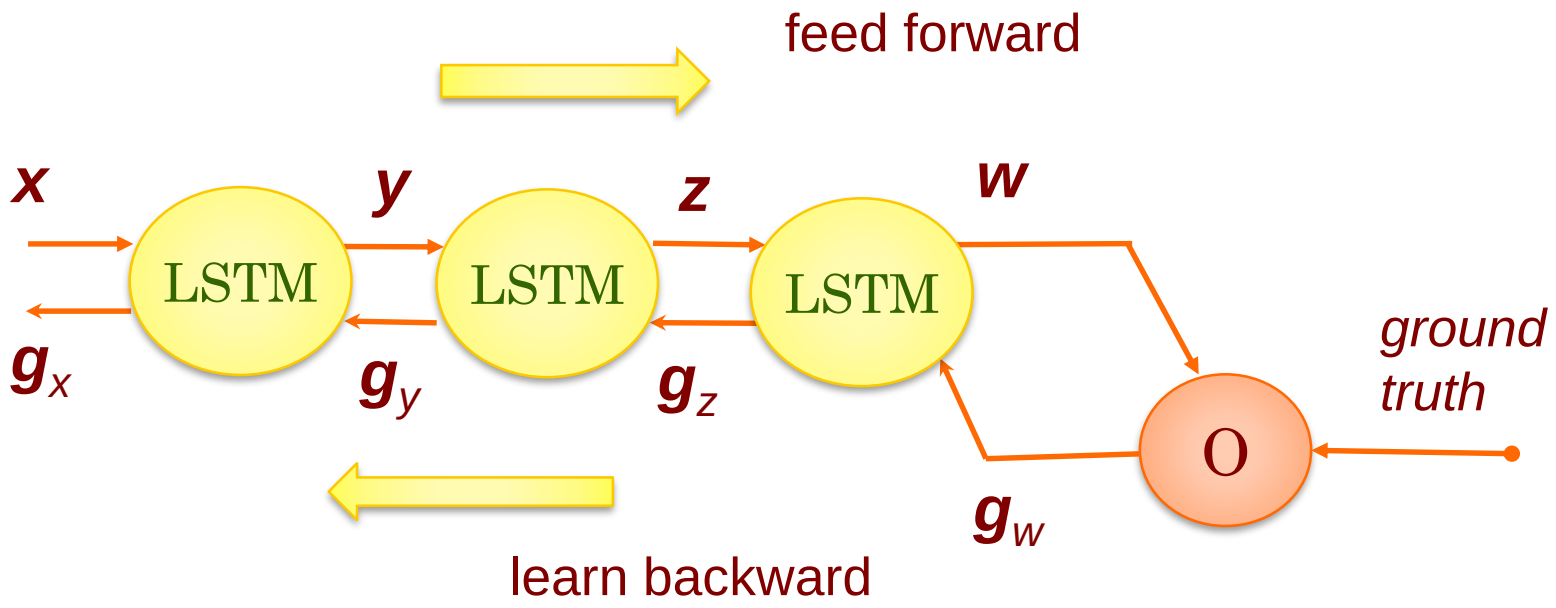
T_f, T_g, and T_o as logic switches

Long Short Term Memory (by Grave)

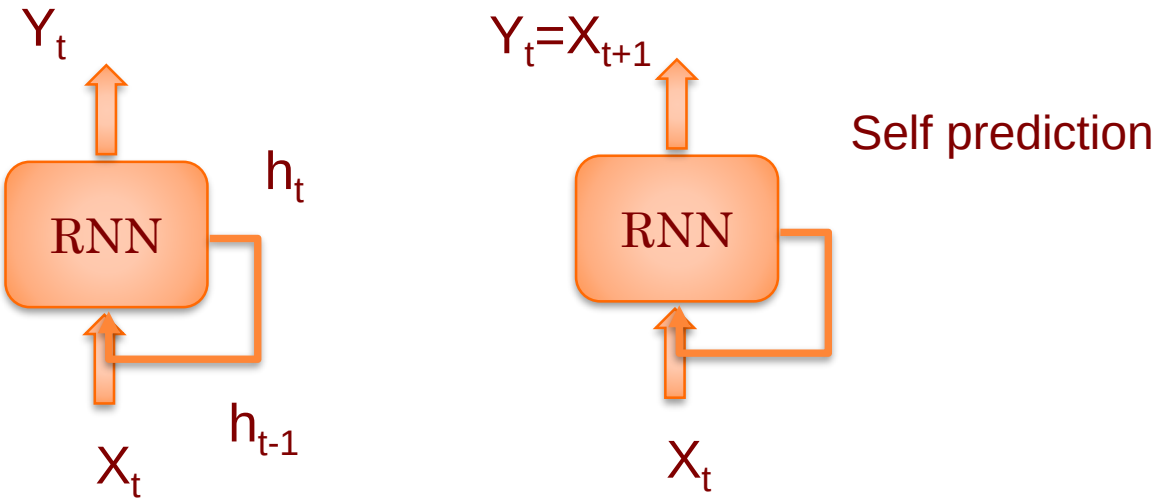


$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$

LSTM with Multiple Layers

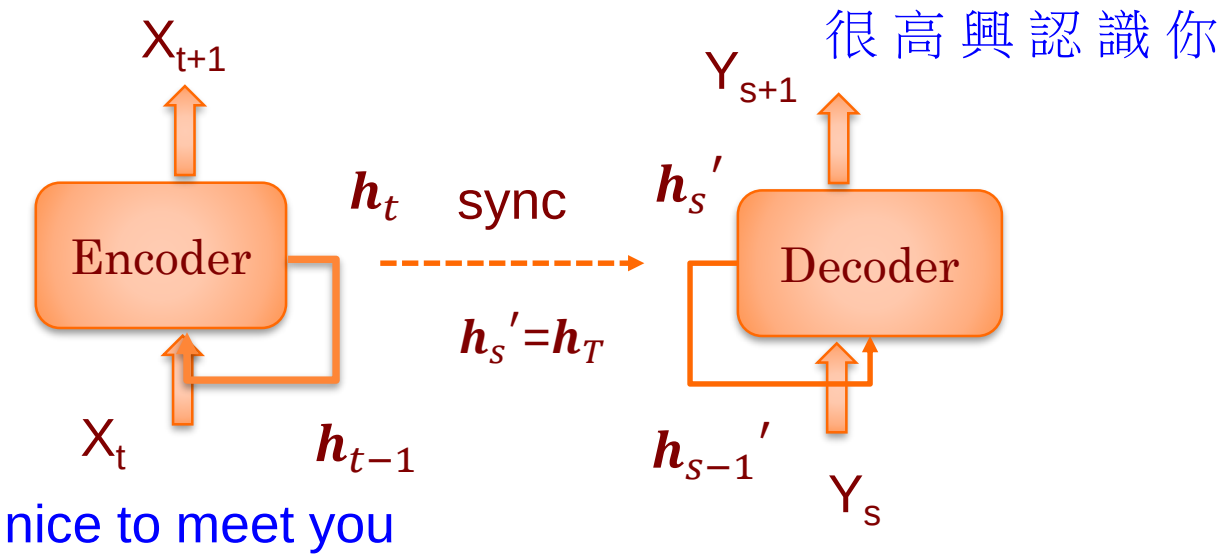


RNN: Model for Sequence Prediction



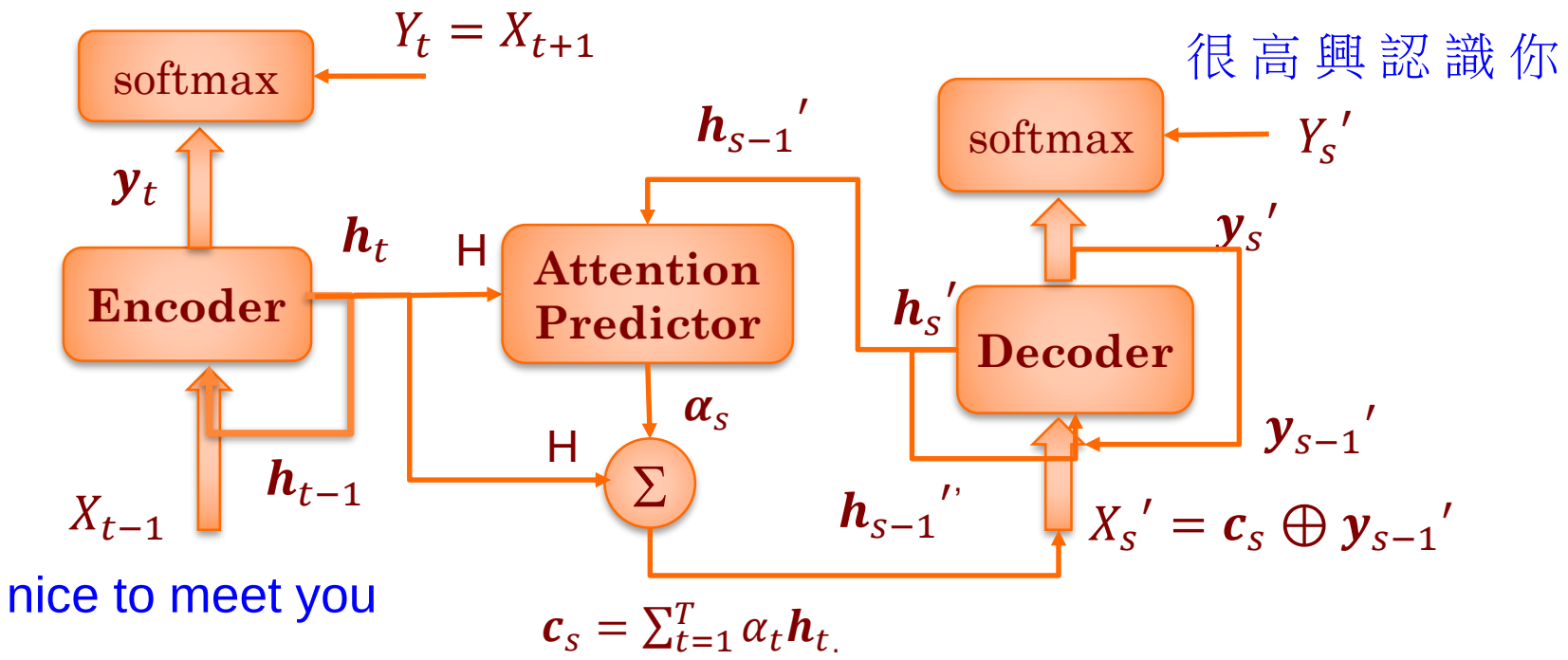
- Prediction sequence Y given X
  - Modeling Y = f(X) between two sequences
  - ~ P(Y | X) if distribution constraint holds
  - PROBLEM: X and Y usually NOT of the same length
- Self prediction for a single sequence
  - e.g. language model

Encoder-Decoder Model



- Input X and output Y could be sequences of different lengths
- Final history state is used for synchronization
- Examples: translation model or dialog model

ATTENTION-BASED ENCODER-DECODER



- H=[$\mathbf{h}_1$, $\mathbf{h}_2$, …, $\mathbf{h}_T$] : all states $\mathbf{h}_t$'s from encoder (bi-direction)
- $\boldsymbol{c}_s = \mathrm{H}\boldsymbol{\alpha}_s = \sum_{t=1}^{T} \alpha_{s,t} \boldsymbol{h}_t$. as context vector
  - Average state weighted with the current attention $\boldsymbol{\alpha}_s$
- $\boldsymbol{\alpha}_s$ attention weights estimated by a learnable network
  - $\alpha_{s,t} = \frac{e^{k(s,t)}}{Z}, Z = \sum_t e^{k(s,t)}, k(s,t) = k(\boldsymbol{h}_{s-1}', \boldsymbol{h}_t)$

Some Applications

- Word segmentation (RNN)
- Parsing/POS tagging (RNN)
- Character/word prediction (auto completion)
- Checking for typing errors (backward prediction)
- Chord prediction (Note-seq → chord-seq)
- Melody synthesis (note-seq self prediction)
- Sequence prediction problems

  $X = X1, X2, ..., Xn$

  $Y = Y1, Y2, ..., Ym$
- $Y = f(X)$
- $p(Y \mid X)$

Discussions

- ANN is a *Regression Network*

  i.e. a network that realize regression function
- Network topologies : different families of functions that constrain the solution space for finding the suboptimal solutions
- You might select different function families to find the optimal parameters, but appropriate selection of function families (~ network topologies) might depends on the data or the tasks
- If prior knowledge about data is known (e.g. linear regression, polynomial, or exponential is appropriate), it is perhaps unnecessary to use *deep* network

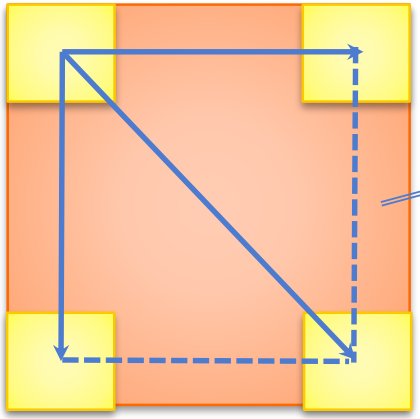
Convolutional Neural Network

- Convolution: filtering operation already used in signal processing (e.g. image/speech)
  - 1D: $y[n] = \sum_k h[k]x[n-k]$
  - 2D: $\mathrm{Y(x,y)} = \sum_{i,j} k(i,j)X(x-i,y-j)$

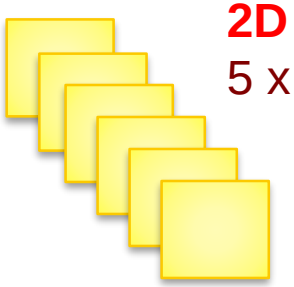
    e.g. Gaussian blur, edge detector
- Fully connected NN : very slow to converge because of too many parameters
- CNN can
  - Reduce # of weights since *the weights of each kernel can be shared* across the output neurons
  - Make the NN to converge fast
  - *Learnable* filter patterns (instead of fixed)

2D Convolution (with 2D kernels)

Input
28 x 28 image



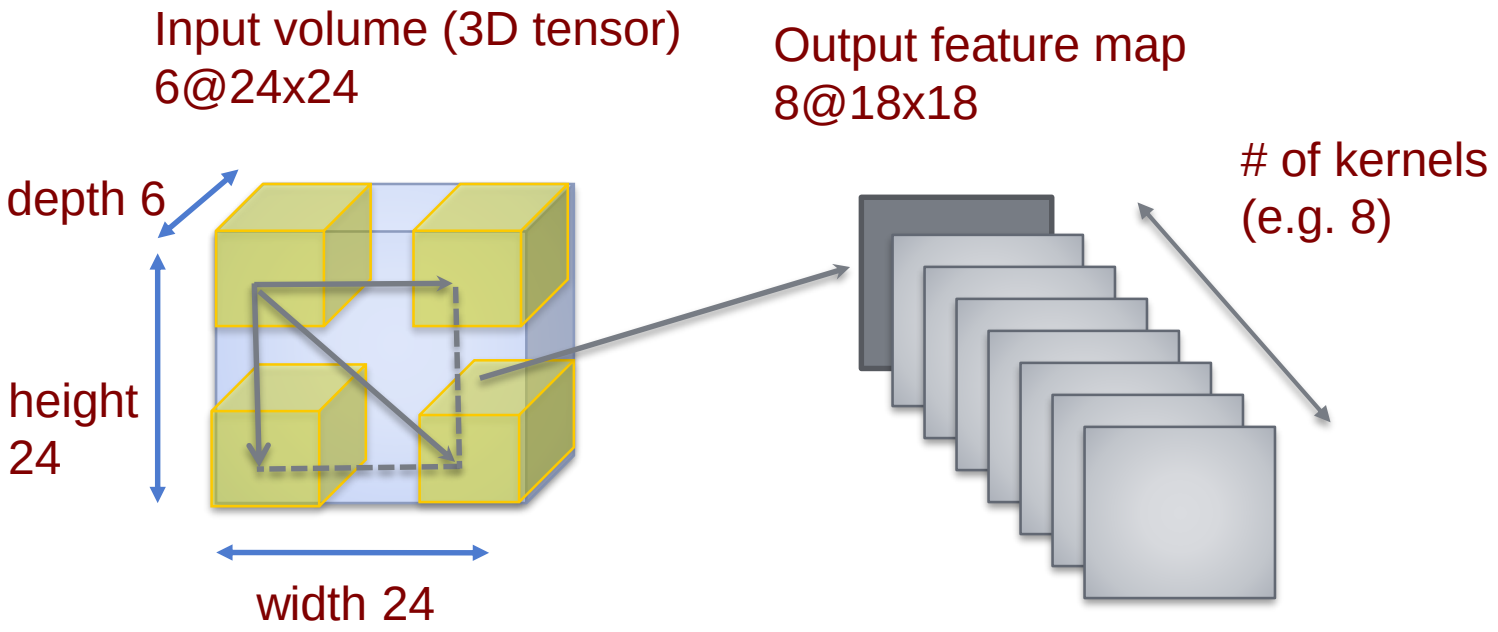
Output feature map
6@24 x 24

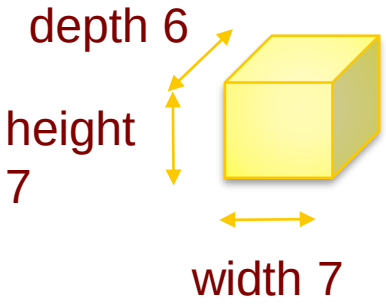


2D kernels
5 x 5

- Performs weighting sum on the input image with a 2D kernel
- Produces a 24 x 24 output pattern for each kernel if stride/shift is 1
- Output is a 3D tensor

2D Convolution (with 3D kernels)

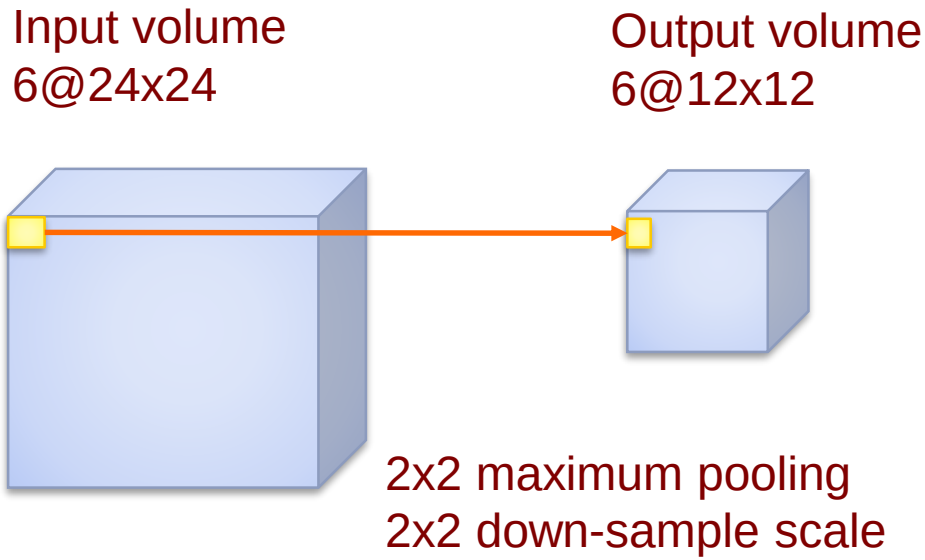




3D kernels
6 x 7 x 7

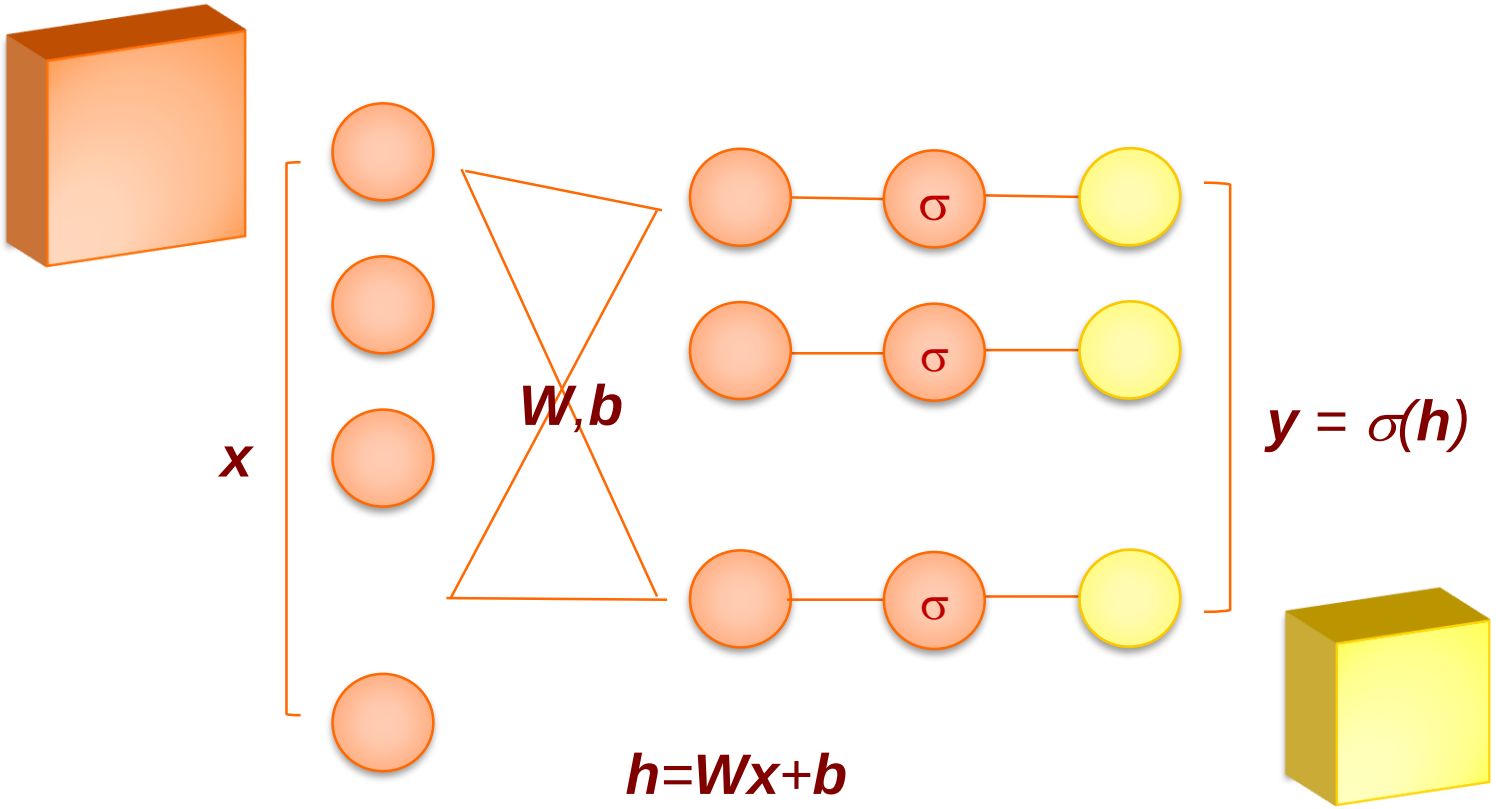
- Performs weighting sum on the input volume with a 3D kernel
- Produces a 18 x 18 pattern for each kernel if stride/shift is 1
- Ouput is 3D tensor

Pooling & Down Sample



- Pooling: pick the maximum or average for each region
- Down-sample: reduce the amount of computations

Logical Mapping for Neurons



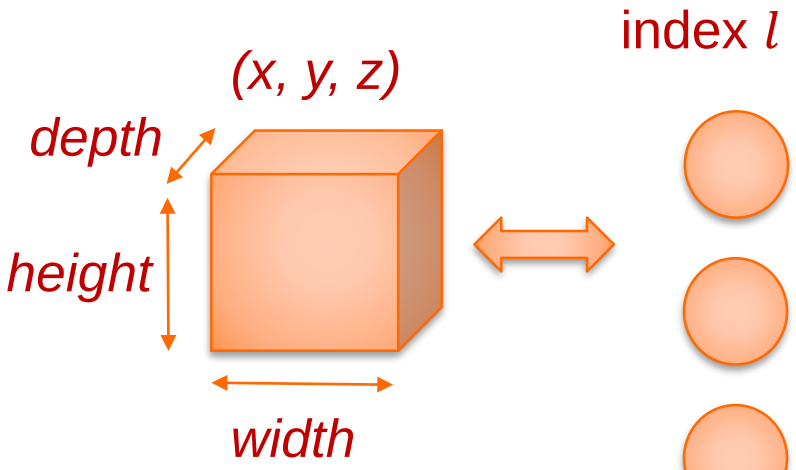
Volume V: 3D Structure

- *width*, *height*, *depth*

  slice = *width* * *height*

  # of neurons

  = *width* * *height* * *depth*



- Coordinate *(x, y, z)*

  $x \in [0, width), y \in [0, height), z \in [0, depth)$

- 3D coordinate converted to 1D index

  $l = loc(x,y,z) = z * slice + y * width + x$

- 1D index converted to 3D coordinate

  $z = l \;/\; slice,\ y = (l \;\%\; slice) \;/\; width;\ x = l \;\%\; width$

Interpretation of CNN

- What *kernel* means?
  - filter pattern (LPF, BPF, HPF)

    e.g. moving average, Gaussian, difference, Sobel,…
  - feature map could be regarded as output of filter banks
  - the kernels in CNN are TRAINABLE instead of manually assigned
- Neuron: pattern detector
  - Inner product to measure the similarity
  - Threshold function (sigmoid): check if some pattern exists
- Meaning of output as 3D feature map v(x, y, z)
  - v as the evidence of the pattern (0~1)
  - (x, y) denotes the location of the pattern
  - z: which pattern(kernel)

Interpretation of CNN

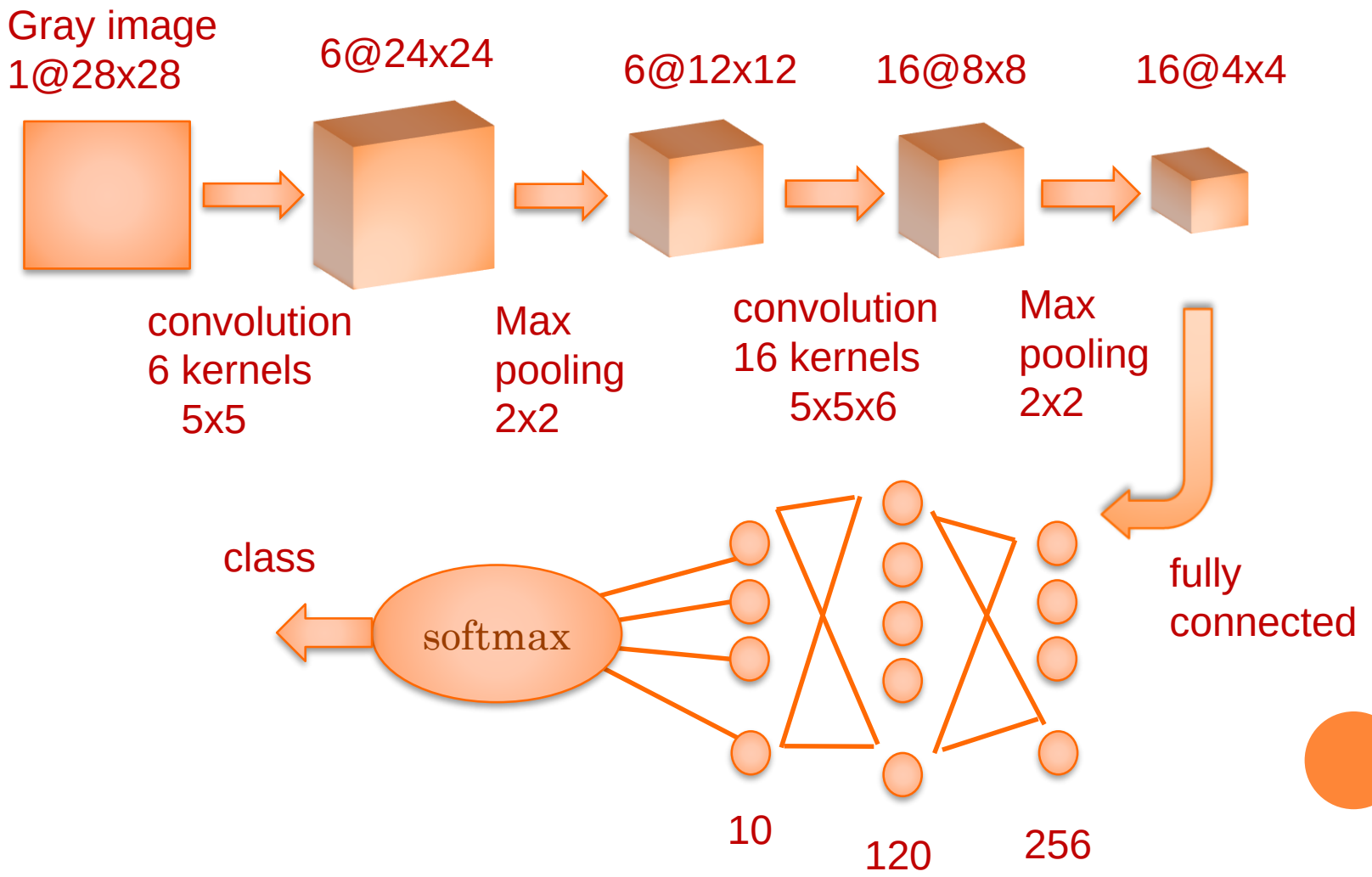
- Multi-layer neurons
  - Concatenation of filters ~ convolution of kernels that corresponds to a larger kernel size (large pattern)
  - Represent hierarchy of semantics for images or NL
- Features of CNN
  - Fewer parameters (weights) that may be shared
  - Fast convergence
  - Detecting local pattern (small kernel size)

    (Fully connected network detects global pattern)

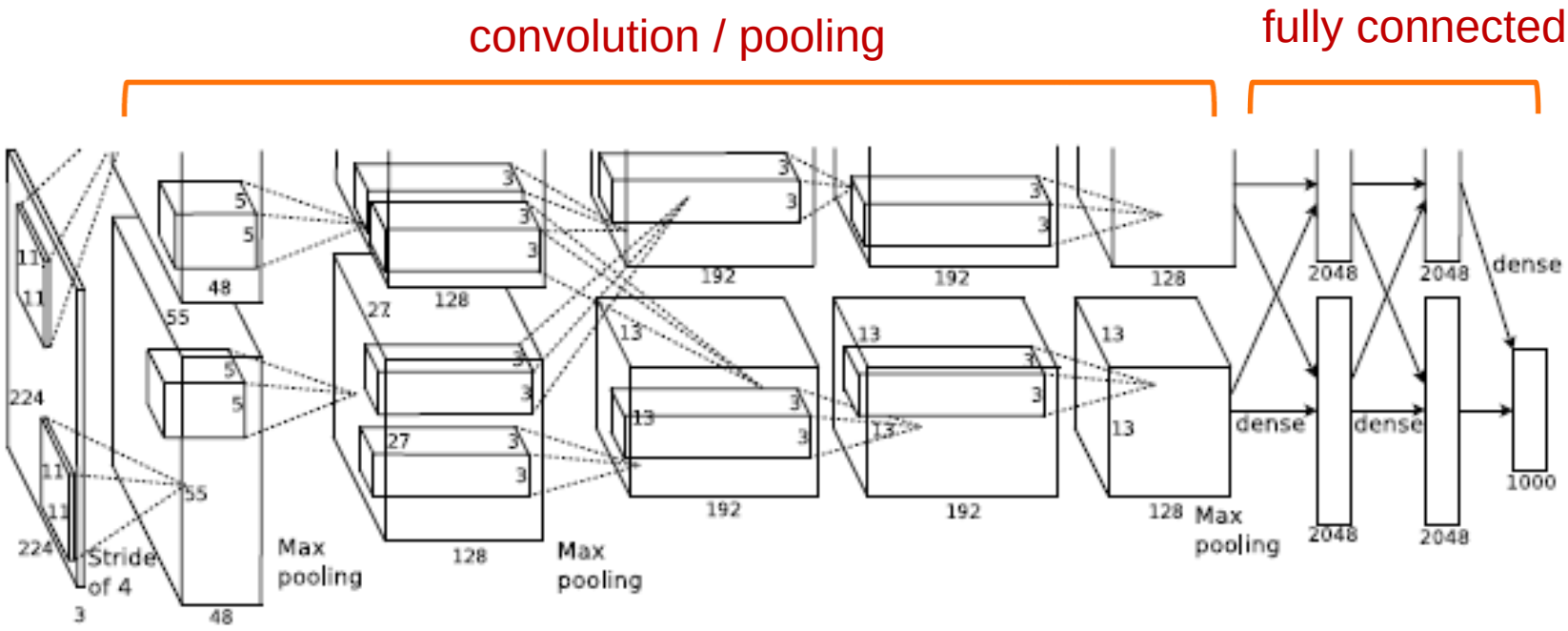
Deep Learning on GPU

- Parallel processing for lots of neurons
  - Every neuron performs simple computation (summation & threshold) in an independent thread
  - Data uploaded to GPU memory
- Processing stages
  1. Feed-forward
     - Summation and threshold function
  2. Propagate-backward
     - Error (gradient) propagation
  3. Learn weights
     - Accumulate the deltas and update in batch
     - Momentum could applied
- Example: *Nvidia Cuda API* / *cuDNN* library

Hand Writing Digit Recognition (LeNet5)

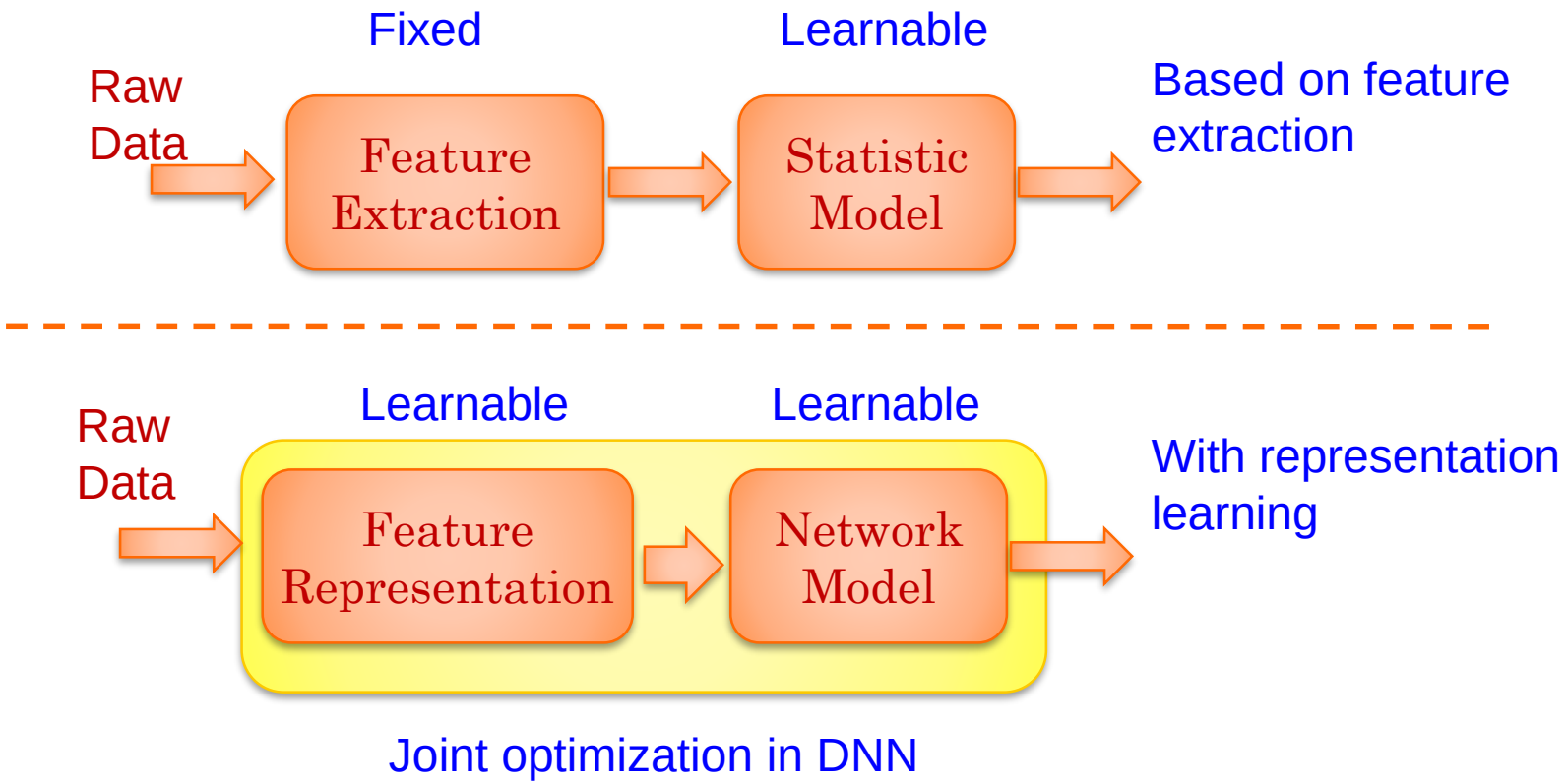


Deep NN for Image Classification



- 2012ImageNet classification
- AlexNet by Krizehevsky
- Bottom level: convolutional layers
- Top level: fully connected layers

Deep Neural Network



- Conventional feature extraction : based on domain knowledge
- Example: Gaussian pyramid for image or MFCC for speech

Deep Neural Network

- Discriminative features could be learned automatically
- Features with high-level semantics could be trained layer by layer
- Parameters can be optimized jointly for both feature learning and model learning
- Obtained features (or patterns) obtained through training may have good discriminative capability without using domain knowledge
- Features can be pre-trained using unsupervised learning approaches such as RBMs or auto-encoders.