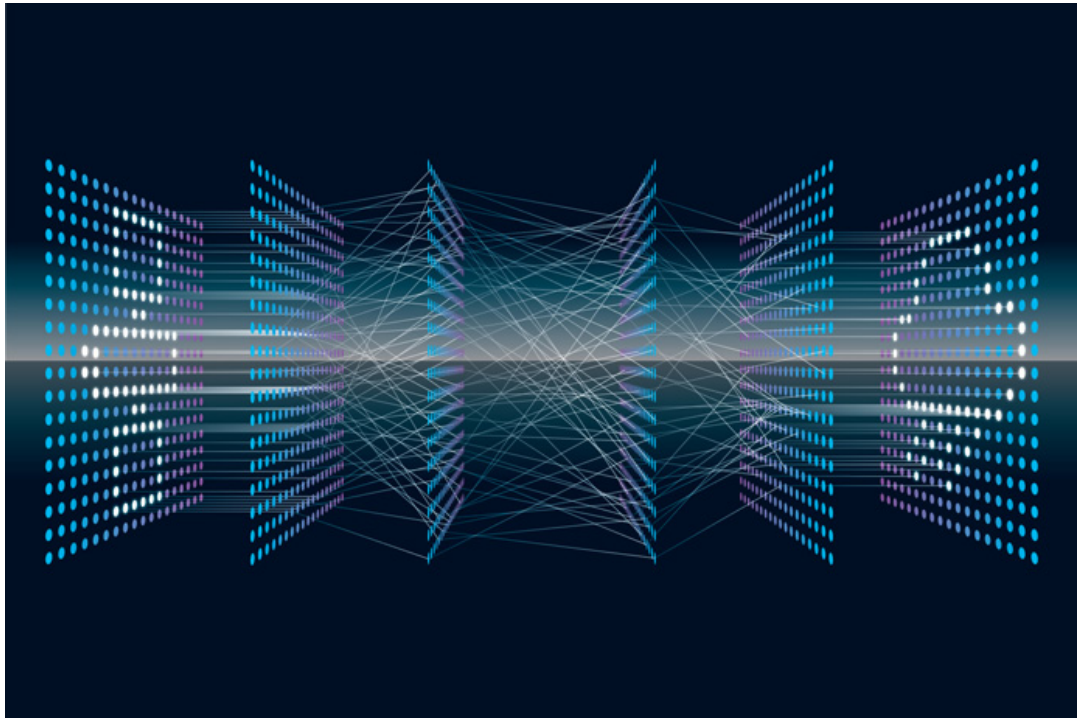
2.160 Identification, Estimation, and Learning

Part 4 Machine Learning and Nonlinear System Modeling

Lecture 21

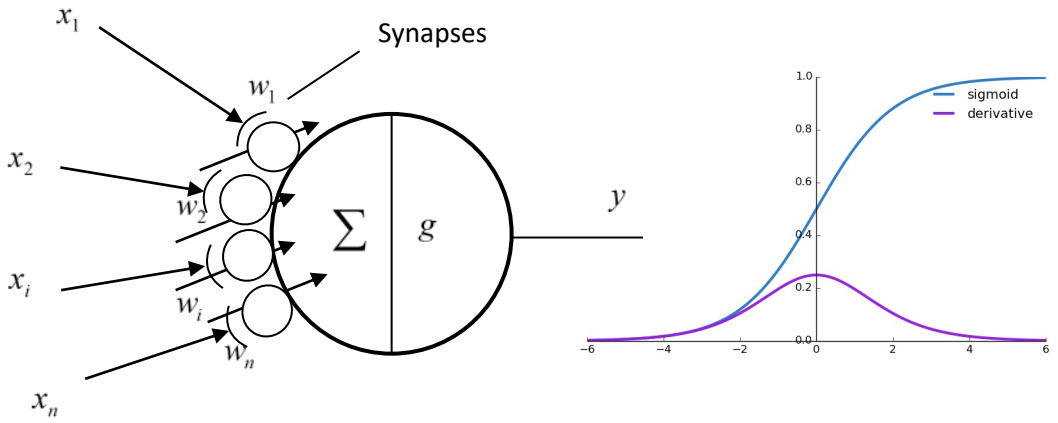
Deep Learning: CNN and RNN

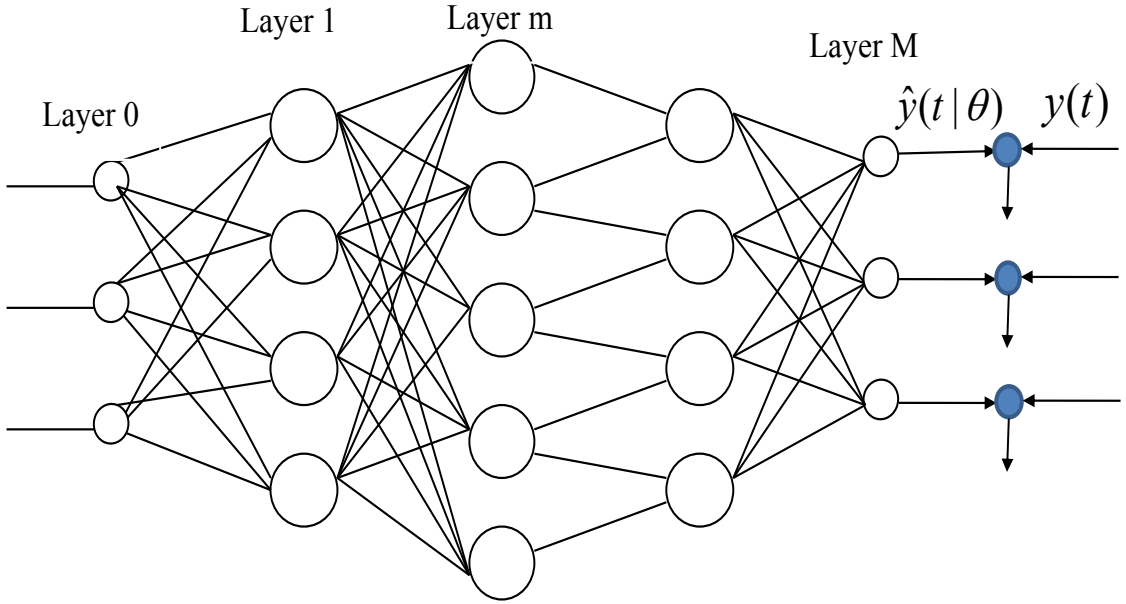


H. Harry Asada
Department of Mechanical Engineering
MIT

Recap

- Artificial Neural Network
  - Basic neural network model
  - Widrow-Hoff stochastic gradient descent method
  - Nonlinear classification : XOR problem
  - Multi-layer neural net
  - The Error Back Propagation Algorithm
  - Sigmoid output function and stability
  - Momentum term for smoothing
  - Local minima
  - Mini-batch training
  - Hyperparameters





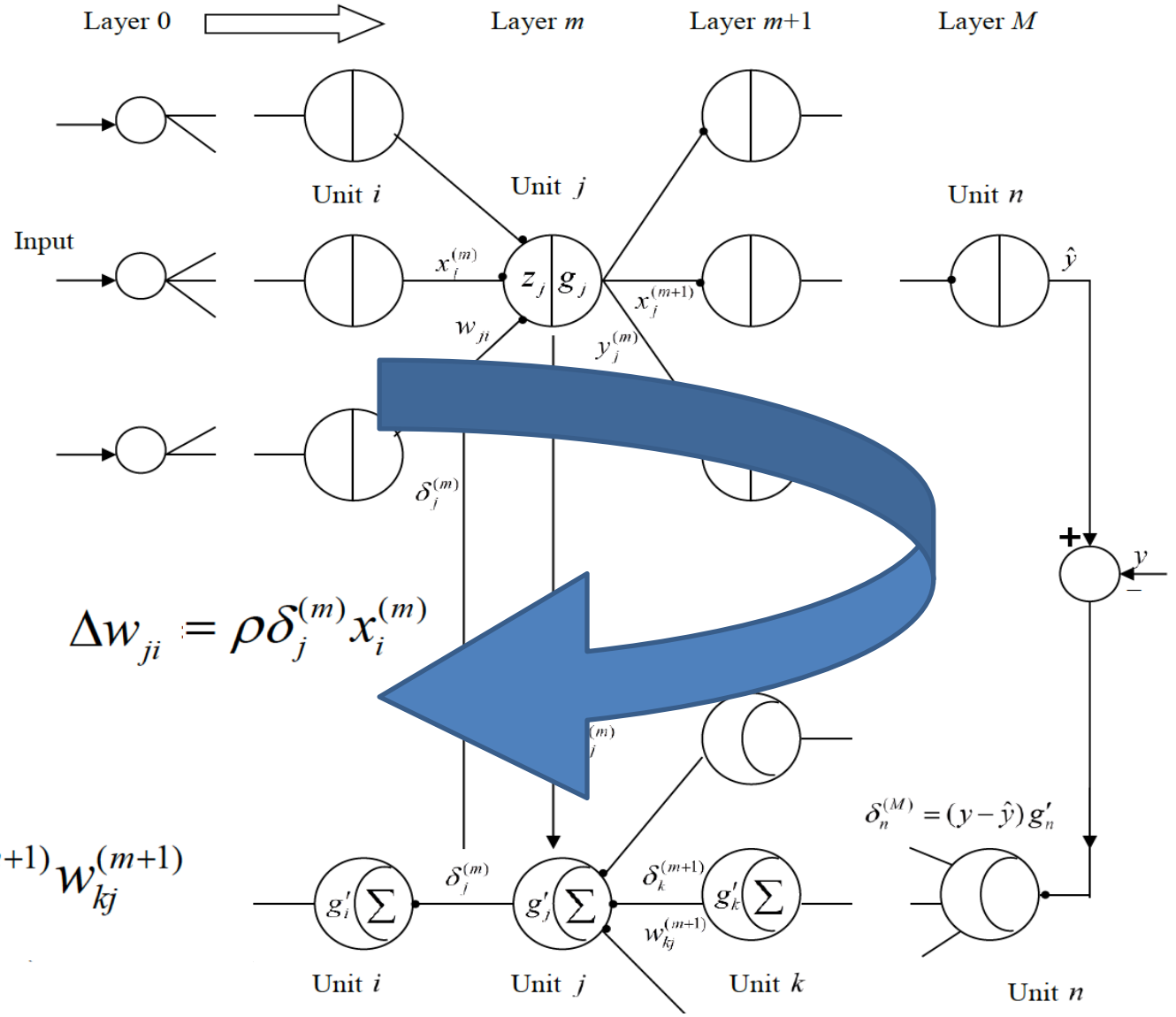
Recap

The Error Back Propagation Algorithm

Forward Computation

$$z_j^{(m)} = \sum_i w_{ji}^{(m)} x_i^{(m)}$$

$$y_j^{(m)} = g_j(z_j^{(m)}) = x_j^{(m+1)}$$



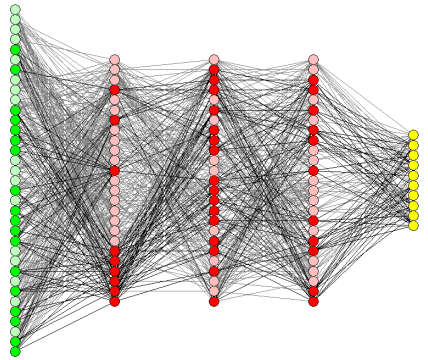
Backward Computation

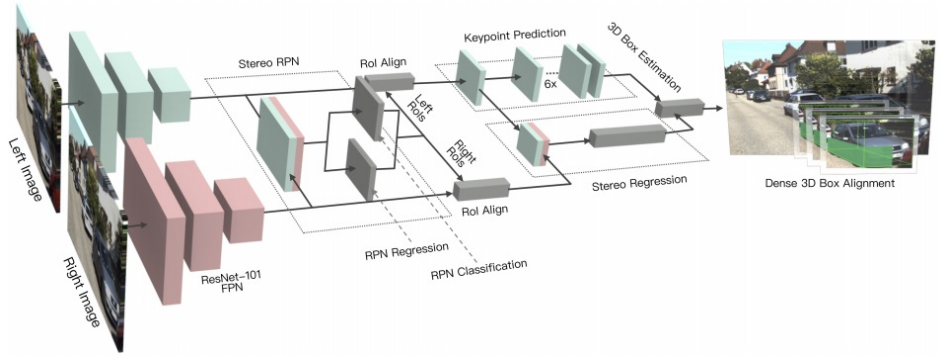
$$\delta_j^{(m)} = g_j'(z_j^{(m)}) \sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)}$$

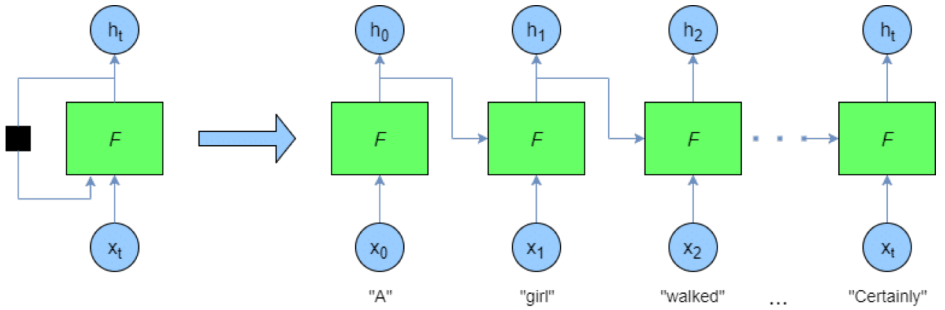
The Error Backpropagation Algorithm
[Wobas 1974, 1994] [Rumelhart, Hinton, & Williams,1986]

Deep Learning

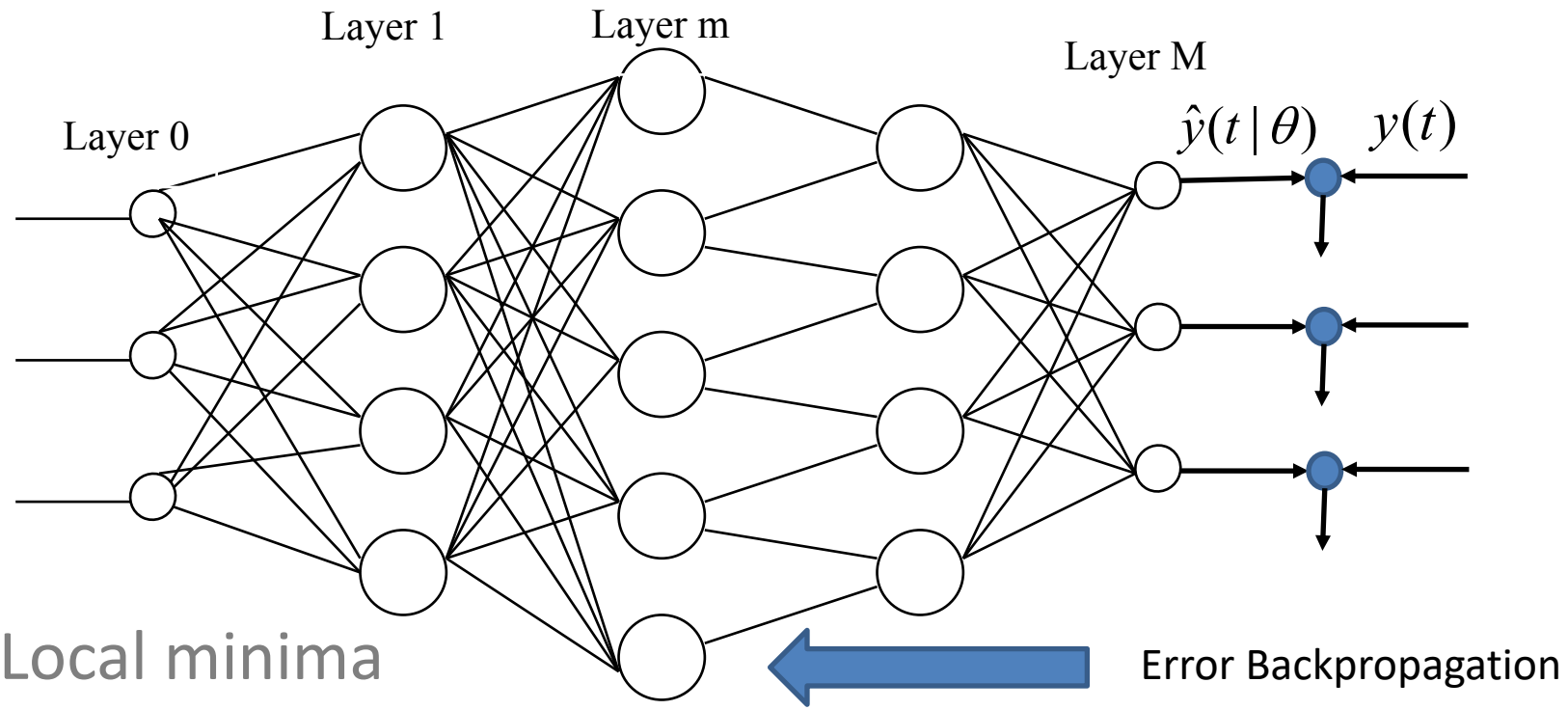
- Issues on training a Multi-Layer Neural Network
  - Over fitting and validation of training
  - Gradient vanishing
- Convolutional Neural Network (CNN)
  - Dropout and focused connection
  - Automatic feature extraction
  - Abstraction via pooling
- Recurrent Neural Network (RNN)
  - Time series data processing
  - Backpropagation through time (BPTT)
  - Long-Short Term Memory (LSTM)





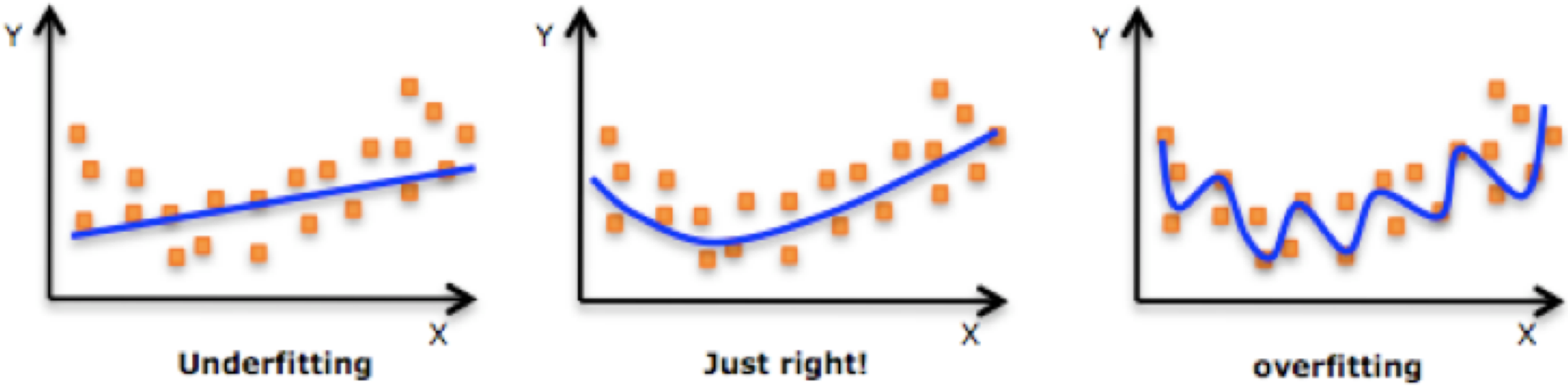


Issues of Multi-Layer NN and Error Back Propagation Algorithm



- Local minima
- Slow convergence
- Over fitting
- Gradient vanishing
- Hyperparameter tuning: how many layers, how many hidden units?

Over-fitting



- Overfitting is a fundamental problem in all areas of system identification and data-driven system modeling;
- Theory: Akaike's Information Criterion (AIC) tells us a reasonable system order, given a dataset (N>>1);
- No other theoretical method is available: Empirical judgement of the engineer

Validation of Learned (Identified) Model

Neural Network can be trained for a set of training data, but it may not work for new data. How can we validate a trained N/N whether it works well for data not involved in the training data?

$$\boldsymbol{X} = \begin{pmatrix} x_1(1) & \cdots & x_1(N) \\ \vdots & \cdots & \vdots \\ x_n(1) & \cdots & x_n(N) \end{pmatrix}$$

$$\boldsymbol{Y} = \begin{pmatrix} y(1) \\ \vdots \\ y(N) \end{pmatrix}$$

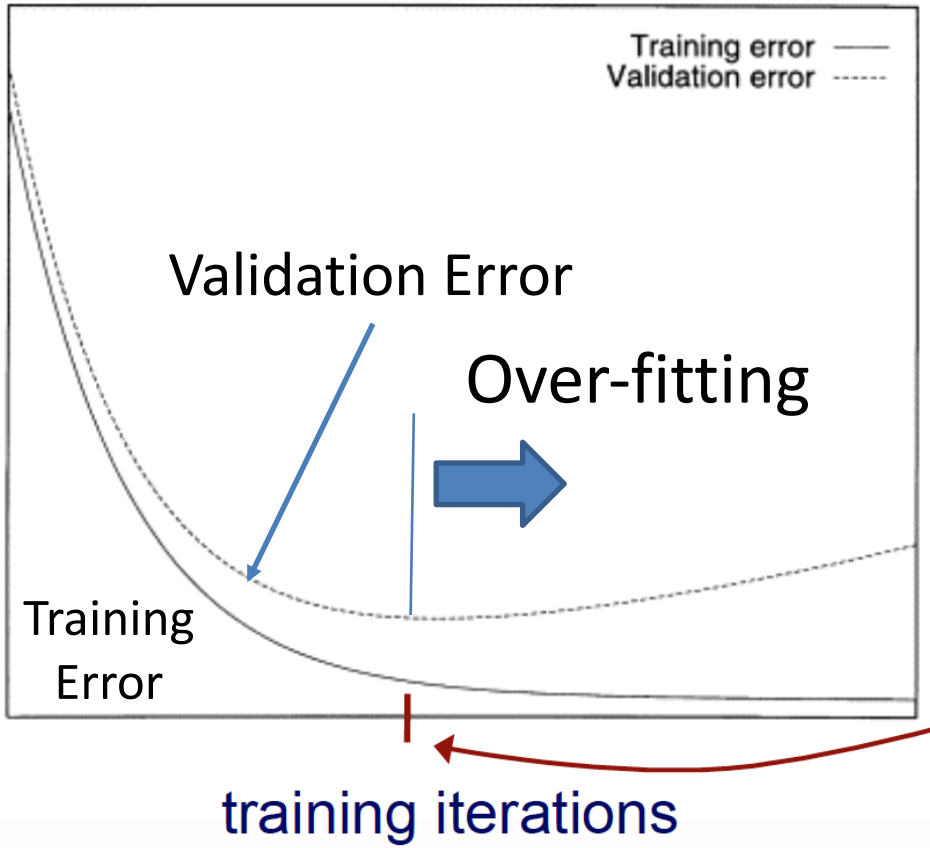
COP #4
Please try Validation Tests

Validation

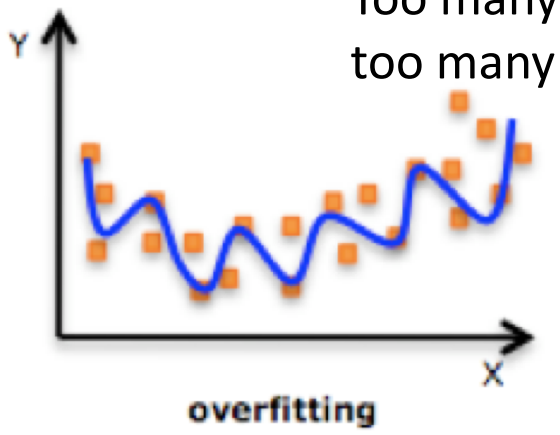
$$\{(u(i), y(i)) \mid i = 1, \cdots, N\}$$ Training Data

$$\{(u(i), y(i)) \mid i = N+1, \cdots, N+N_{Valid}\}$$ Validation Data

Validation
Test Error
&
Training Error



Too many units;
too many hidden units

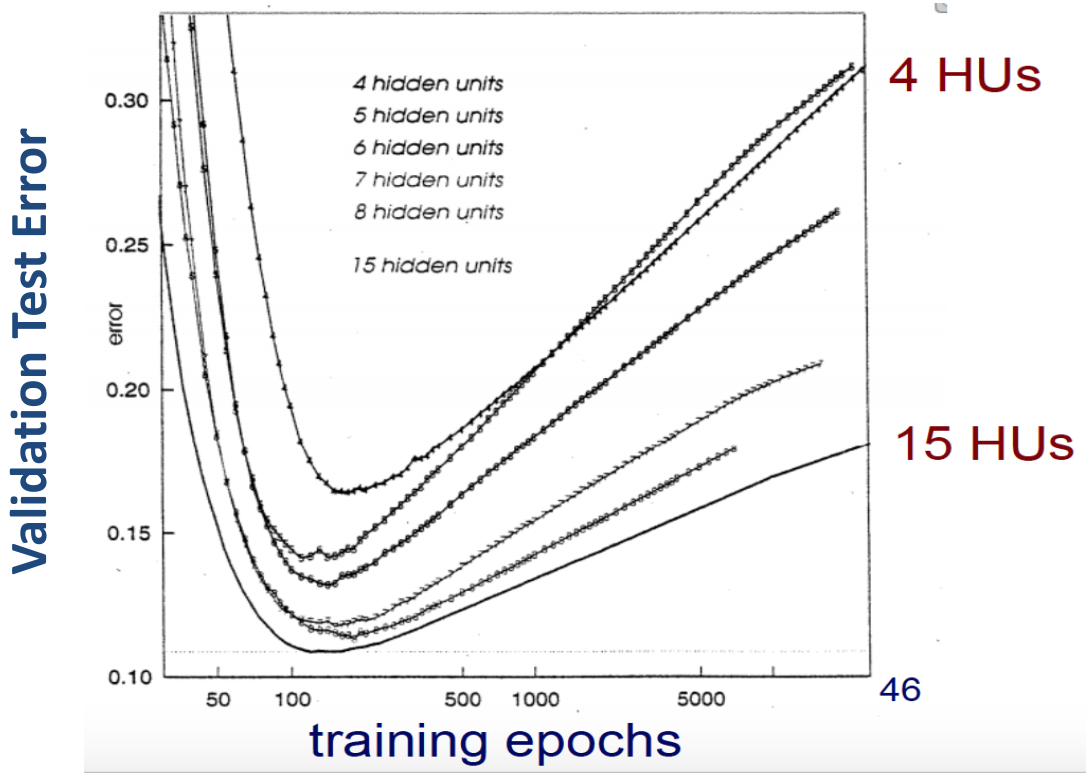


stop training here

37

How Many Hidden Units to Use

- Conventional wisdom in the early days of neural nets: prefer small networks because fewer parameters (i.e. weights & biases) will be less likely to over-fit
  - Asymptotic variance analysis



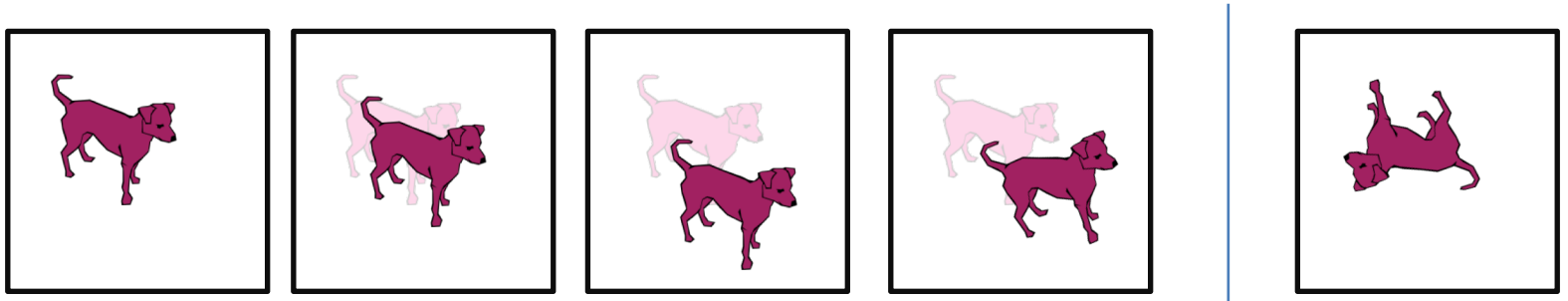
- Somewhat more recent wisdom: if early stopping is used, larger networks often behave as if they have fewer "effective" hidden units, and find better solutions

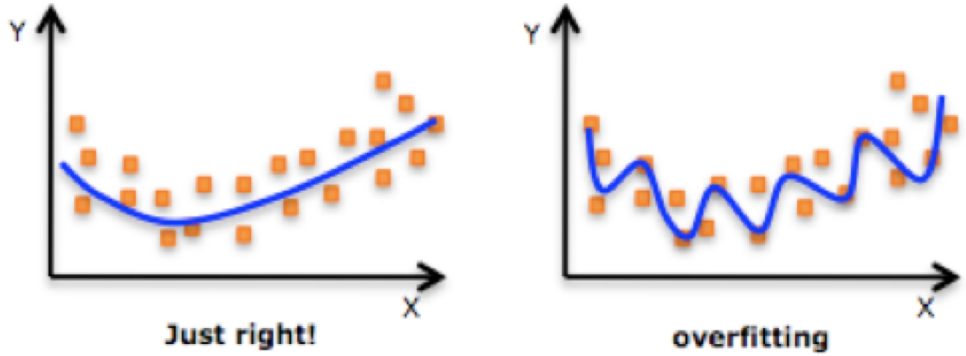
Tips

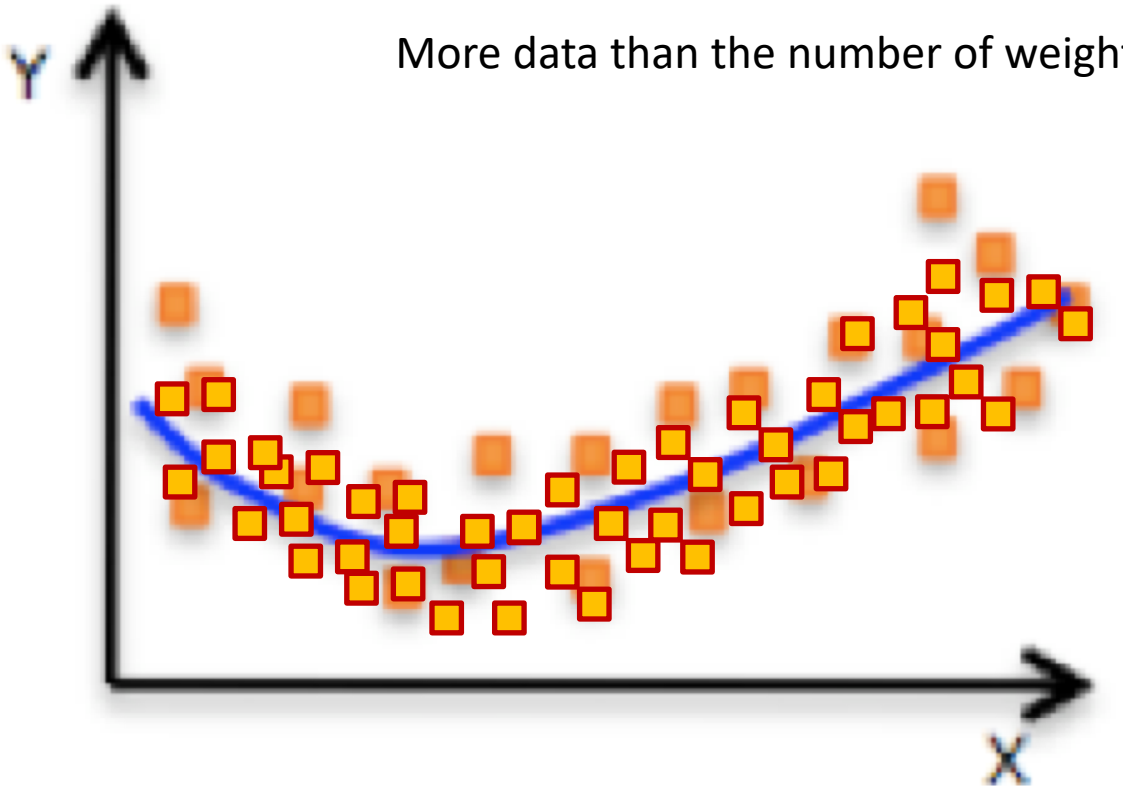
Data augmentation

for avoiding overfitting and improving robustness

- Turn one positive (negative) example into many positive (negative) examples: Proliferation
- Image data: rotate, re-scale, or shift image, or flip image about axis; image still contains the same objects, exhibits the same event or action



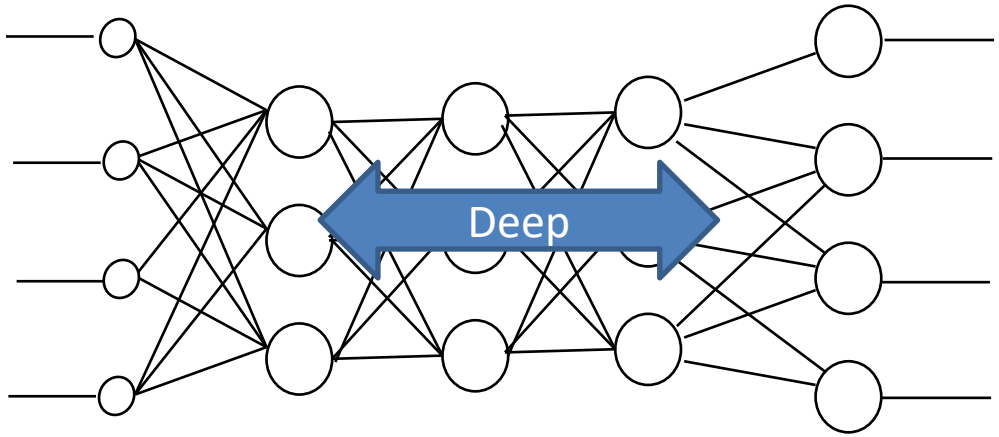




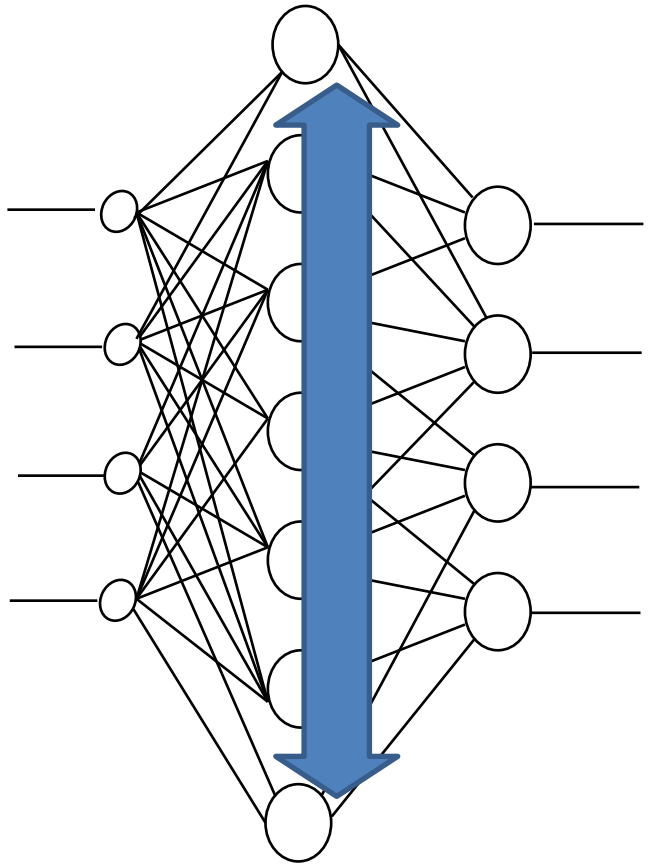
More data than the number of weights

Deep or Shallow

An Architecture Issue

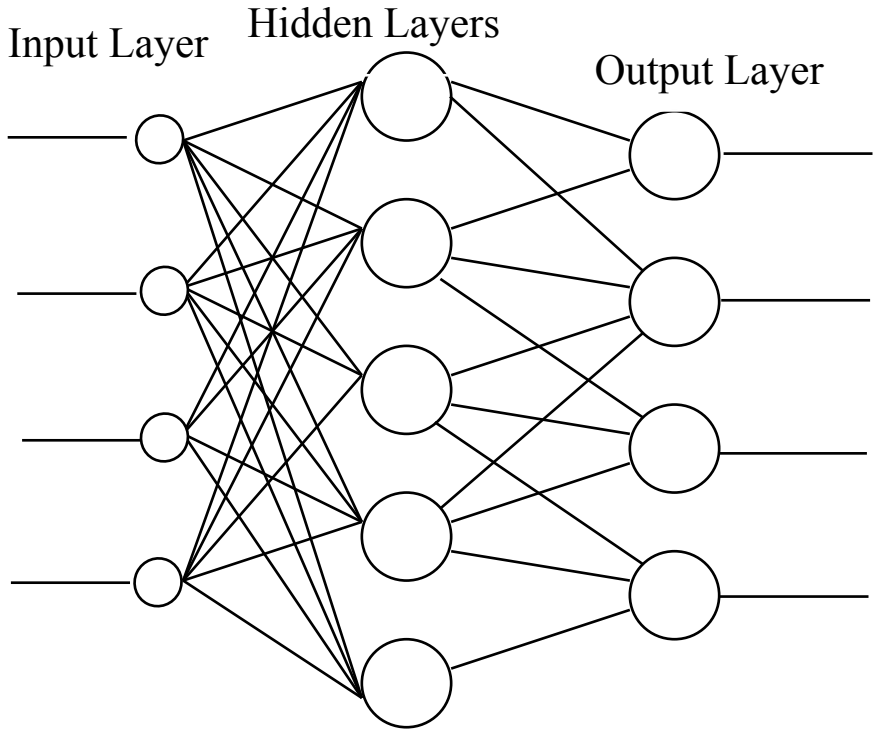


Shallow but wide

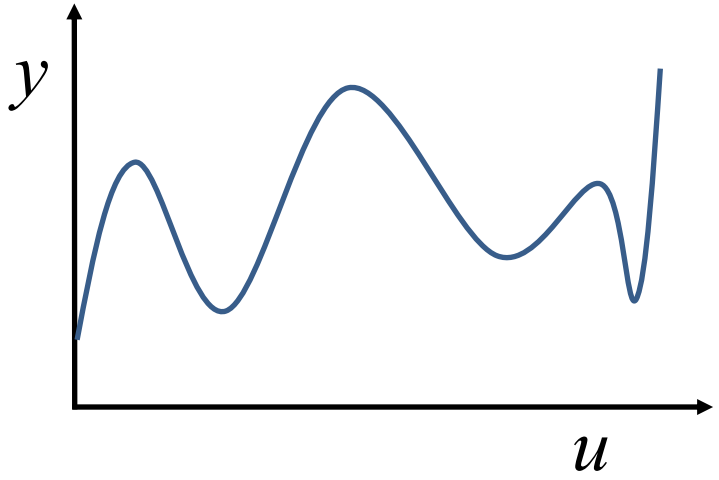


Universal Function Approximation Theory

3-Layer Neural Network



Hidden units are essential for representing a nonlinear function.



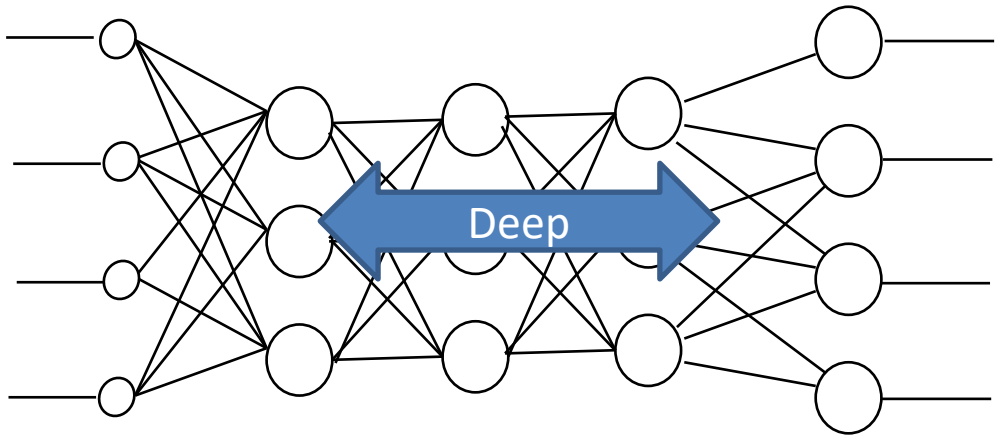
The 3-layer neural network is a universal approximation function that can approximate an arbitrary (measurable) function to any accuracy.

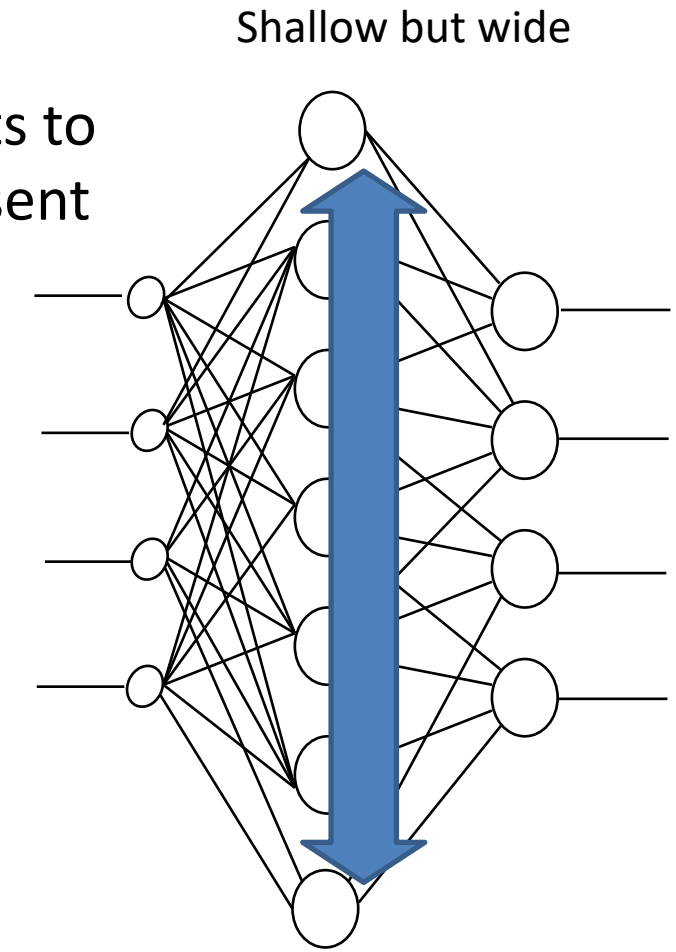
For an arbitrary (small) $\varepsilon > 0$, there exists a finite number of neural net units m, such that

$$\left| y(u) - \hat{y}(u;\theta,m) \right| < \varepsilon$$

Deep or Shallow: A debate

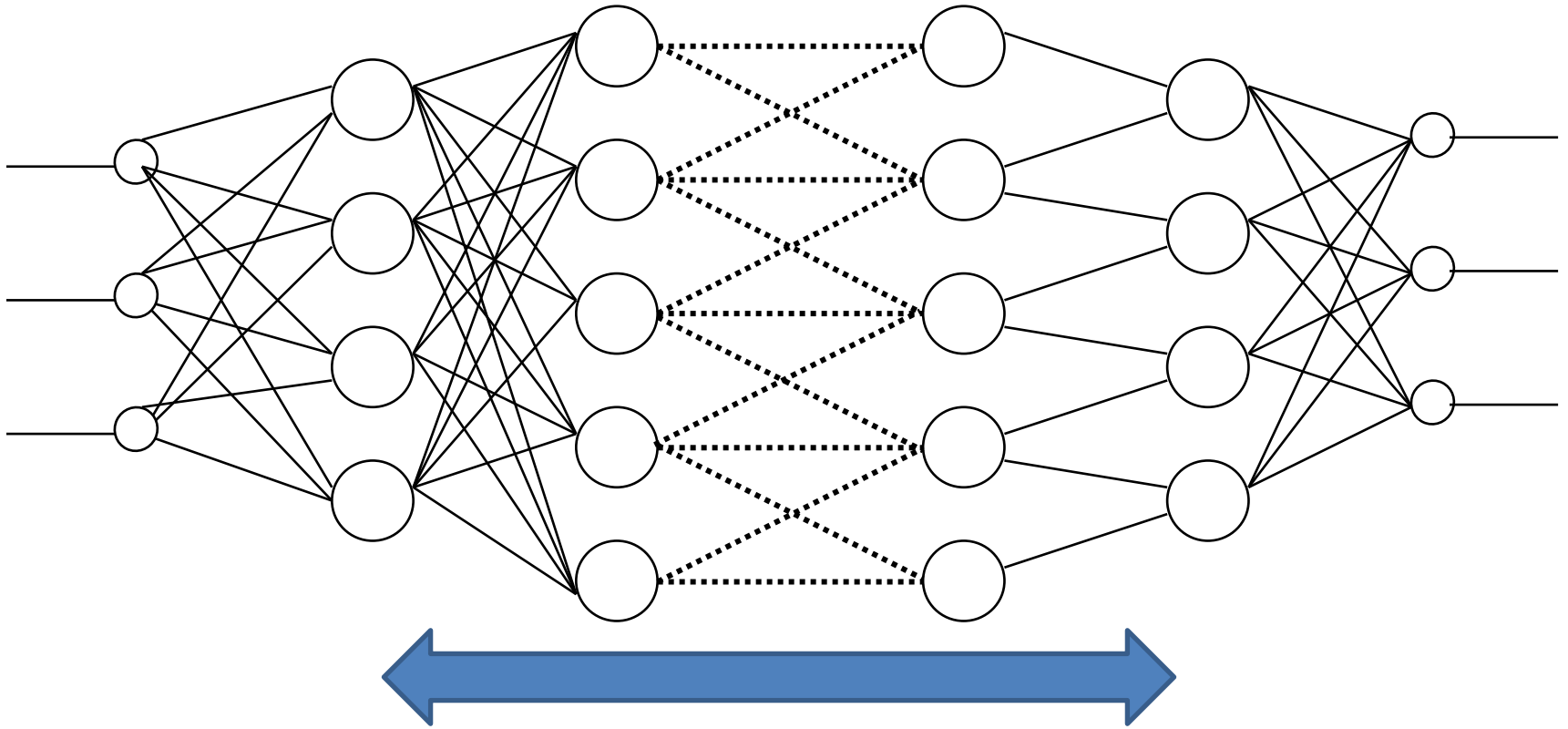
- According to the universal approximation theory, 3-layer network with one hidden layer is good enough to approximate any measureable nonlinear function.
- However, Deep Neural Net has fewer weights to tune than shallow / wide networks to represent the same nonlinear function.
- Deep Neural Net has more flexibility in architecture and algorithm.





Hypothesis

The more hidden layers a multi-layer NN has, the better it represents a highly nonlinear relationship.

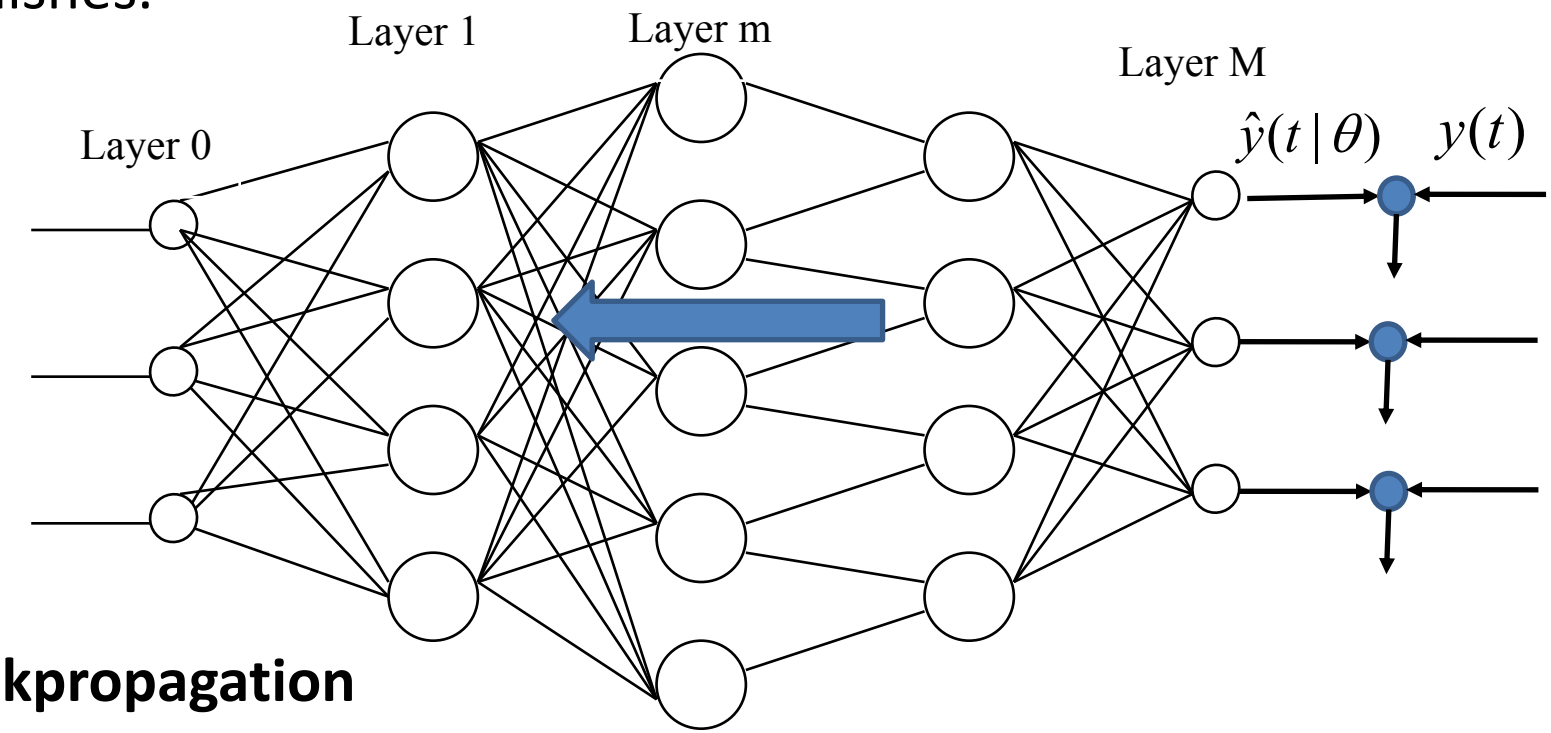


Deep

How can we train many layers of hidden units?

Gradient Vanishing: A roadblock

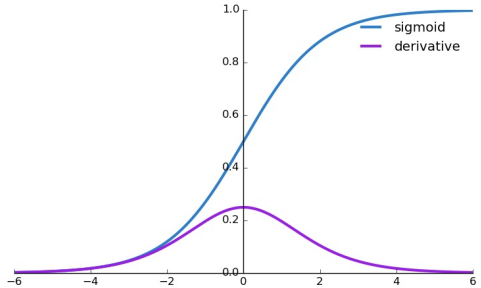
- Error does not propagate deep into early layers;
- Delta δ consists of products of g' and weights w; if one of them becomes zero, it vanishes.



Error Backpropagation

$$\delta_{deep} = -\frac{\partial E}{\partial z_{deep}} = \sum\sum\cdots\sum (y_i - \hat{y}_i) g'(z_i) \frac{\partial z_i}{\partial x_j} g'(z_j) \frac{\partial z_j}{\partial x_k} g'(z_i) \frac{\partial z_k}{\partial x_\ell} \cdots$$

$$= \sum\sum\cdots\sum (y_i - \hat{y}_i) g'(z_i) g'(z_j) g'(z_i) \cdots w_{ij} w_{jk} w_{k\ell} \cdots$$

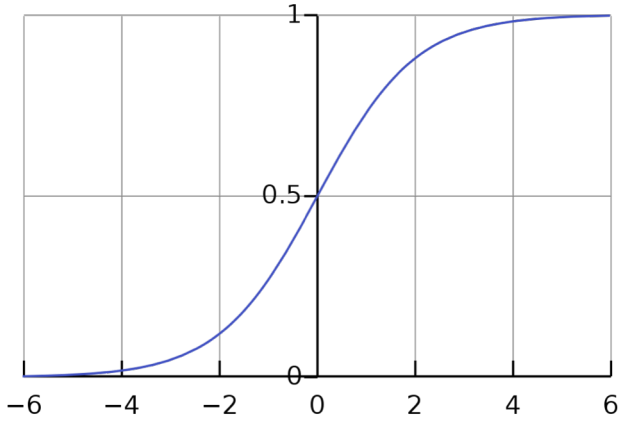


Recap

$$g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(1-g)$$

Sigmoid Function

$$g(z) = \frac{1}{1+e^{-z}}$$

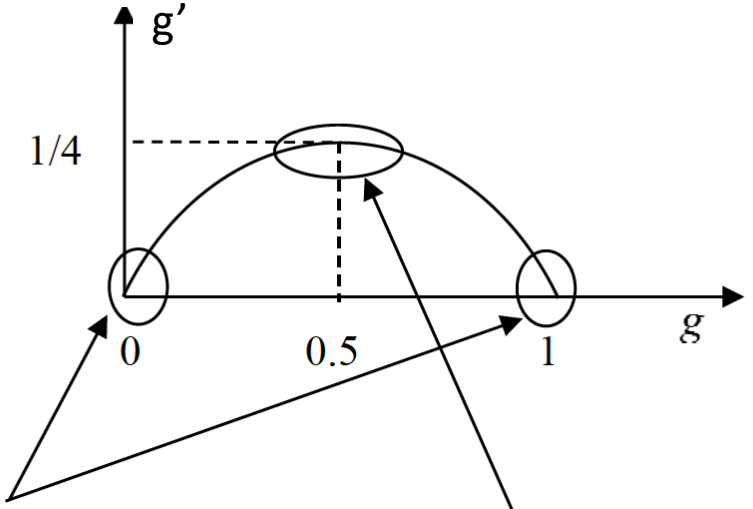


$$\Delta w_{ji} = \rho \delta_j^{(m)} x_i^{(m)}$$

$$\delta_j^{(m)} = g'_j \sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)}$$

(33) $\Delta w_{ji} \propto g'_j(z_j)$

The incremental weight change is proportional to the derivative of $g(z)$.

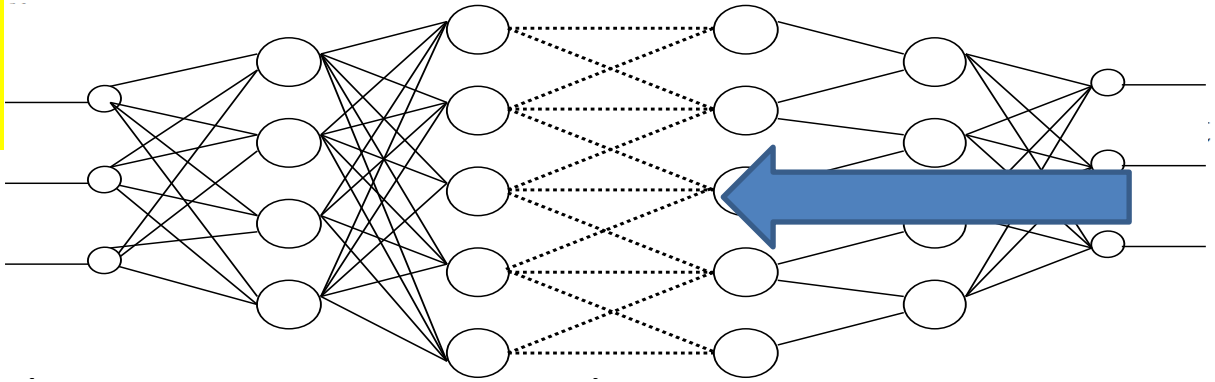


For $-\infty < z < \infty$.
g varies $0 < g < 1$.
Max $g' = 1/4$
at $z = 0$ $g = 0.5$

In these ranges weight changes are small.
$g \cong 0$ or $g \cong 1$
$|z| >> 1$.

The largest weight change occurs in this range.
$g = 0.5, z = 0$

The error does not propagate further.



Alternative activation functions

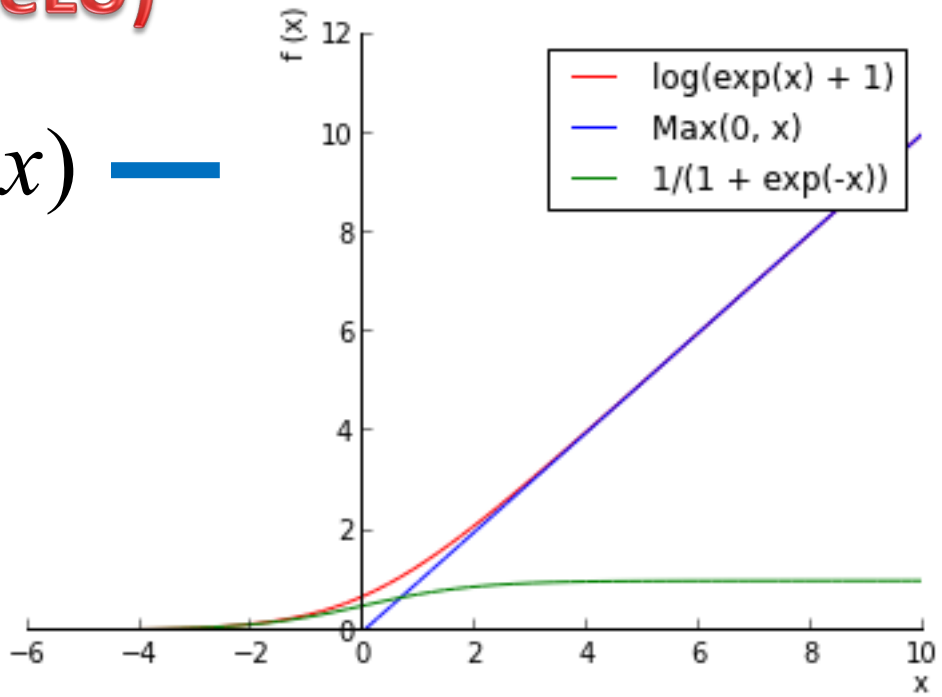
Propagation of error δ does not diminish.

Rectified Linear Unit (ReLU)

$$f(x) = x^+ = \max(0, x)$$

SoftPlus function

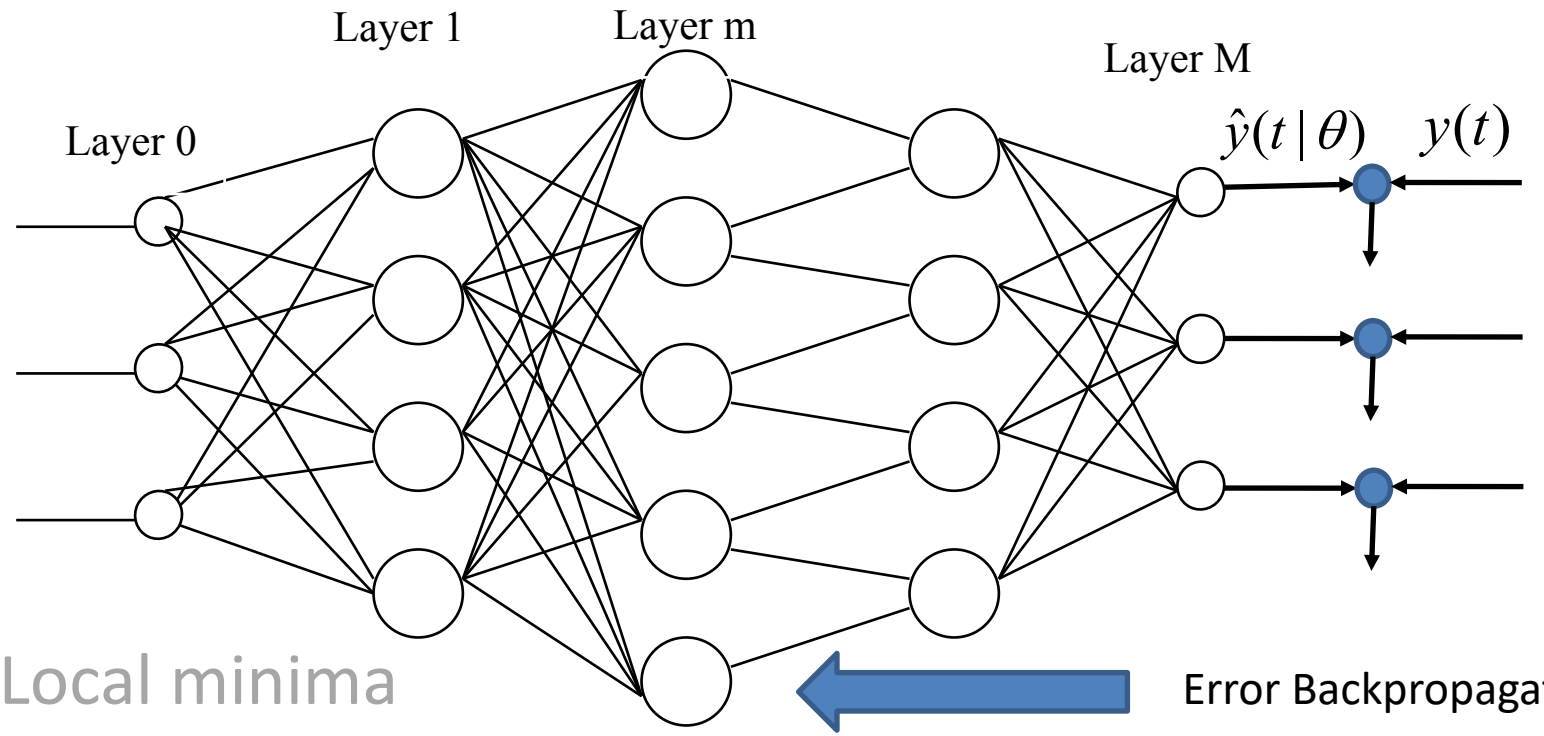
$$f(x) = \log(1 + \exp x)$$



$$f'(x) = \exp x/(1 + \exp x) = 1/(1 + \exp(-x)),$$

The derivative is the sigmoid function.

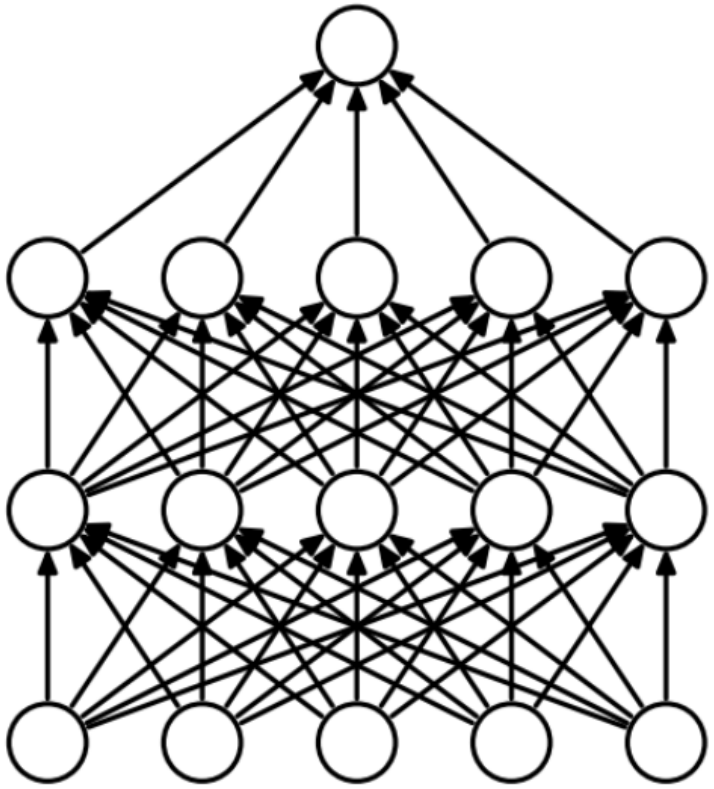
Issues of Multi-Layer NN and Error Back Propagation Algorithm



- Local minima
- Over fitting
- Slow convergence
- Error does not propagate deep into early layers;
- Architectural parameter tuning: how many layers, how many hidden units?

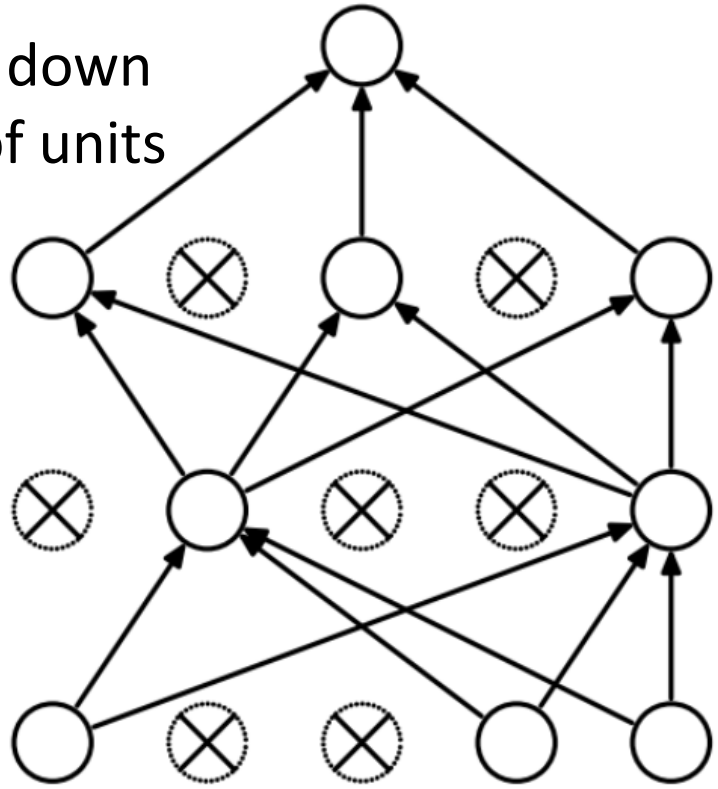
Dropout Training

- Training of a fully-connected multi-layer neural net is a challenge;
- Divide and Concur: Train a set of smaller-scale networks and combine them.



(a) Standard Neural Net

Shut down
(50%) of units

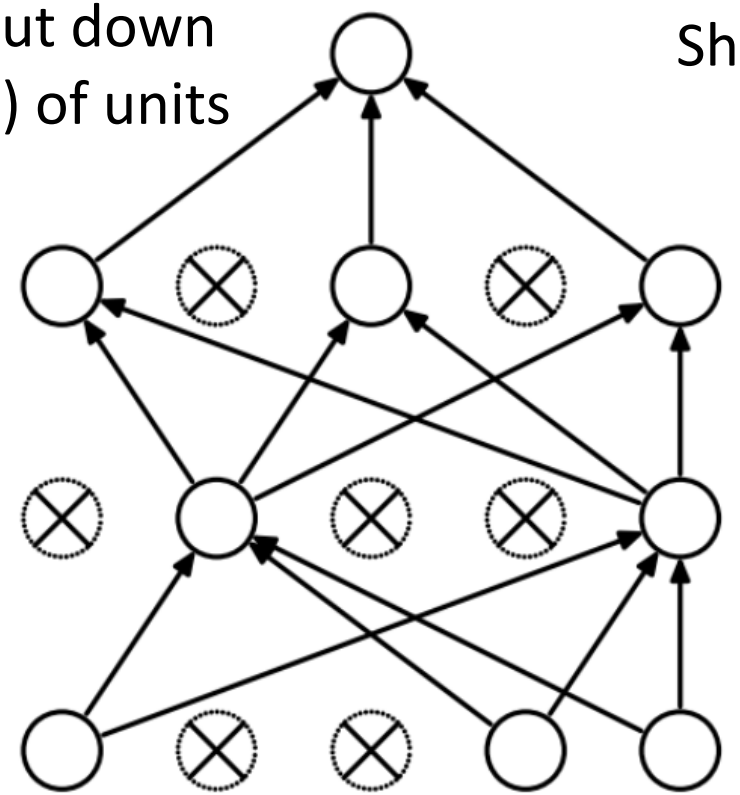


(b) After applying dropout.

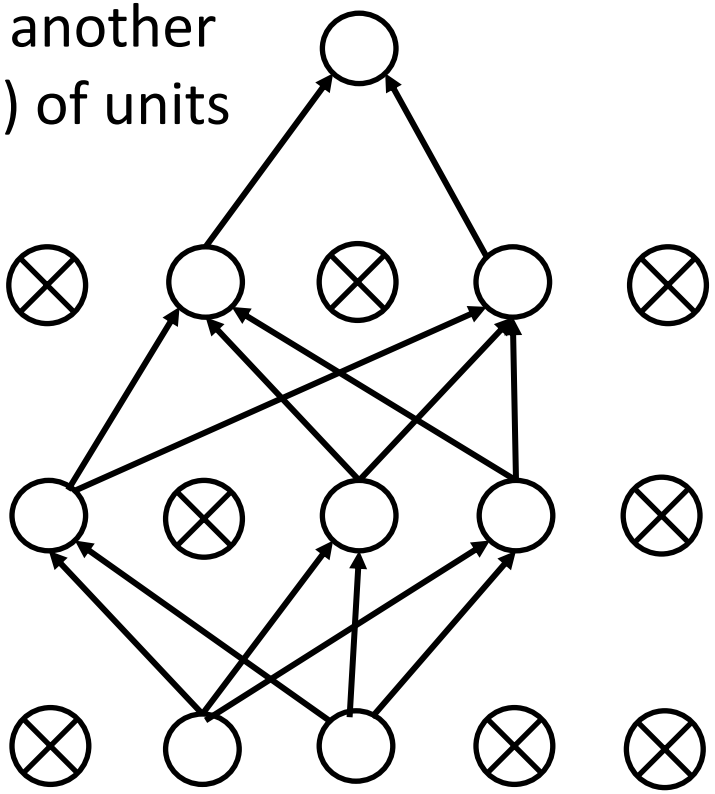
Dropout Training

- Divide the entire units into several groups, and train each group at a time;
- Fewer parameters: faster in conversion.

Shut down (50%) of units

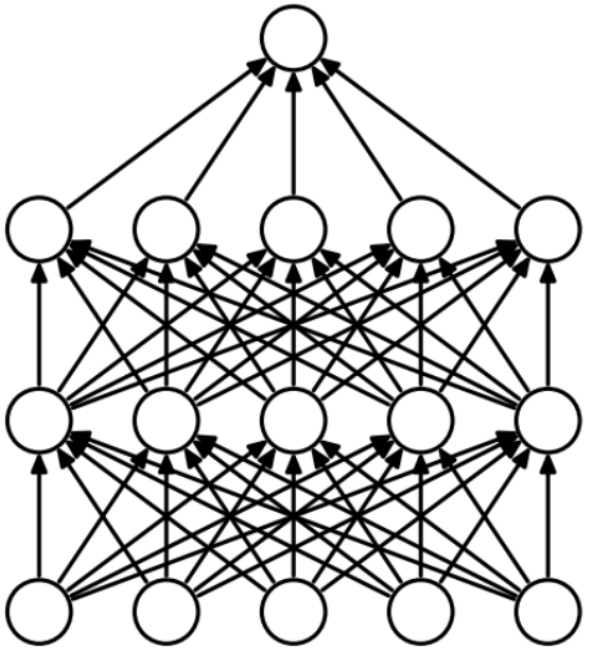


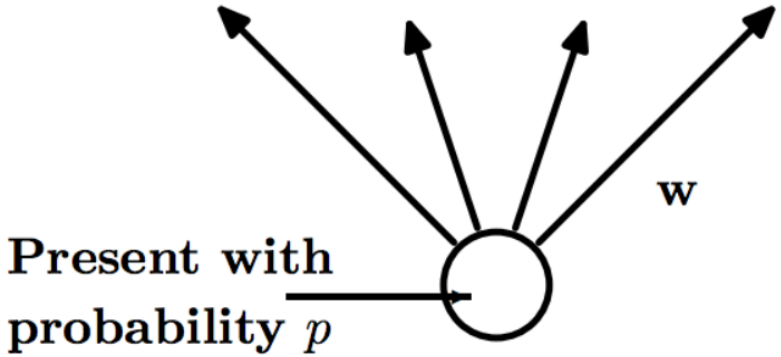
Shut down another (50%) of units



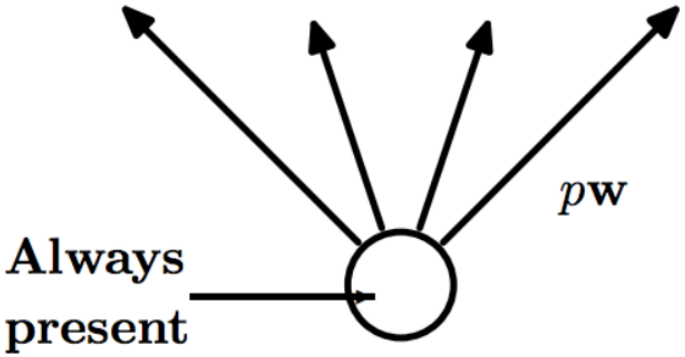
Dropout Training

- Combine all the trained units;
- Multiply each trained weight by fraction of times node was used during training;
- Ensemble mean: more robust.





(a) At training time



(b) At test time

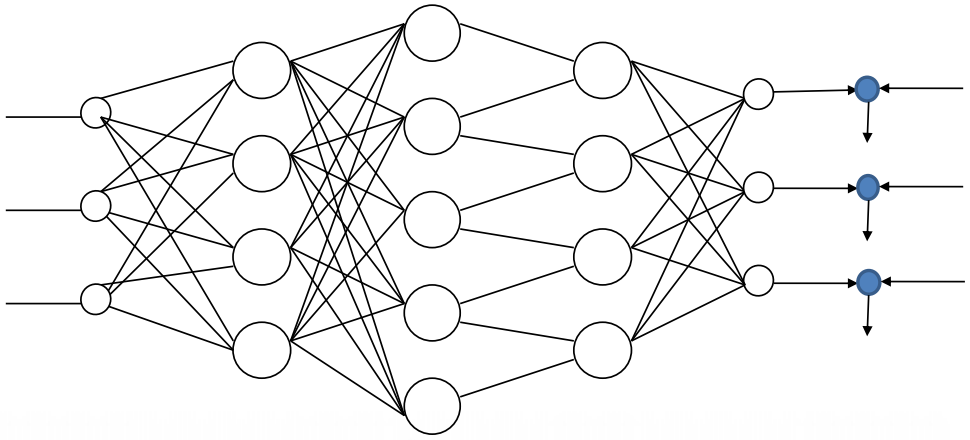
If one unit was shut down 80% of the time during individual dropout training, the weights trained are multiplied by 0.2 when combined: statistical mean.

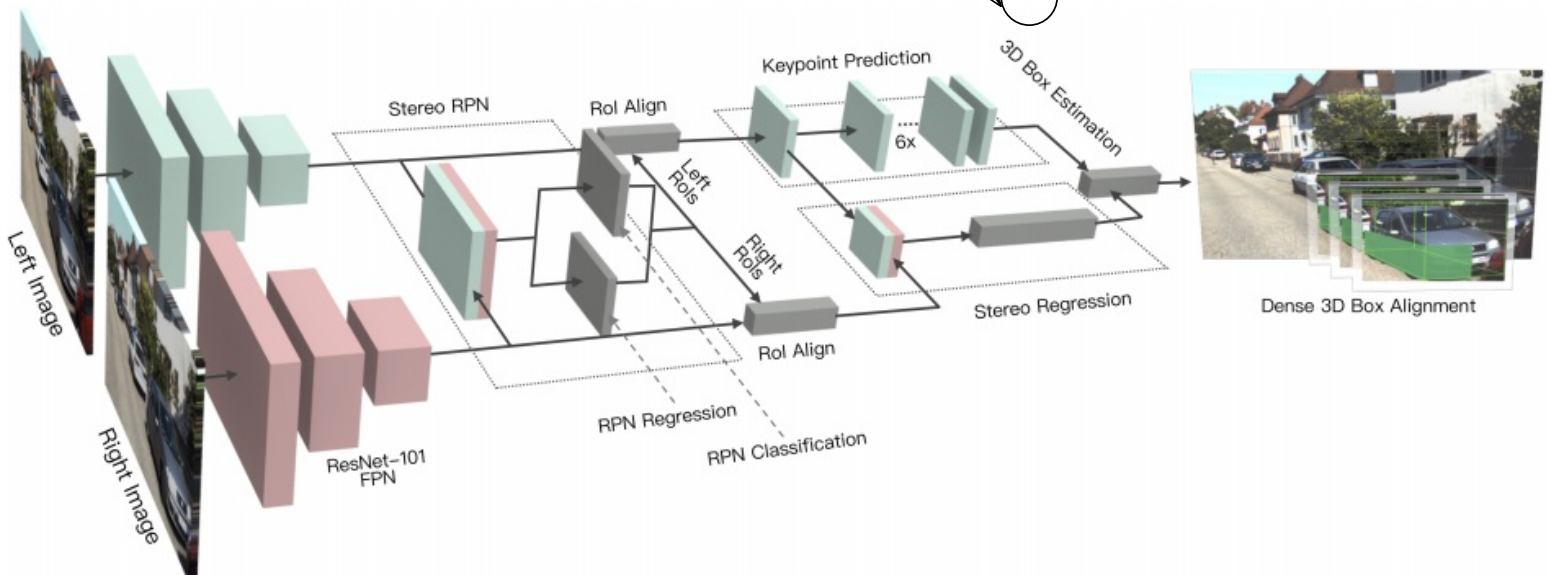
Convolutional Neural Network (CNN)

- Fully-connected multi-layer neural network does not scale well, particularly for processing a visual image;
- Focused connection is more effective.

300 x 300 x 3 = 270,000 pixels

At least, 270,000 x # units for the first hidden layer must be tuned.

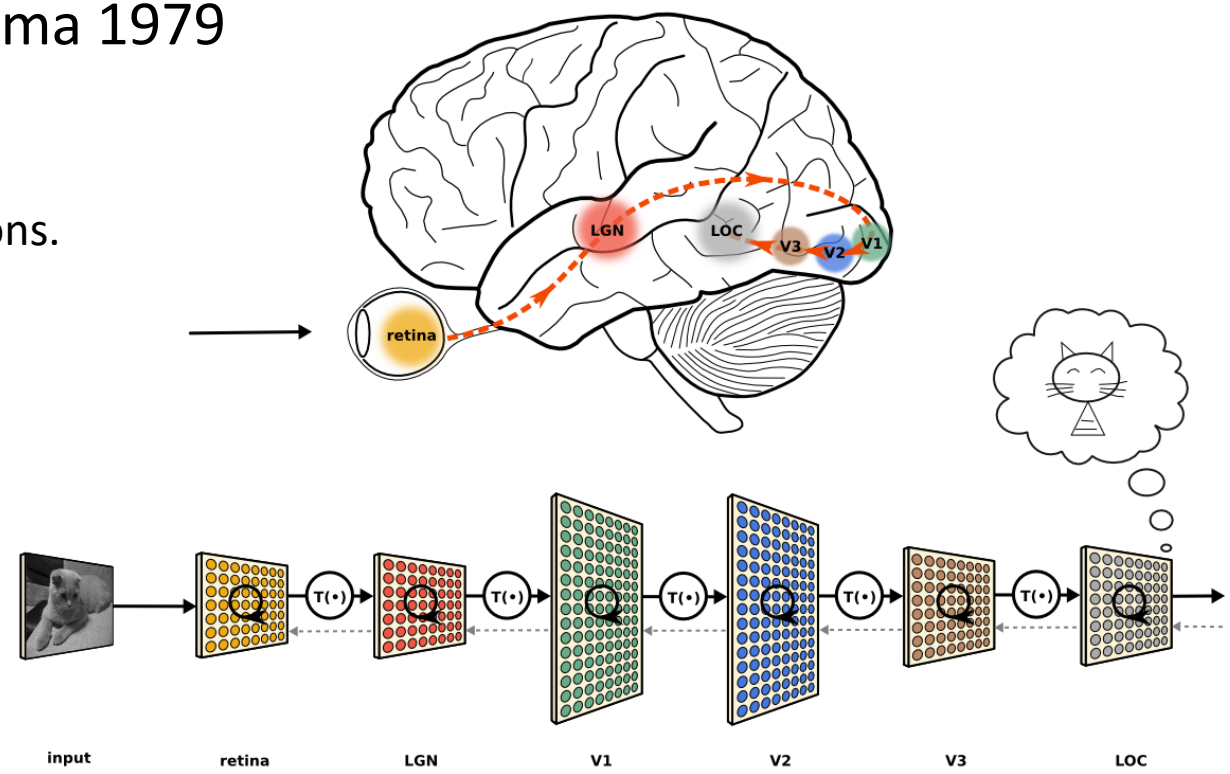


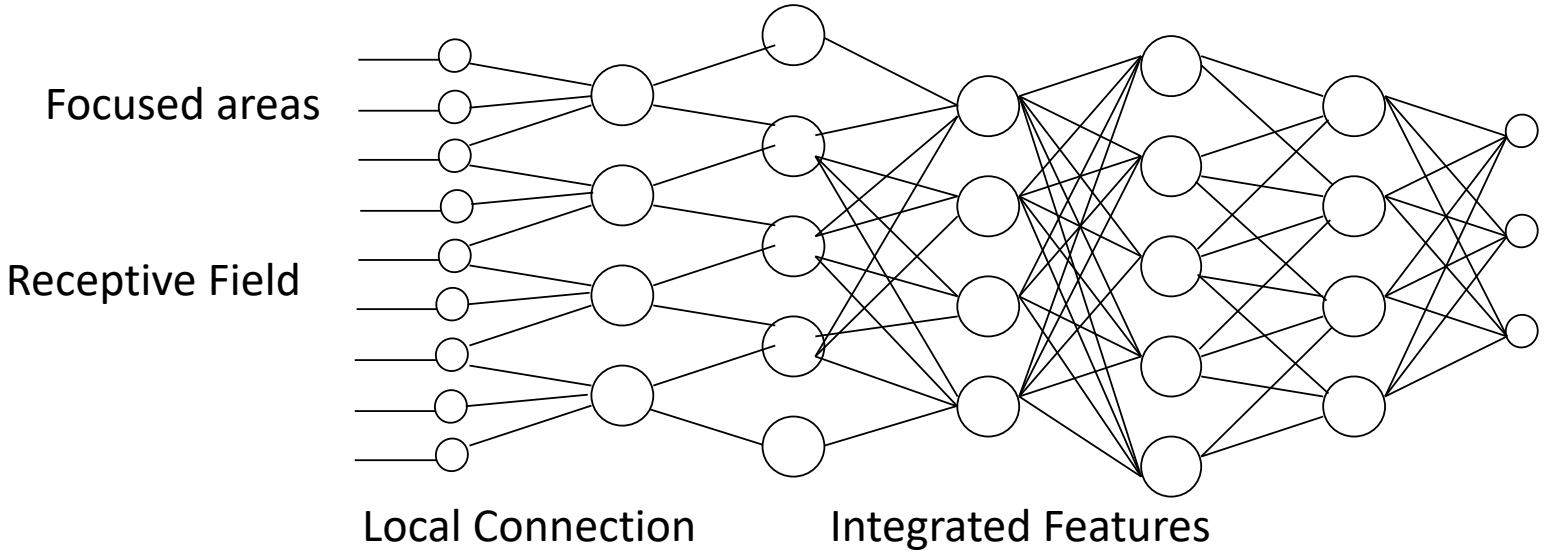


Convolutional Neural Network

- Research on the brain suggests that an image captured at the retina is first processed locally, extracting low-level features, and is passed on to a next level , where more complex features are extracted. Inspired by this, a hierarchical artificial neural network has been developed.
- Hubel and Wiesel, 1959
- Neocognitron, Fukushima 1979

The visual cortex is sensitive to sub-regions.





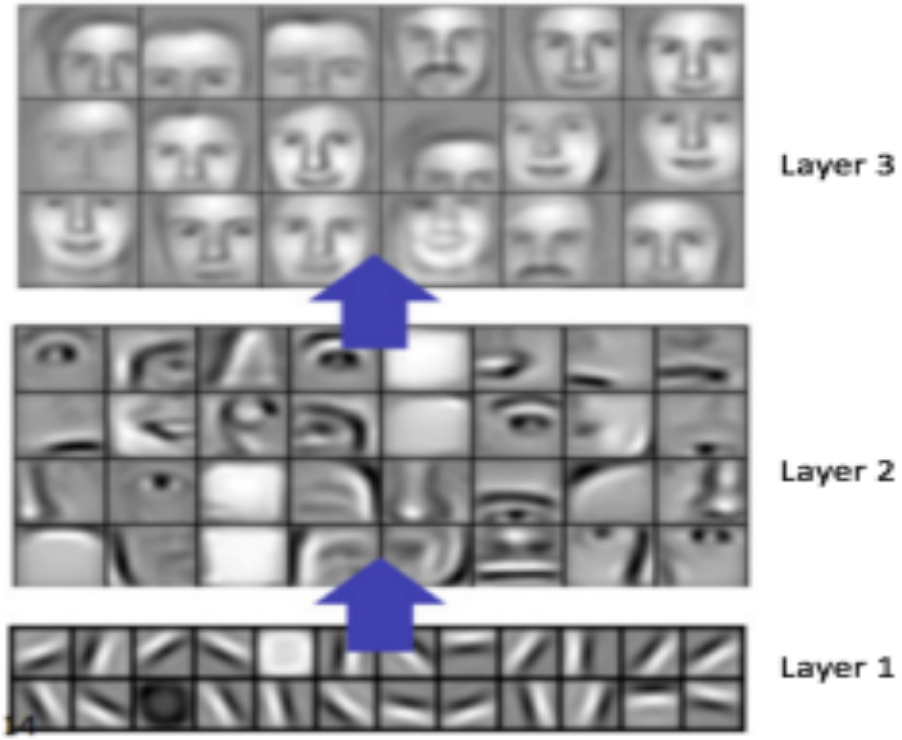
Perception of a face

Integration, Abstraction

Detection of eyes, nose, mouth, etc.

Integration, Abstraction

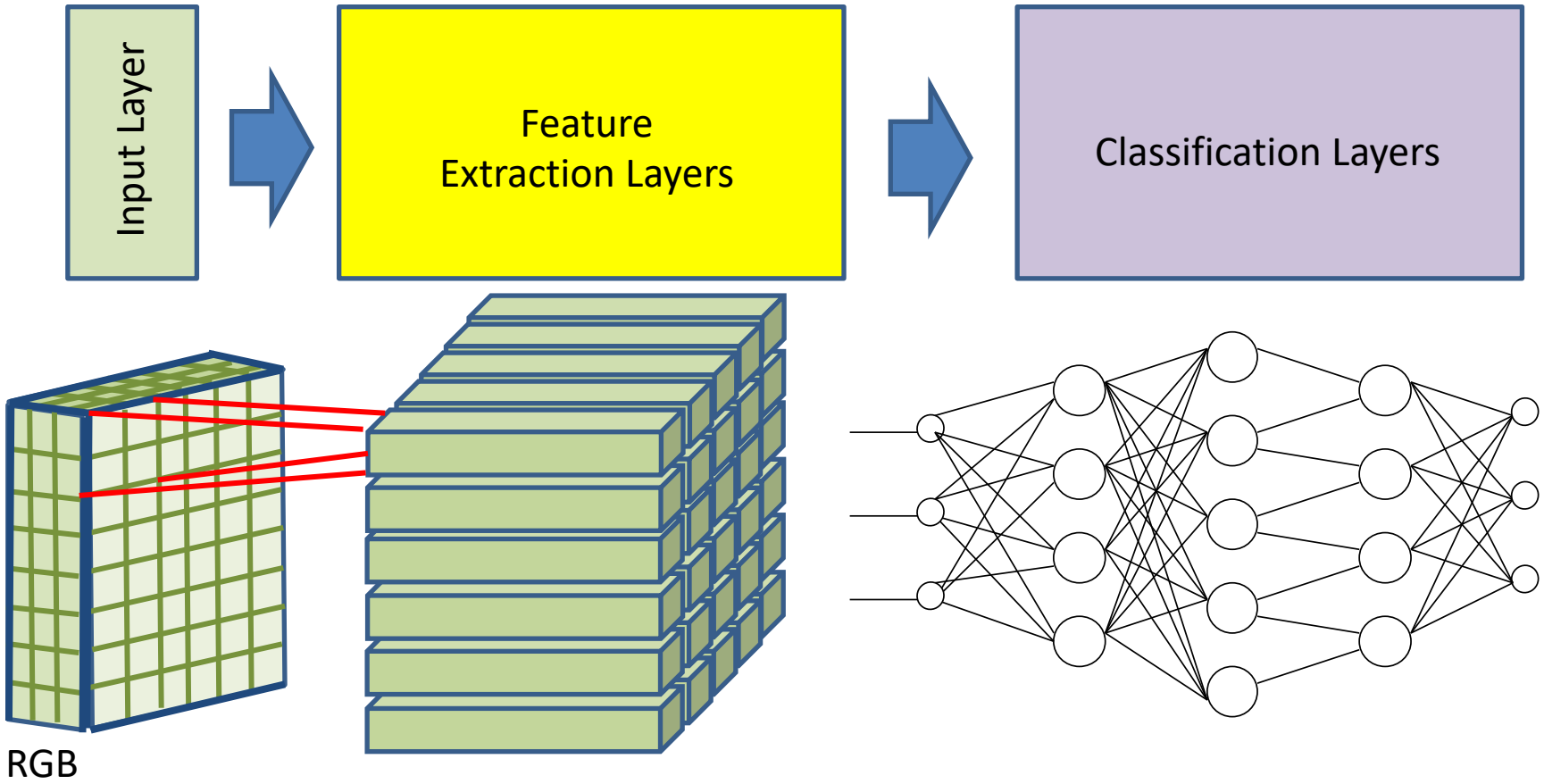
Detection of edges



Convolutional Neural Network

A Convolutional Neural Network consists of three different types of layers:

- Input Layer distributes inputs to the following layers;
- Feature extraction layers are connected to specific local regions of input layers, extracting local features; and
- Classification Layers are a fully-connected multi-layer neural network, producing outputs.



Feature Extraction via convolution

- In computer vision, local features, such as edges, are detected by using spatial filters;
- A spatial filter (2D template) is overlaid with the original image to evaluate whether the local image matches the template.
- The computation used is basically ***correlation***, or convolution (abuse of terminology).

Local image pixcels

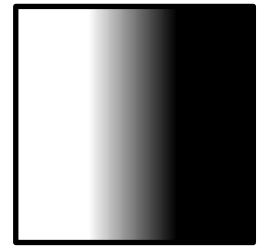
.8	.4	.1
.8	.5	.2
.7	.6	.3

Spatial filter / kernel

-1	0	1
-2	0	2
-1	0	1

$\frac{\partial f}{\partial x}$

Detecting a vertical edge



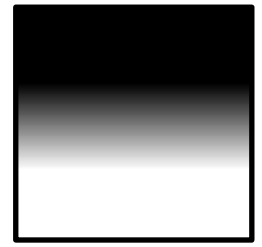
= .8x(-1)+.4x0+.1x1
+.8x(-2)+.5x0+.2x2
+.7x(-1)+.6x0+.3x1
= -2.3

.8	.4	.1
.8	.5	.2
.7	.6	.3

-1	-2	-1
0	0	0
1	2	1

$\frac{\partial f}{\partial y}$

Detecting a horizontal edge



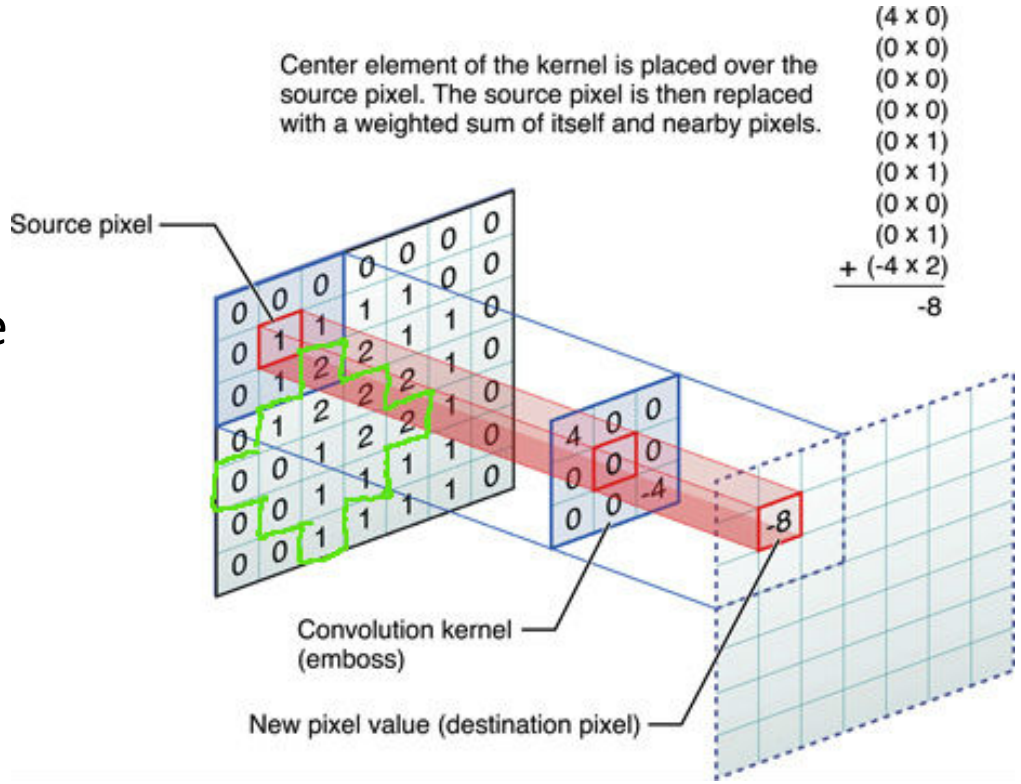
= 0.5

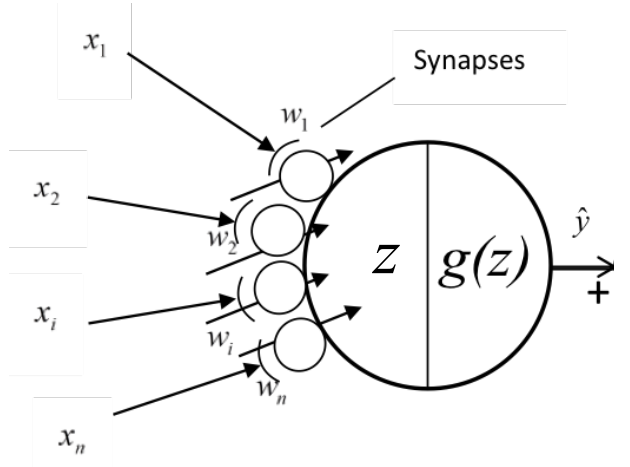
Convolution (Correlation)

- Convolutional Neural Network uses the same local spatial filter / kernel for extracting local features.

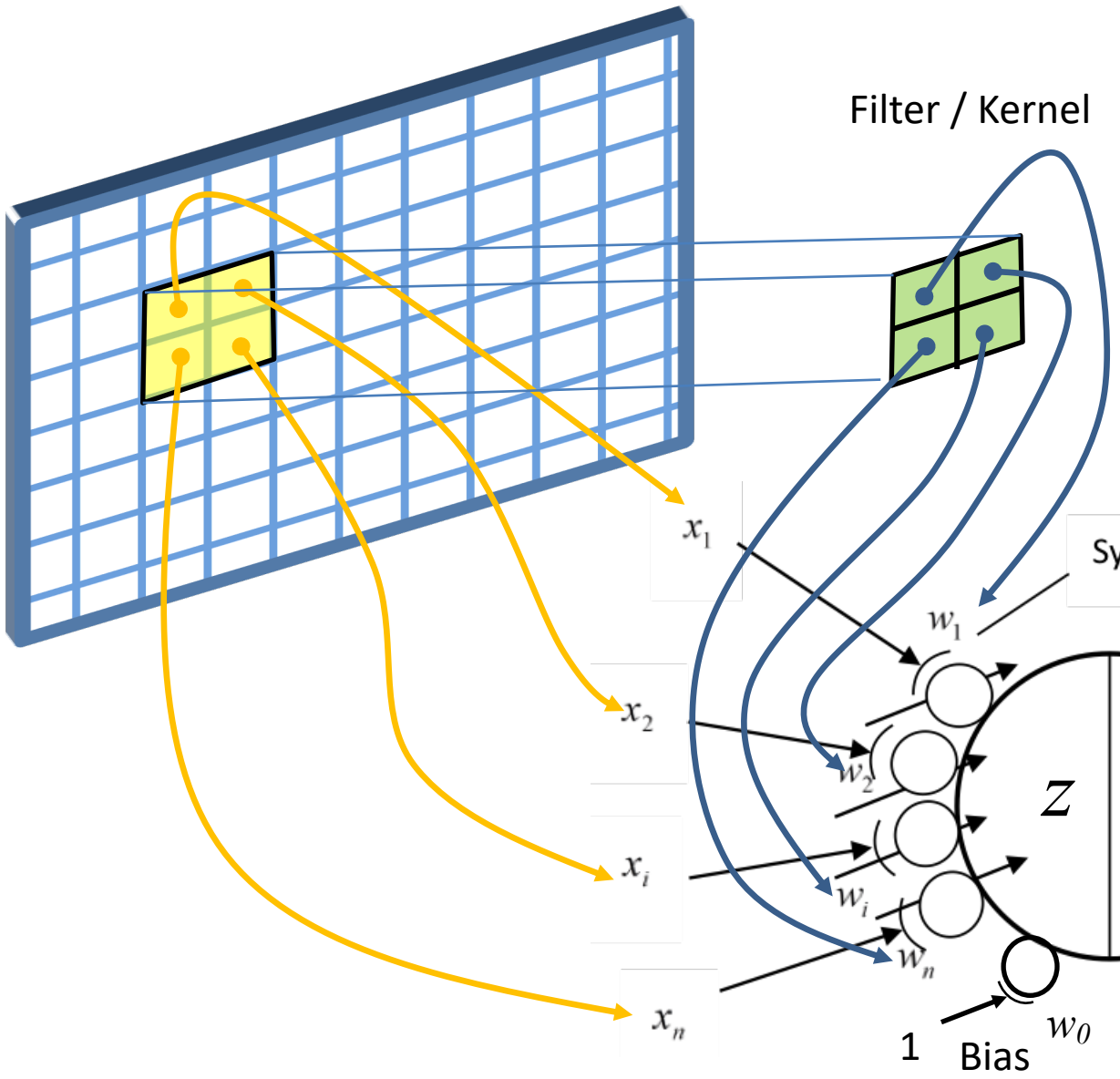
- Each segment of input array (image pixels) is convoluted with a template (convolutional kernel);
- The weighted sum of the input segment is computed for all the segments, by shifting the window to cover all the input array.

- The major difference from the standard spatial filters is that in CNN these filters are weights of neural units and are generated through learning.





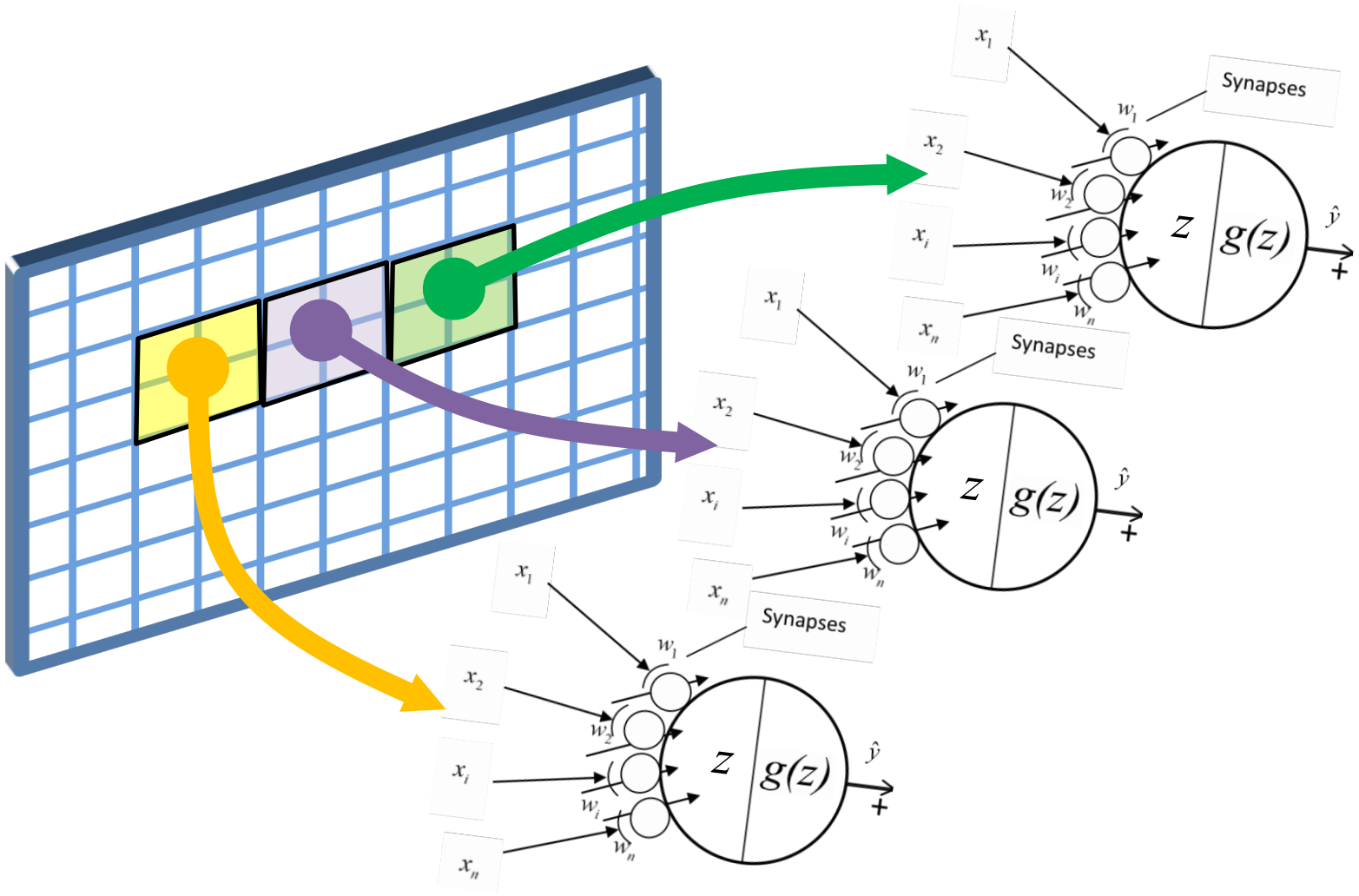
Feature Extraction



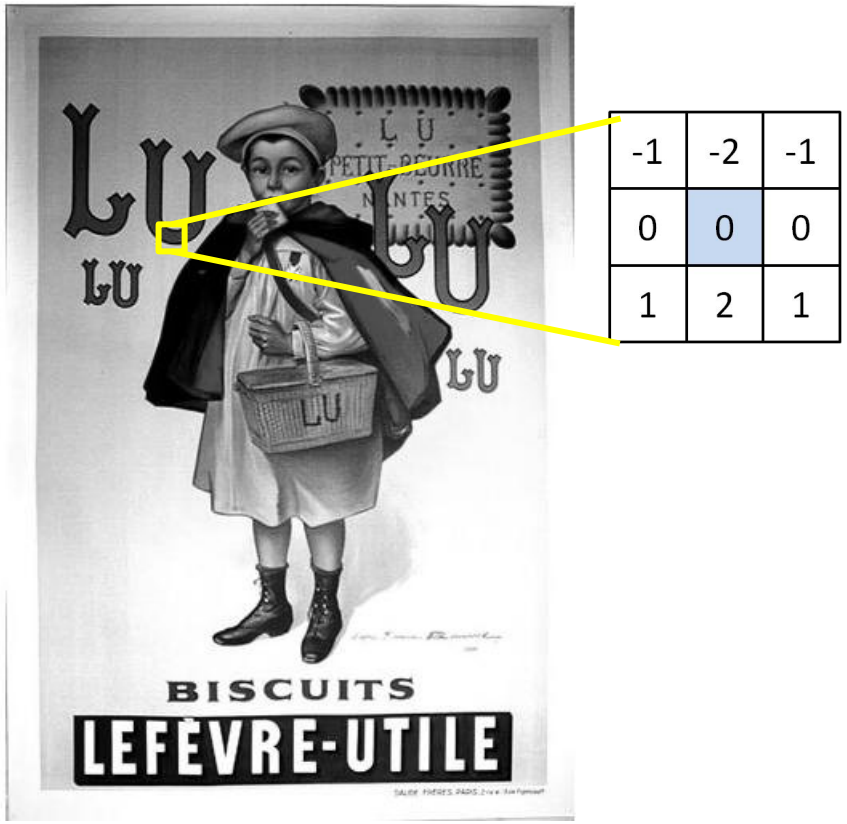
The parameters of a filter / kernel are weights of a neural unit, while the corresponding local array of data (e.g. pixels) are fed into the inputs to the neural unit.

$$z_j^{(m)} = \sum_j w_{ji}^{(m)} \, x_i^{(m)}$$

Local Feature Extraction through Convolution



Parameter Sharing

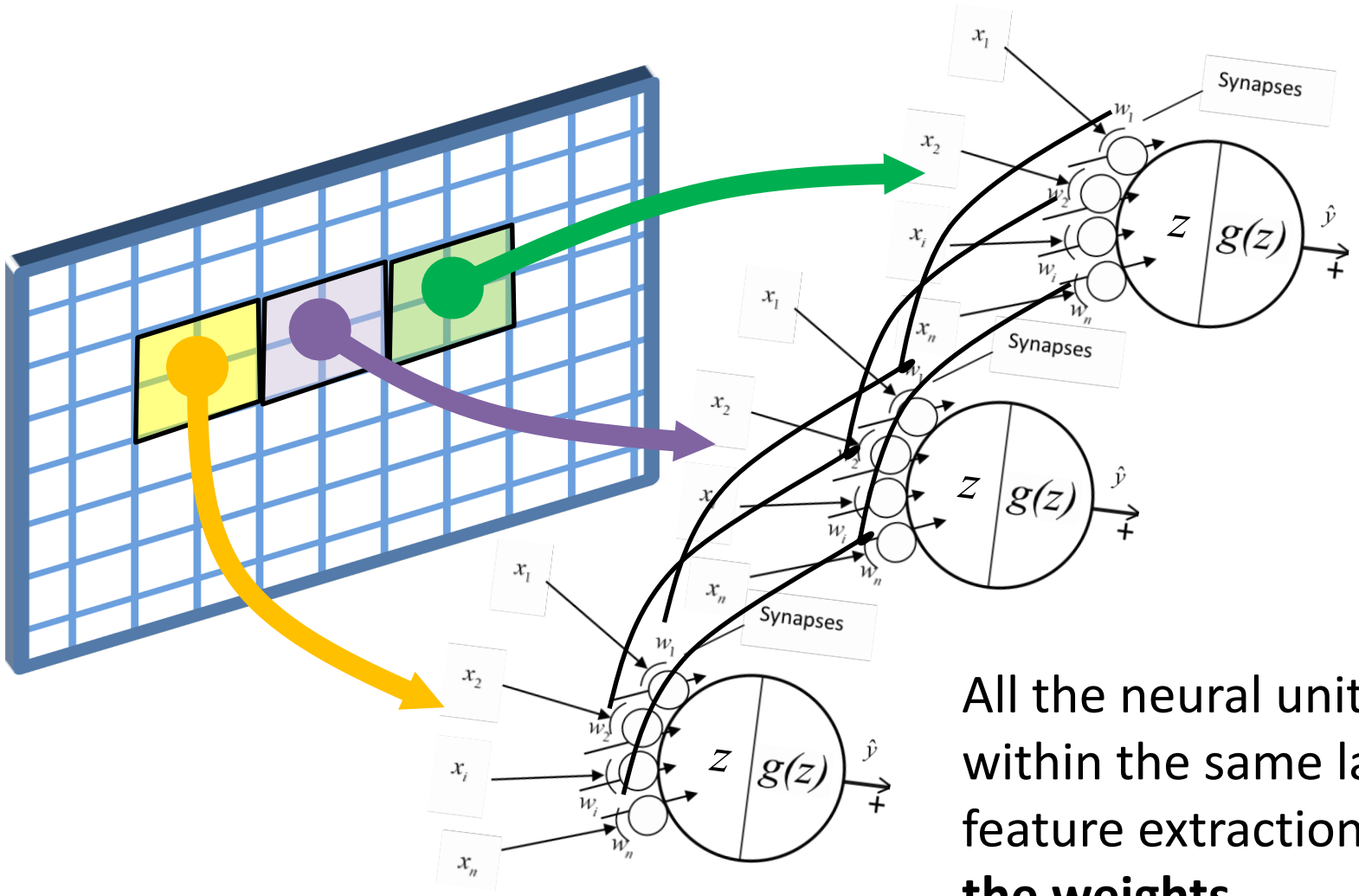


-1	-2	-1
0	0	0
1	2	1

- Local features, such as horizontal edge and corner, should be found across the entire image (data).
- Therefore, the same filter / kernel should be used everywhere: parameter sharing.
- This also implies that such features are location invariant: applicable to all regions.

- This parameter sharing also reduces the number of parameters to train.

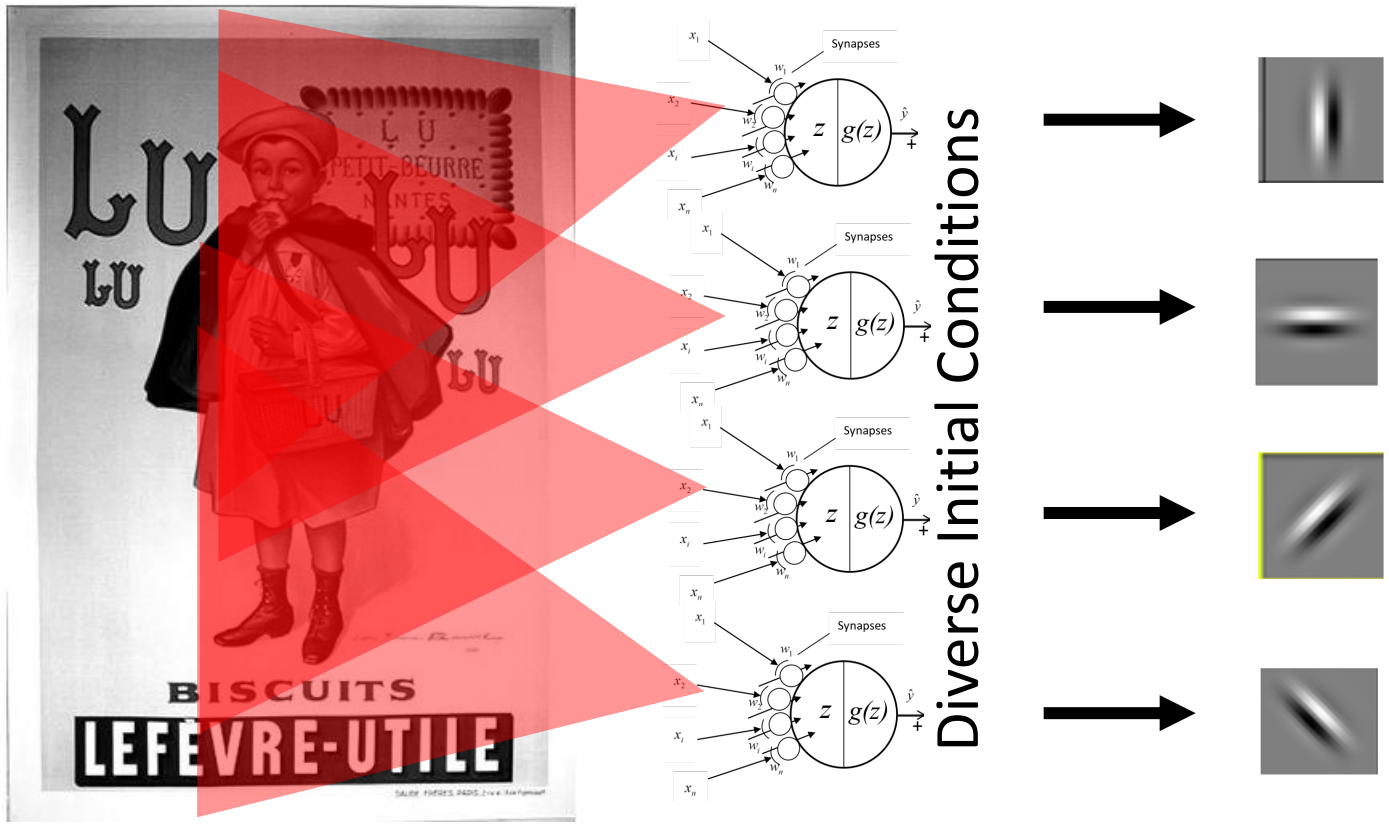
Parameter Sharing



All the neural units within the same layer of feature extraction **share the weights**.

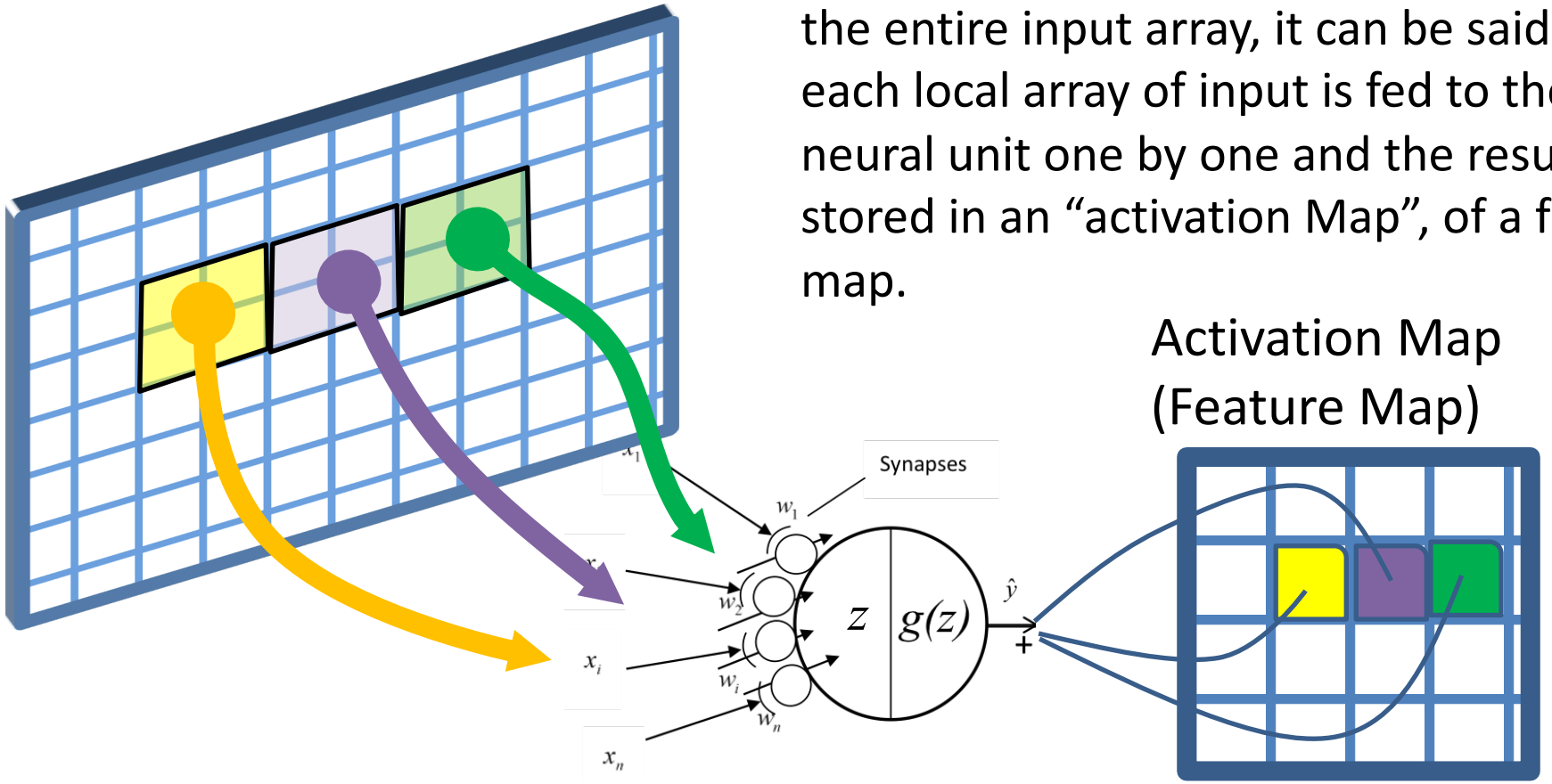
Creating Multiple Local Filters

- Multiple filters are needed for detecting various local features, including vertical, horizontal, oblique edges.
- Independent multiple neurons are used for creating multiple local filters, each of them is trained with different initial weights.

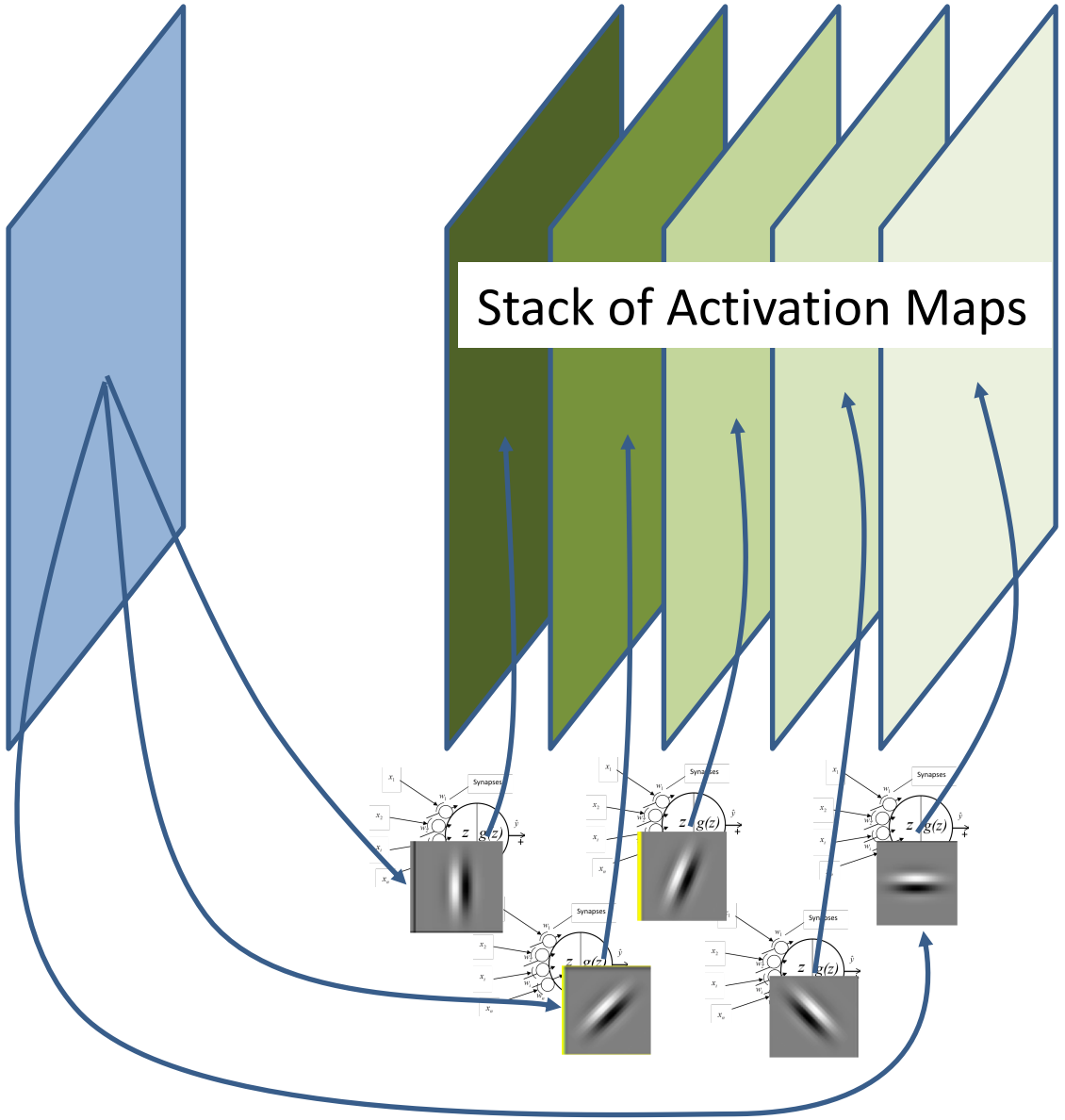


Feature extraction with neural units

Since the same neural unit (filter) is used for the entire input array, it can be said that each local array of input is fed to the single neural unit one by one and the result is stored in an “activation Map”, of a feature map.



Multiple Filters and a Stack of Activation Maps

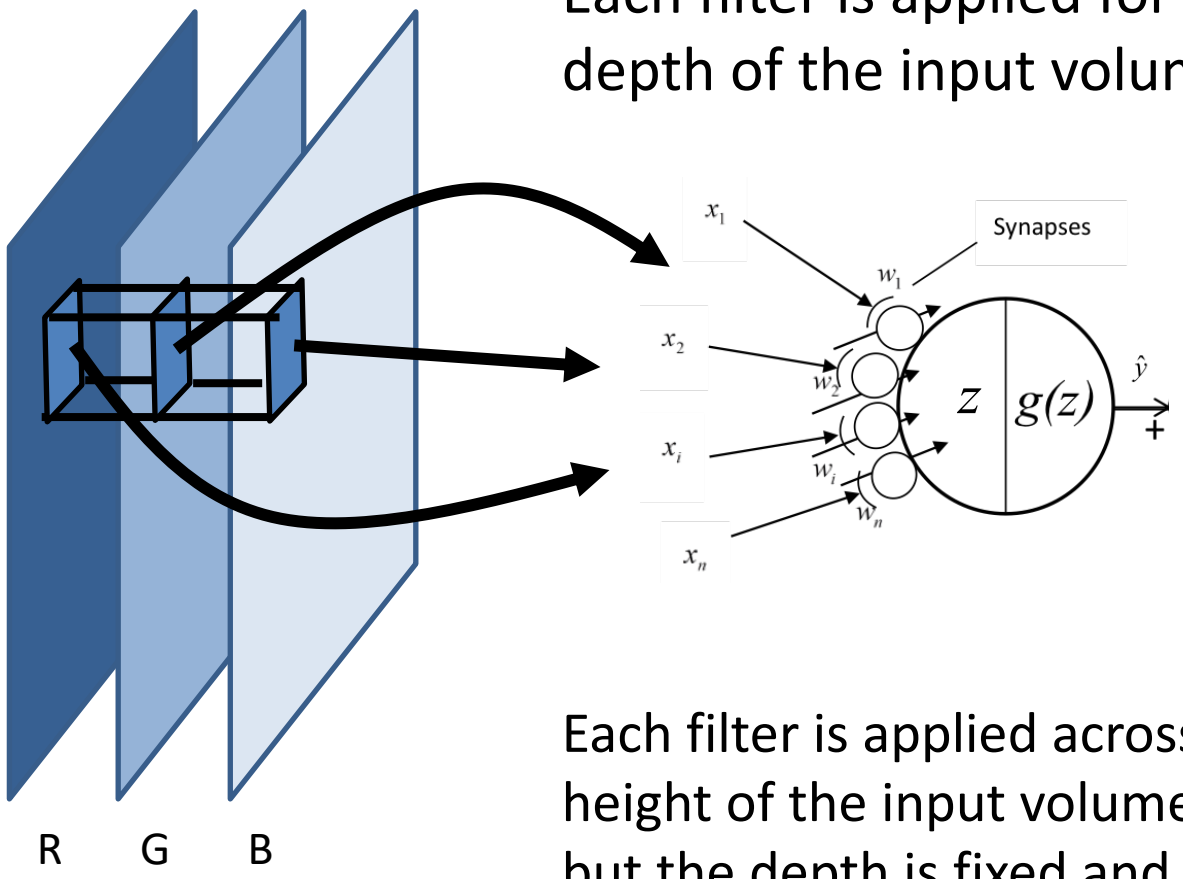


Each filter (kernel) detects a particular feature associated with a single neural unit.

We can use multiple filters to detect various features.

As a result, a stack of activation maps are generated.

Input Volume

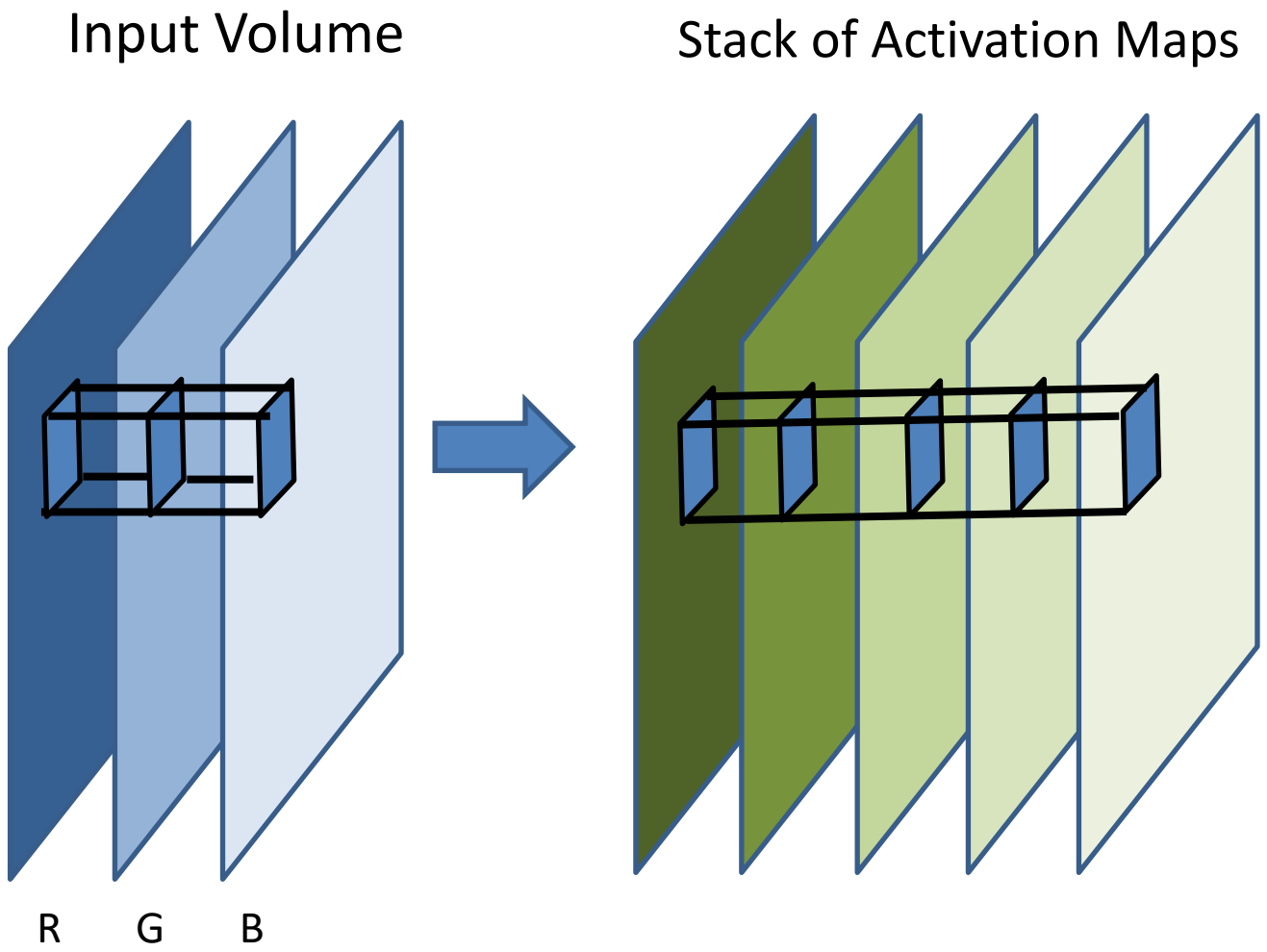


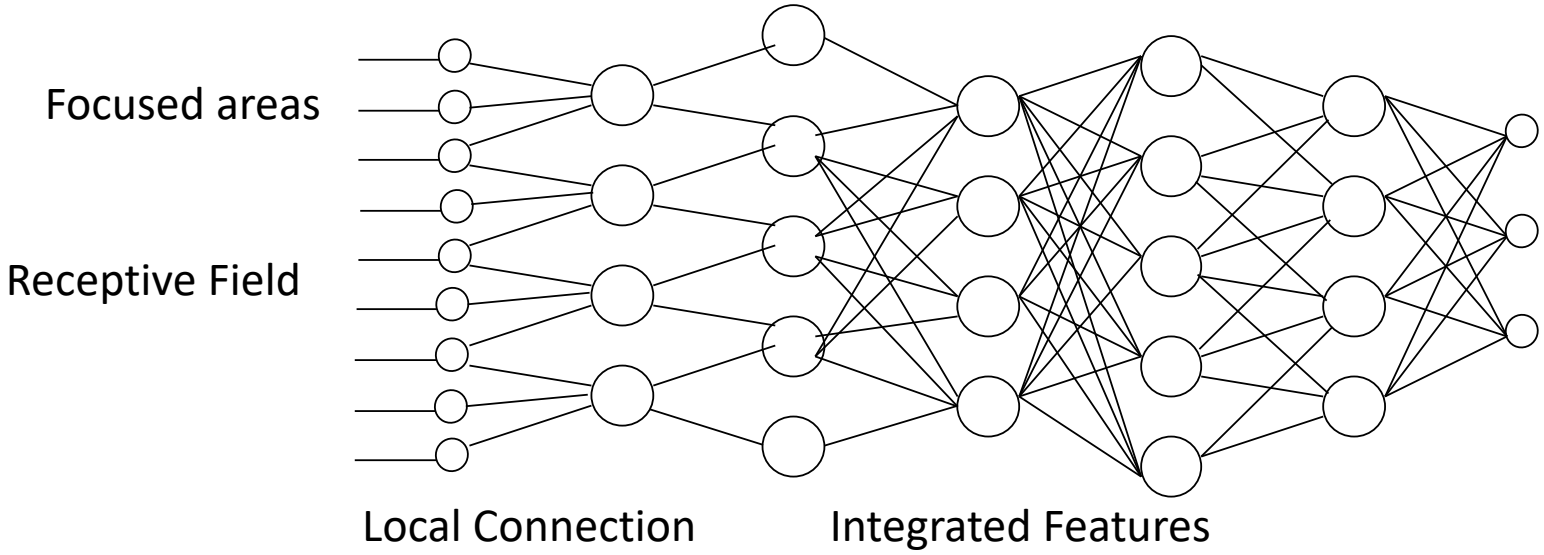
Each filter is applied for every depth of the input volume.

Each filter is applied across the width and height of the input volume by sliding on it, but the depth is fixed and fully connected.

CNN feature extraction finds strong local correlations in Receptive Field.

Volume to Volume Processing all created by structuring the connectivity of neural network.





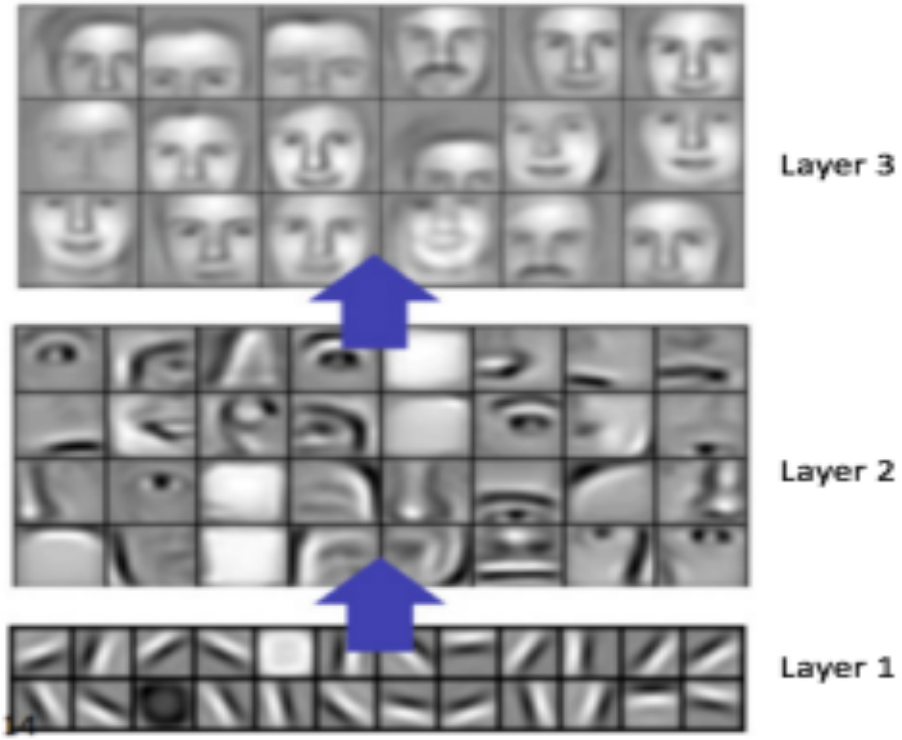
Perception of a face

Integration, Abstraction

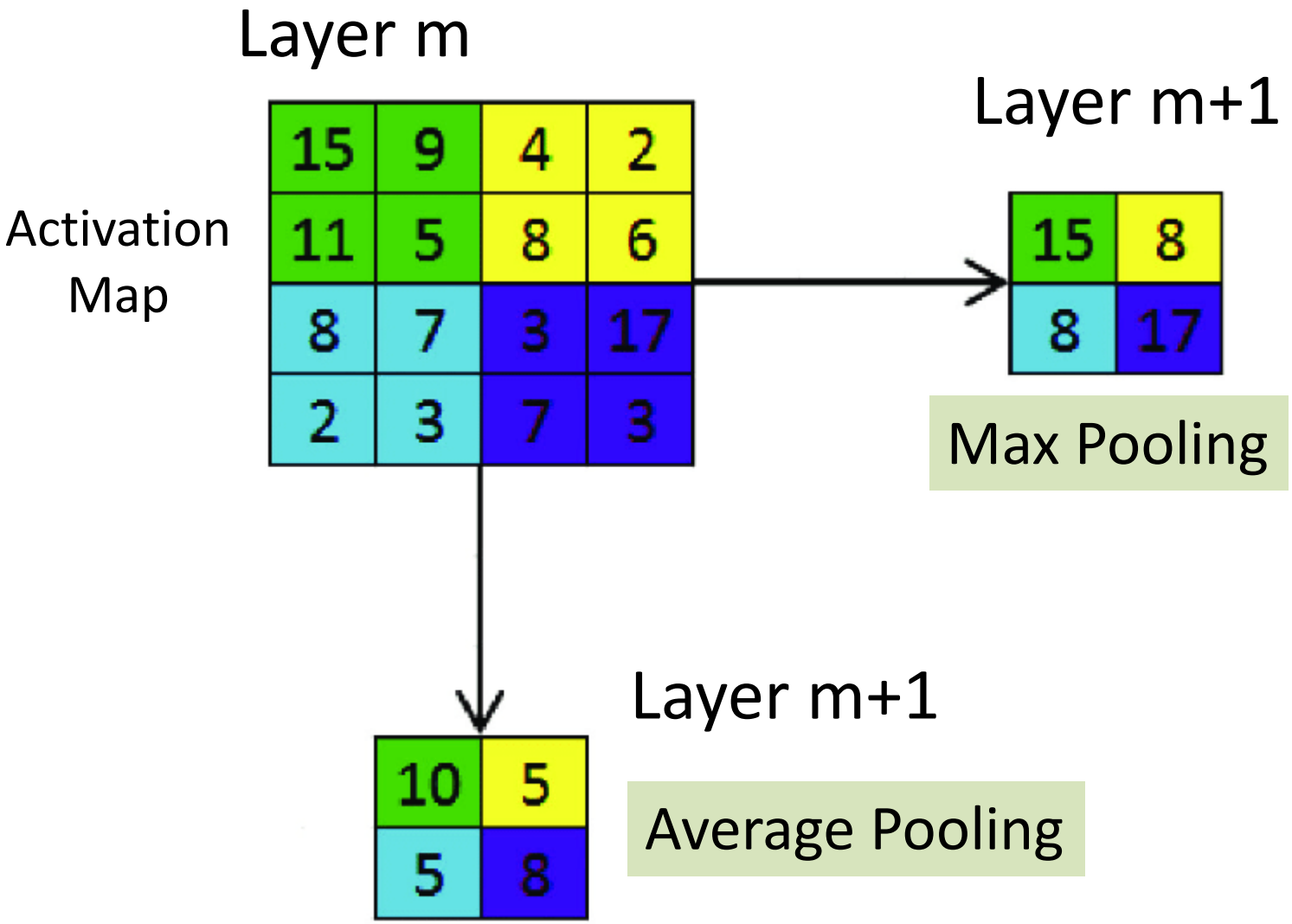
Detection of eyes, nose, mouth, etc.

Integration, Abstraction

Detection of edges

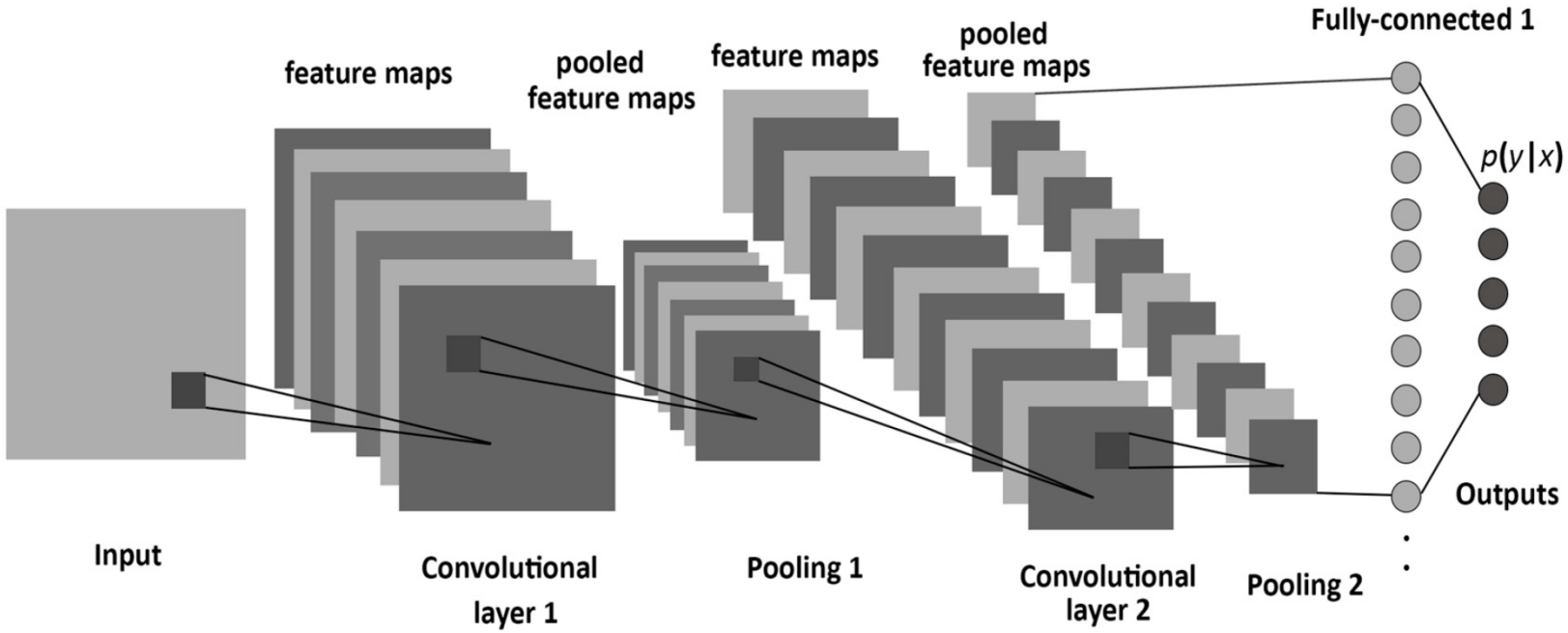


Pooling: Integration and Abstraction



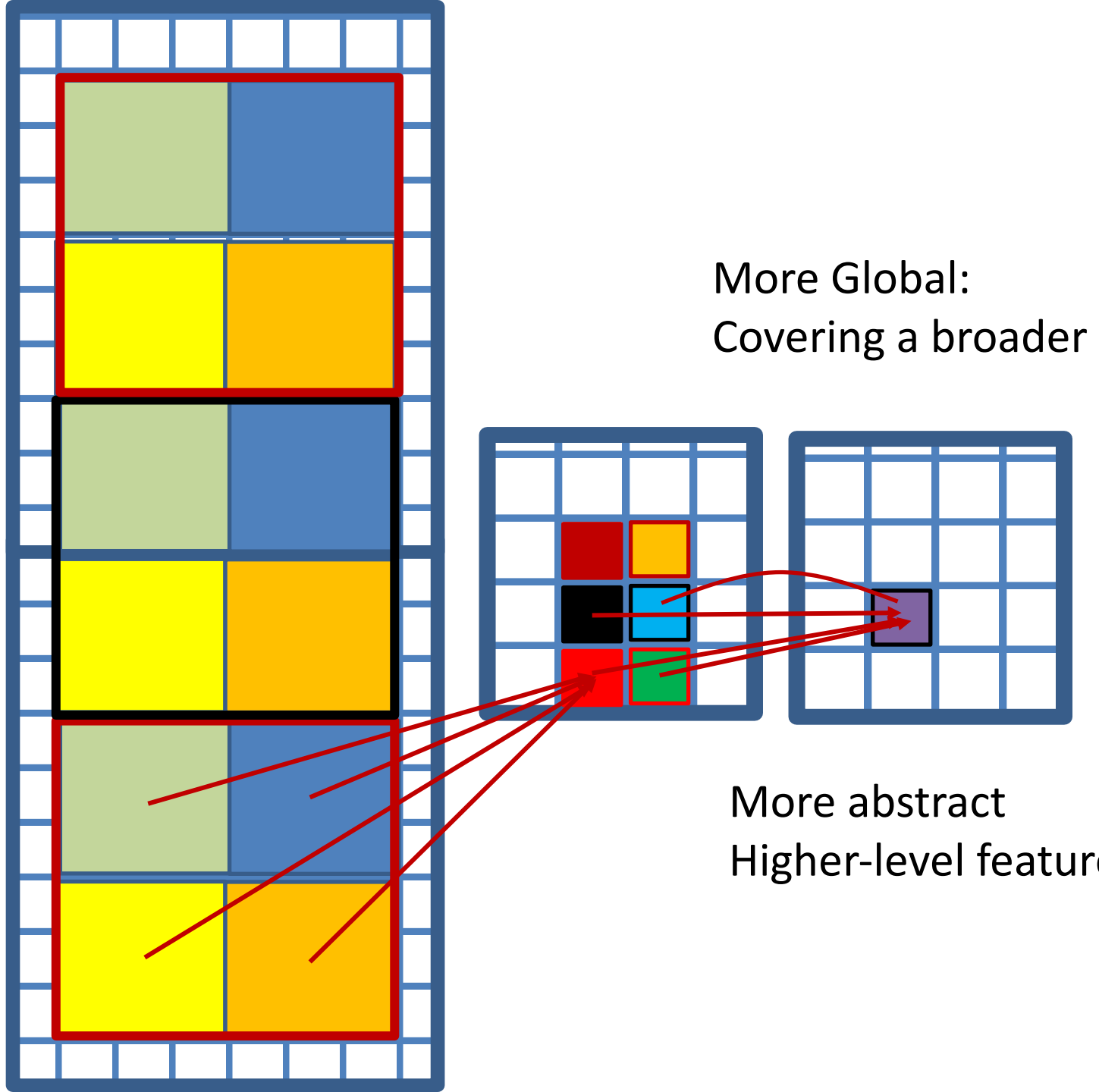
Convolutional Neural Network

Alternating Convolutional Layer and Pooling Layer



- Convolutional layer and pooling layer are paired and repeated several times.
- As the layers proceed, more abstract and higher-level features are detected.

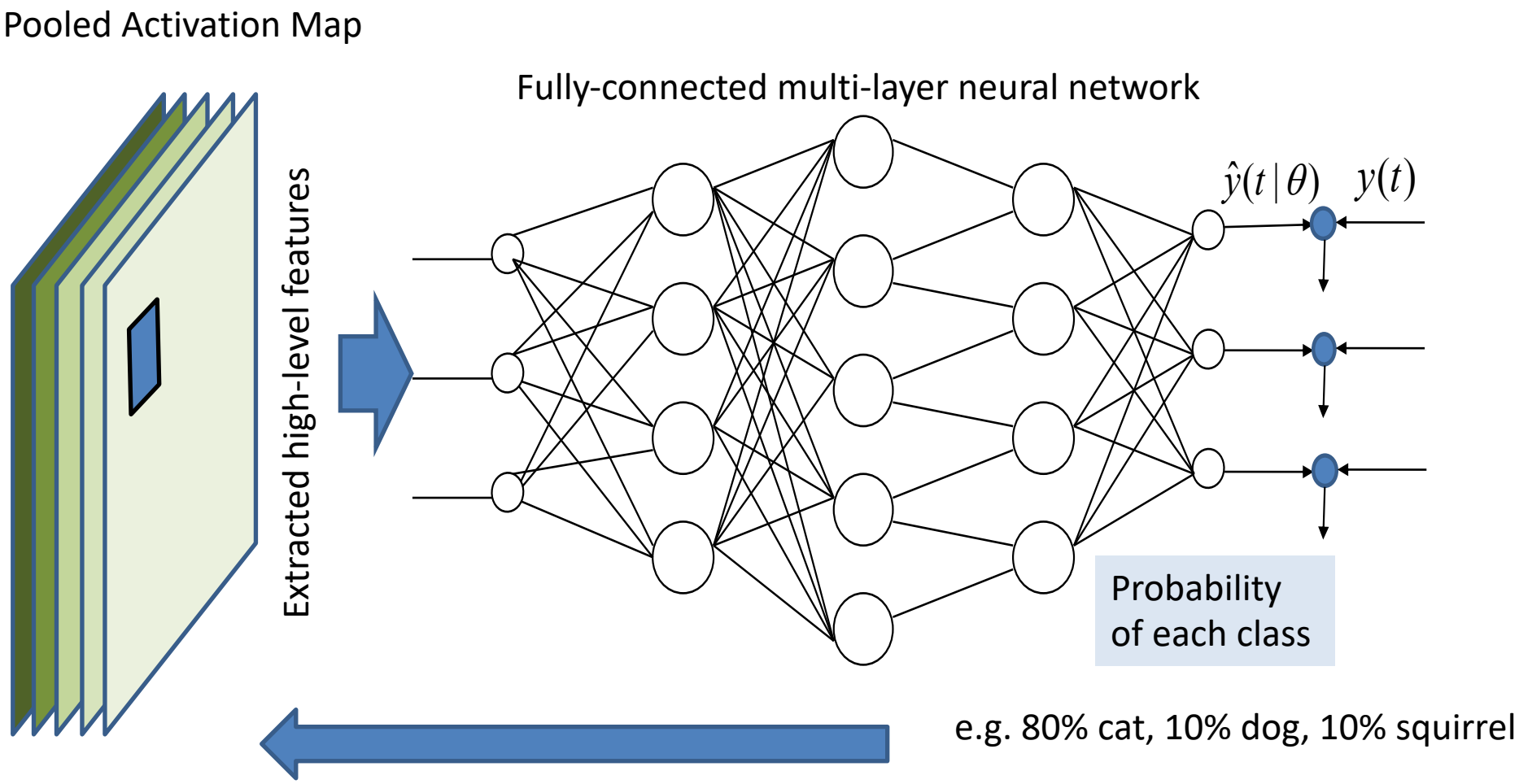
Local



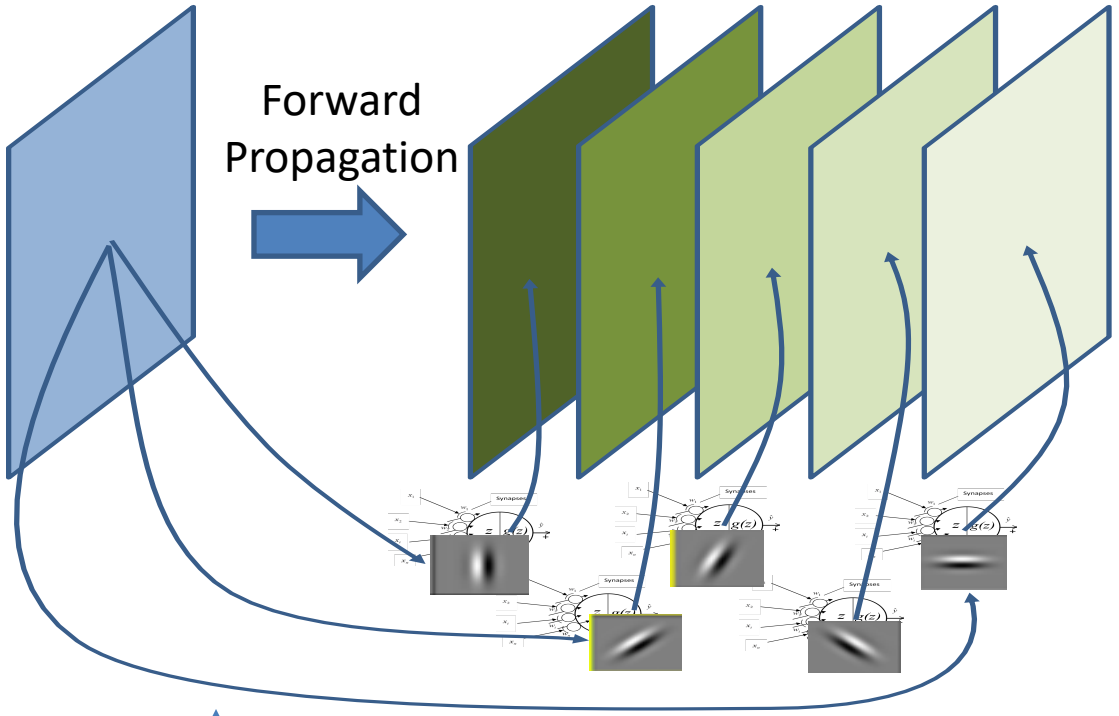
More Global:
Covering a broader area

More abstract
Higher-level features

Classification Layer

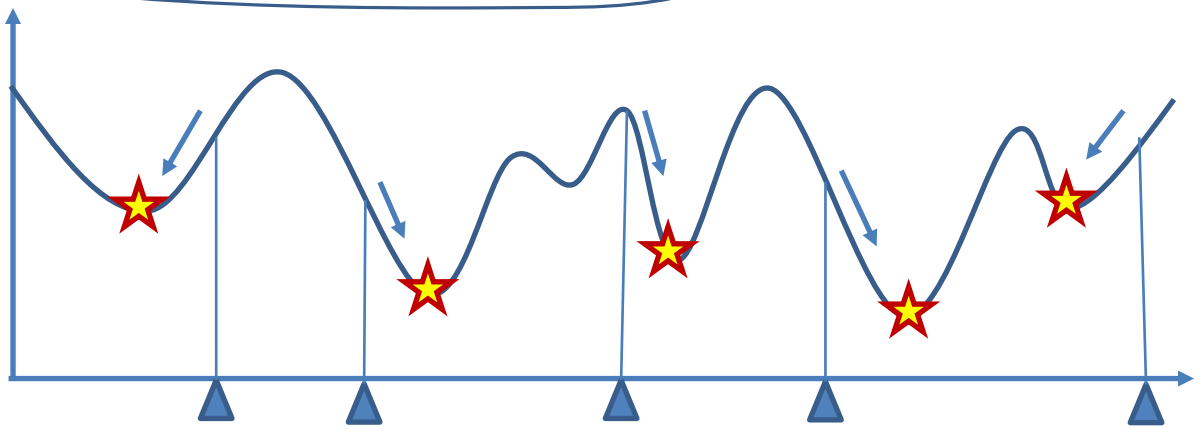


Training of feature detector filters



Error Backpropagation

Randomize the initial conditions (Values of weights) for individual units, and let them converge to local minima.



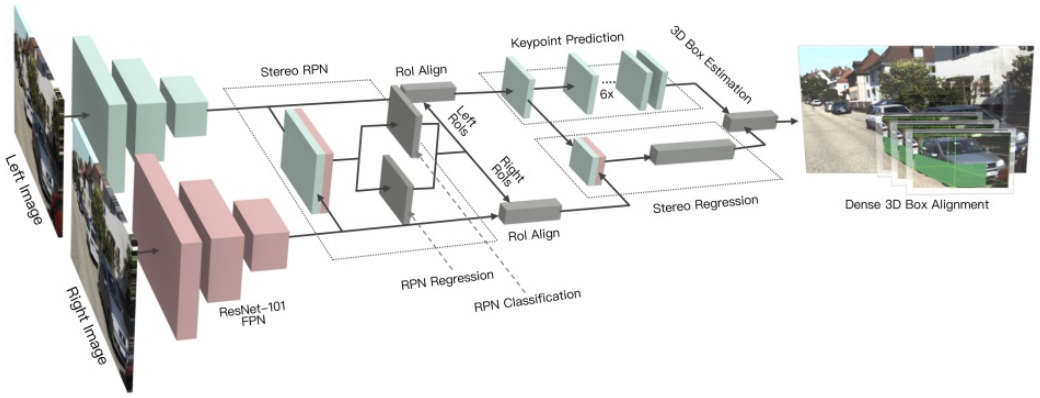
Still challenging, so…

Training a deep neural network based on End-to-End error backpropagation remains a challenge.

- Feature extraction layers can be trained more easily by using existing (already-trained) layers of neural units. We can use the trained weights as “initial conditions” for the feature extraction training for a specific class of images (data).
- The classification layer training can be performed by training layer by layer using the auto-encoding technique.
- Recent trend is more relying on the computing power that increased dramatically.

Summary of CNN

- CNN feature extraction layers seek features through spatially local correlation/convolution in a receptive field.
- This is made possible by limiting the connectivity and structuring the network where individual units are connected only to specific group of units.
- The structure is described with specific parameters, called hyper-parameters:
  - Filter / kernel size: width, height, depth
  - Filter count
  - Stride: Horizontal and vertical
  - (Zero-padding handling edge / boarder effect)
- Use of CNN further requires specifications of
  - Pooling: Max Pooling, Average Pooling
  - Pooling filter size
  - The number of alternating convolutional layers and pooling layers
  - The structure hyperparameters associated with classification layers.



Time Series Data Analysis

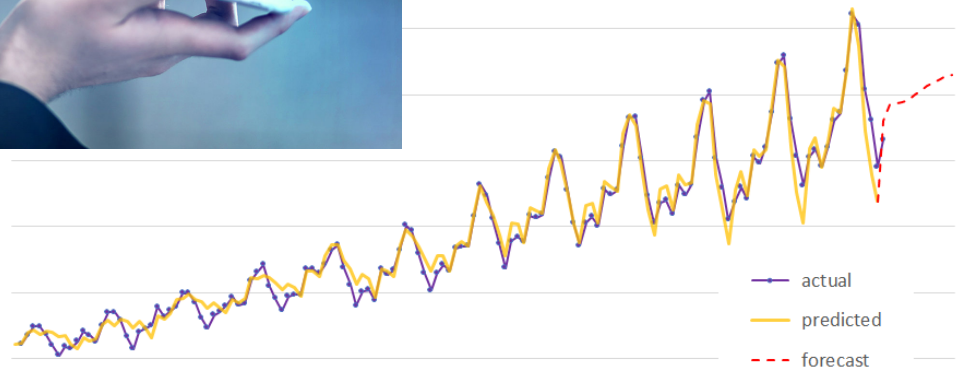
A central theme of 2.160

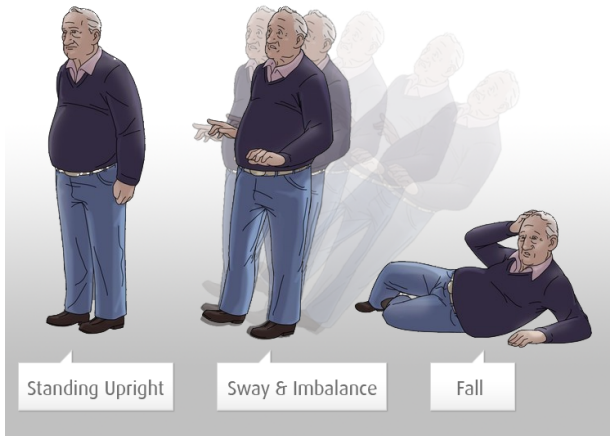
- ❑ Voice recognition
- ❑ Language translation
- ❑ Video processing
- ❑ Word completion

- ❑ Stock market prediction
- ❑ Airline passenger prediction

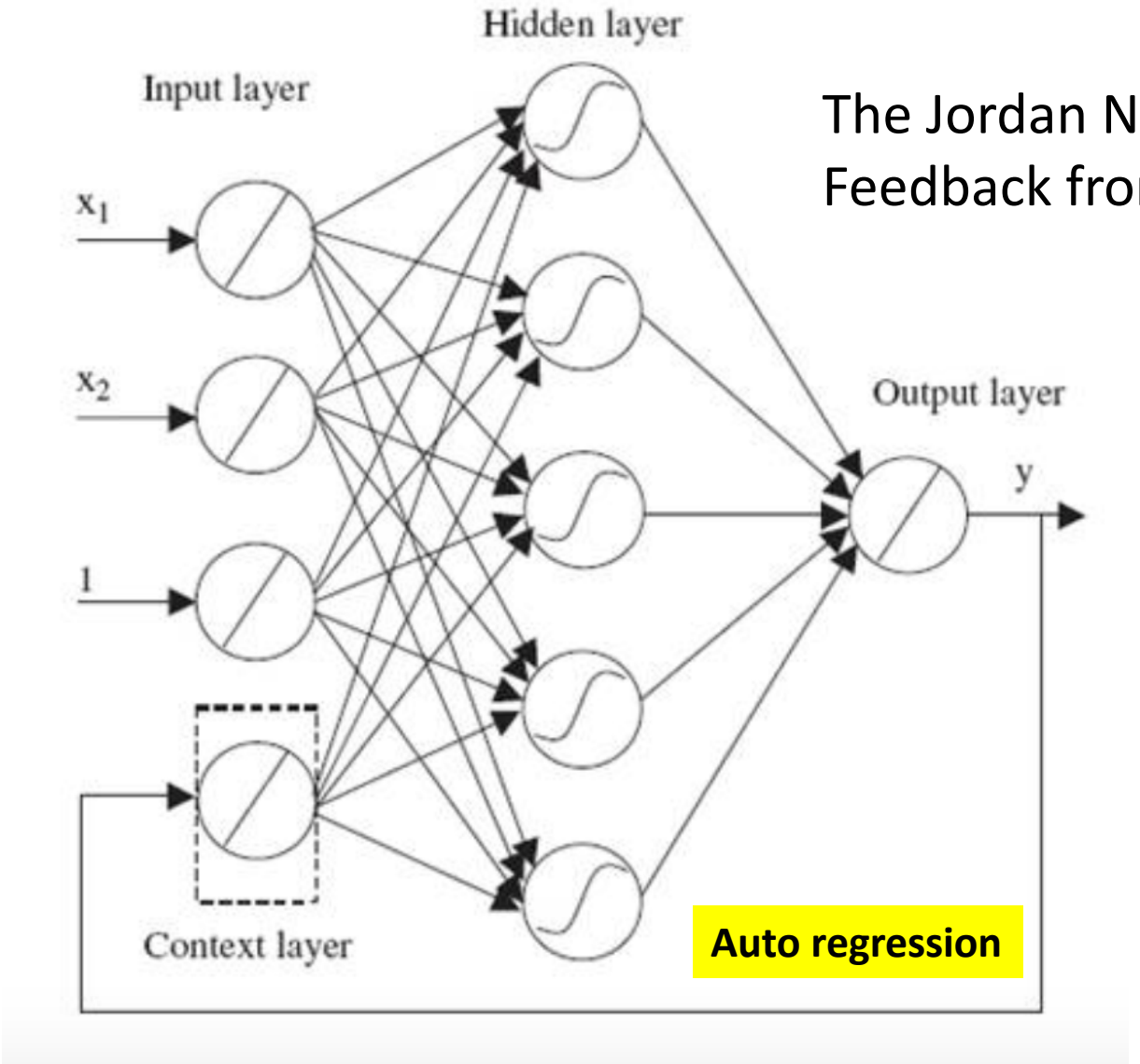
- ❑ Weather forecast
- ❑ Ocean monitoring

- ❑ Cardiovascular monitoring
- ❑ Fall prediction





Recurrent Neural Network



The Jordan Network:
Feedback from the output unit.

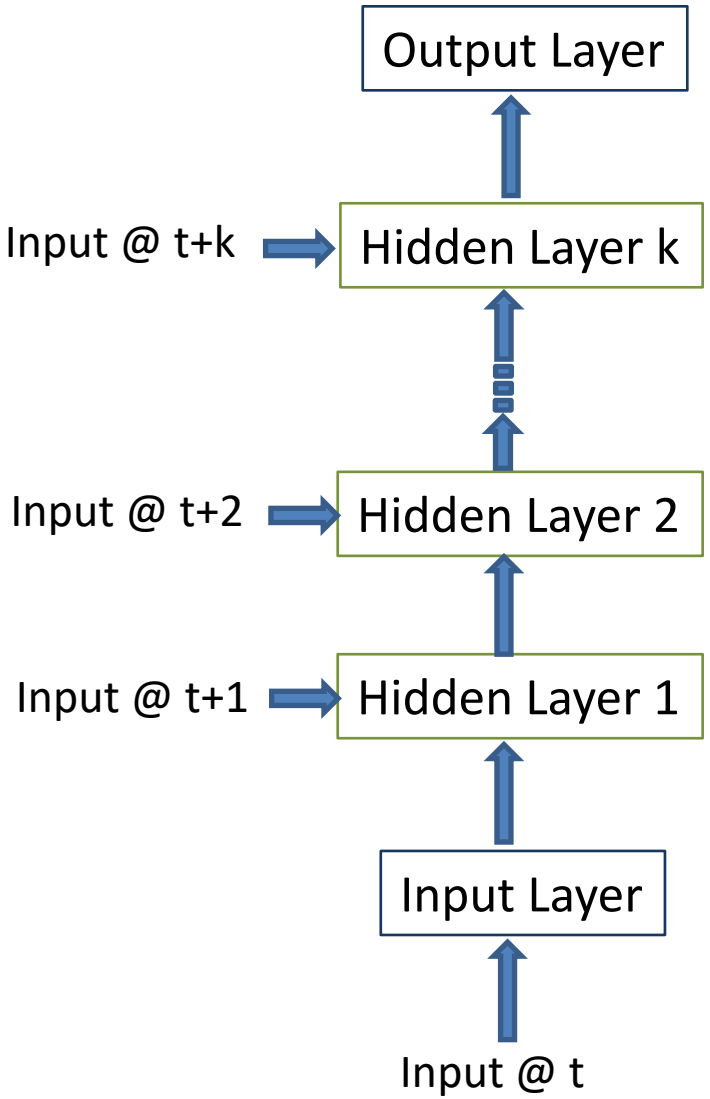
Elmer Network, LSTM, GRU: Feedback from hidden units.

Recurrent Neural Network in a Nutshell: Vanilla Elmar Network

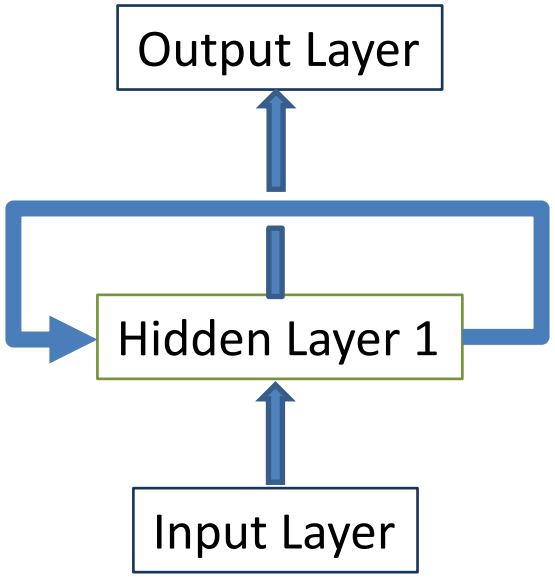
- Consider a multi-layer neural network with T-hidden layers, receiving a discrete-time input sequence,

  u(t), t = 1,2,3,…, T

- In the k-th hidden layer, neural units receive signals from the previous hidden layer, k-1, as well as from the (t+k)-th input, u(t+k).
- Blending both signals, the hidden units produce outputs for the next layer until it reaches the output layer.

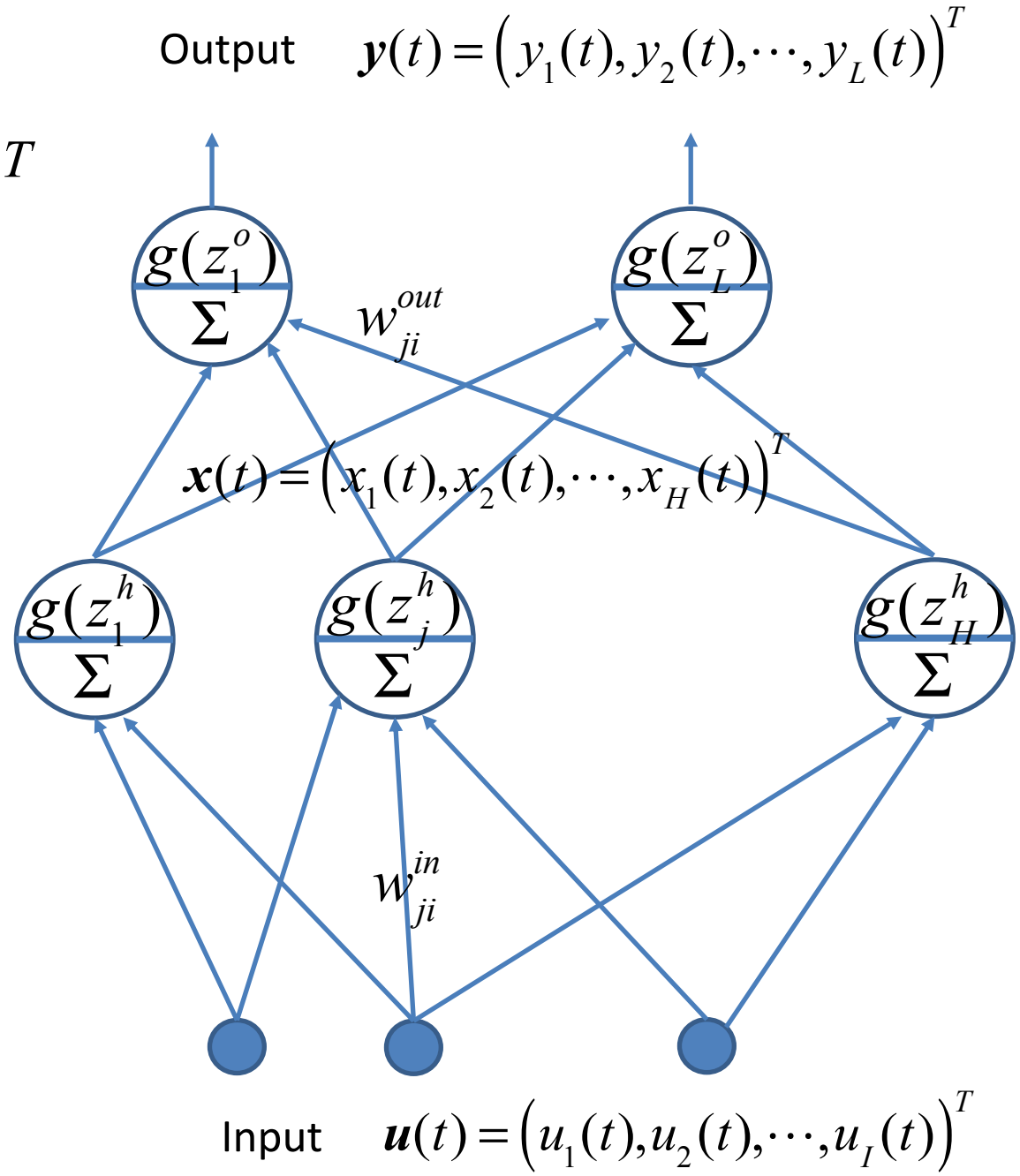


- If we want to deal with an input sequence having an arbitrary length, this architecture is not appropriate.
- Assuming all weights in the hidden layers are the same, this multi-layer network can be folded down to the recurrent network below.



Notation

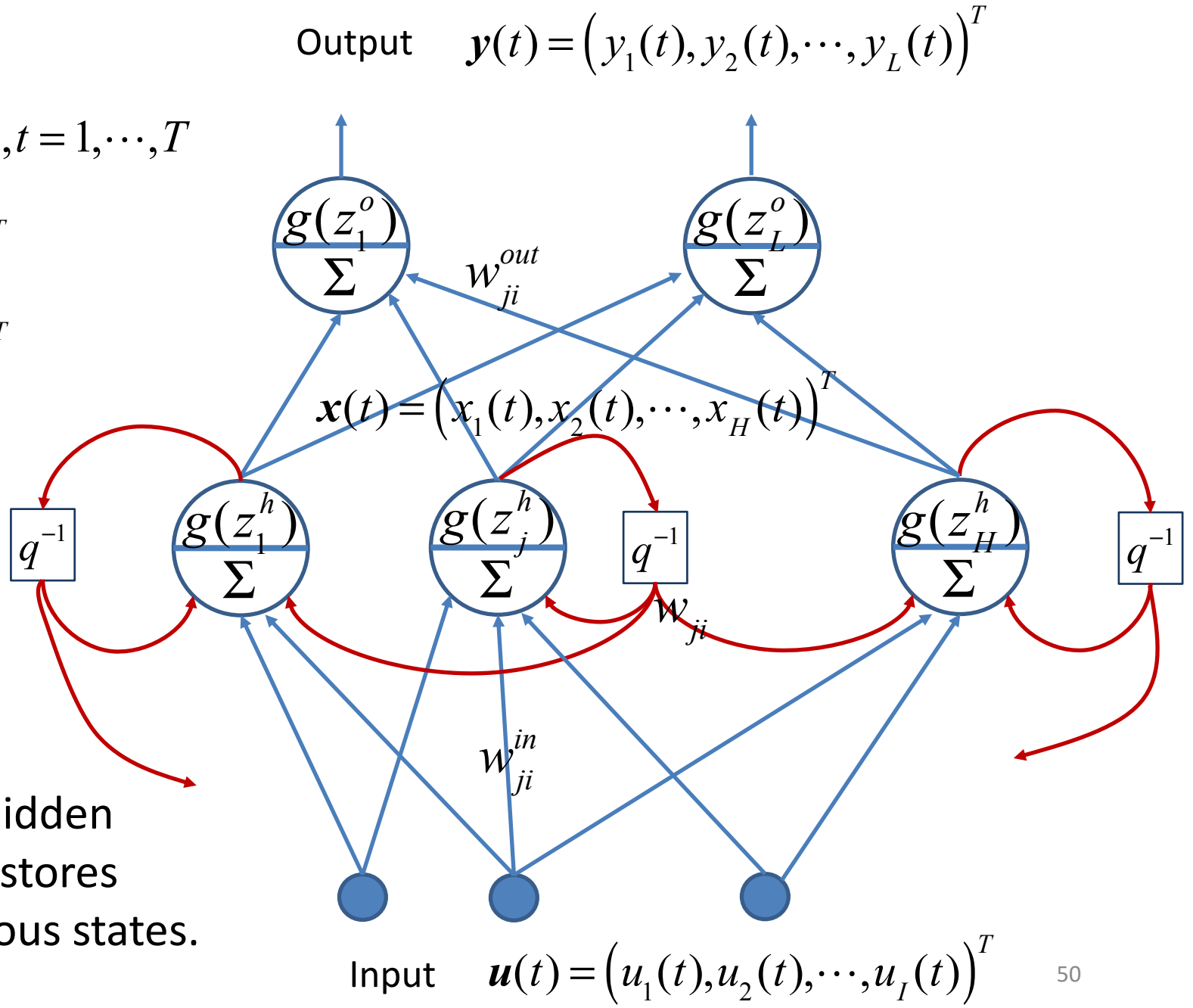
- Input:
$$\boldsymbol{u}(t) = \left(u_1(t), u_2(t), \cdots, u_I(t)\right)^T, t = 1, \cdots, T$$
- Hidden unit outputs (State):
$$\boldsymbol{x}(t) = \left(x_1(t), x_2(t), \cdots, x_H(t)\right)^T$$
- Output:
$$\boldsymbol{y}(t) = \left(y_1(t), y_2(t), \cdots, y_L(t)\right)^T$$
- Weight from input unit i to hidden unit j: w_{ji}^{in}
- Weight from hidden unit i to hidden unit j:
- Weight from hidden unit i to output unit j: w_{ji}^{out}



Notation

- Input:
$$\boldsymbol{u}(t) = \left(u_1(t), u_2(t), \cdots, u_I(t)\right)^T, t = 1, \cdots, T$$
- Hidden unit outputs (State):
$$\boldsymbol{x}(t) = \left(x_1(t), x_2(t), \cdots, x_H(t)\right)^T$$
- Output:
$$\boldsymbol{y}(t) = \left(y_1(t), y_2(t), \cdots, y_L(t)\right)^T$$
- Weight from input unit i to hidden unit j: w_{ji}^{in}
- Weight from hidden unit i to hidden unit j: w_{ji}
- Weight from hidden unit i to output unit j: w_{ji}^{out}

The hidden layer stores previous states.



Forward Pass Computation

- Hidden units

$$z_j^h(t) = \sum_{i=1}^{I} w_{ji}^{in} u_i(t) + \sum_{k=1}^{H} w_{jk} x_k(t-1)$$

- Hidden unit outputs (State):

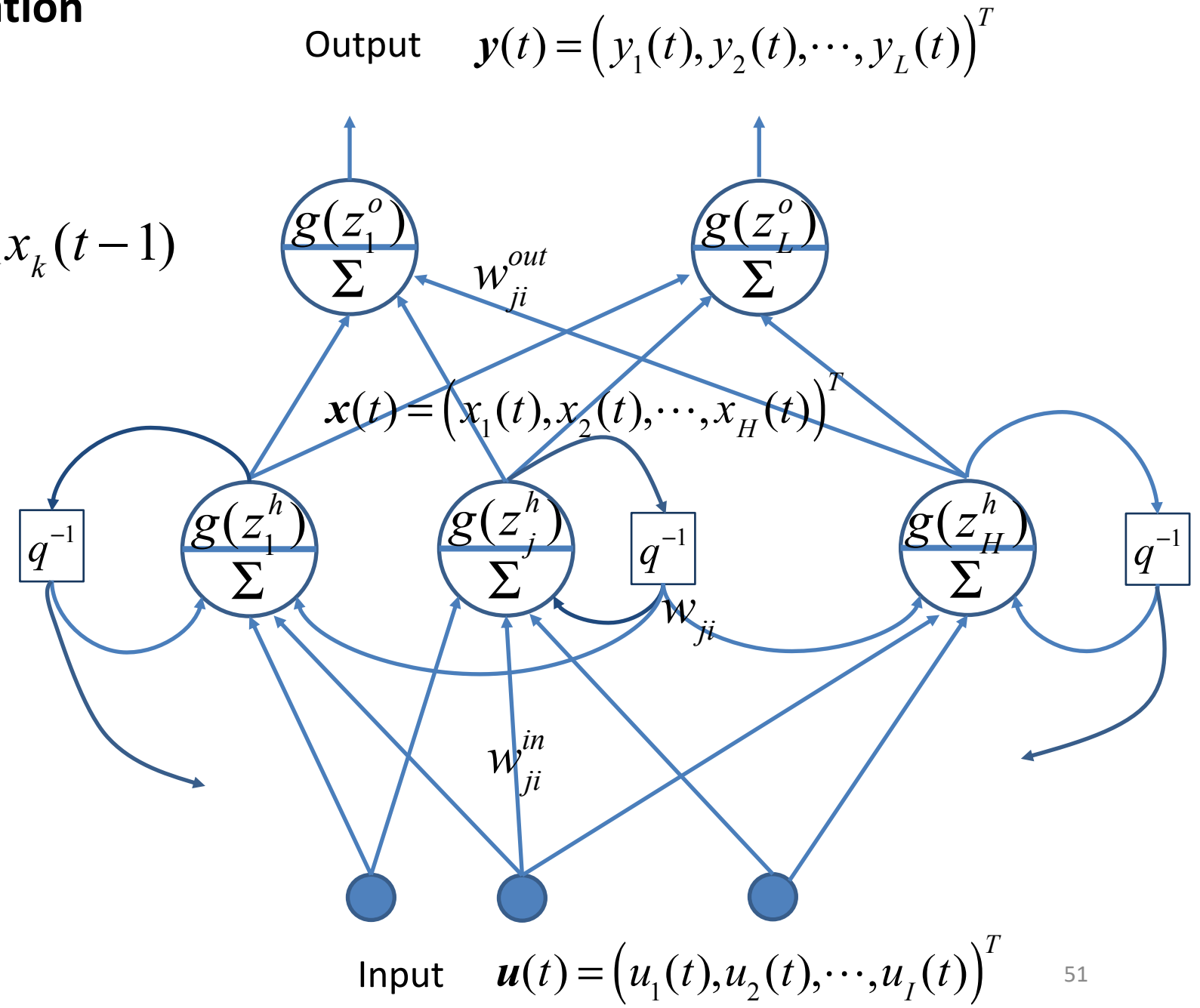
$$x_j(t) = g(z_j^h(t))$$

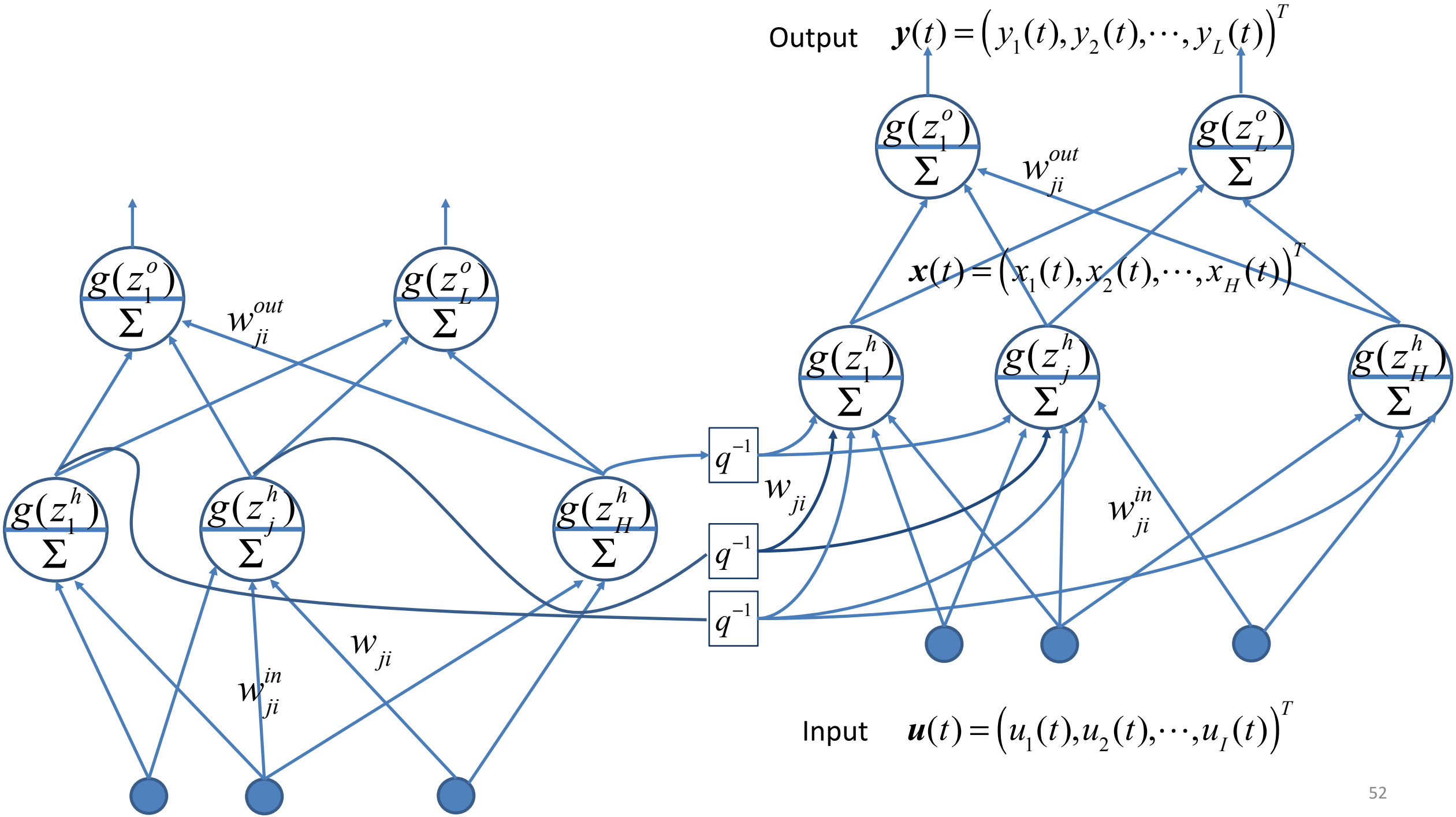
- Output units:

$$z_j^o(t) = \sum_{i=1}^{H} w_{ji}^{out} x_i(t)$$

- Outputs

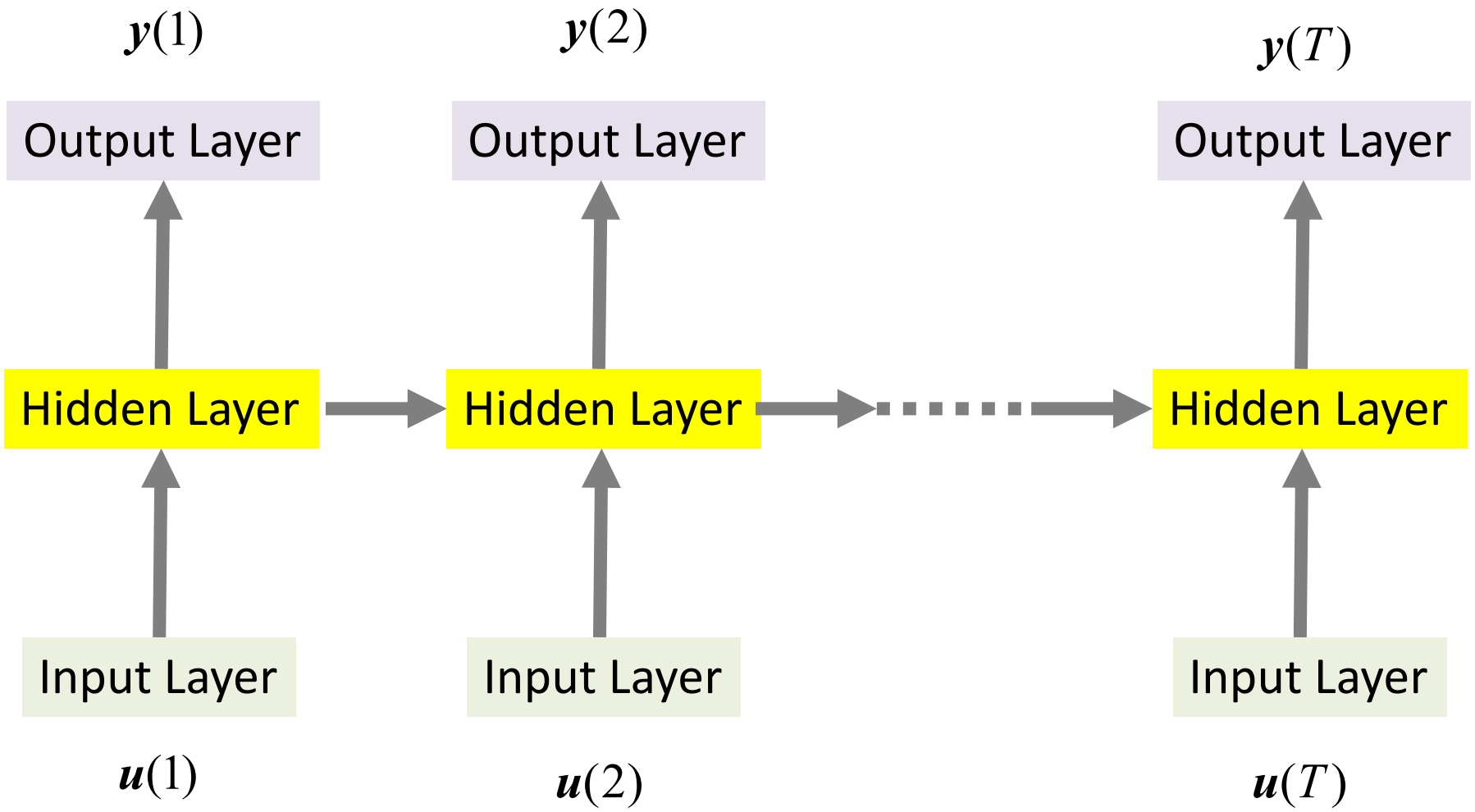
$$y_j(t) = g(z_j^o(t))$$




Output
$\boldsymbol{y}(t) = \left(y_1(t), y_2(t), \cdots, y_L(t)\right)^T$
$g(z_1^o)$
$g(z_L^o)$
Σ
w_{ji}^{out}
$\boldsymbol{x}(t) = \left(x_1(t), x_2(t), \cdots, x_H(t)\right)^T$
$g(z_1^h)$
$g(z_j^h)$
$g(z_H^h)$
q^{-1}
w_{ji}
w_{ji}^{in}
Input
$\boldsymbol{u}(t) = \left(u_1(t), u_2(t), \cdots, u_I(t)\right)^T$

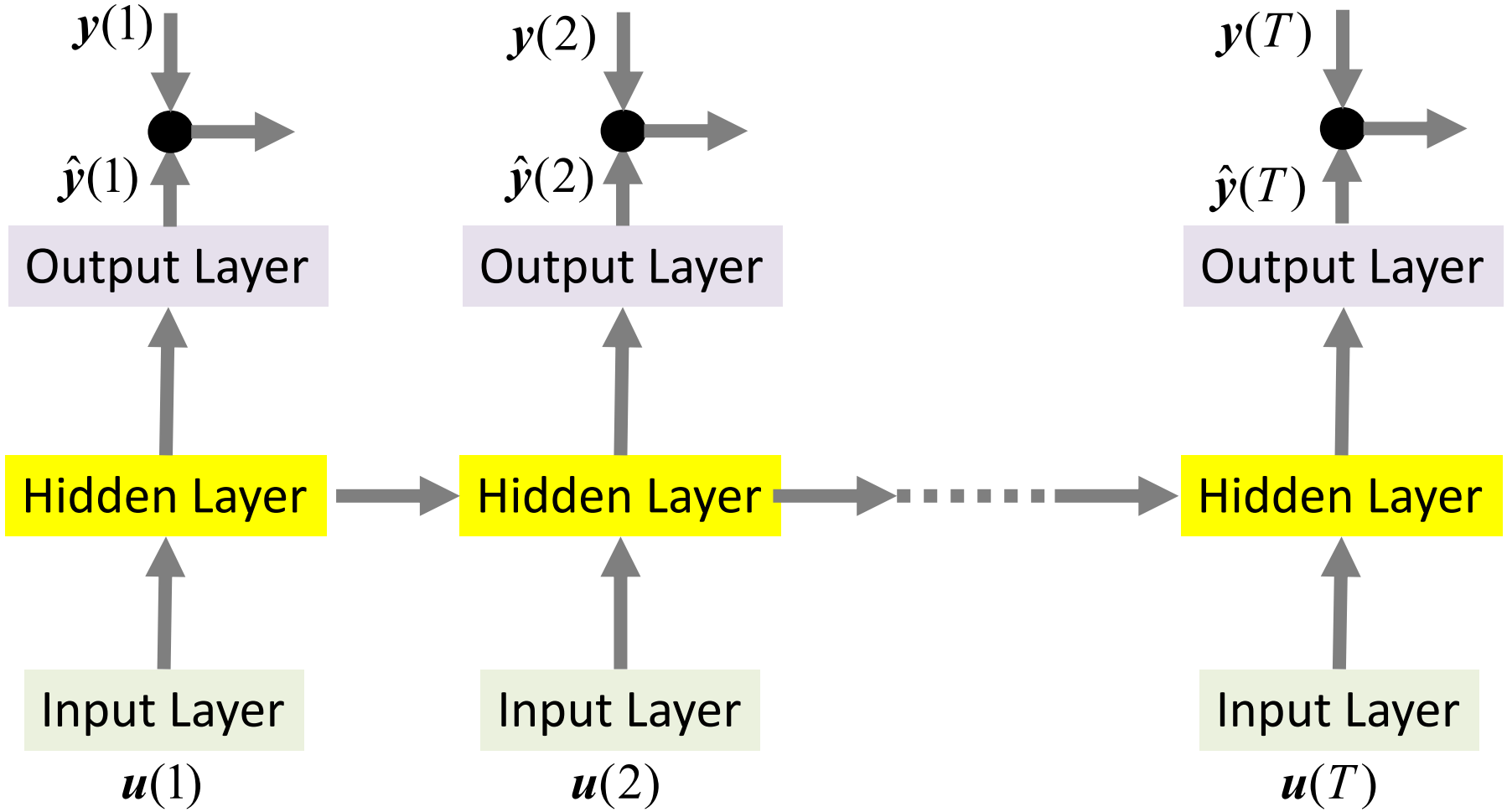
Unfolding the RNN in time

- Unfolding the RNN in time by stacking the identical copies of the RNN and redirecting connections within the network to obtain connections between subsequent copies.



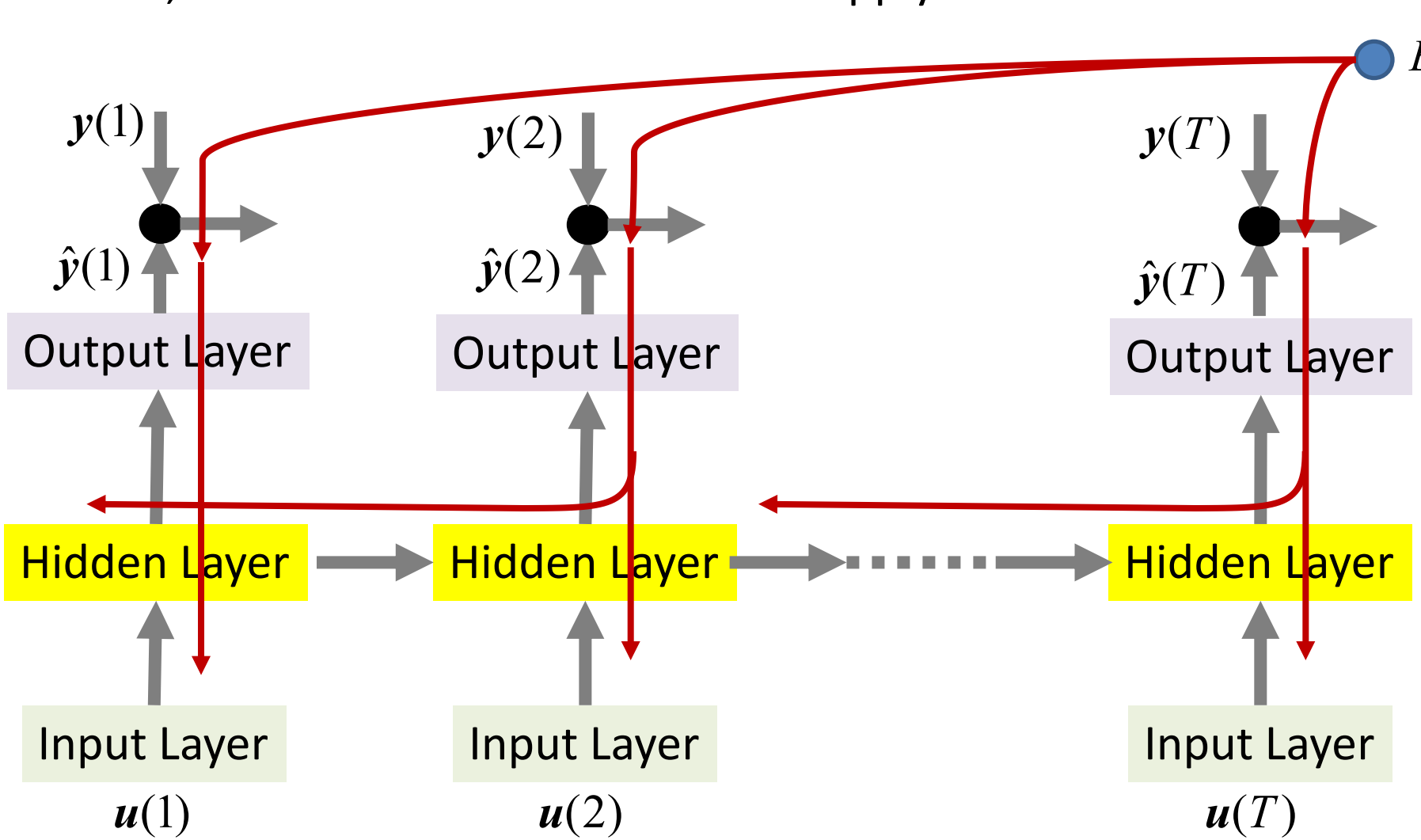
Training of RNN

- Target output (training data) is given to each output: $\boldsymbol{y}(1)\ \boldsymbol{y}(2)\cdots\ \boldsymbol{y}(T)$
- Loss Function to minimize $E=\sum_{t=1}^{T}\left|\boldsymbol{y}(t)-\hat{\boldsymbol{y}}(t)\right|^2=\sum_{t=1}^{T}E(t)$



Back Propagation Through Time (BPTT)

- The standard Error Back Propagation algorithm cannot be directly applied to RNN.
- However, the unfolded RNN allows us to apply the same chain rule.



Back Propagation Through Time (BPTT)

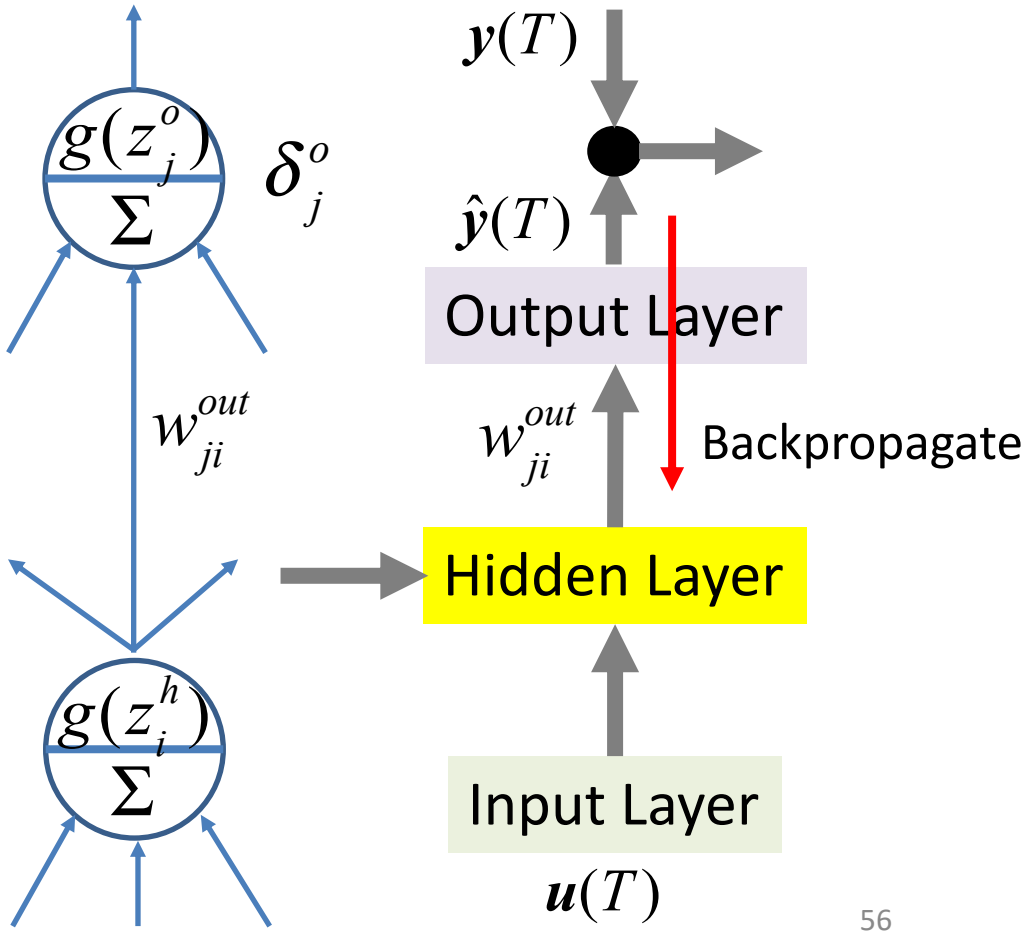
$$\delta_j = -\frac{\partial E}{\partial z_j} \qquad E = \frac{1}{2}\sum_{t=1}^{T}\left|\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right|^2$$

- We start with the final time layer T, where output $\boldsymbol{y}(T)$ is directly provided:
  - Output units in time layer T

$$\delta_j^o = -\frac{\partial E}{\partial z_j^o} = (y_j(T) - \hat{y}_j(T))g'(z_j^o(T))$$

  - Hidden units in time layer T

$$\delta_i = -\frac{\partial E}{\partial z_i} = \sum_j \delta_j^o(T) w_{ji}^{out} \cdot g'(z_i(T))$$



Back Propagation Through Time (BPTT)

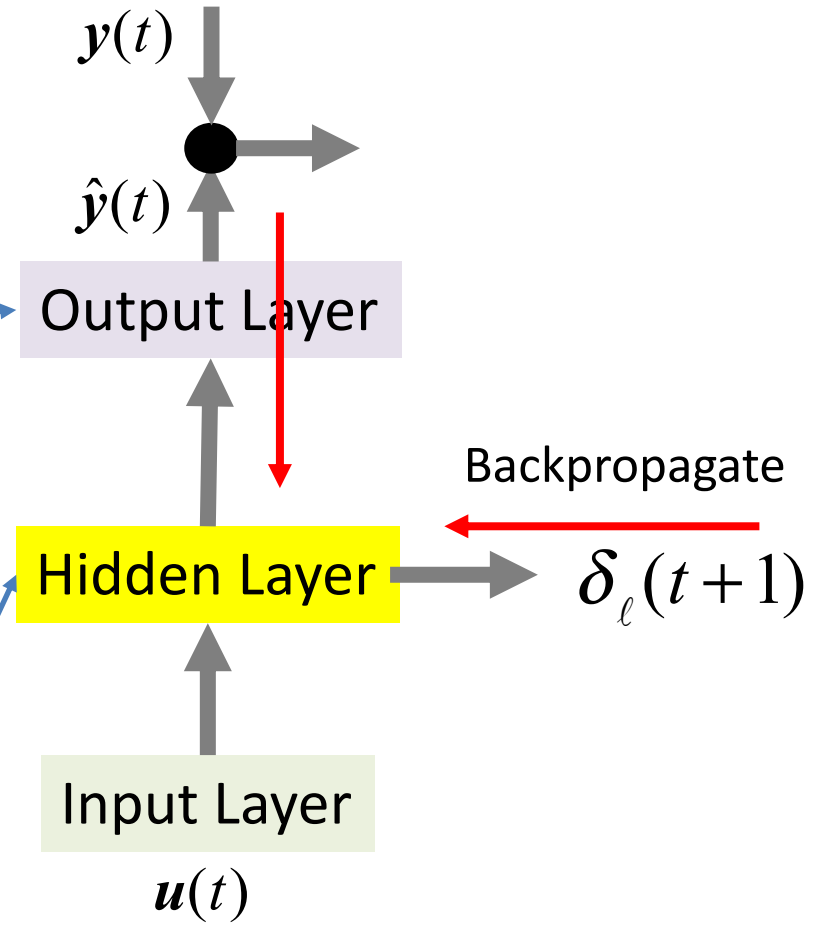
$$\delta_j = -\frac{\partial E}{\partial z_j} \qquad E = \frac{1}{2}\sum_{t=1}^{T}\left|\boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t)\right|^2$$

- Move back to earlier time layer: $1 \le t \le T-1$
  - Output units in time layer t

$$\delta_j^o = (y_j(t) - \hat{y}_j(t))\,g'(z_j^o(t))$$

  - Hidden units in time layer t:

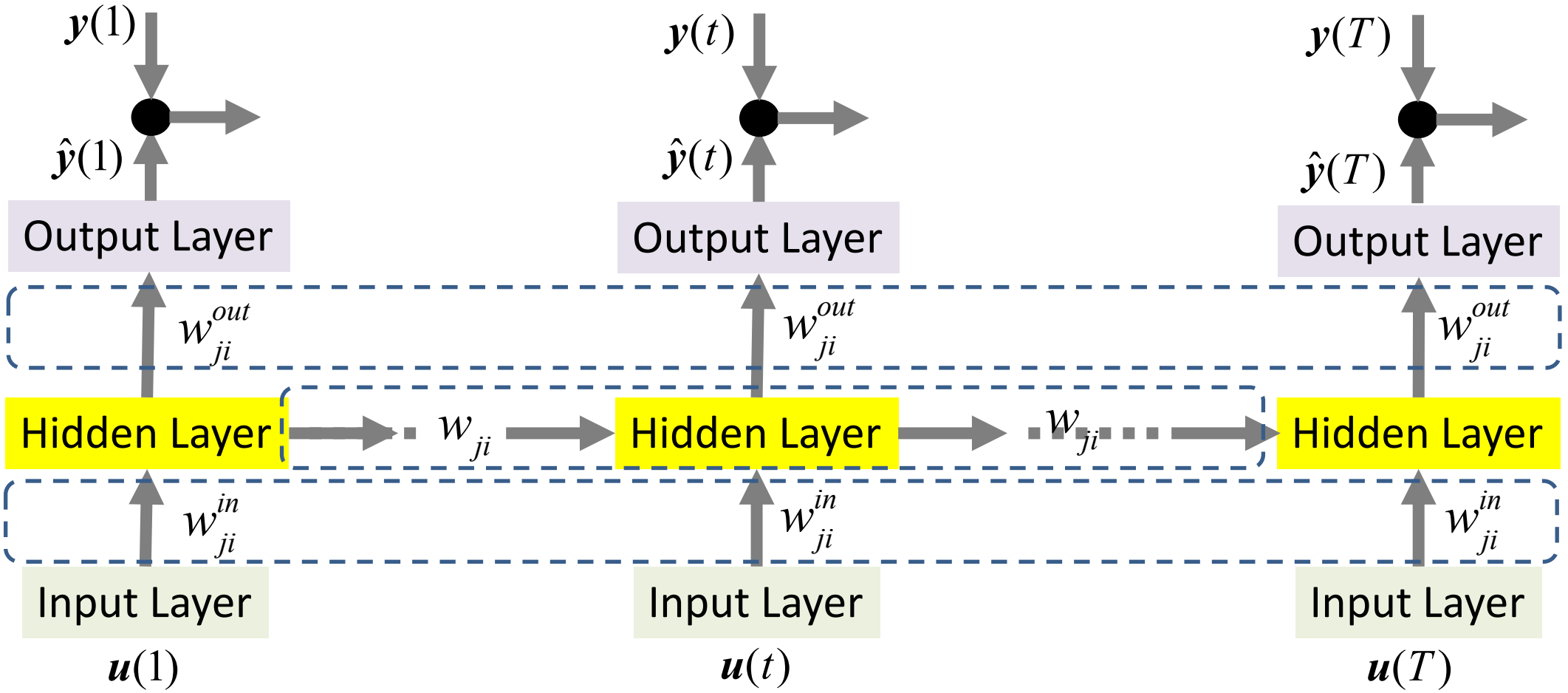
$$\delta_i = \left(\sum_j \delta_j^o(t) w_{ji}^{out} + \sum_\ell \delta_\ell(t+1) w_{\ell i}\right) g'(z_i(t))$$



Weight Changes for BPTT

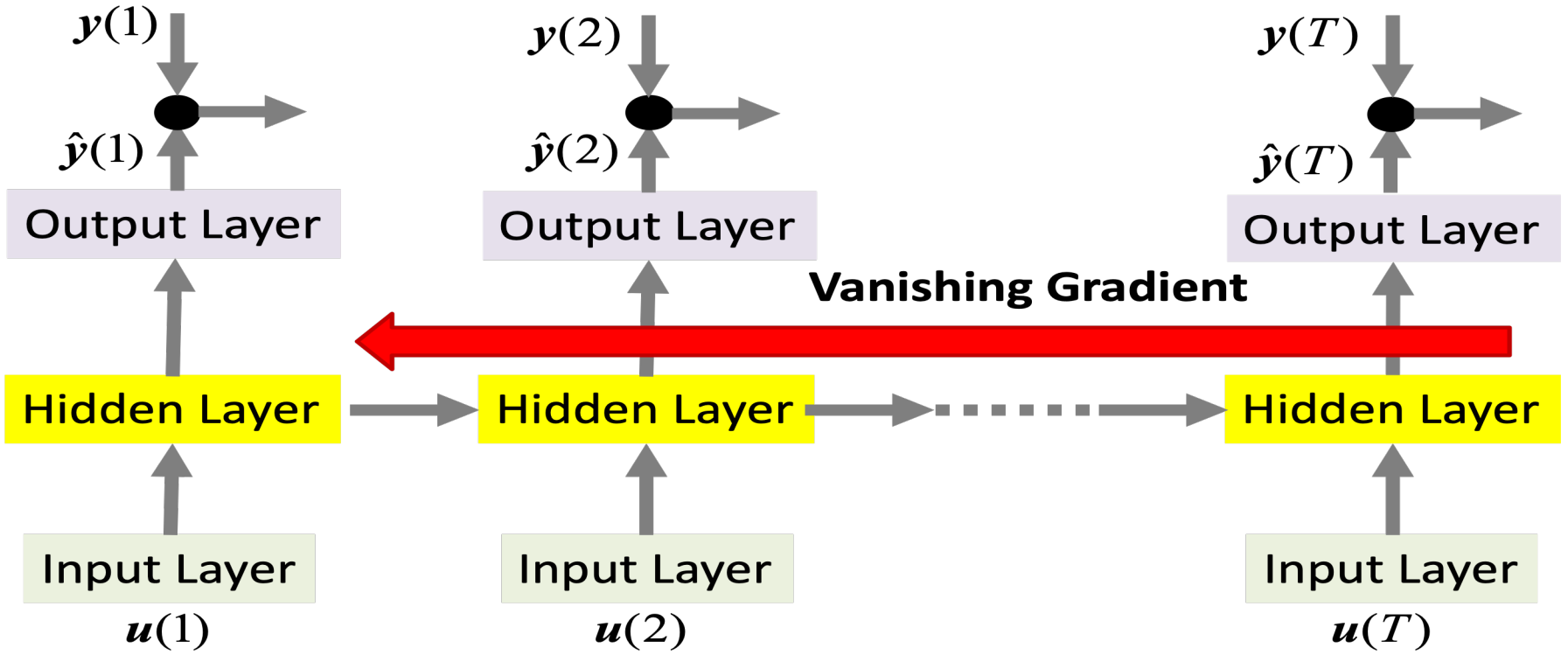
- The units in all time layers share the weights.
- (weight update)=(sum of corrections of all time layers)

$$\Delta w_{ji}^{in} = \rho \sum_{t=1}^{T} \delta_j(t) u_i(t) \qquad \Delta w_{ji} = \rho \sum_{t=1}^{T} \delta_j(t) x_i(t-1) \qquad \Delta w_{ji}^{out} = \rho \sum_{t=1}^{T} \delta_j^o(t) x_i(t)$$



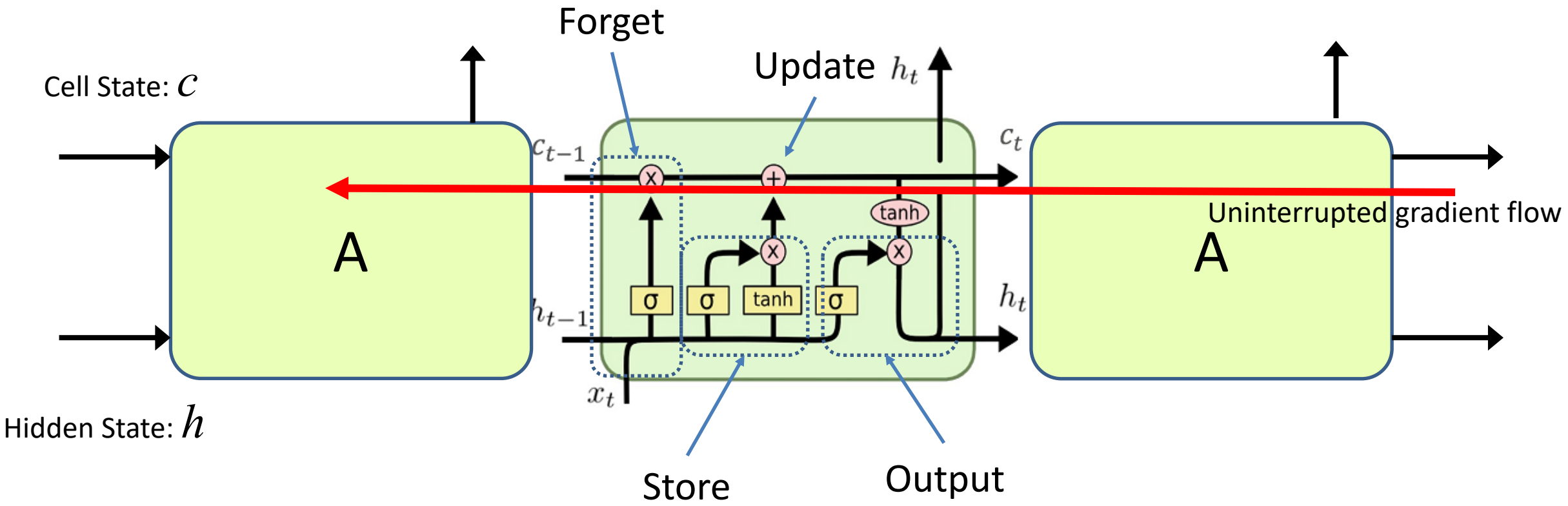
Drawbacks of Back Propagation Through Time (BPTT)

- As the time layers extend, (large T), the error backpropagation through time tends to vanish.
- There are a few techniques and network architectures that have been proven to be effective for coping with the vanishing gradient problem.
- These include Long-Short Term Memory (LSTM) network, which has been used extensively.



Long Short Term Memory network architecture

- ❑ More control over information transmitted
- ❑ Discard irrelevant information based on new input and the previous state h_{t-1}, Forget
- ❑ Store relevant information taken from new input and the previous state: Store and Update
- ❑ Maintain an uninterrupted gradient flow : Separate cell states from outputs → Highway



Reflection

- Strong points of deep neural nets
  - Error backpropagation with improved activation functions, e.g. ReLU, and control of information flow (LSTM)
  - Automatic feature extraction through end-to-end learning, e.g. CNN
  - Focused connection and hierarchical structure (CNN)
  - Capturing of time series information (RNN)
- Clever, but unsure
  - Early stopping for preventing over fitting
  - A lot of hacks: Dropout; Momentum term; Randomized weights, Data augmentation, etc.
  - Altering convolution and pooling
  - The lack of fundamental theory: unaccountable results