Australia's Global University

COMP9321

Data Services Engineering

Term 1, 2019

Week 7 Lecture 2

Outline

Supervised Learning

- Convolutional Neural Network
- Sequence Modelling: RNN and its extensions

Unsupervised Learning

- Autoencoders
- Deep Autoencoders

Building blocks of DL

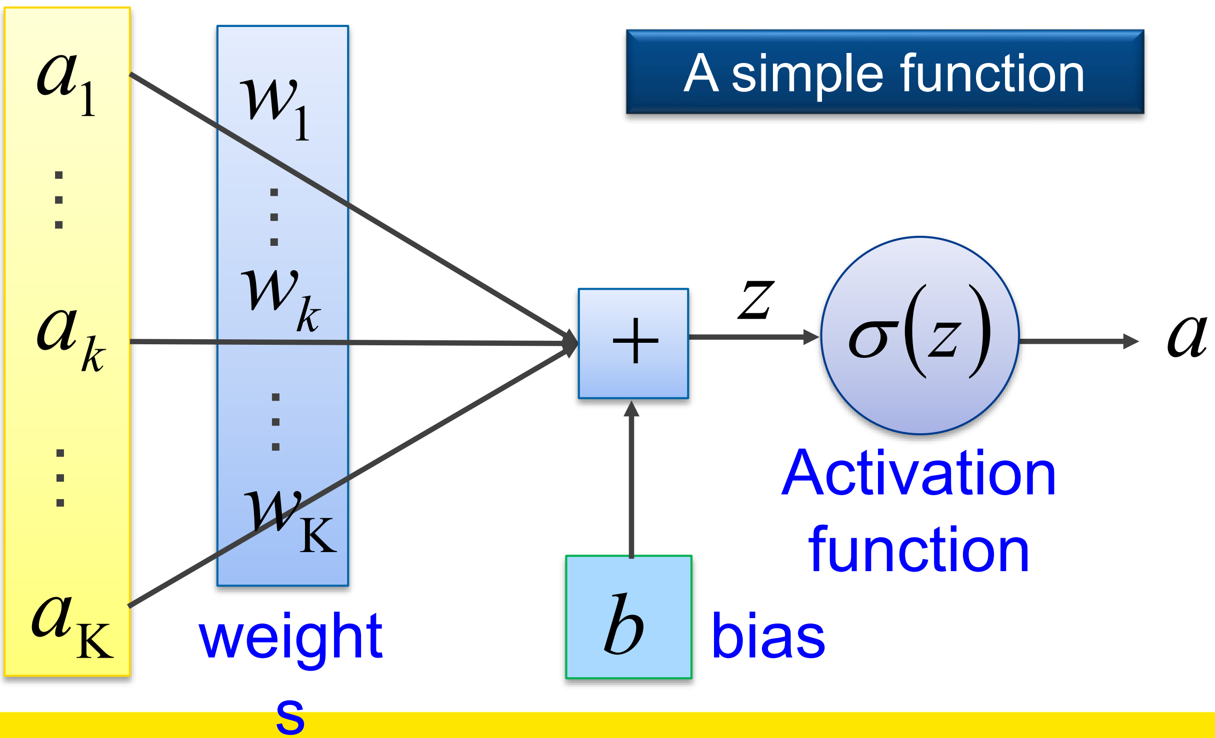
Linear Combinations and No-linear activation functions.

Deep learning is a composition of many functions

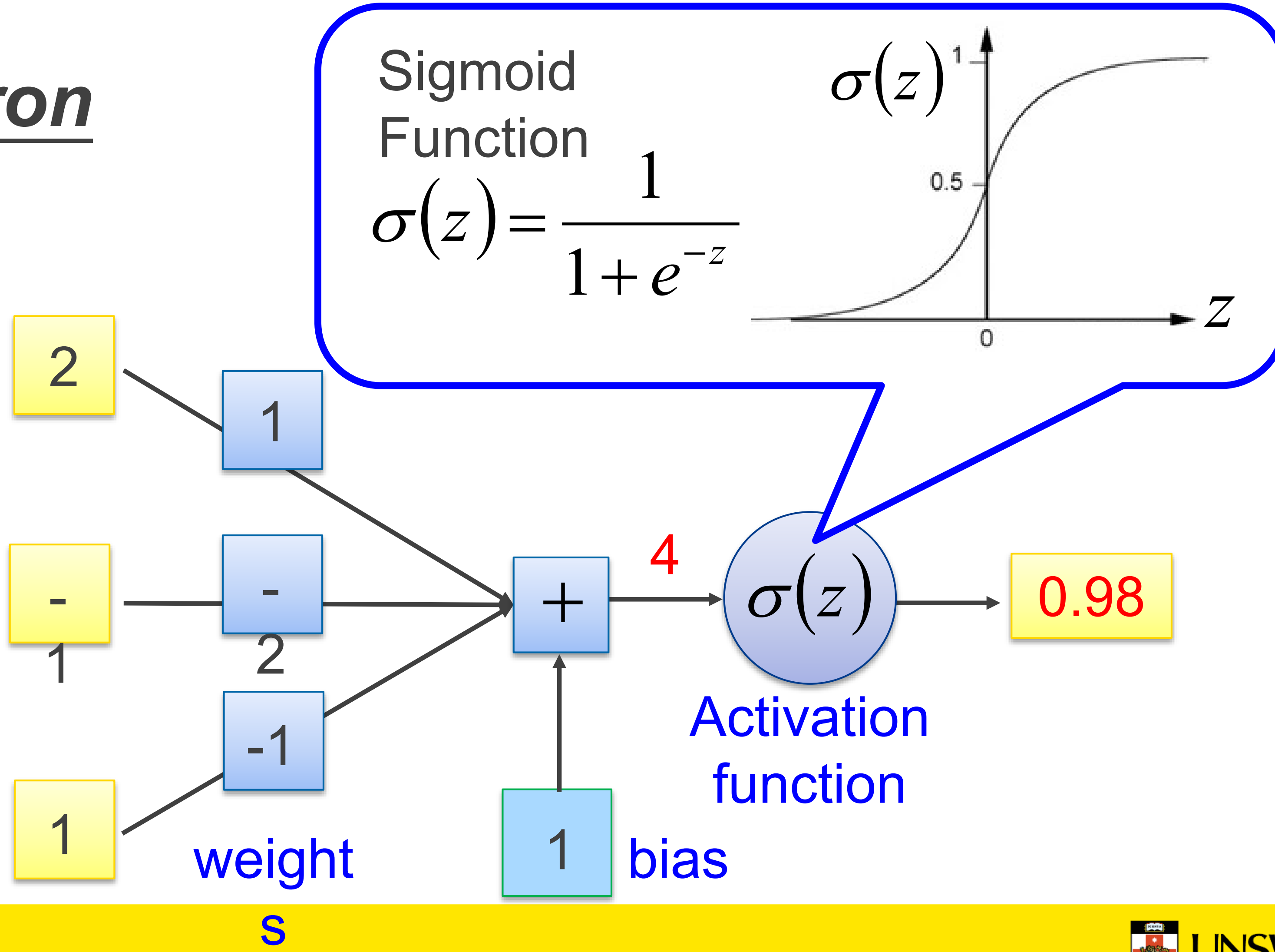
The gradients can be propagated by using chain rule

Neuron

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$

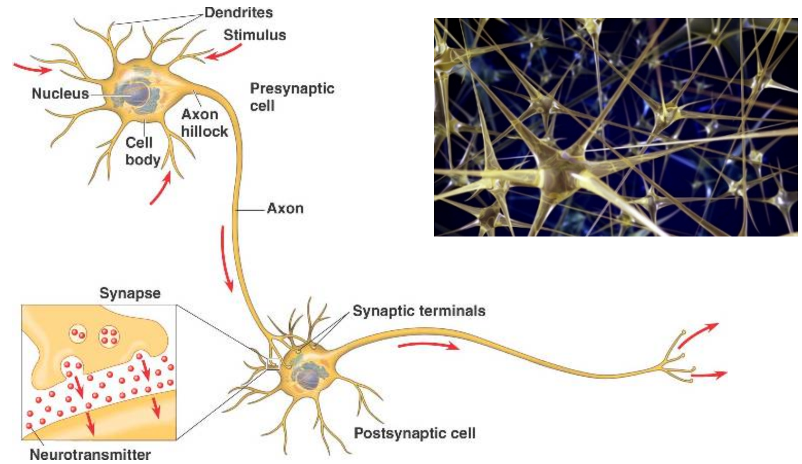


Neuron

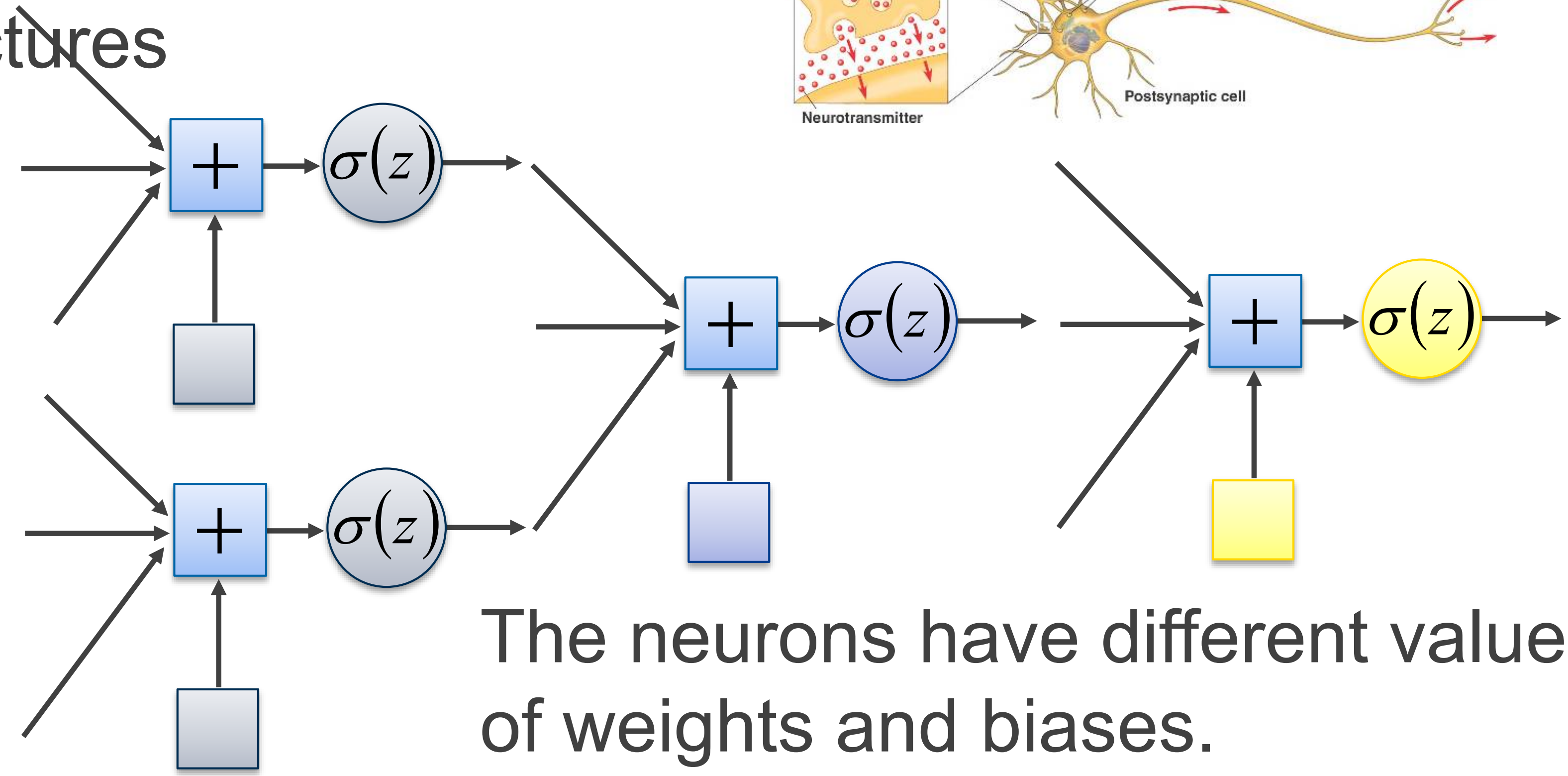


Sigmoid
Function
$\sigma(z) = \frac{1}{1+e^{-z}}$
$\sigma(z)$
1
0.5
0
z
2
1
-
-
1
2
-1
1
weight
s
+
1
bias
4
$\sigma(z)$
Activation
function
0.98

Neural Network



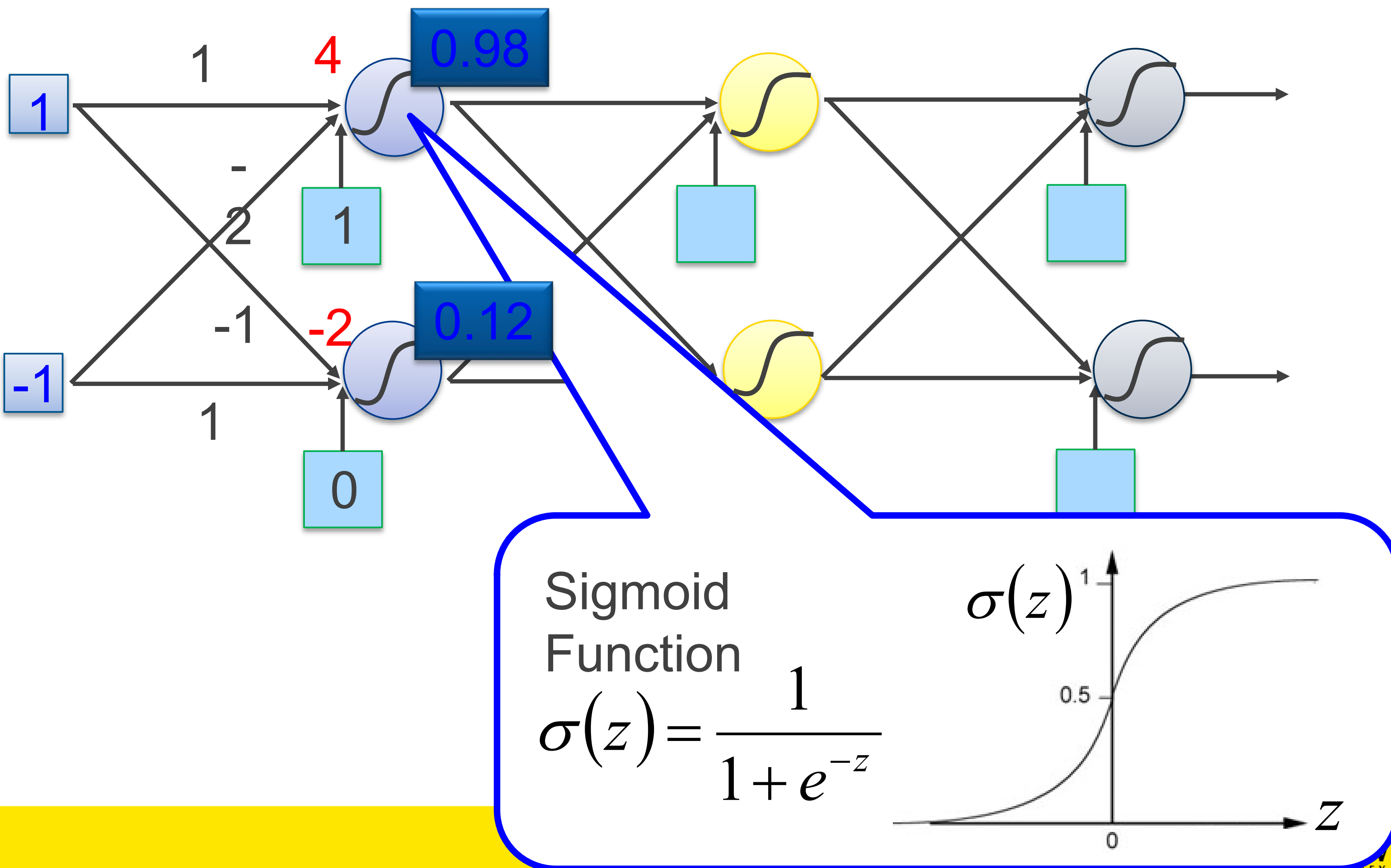
Different connections lead to different network structures



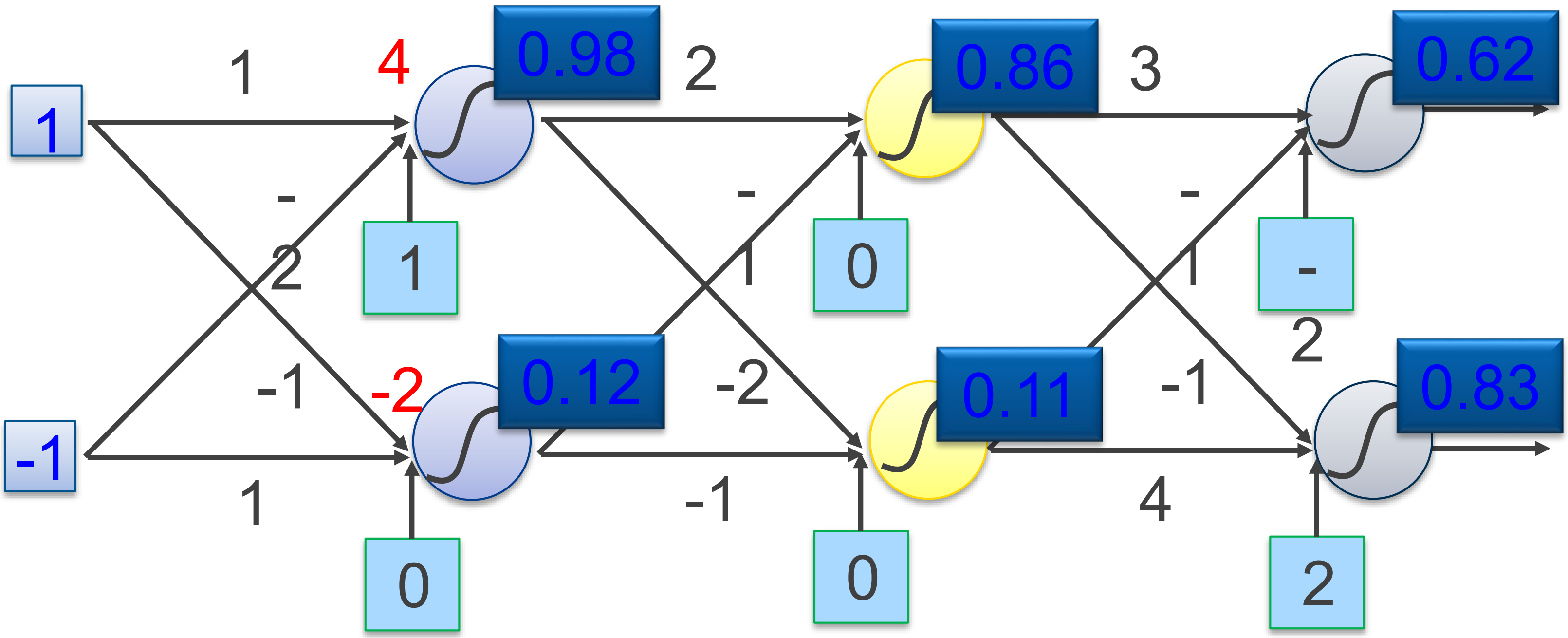
The neurons have different values of weights and biases.

Weights and biases are network parameters

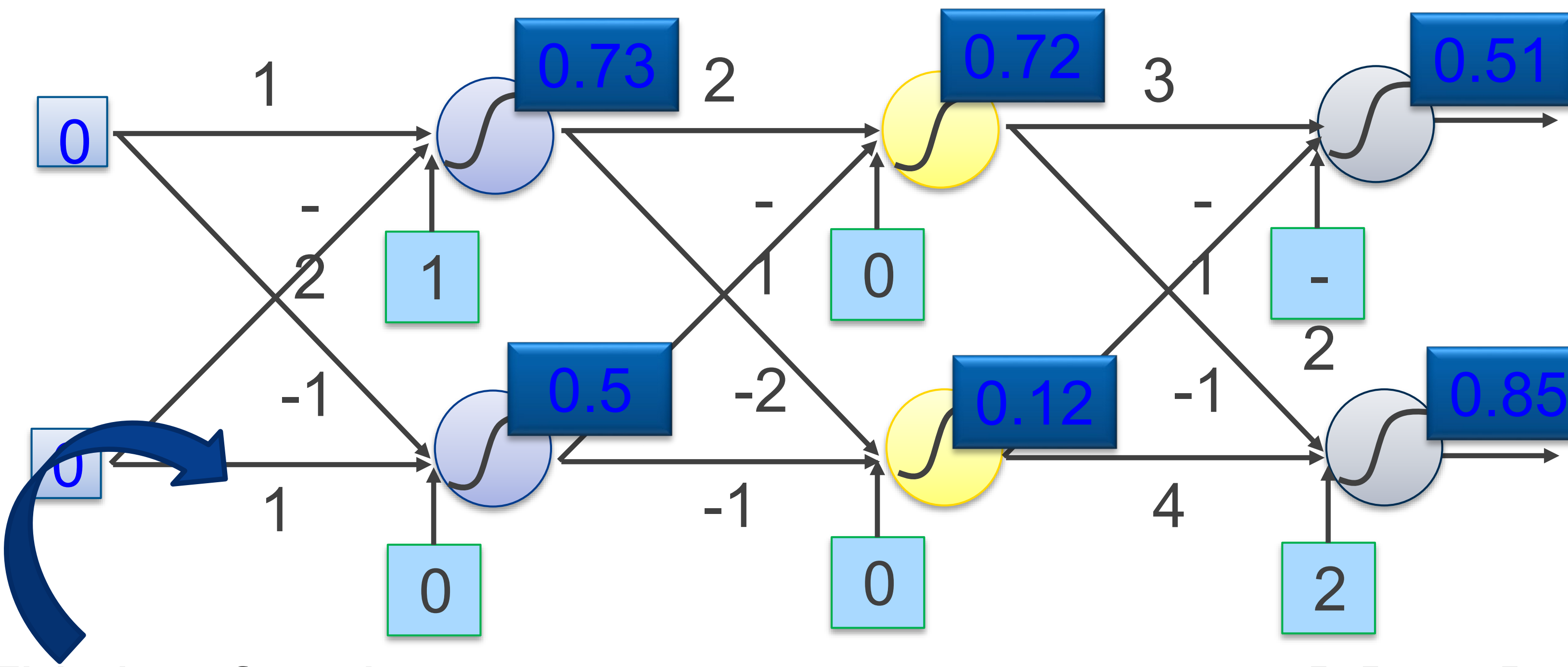
Fully Connect Feedforward Network



Fully Connect Feedforward Network



Fully Connect Feedforward Network



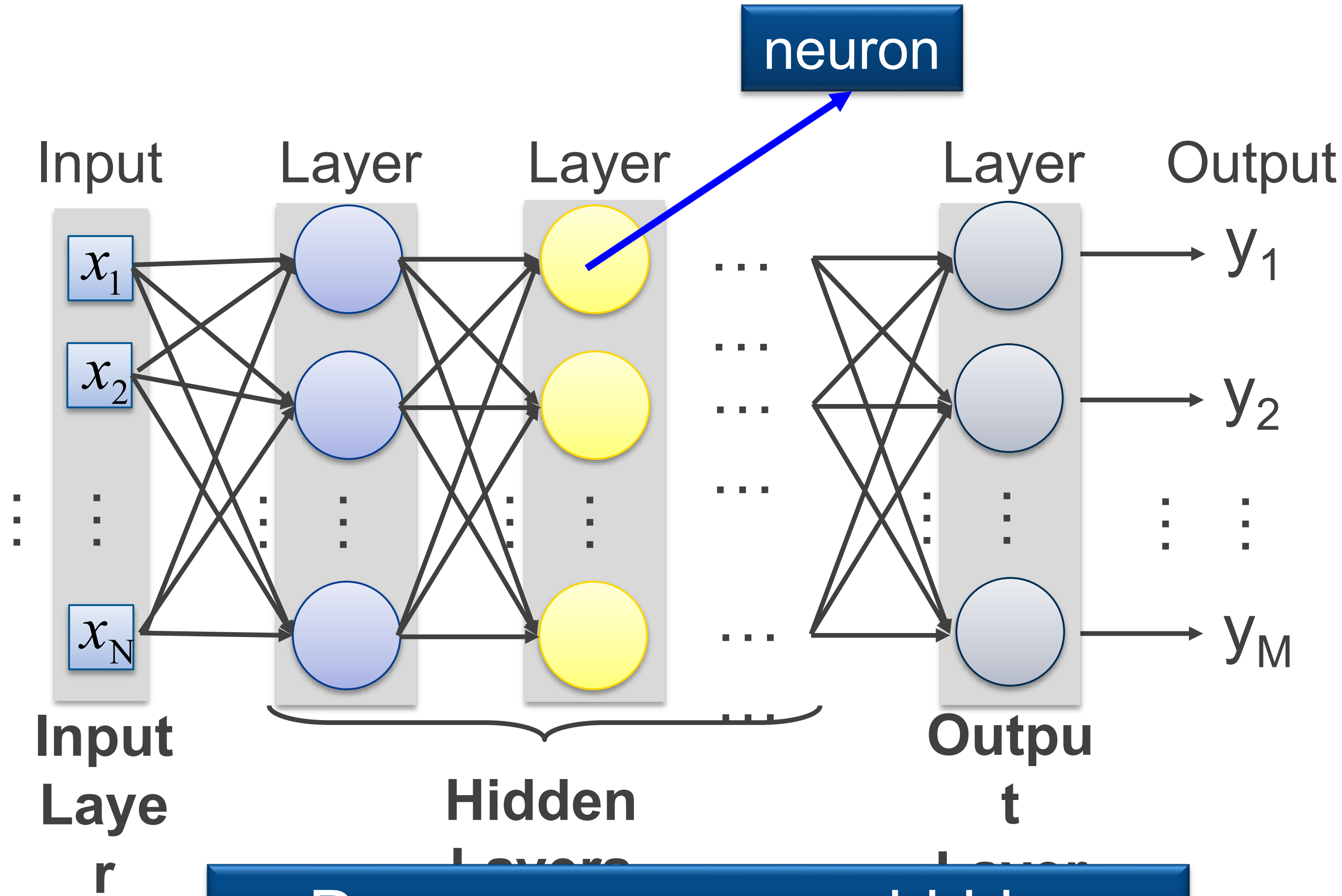
This is a function.
Input vector, output vector

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Given parameters θ, define a function

Given network structure, define ***a function set***

Fully Connect Feedforward Network



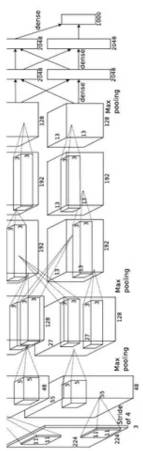
Deep means many hidden layers

Deep = Many hidden layers

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

16.4%



AlexNet (2012)

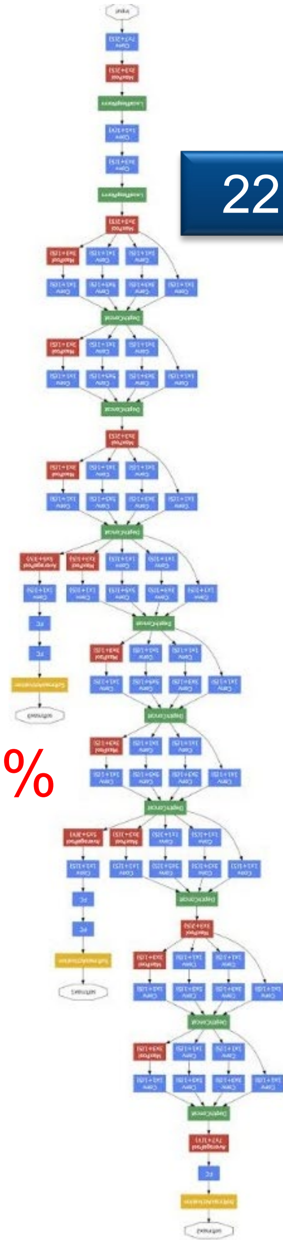
19 layers

7.3%

VGG (2014)

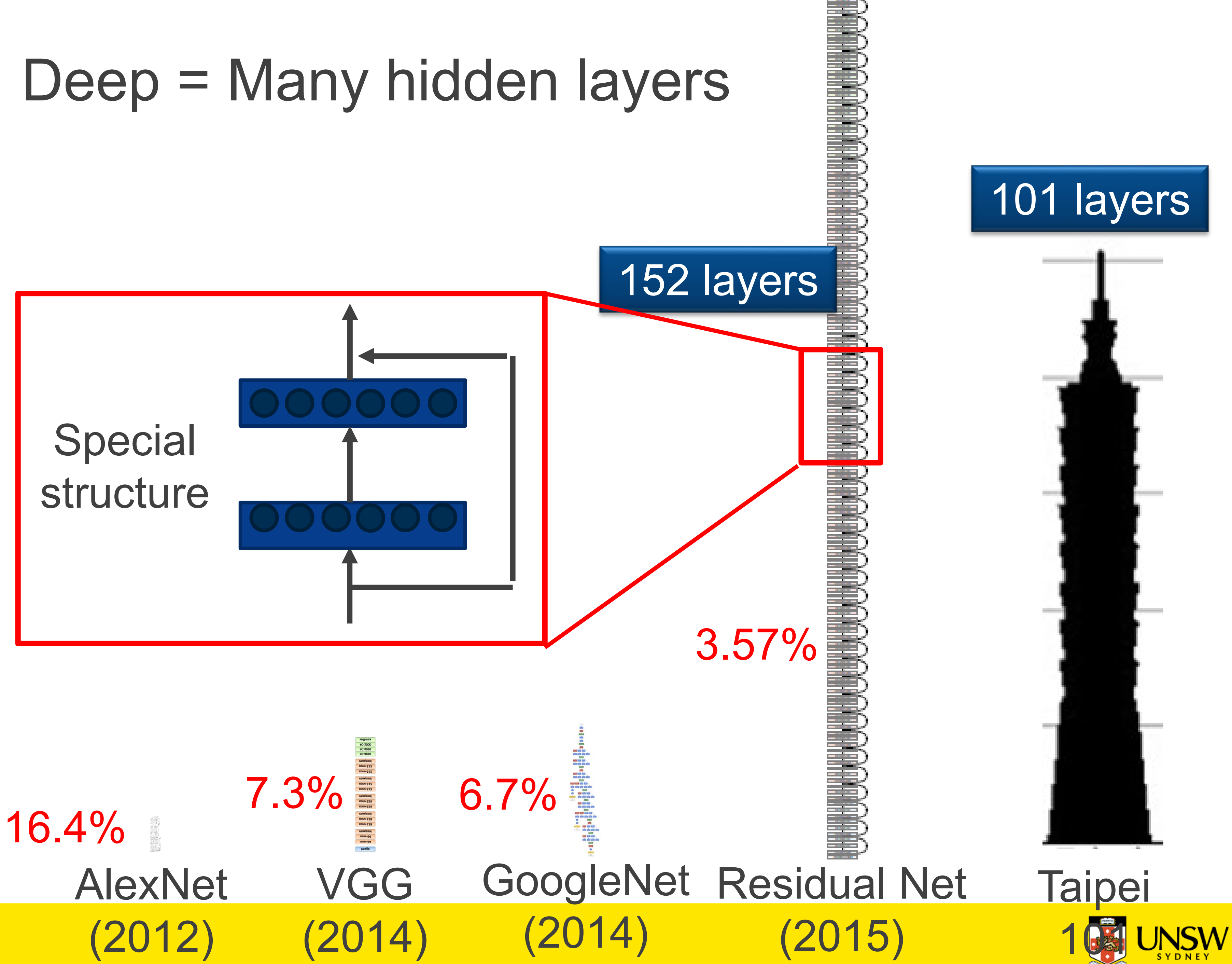
22 layers

6.7%



GoogleNet (2014)

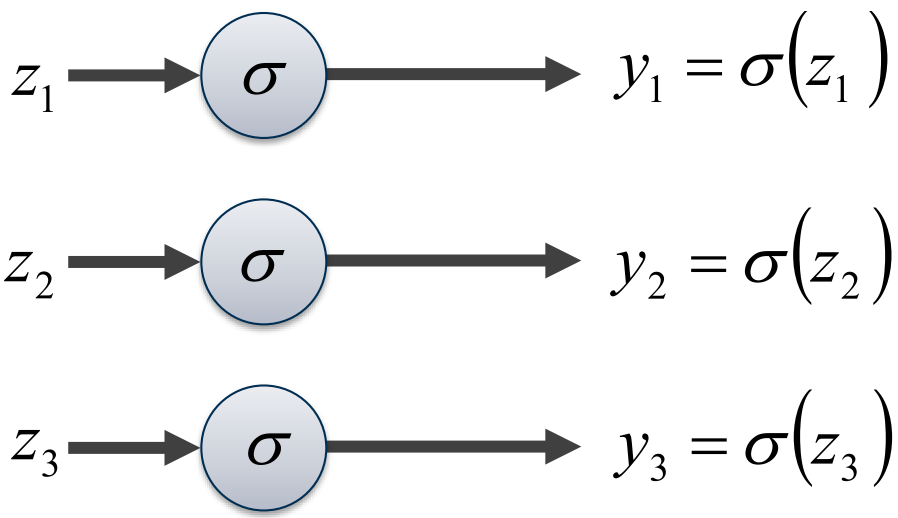
Deep = Many hidden layers



Output Layer

Softmax layer as the output layer

Ordinary Layer



In general, the output of network can be any value.

May not be easy to interpret

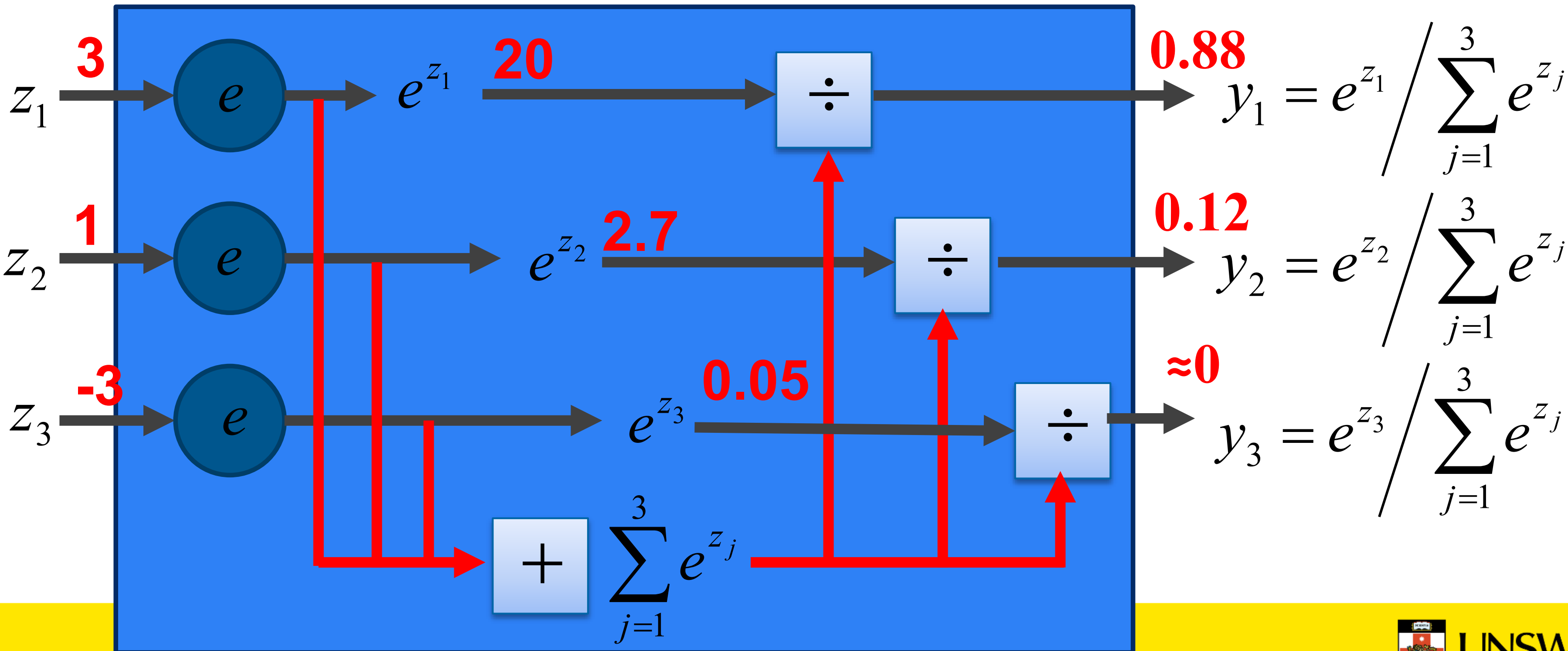
Output Layer

Softmax layer as the output layer

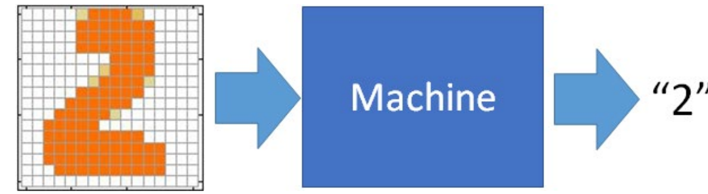
Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

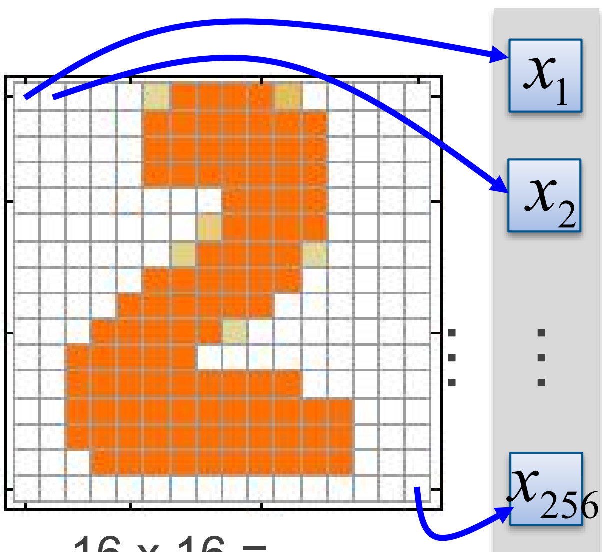
Softmax Layer



Example Application



Input

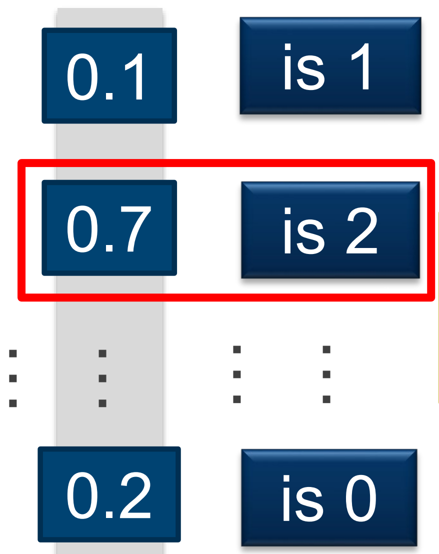


16 x 16 = 256

Ink → 1

No ink → 0

Output

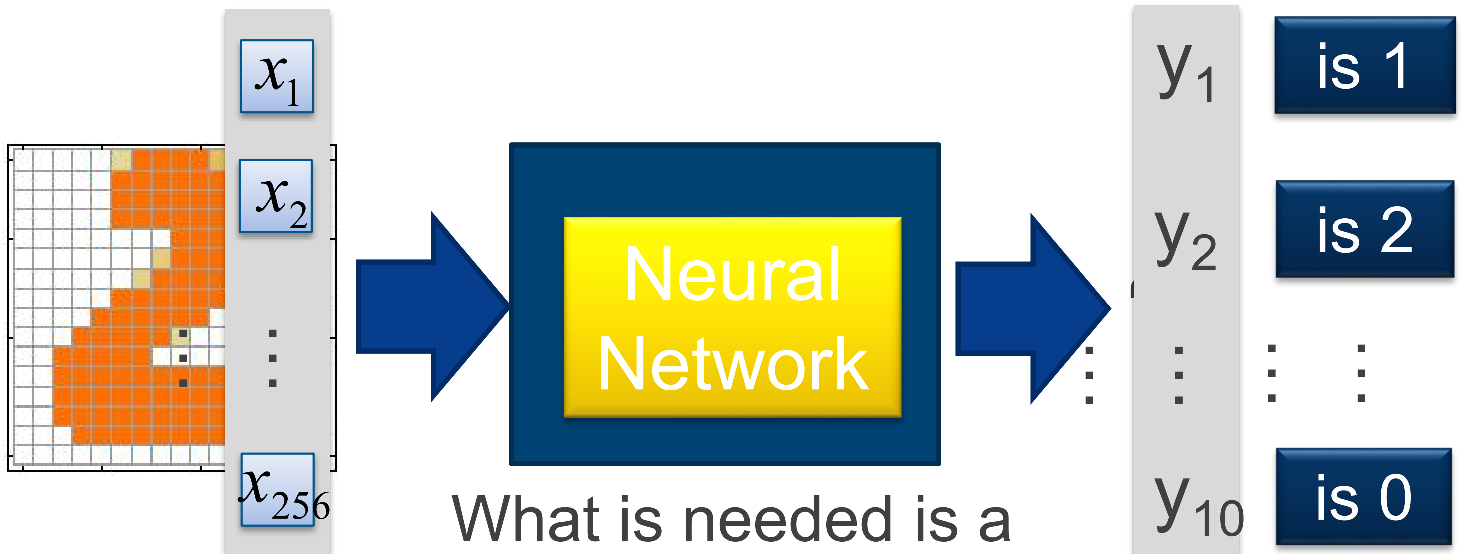


The image is “2”

Each dimension represents the confidence of a digit.

Example Application

Handwriting Digit Recognition

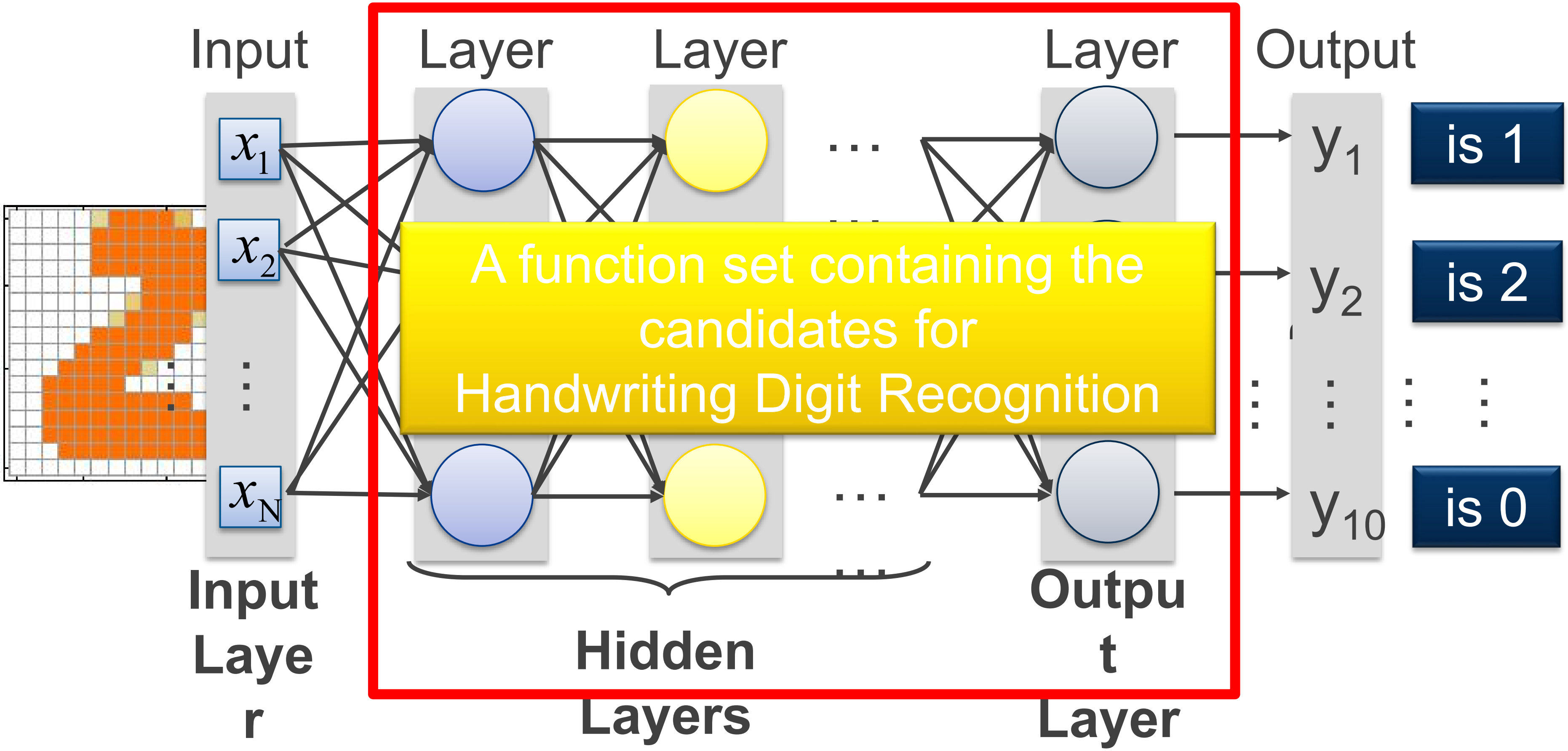


What is needed is a function ……

Input: 256-dim vector

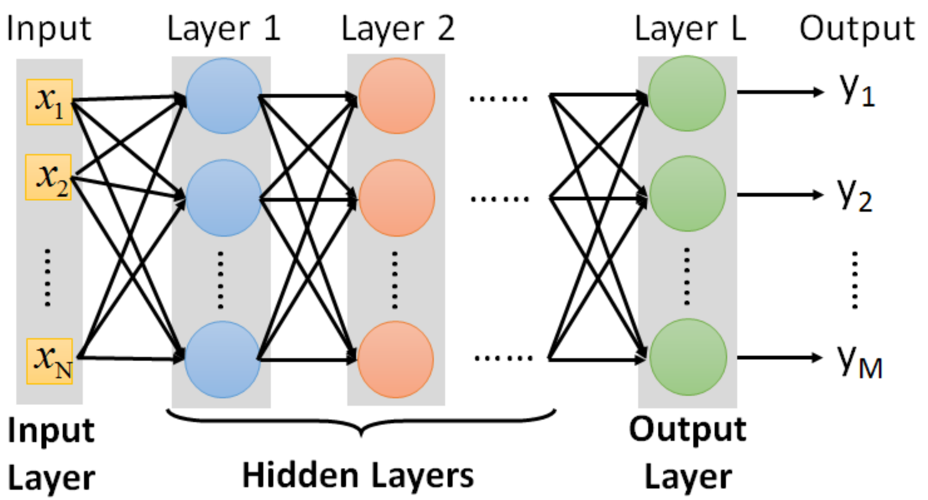
output: 10-dim vector

Example Application



You need to decide the network structure to let a good function in your function set.

FAQ



Q: How many layers? How many neurons for each layer?

Trial and Error + Intuition

Q: Can we design the network structure?

Convolutional Neural Network (CNN)

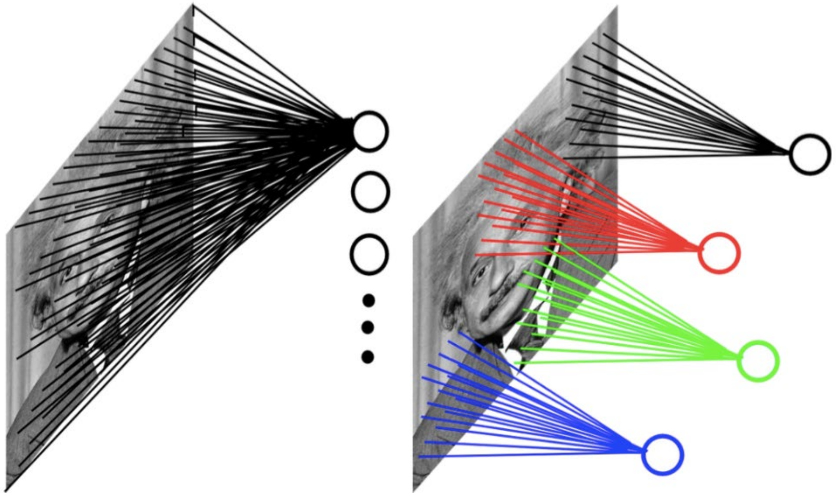
Q: Can the structure be automatically determined?

- Yes, but not widely studied yet.

Convolutional Neural Network

Input can have very high dimension. Using a fully-connected neural network would need a large amount of parameters.

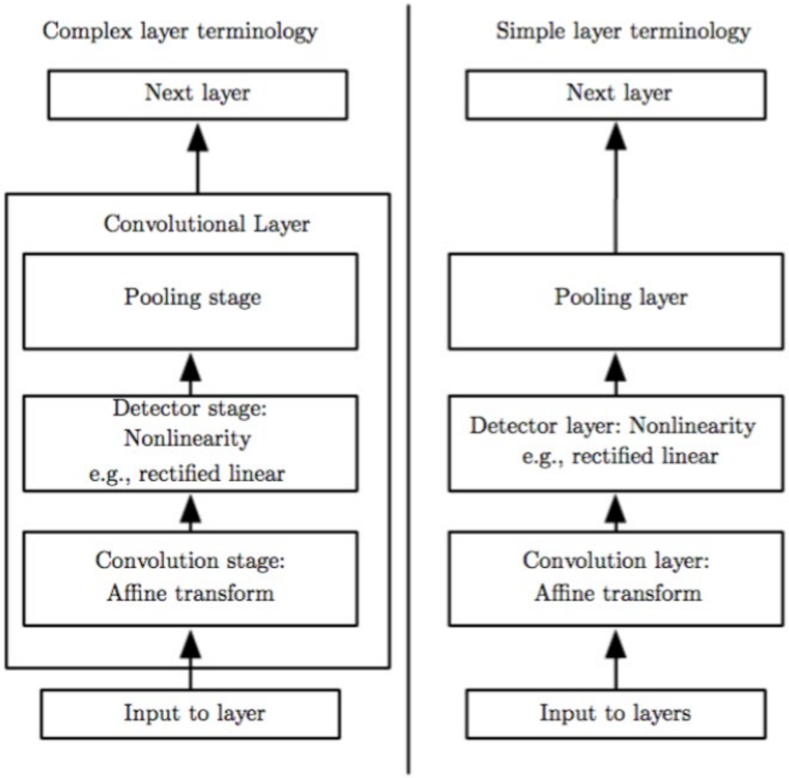
Inspired by the neurophysiological experiments conducted by [Hubel & Wiesel 1962], CNNs are a special type of neural network whose hidden units are only connected to local receptive field. The number of parameters needed by CNNs is much smaller.



Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters
b) CNN: 5x5 kernel, 100 feature maps => 2,500 parameters

Three Stages of a Convolutional Layer



1. Convolution stage
2. Nonlinearity: a nonlinear transform such as rectified linear or $\tanh$
3. Pooling: output a summary statistics of local input, such as max pooling and average pooling

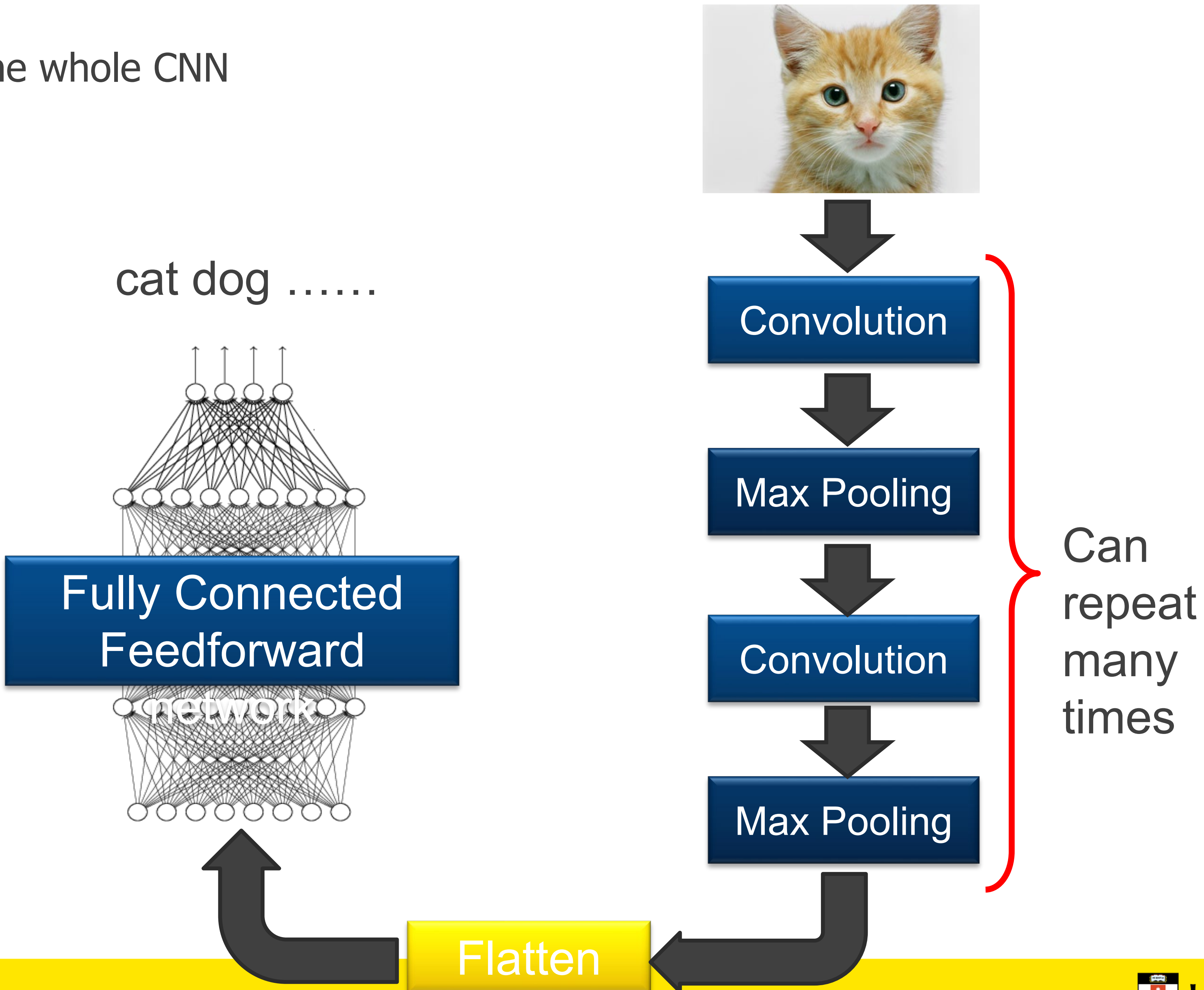
Convolutional Neural Network

Neural Networks that use convolution in place of general matrix multiplication in atleast one layer

Next:

- What is convolution?
- Nonlinearity
- What is pooling?
- What is the motivation for such architectures?

The whole CNN



The whole CNN

Property 1

- Some patterns are much smaller than the whole

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

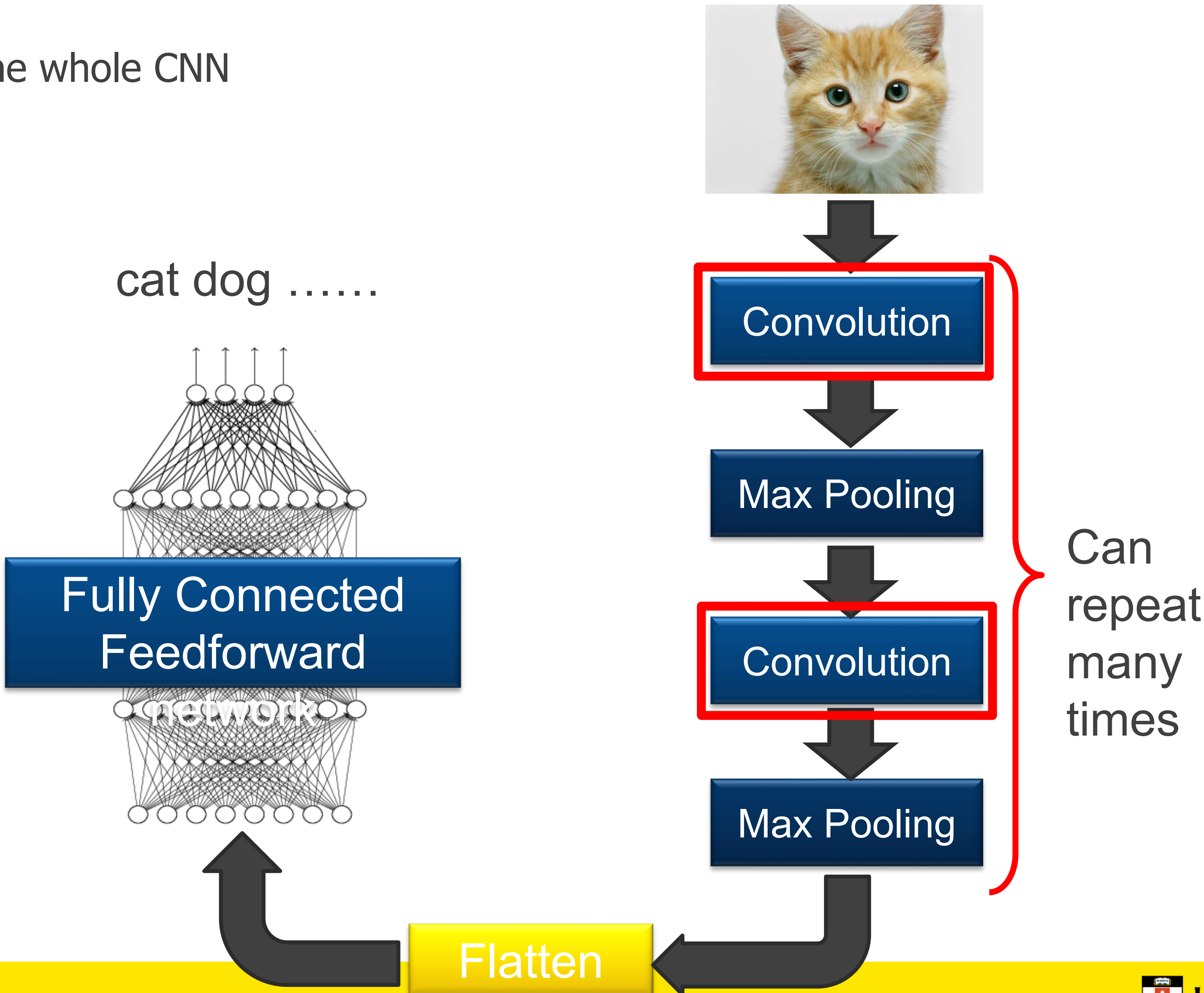
Convolution

Max Pooling

Can repeat many times

Flatten

The whole CNN



CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮ ⋮

Property 1

Each filter detects a small pattern (3 x 3)

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

We set stride=1 below

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2

CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

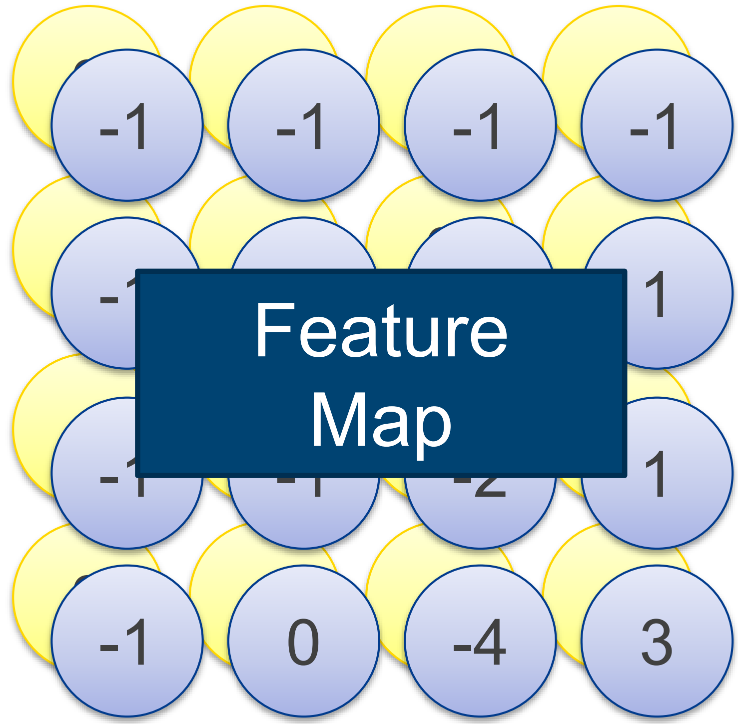
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Do the same process for every filter



4 x 4 image

CNN – Zero Padding

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

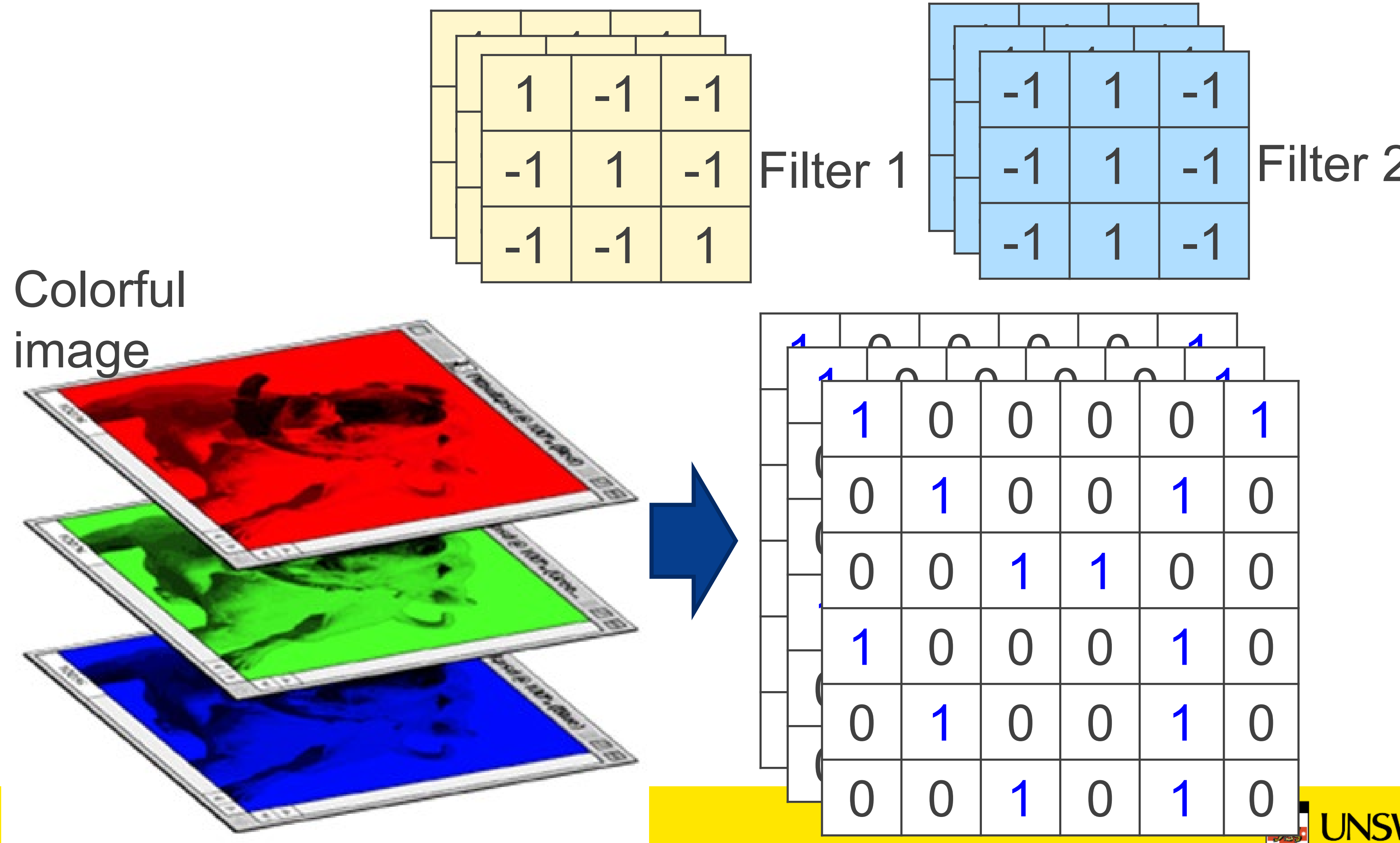
0	0	0					
0	1	0	0	0	0	1	
0	0	1	0	0	1	0	
	0	0	1	1	0	0	
	1	0	0	0	1	0	
	0	1	0	0	1	0	0
	0	0	1	0	1	0	0
					0	0	0

6 x 6 image

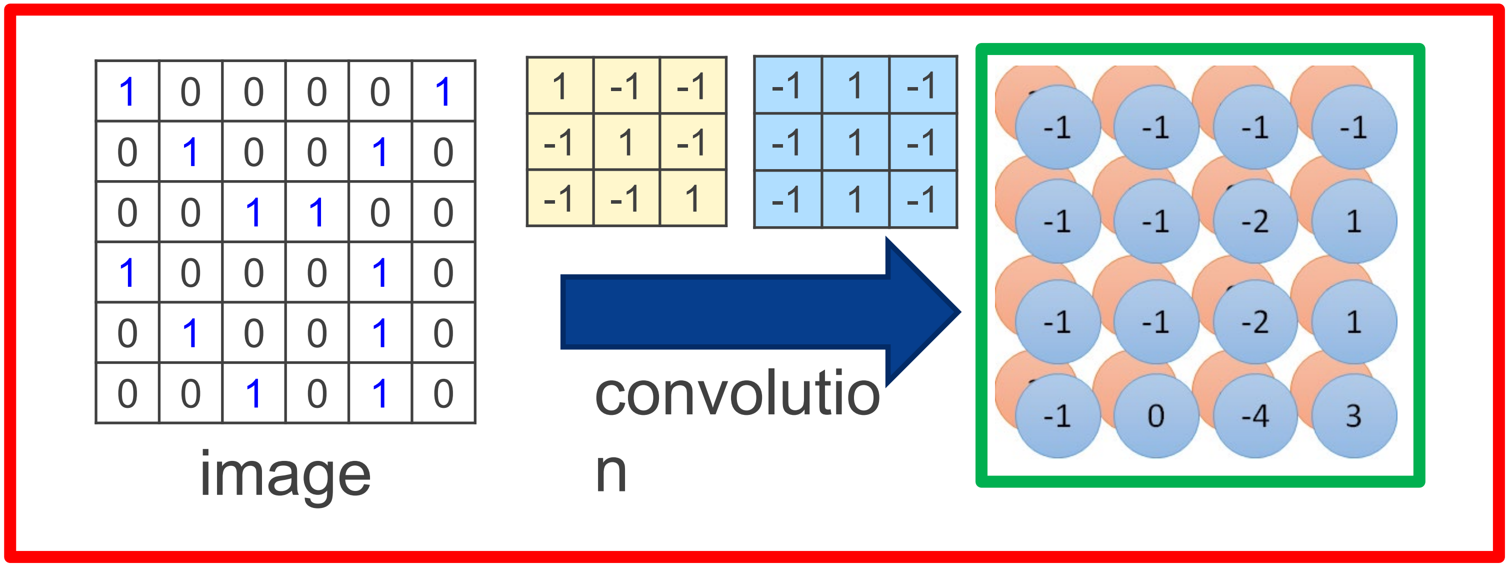
You will get another 6 x 6 images in this way

→ Zero padding

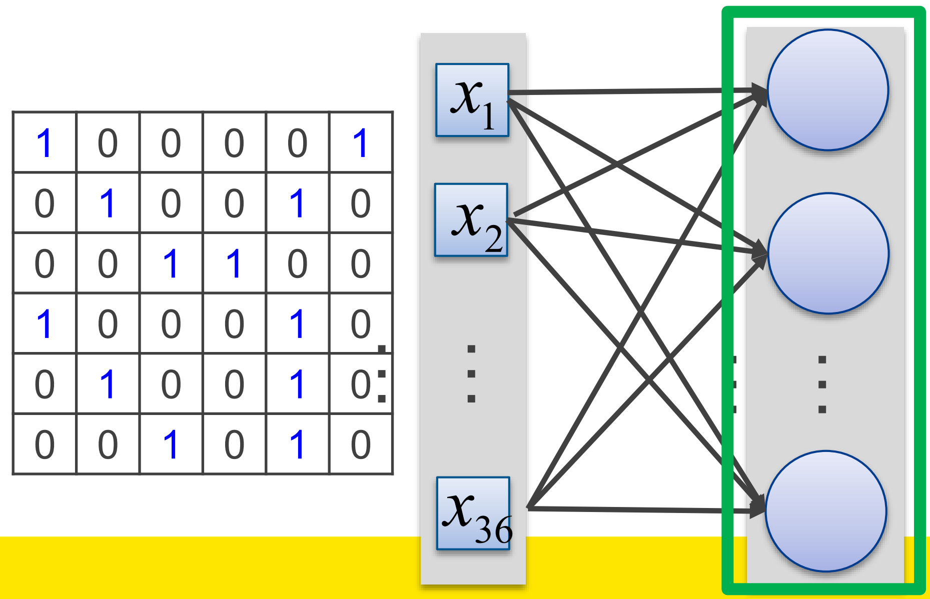
CNN – Colorful image

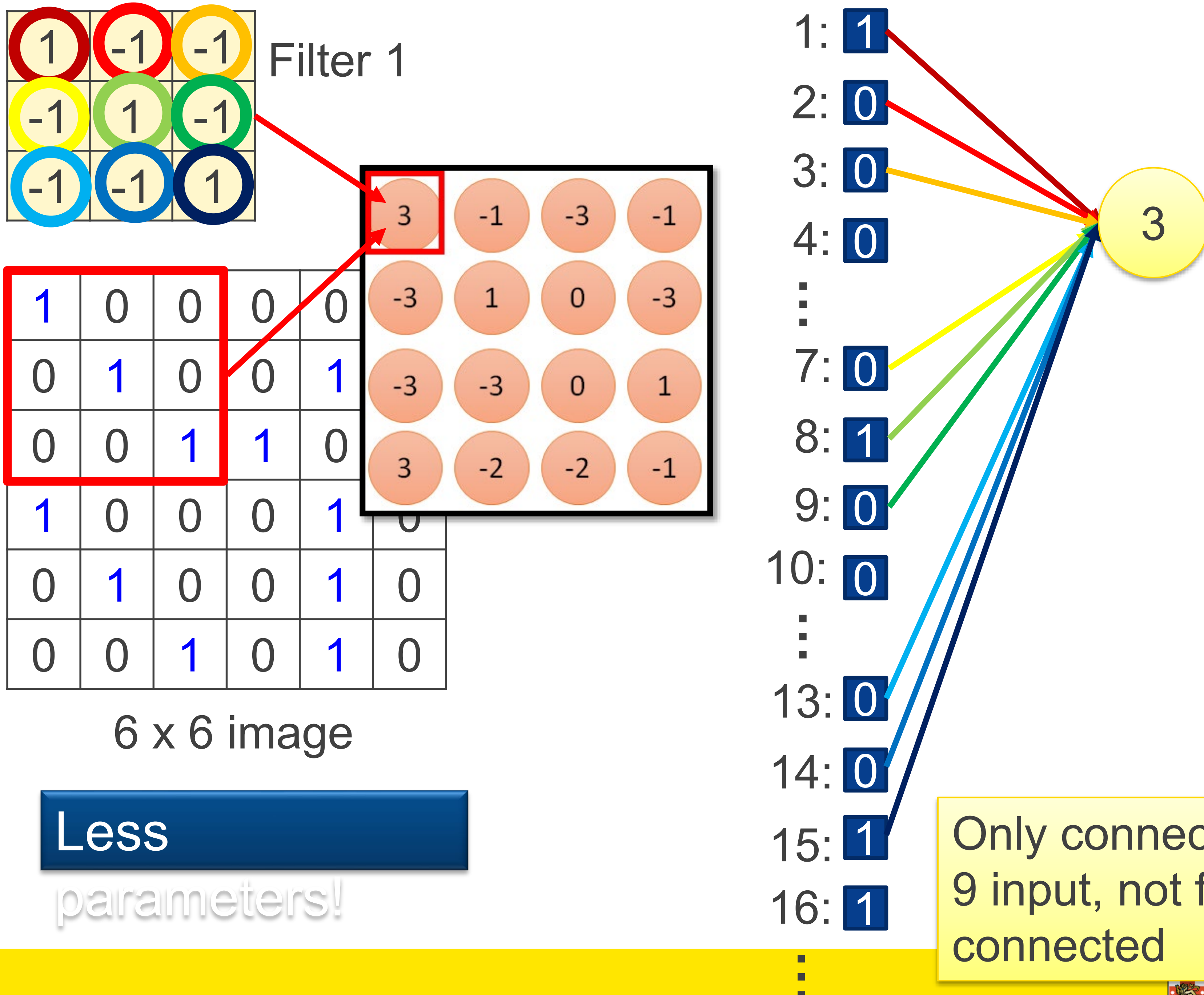


Convolution v.s. Fully Connected



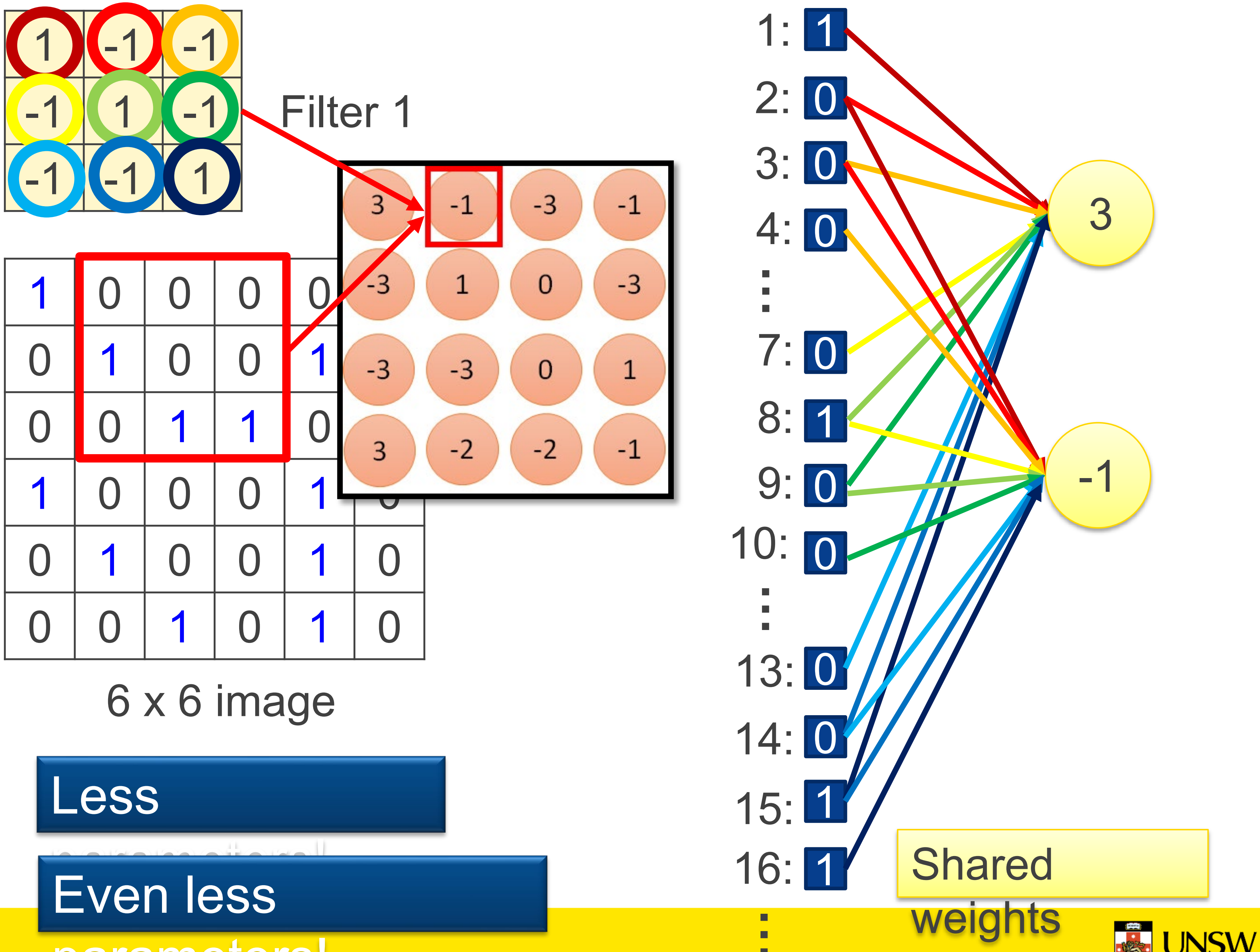
Fully-connected





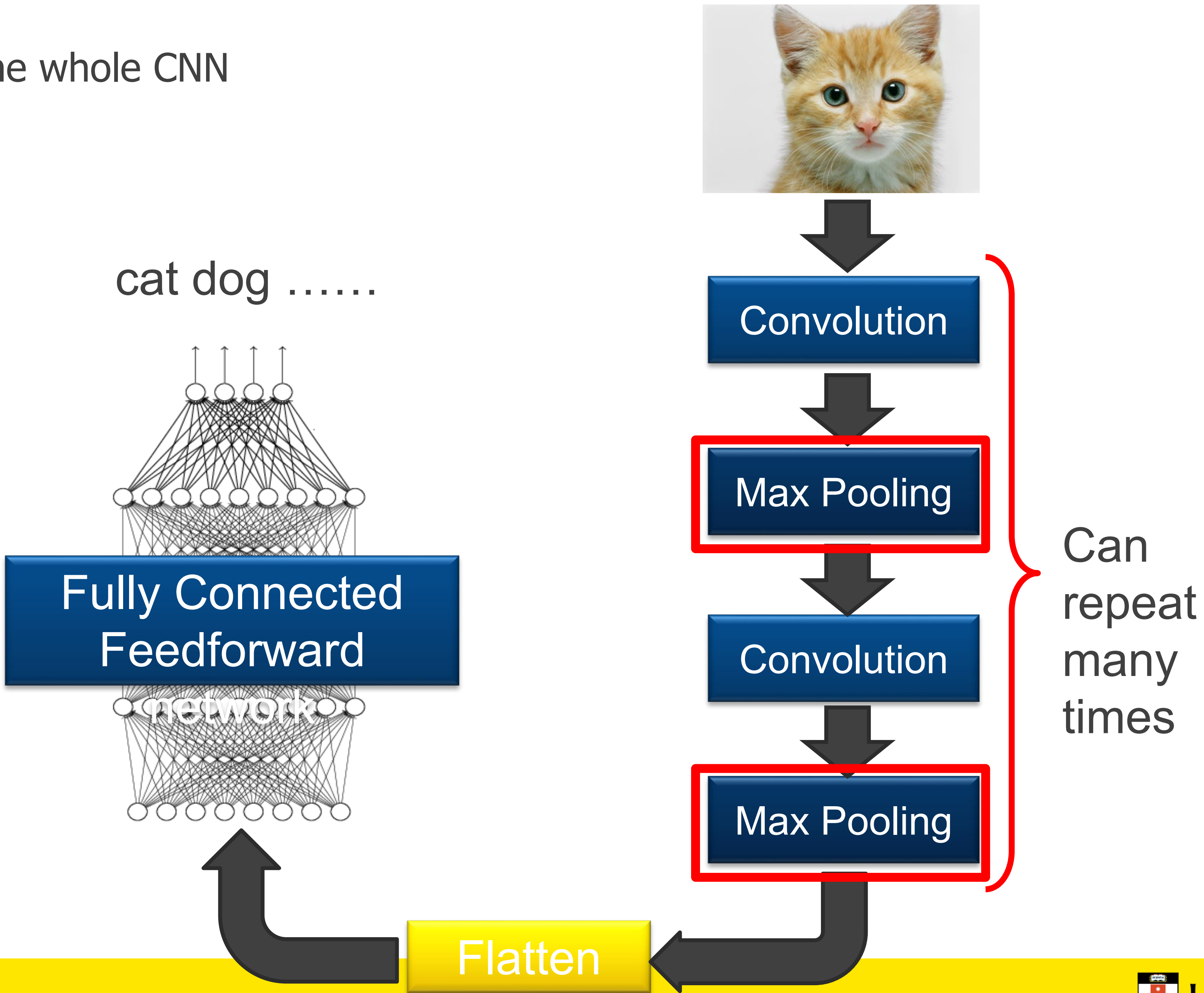
Less parameters!

Only connect to 9 input, not fully connected



Filter 1
6 x 6 image
Less parameters!
Even less parameters!
Shared weights

The whole CNN



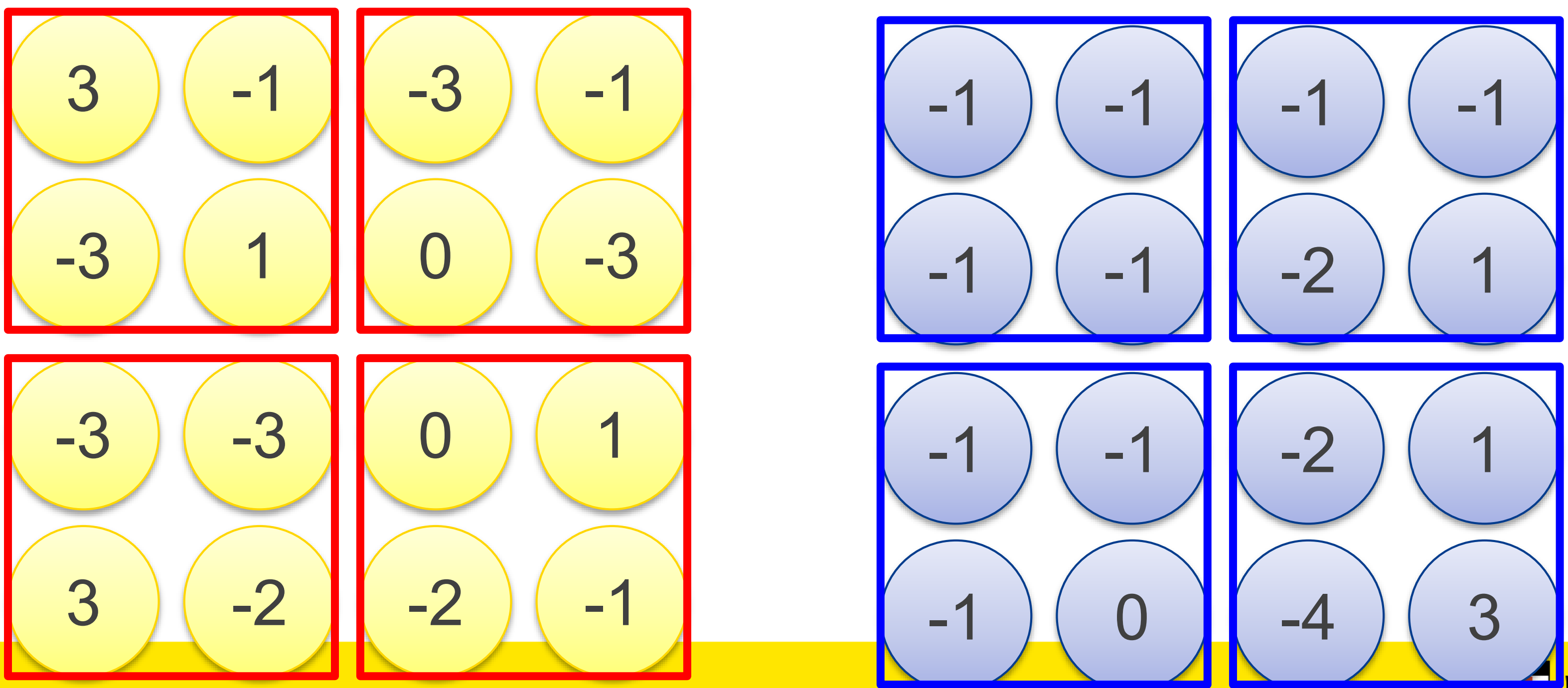
CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

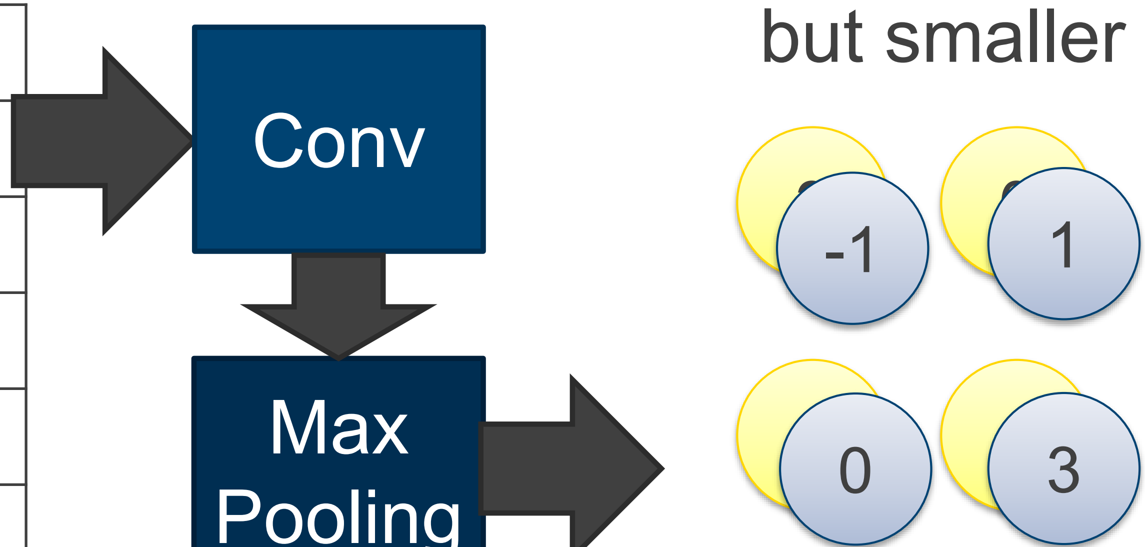


CNN – Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

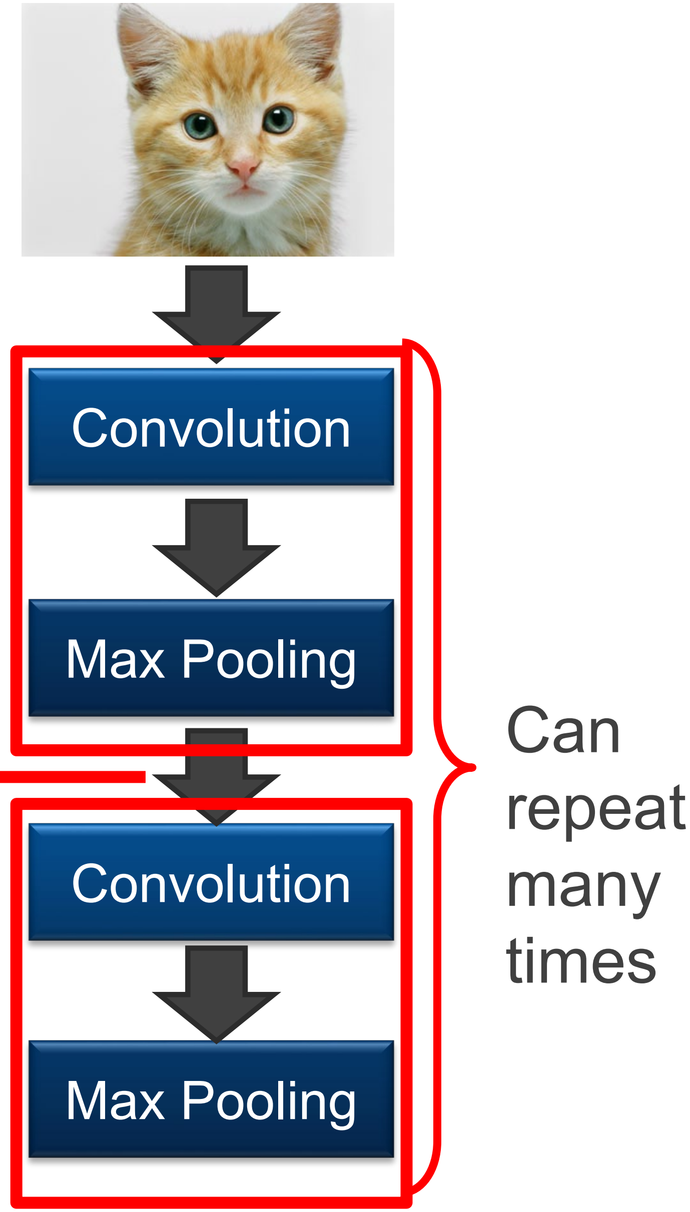
New image but smaller

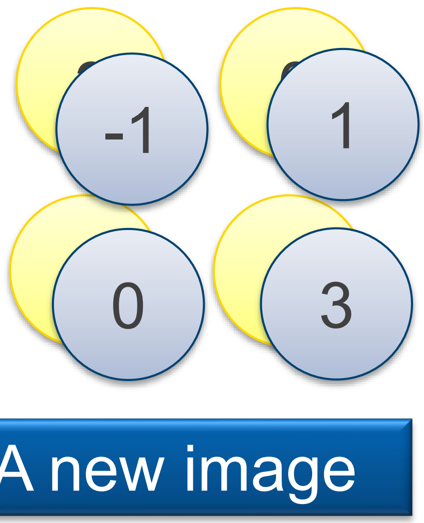


2 x 2 image

Each filter is a channel

The whole CNN

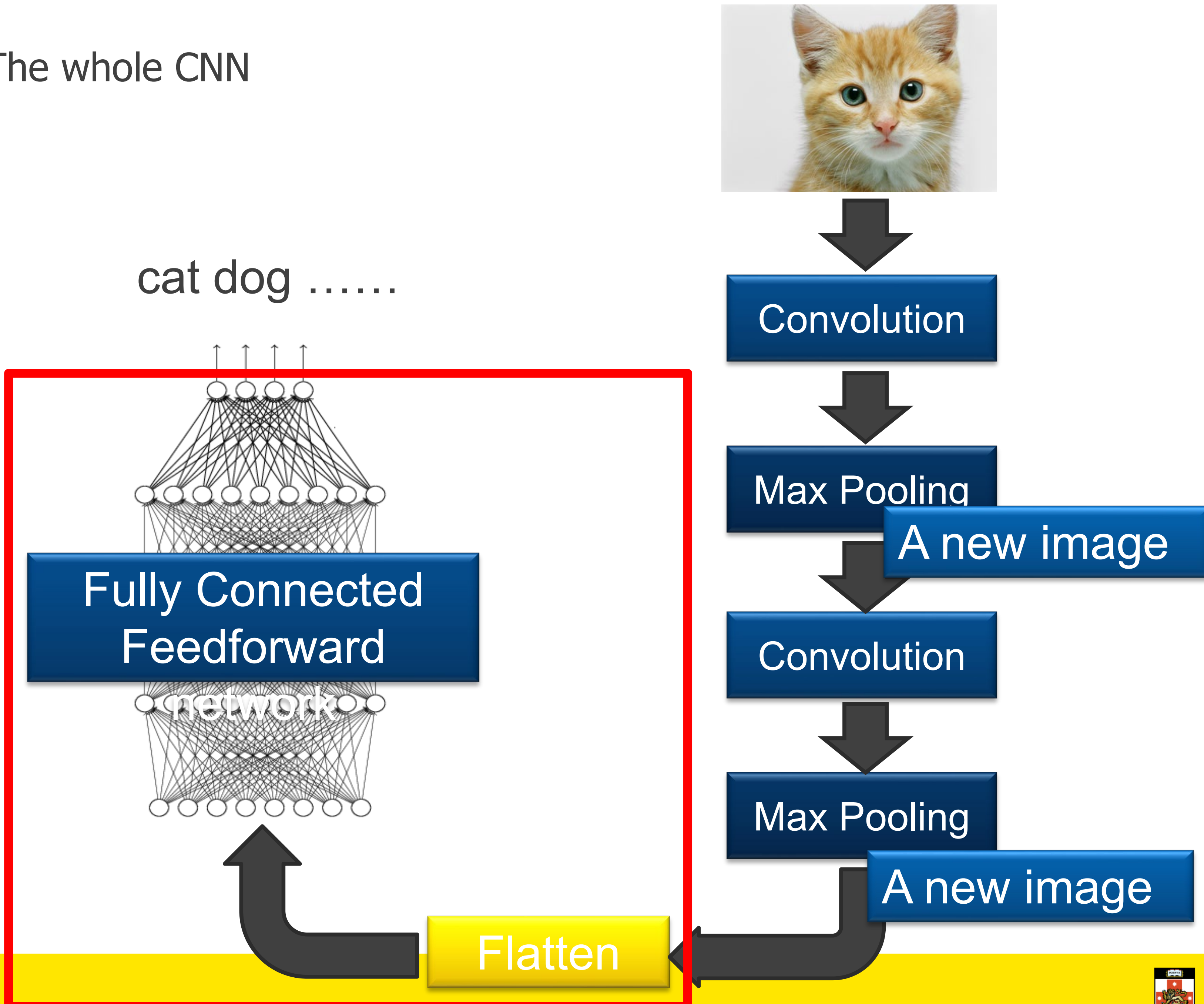




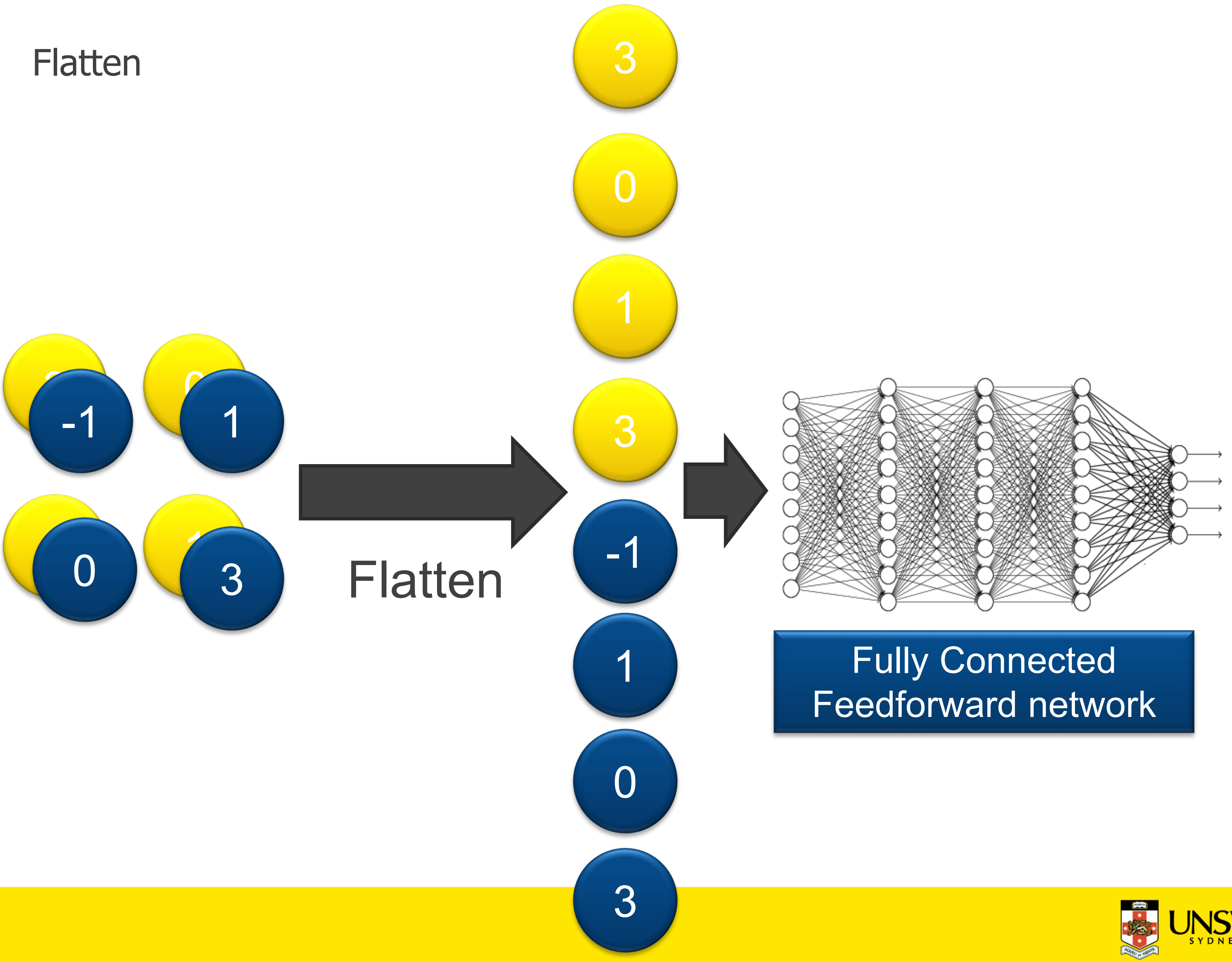
Smaller than the original image

The number of the channel is the number of filters

The whole CNN



Flatten

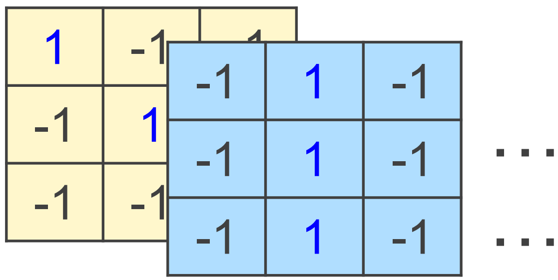


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(1,28,28) ) )
```



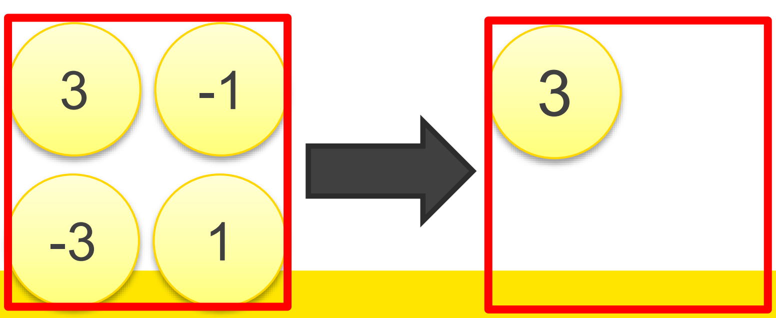
There are 25 3x3 filters.

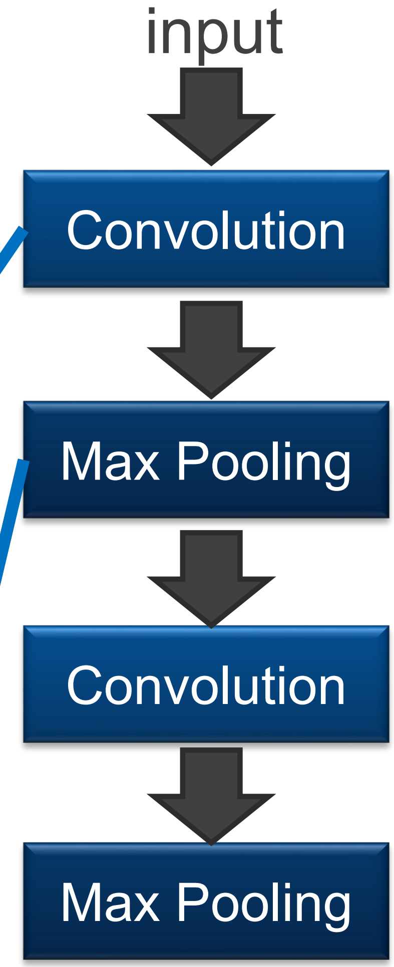
Input_shape = (1 , 28 , 28)

1: black/weight, 3: RGB

28 x 28 pixels

```
model2.add(MaxPooling2D((2,2)))
```





CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

input

1 x 28 x 28

Convolution

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(1,28,28) ) )
```

25 x 26 x 26

How many parameters for each filter? **9**

Max Pooling

```
model2.add(MaxPooling2D((2,2)))
```

25 x 13 x 13

Convolution

```
model2.add(Convolution2D(50,3,3))
```

50 x 11 x 11

How many parameters for each filter? **225**

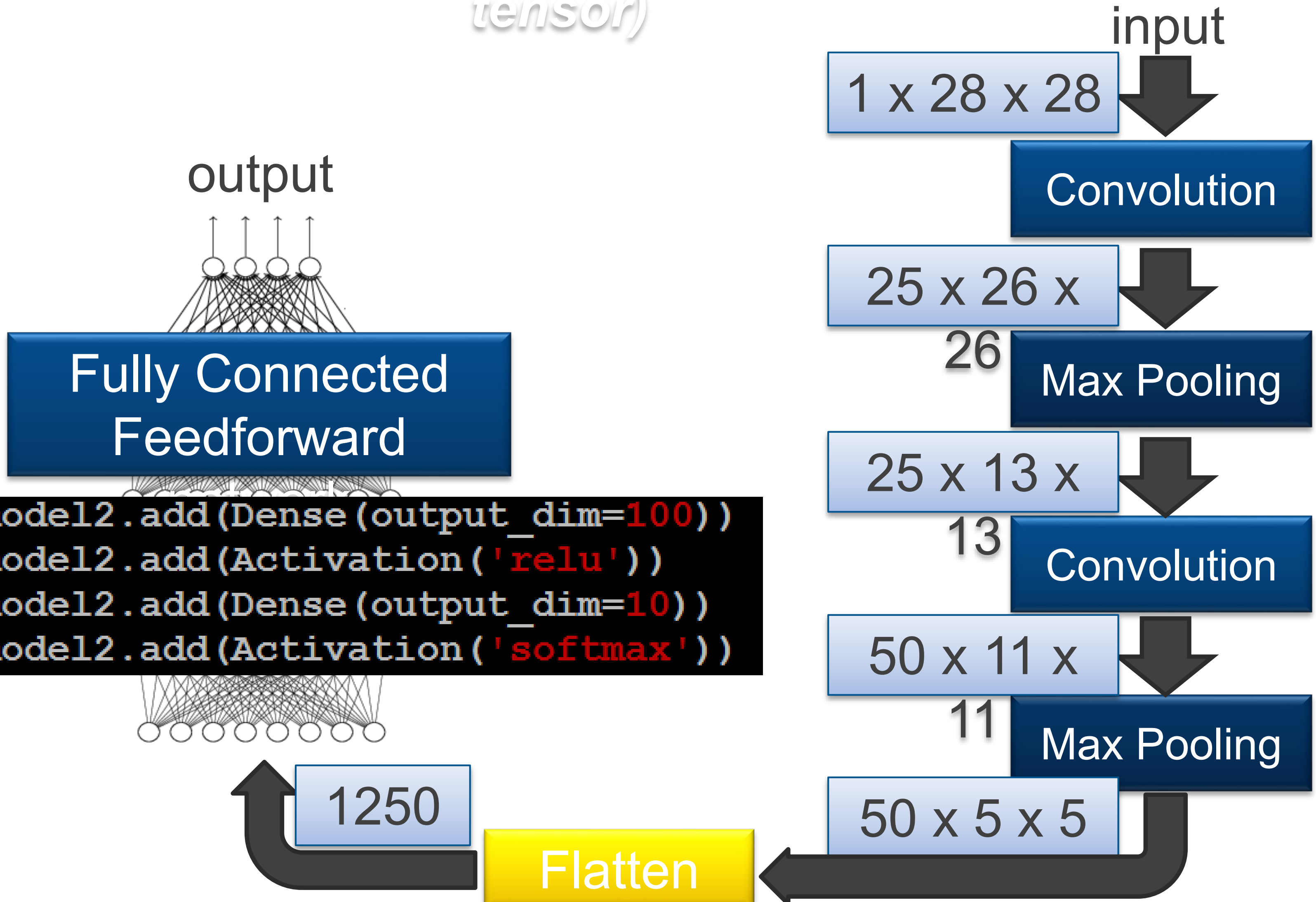
Max Pooling

```
model2.add(MaxPooling2D((2,2)))
```

50 x 5 x 5

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



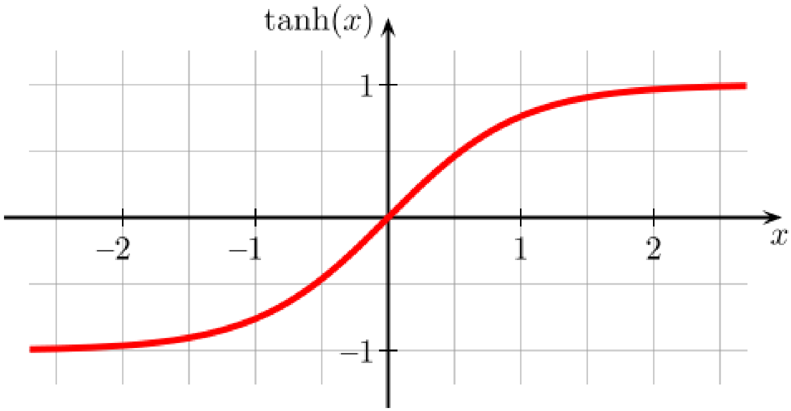

```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

```
model2.add(Flatten())
```

Non-linearity

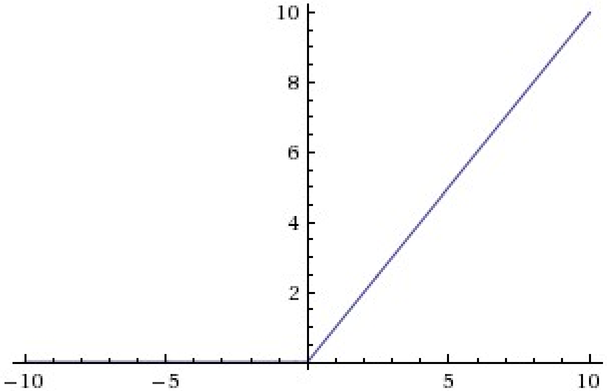
```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

Tanh(x)



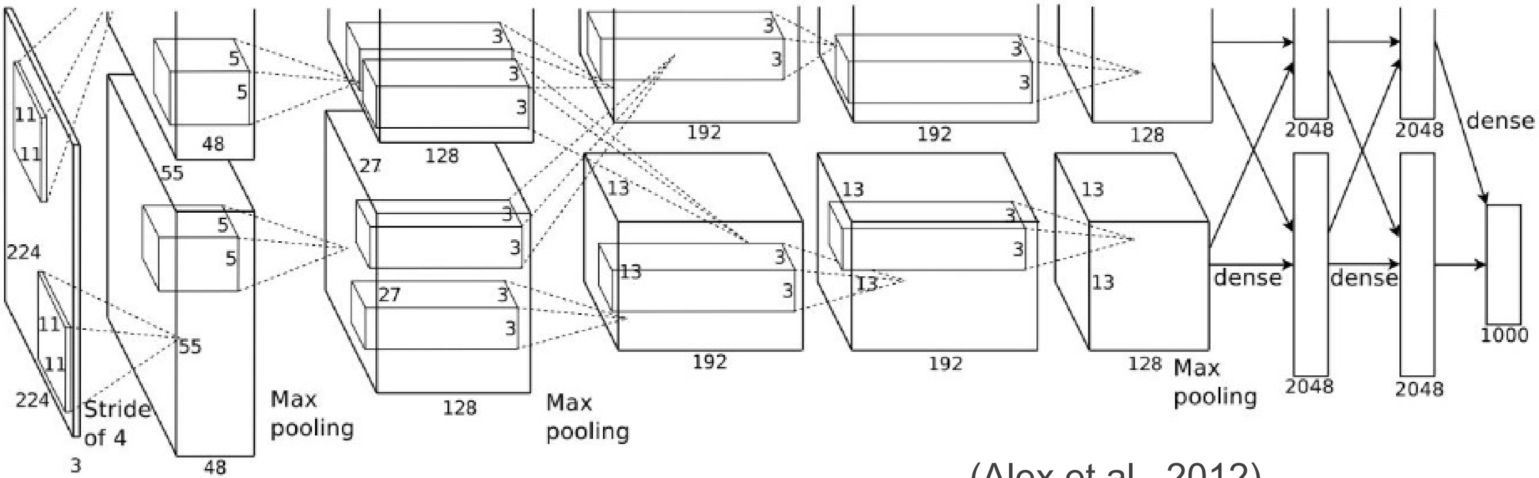
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU



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

Deep CNN: winner of ImageNet 2012



(Alex et al., 2012)

Multiple feature maps per convolutional layer.

Multiple convolutional layers for extracting features at different levels.

Higher-level layers take the feature maps in lower-level layers as input.

Deep CNN for Image Classification

Classification

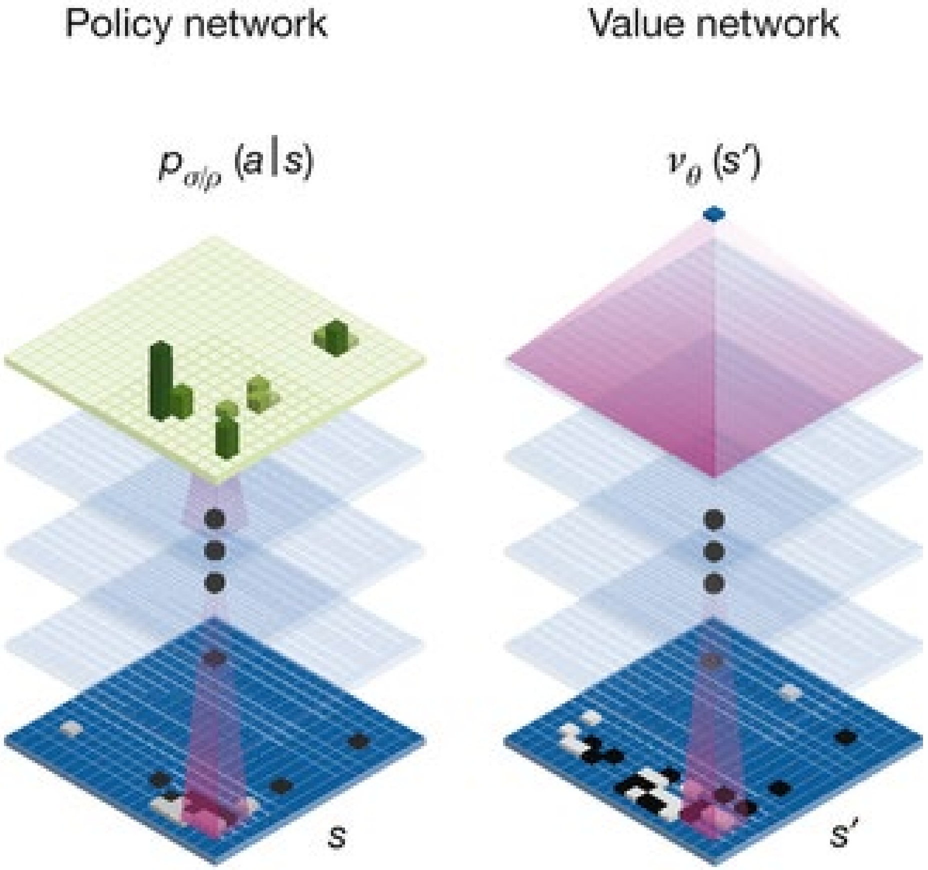
Click for a Quick Example

Maximally accurate	Maximally specific
cat	1.79306
feline	1.74269
domestic cat	1.70760
tabby	0.94807
domestic animal	0.76846

CNN took 0.064 seconds.

Try out a live demo at
http://demo.caffe.berkeleyvision.org/

Deep CNN in AlphaGO



(Silver et al, 2016)

Policy network:

Input: 19x19, 48 input channels

Layer 1: 5x5 kernel, 192 filters

Layer 2 to 12: 3x3 kernel, 192 filters

Layer 13: 1x1 kernel, 1 filter

Value network has similar architecture to policy network

Sequence Modelling

Why do we need RNN?

What are RNNs?

RNN Extensions

What can RNNs can do?

Why do we need RNNs?

The limitations of the Neural network (CNNs)

Rely on the assumption of independence among the (training and test) examples.

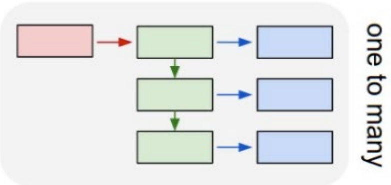
- After each data point is processed, the entire state of the network is lost

Rely on examples being vectors of fixed length

We need to model the data with temporal or sequential structures and varying length of inputs and outputs

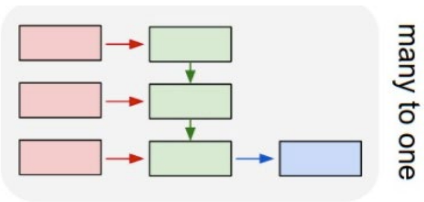
- Frames from video
- Snippets of audio
- Words pulled from sentences

What can RNNs do?



A person riding a motorbike on dirt road

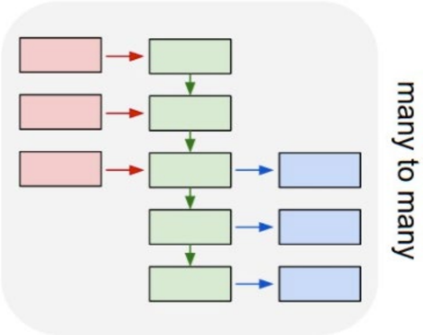
Image Captioning



Awesome tutorial.

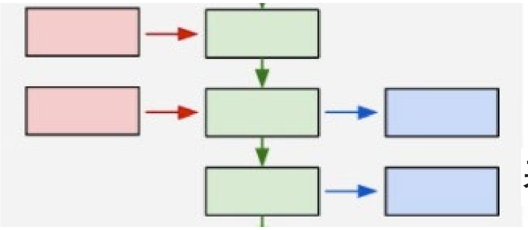
Positive

Sentiment Analysis



Happy

Chinese New Year



春节快乐

Machine Translation

Sentiment Analysis

Classify a restaurant review from Yelp! OR movie review from IMDB OR
…
as positive or negative

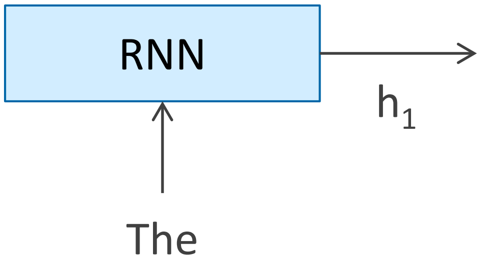
Inputs: Multiple words, one or more sentences

Outputs: Positive / Negative classification

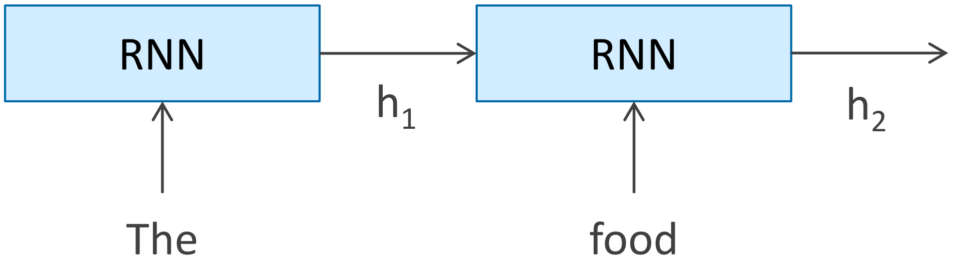
"The food was really good"

"The chicken crossed the road because it was uncooked"

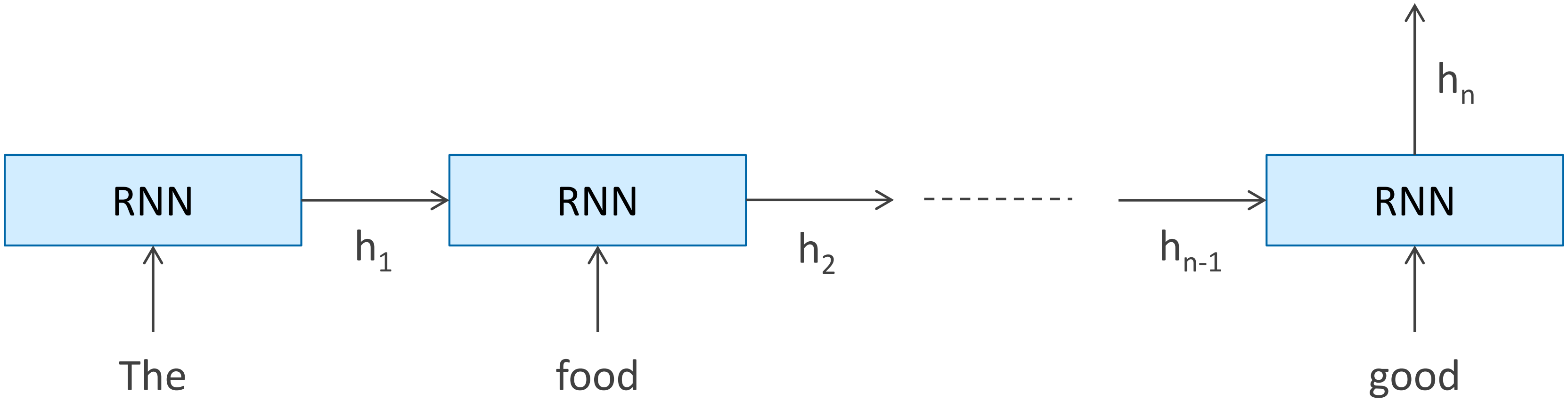
Sentiment Analysis



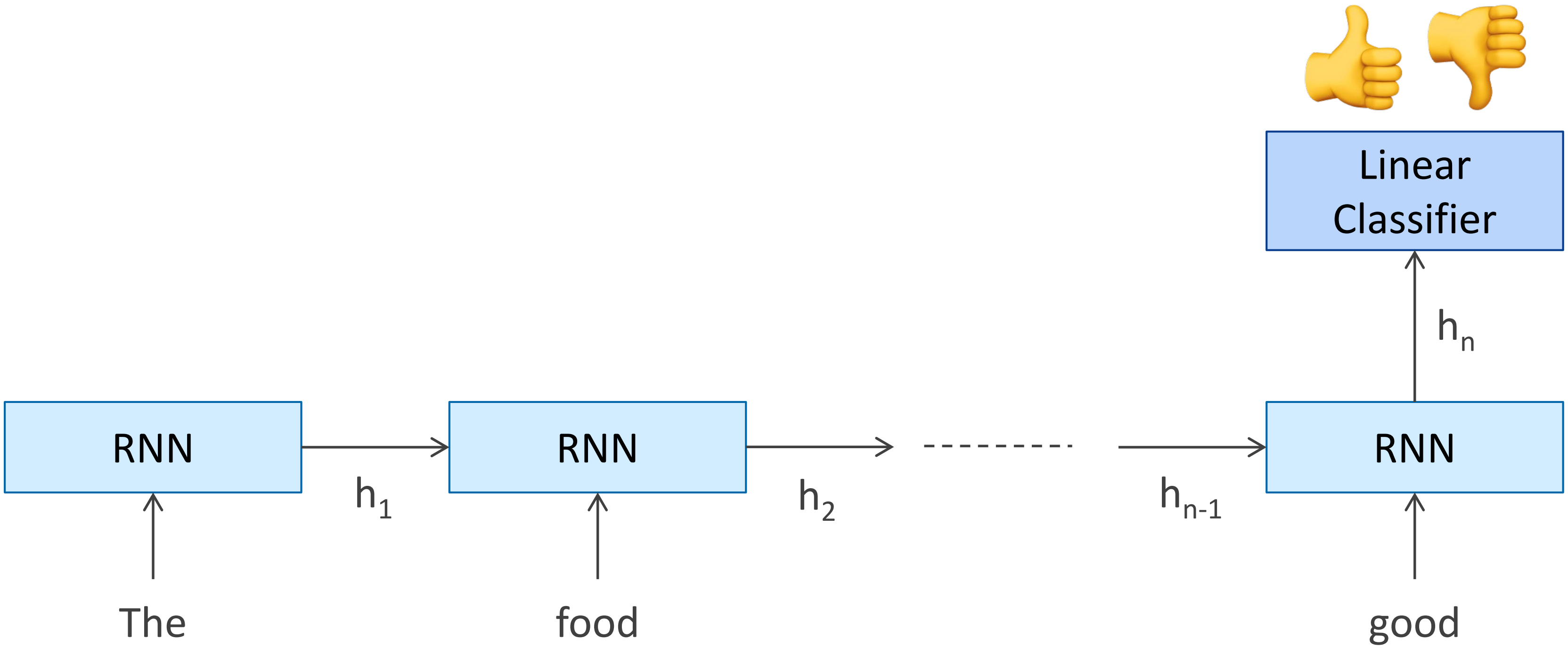
Sentiment Analysis



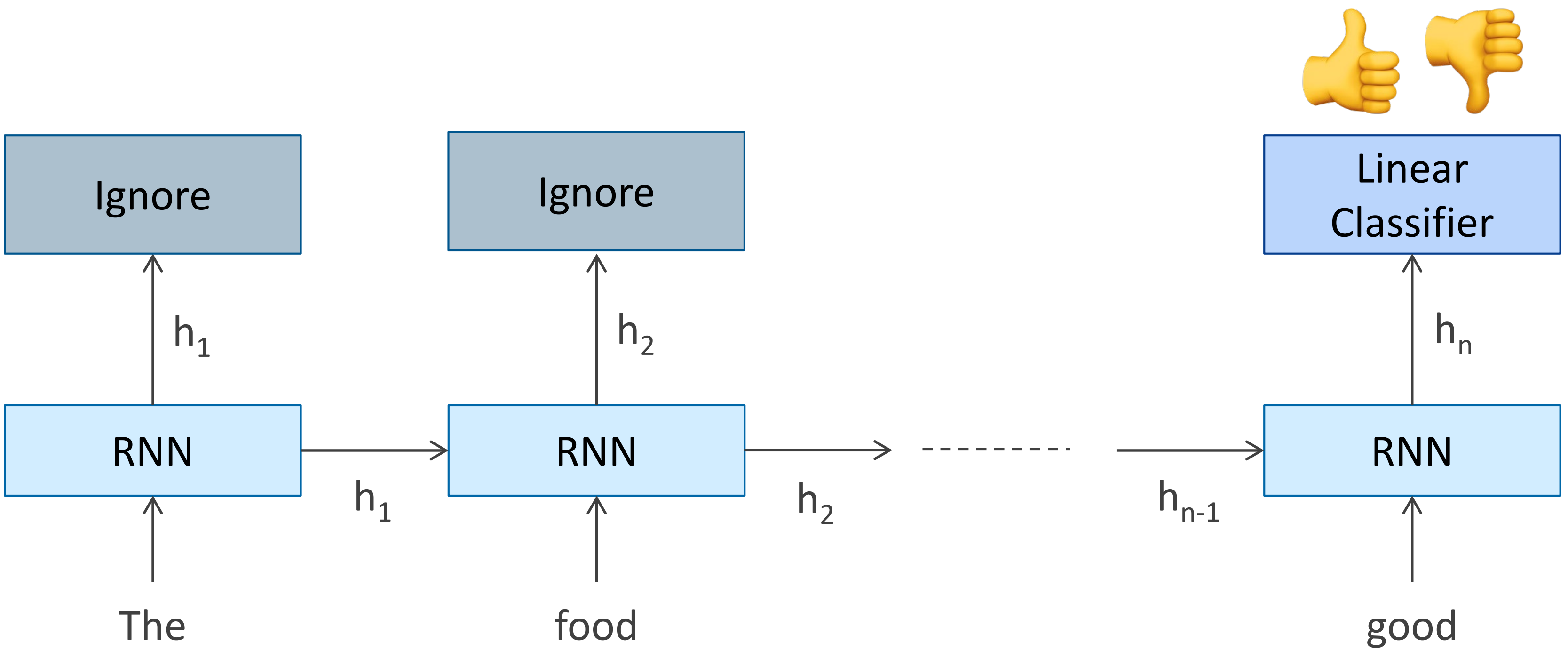
Sentiment Analysis



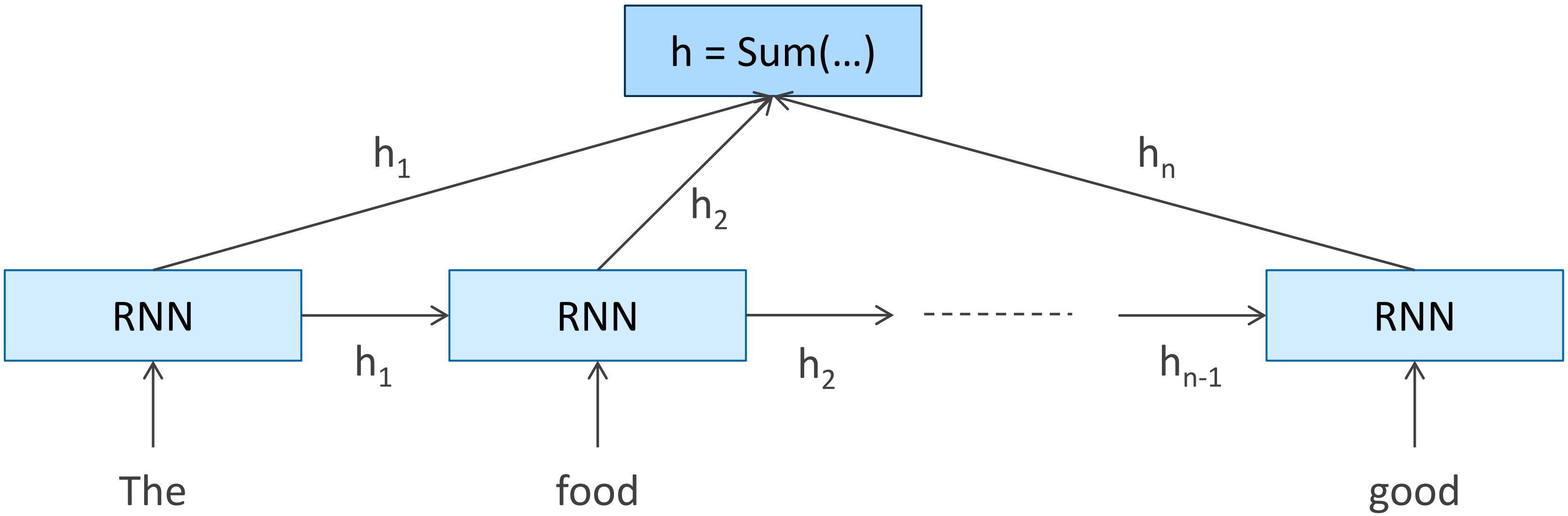
Sentiment Analysis



Sentiment Analysis



Sentiment Analysis



Sentiment Analysis

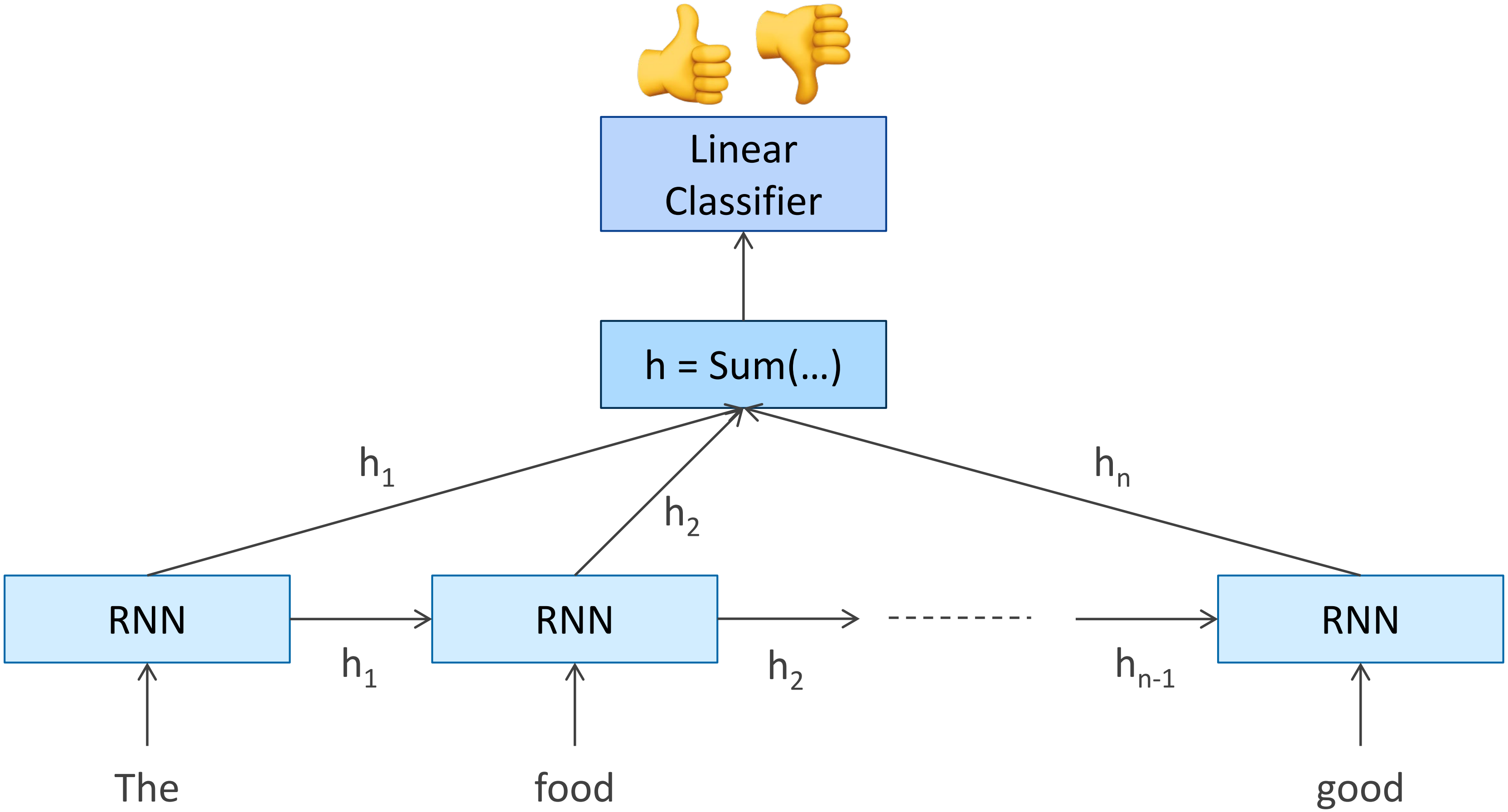


Image Captioning

Given an image, produce a sentence describing its contents

Inputs: Image feature (from a CNN)

Outputs: Multiple words (let's consider one sentence)

: The dog is hiding

Image Captioning

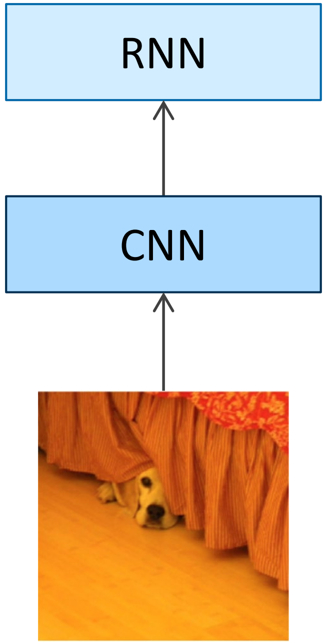


Image Captioning

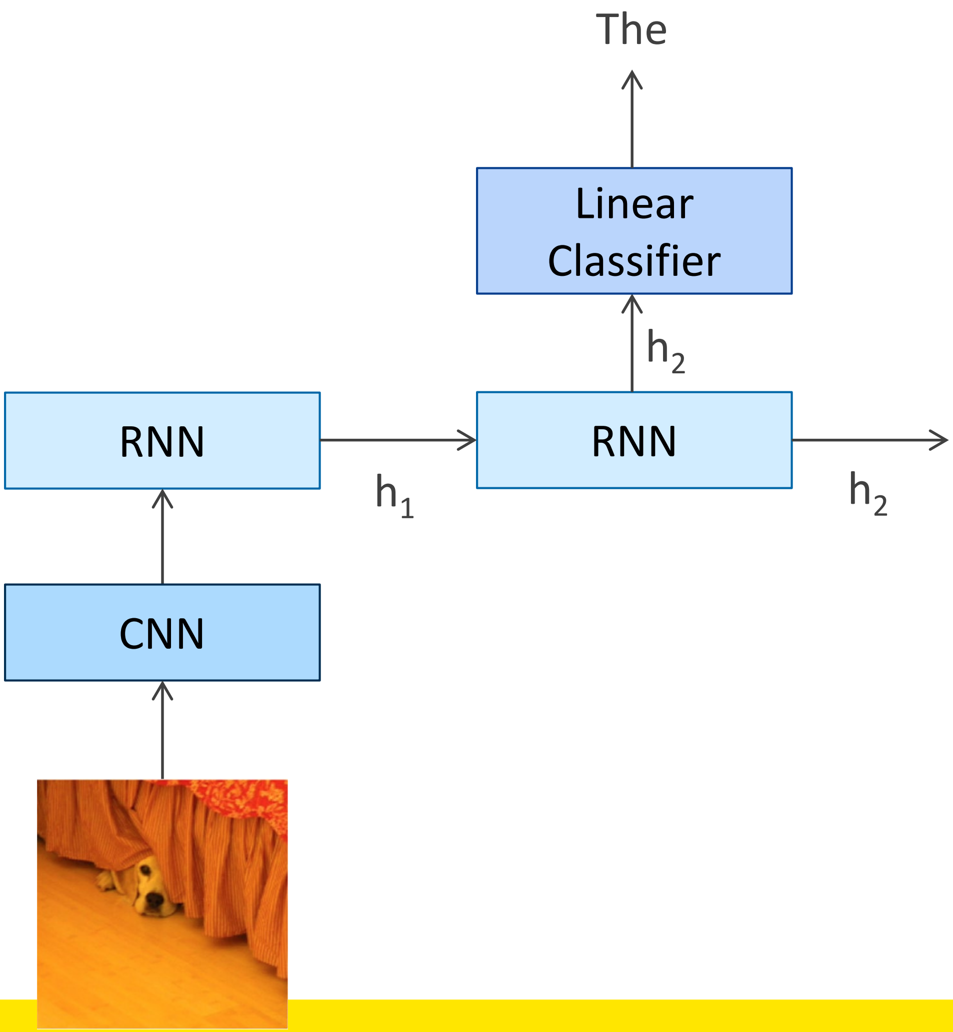
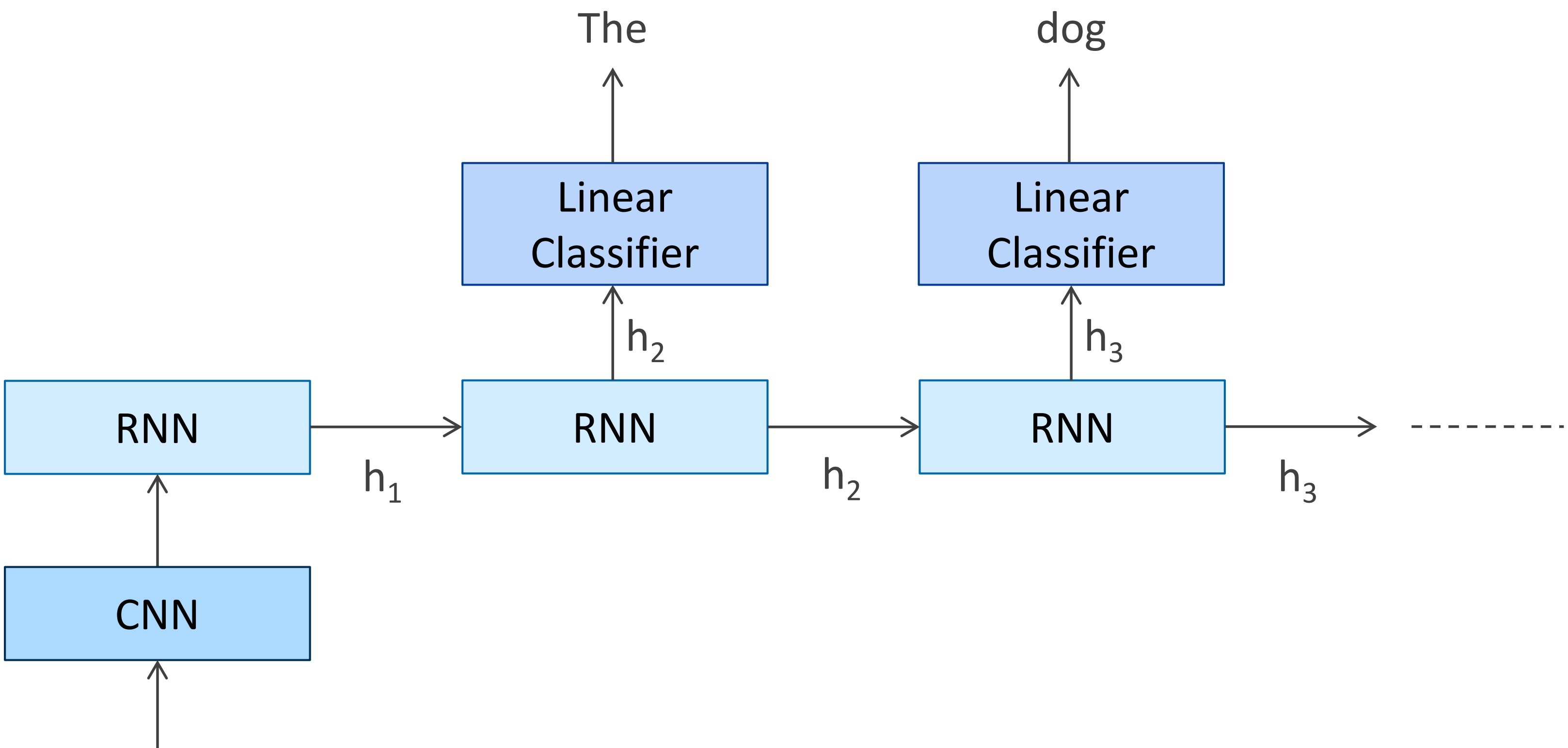


Image Captioning



RNN Outputs: Image Captions

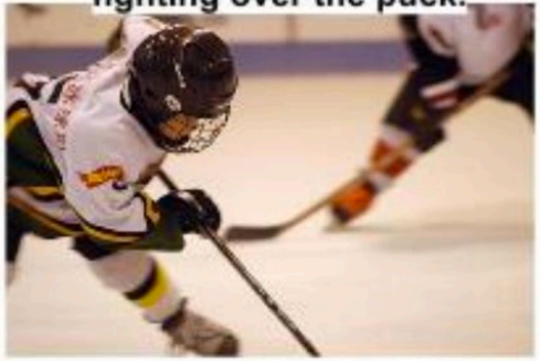
A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.



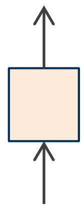
A close up of a cat laying on a couch.

[Show and Tell: A Neural Image Caption Generator, CVPR 15]

UNSW SYDNEY

Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

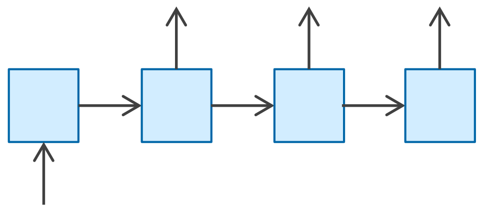
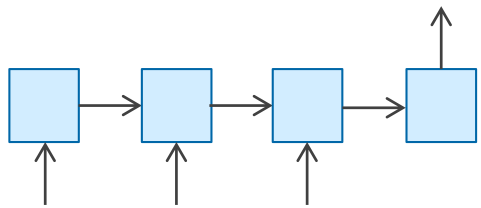


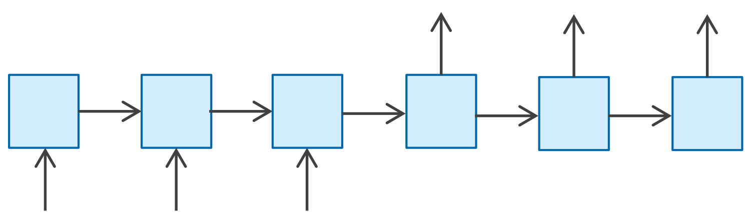
Image Captioning

Multiple - Single



Sentiment Classification

Multiple - Multiple



Translation

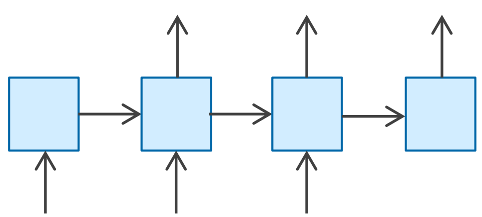


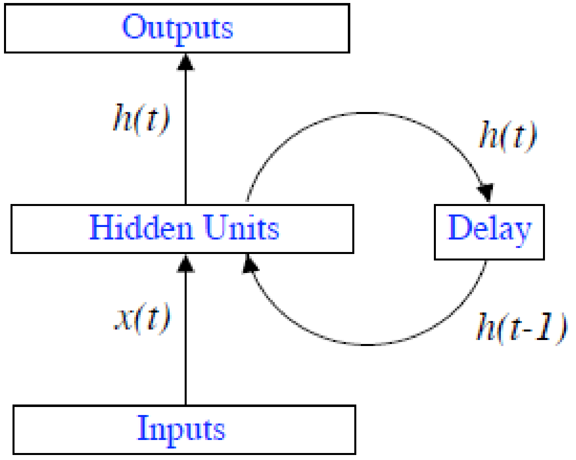
Image Captioning

Input – Output Scenarios

Note: We might deliberately choose to frame our problem as a particular input-output scenario for ease of training or better performance.
For example, at each time step, provide previous word as input for image captioning
(Single-Multiple to Multiple-Multiple).

What are RNNs?

Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs

Allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output

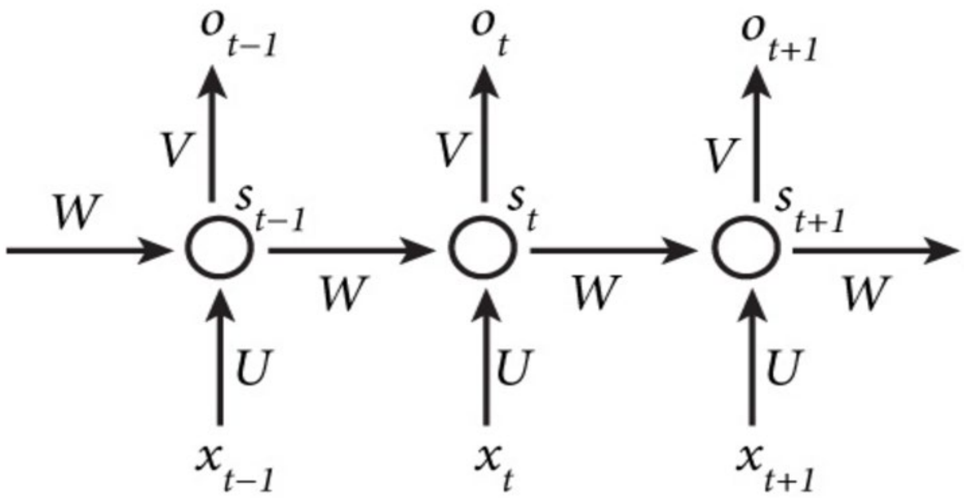
$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$

$$y(t) = f_O(W_{HO}h(t))$$

f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training

What are RNNs?

The recurrent network can be converted into a feed-forward network by unfolding over time

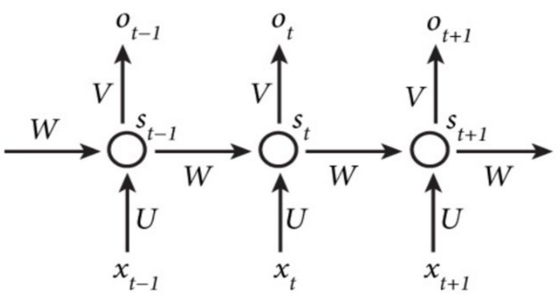


An unfolded recurrent network. Each node represents a layer of network units at a single time step. The weighted connections from the input layer to hidden layer are labelled 'w1', those from the hidden layer to itself (i.e. the recurrent weights) are labelled 'w2' and the hidden to output weights are labelled'w3'. Note that the same weights are reused at every time step. Bias weights are omitted for clarity.

What are RNNs?

Training RNNs (determine the parameters)

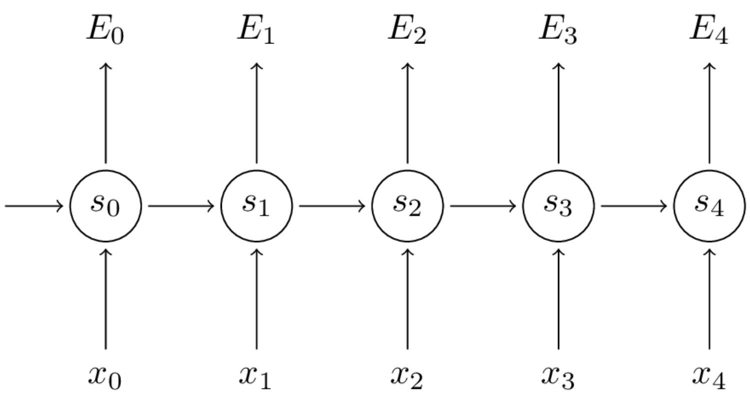
Back Propagation Through Time (BPTT) is often used to learn the RNN
BPTT is an extension of the back-propagation (BP)



- The output of this RNN is $\hat{y}_t$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

- The loss/error function of this network is



$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

The error at each time step

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

the total loss is the sum of the errors at each time step

What are RNNs?

Training RNNs (determine the parameters)

✓ The gradients of the error with respect to our parameters

Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

✓ The gradient at each time step

we use time 3 as an example

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W}$$

Chain Rule

$$s_3 = \tanh(Ux_1 + Ws_2)$$

s_3 depends on W and s_1, we cannot simply treat s_2 a constant

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

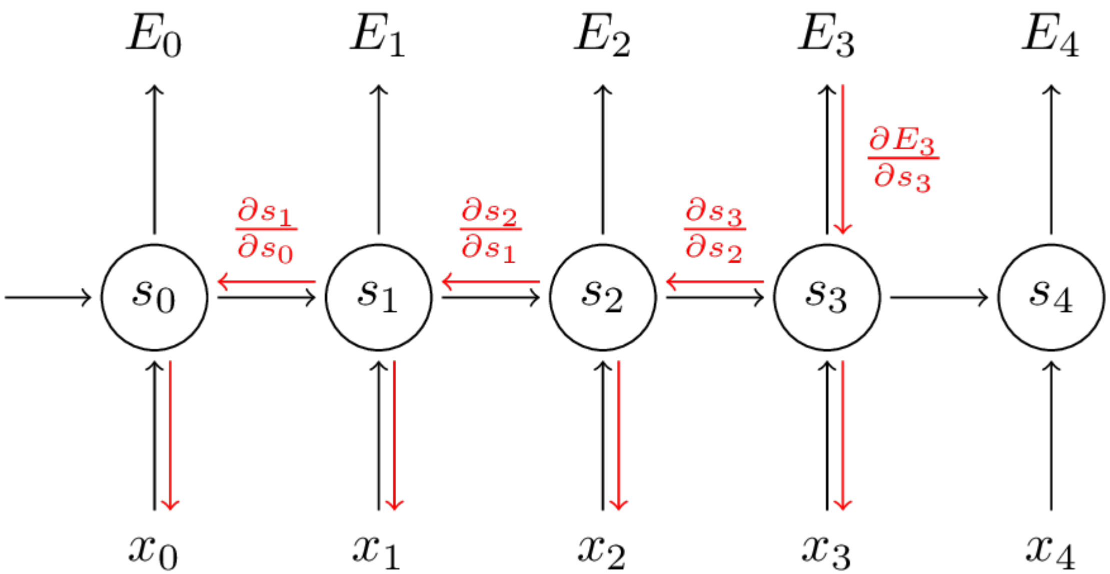
Apply Chain Rule again on s_k

What are RNNs?

- Training RNNs (determine the parameters)

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Becaise W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$



What are RNNs?

- The vanishing gradient problem

To understand why, let's take a closer look at the gradient we calculated above:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \frac{\partial s_k}{\partial W}$$

Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing

Gradient contributions from "far away" steps become zero, and the state at those steps doesn't contribute to what you are learning: You end up not learning long-range dependencies.

What are RNNs?

RNN’s use back propagation.

Back propagation uses chain rule.

- Chain rule multiplies derivatives

If these derivatives are between 0 and 1 the product vanishes as the chain gets longer.

- or the product explodes if the derivatives are greater than 1.

Sigmoid activation function in RNN leads to this problem.

Relu, in theory, avoids this problem but not in practice.

What are RNNs?

- How to sole the vanishing gradient problem?
  - Proper initialization of the W matrix can reduce the effect of vanishing gradients
  - Use ReLU instead of tanh or sigmoid activation function

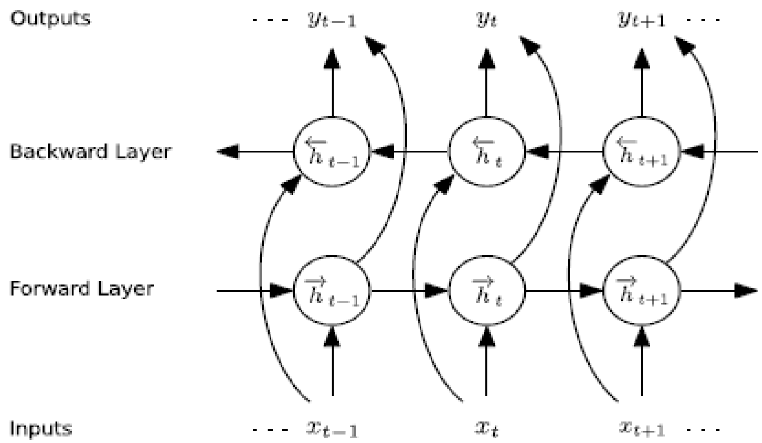
    ReLU derivate is a constant of either 0 or 1, so it isn't likely to suffer from vanishing gradients
  - Use Long Short-Term Memory or Gated Recurrent unit architectures

    LSTM will be introduced later

RNN Extensions: Bidirectional Recurrent Neural Networks

Traditional RNNs only model the dependence of the current state on the previous state, BRNNs (Schuster and Paliwal, 1997) extend to model dependence on both past states and future states.

For example: predicting a missing word in a sequence you want to look at both the left and the right context.



An unfolded BRNN

$$\overrightarrow{h}_t = f(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}})$$

$$\overleftarrow{h}_t = f(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}})$$

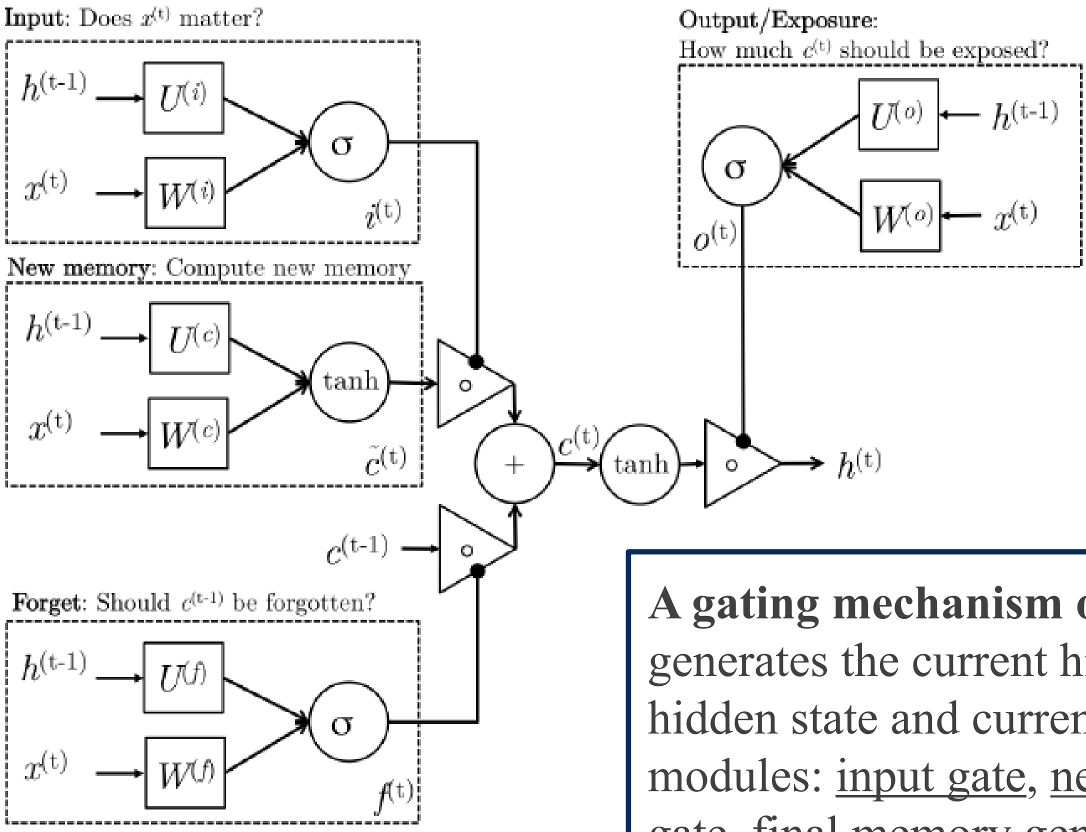
training sequence forwards and backwards to two separate recurrent hidden layers

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$

past and future context determines the output

RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.



A gating mechanism of the LSTM , which generates the current hidden state by the paste hidden state and current input ..It contains five modules: input gate, new memory cell, forget gate, final memory generation, and output gate.

RNN Extensions: Long Short Term Memory (LSTM)

LSTM provide solution to the vanishing/exploding gradient problem.

Solution: Memory Cell, which is updated at each step in the sequence.

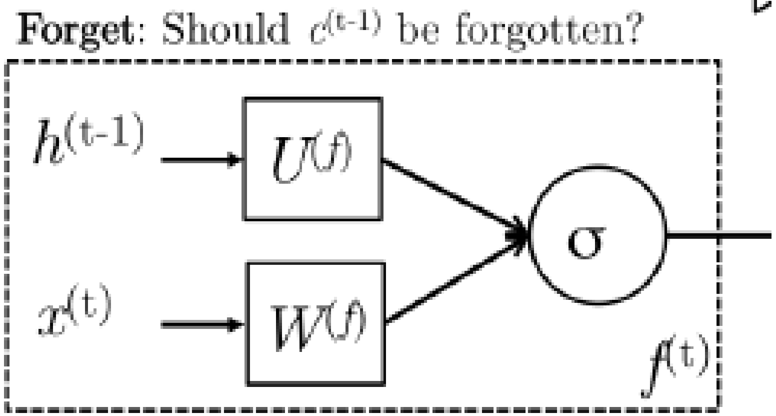
Three Gates control the flow of information to and from the Memory cell

- Input Gate: protect the current step from irrelevant inputs
- Output Gate: prevents current step from passing irrelevant information to later steps.
- Forget Gate: limits information passed from one cell to the next.

RNN Extensions: Long Short-term Memory (LSTM)

A gating mechanism of the LSTM

Forget gate



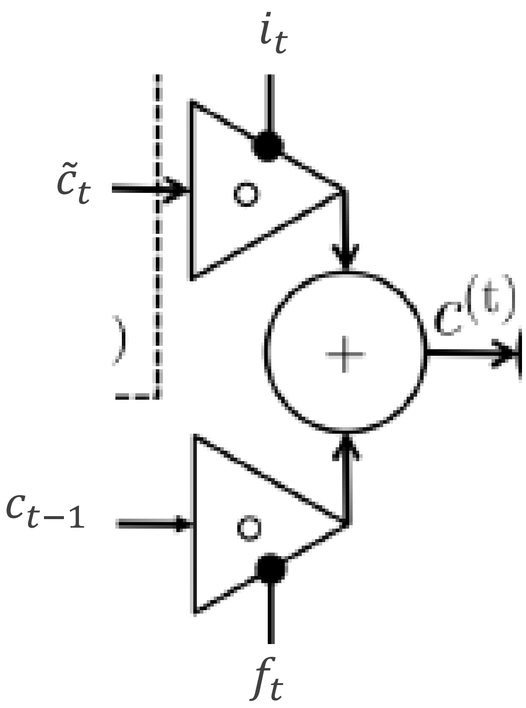
$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$

The forget gate looks at the input word and the past hidden state and makes an assessment on whether the past memory cell is useful for the computation of the current memory cell

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Final memory cell



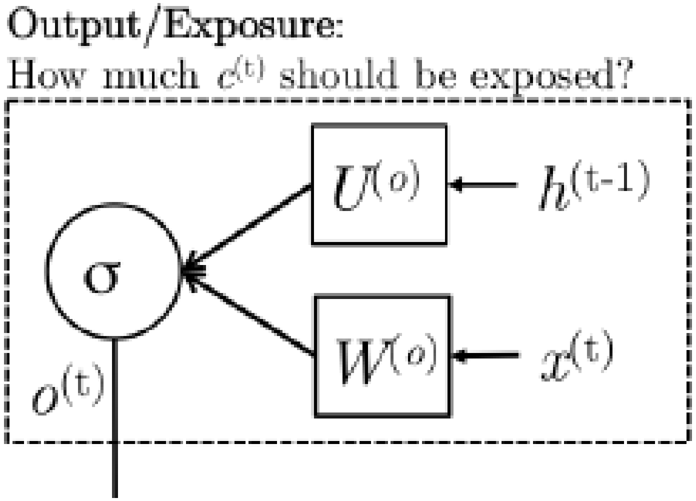
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

This stage first takes the advice of the forget gate f_t and accordingly forgets the past memory c_{t-1}. Similarly, it takes the advice of the input gate i_t and accordingly gates the new memory. It then sums these two results to produce the final memory

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Output gate



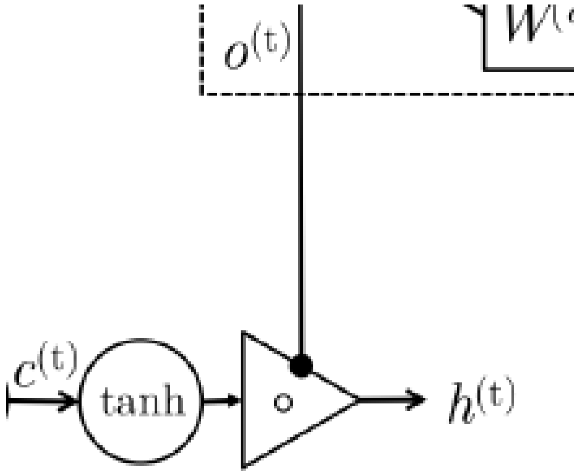
$$o_t = \sigma(W_o x_t + U_o h_{t-1})$$

This gate makes the assessment regarding what parts of the memory c_t needs to be exposed/present in the hidden state h_t.

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

The hidden state



$$h_t = o_t \circ \tanh(c_t)$$

RNN extensions: Long Short-term memory

Conclusions on LSTM

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, back-propagating error, and adjusting weights via gradient descent.

RNN extensions: Long Short-term Memory

Why LSTM can combat the vanish gradient problem?

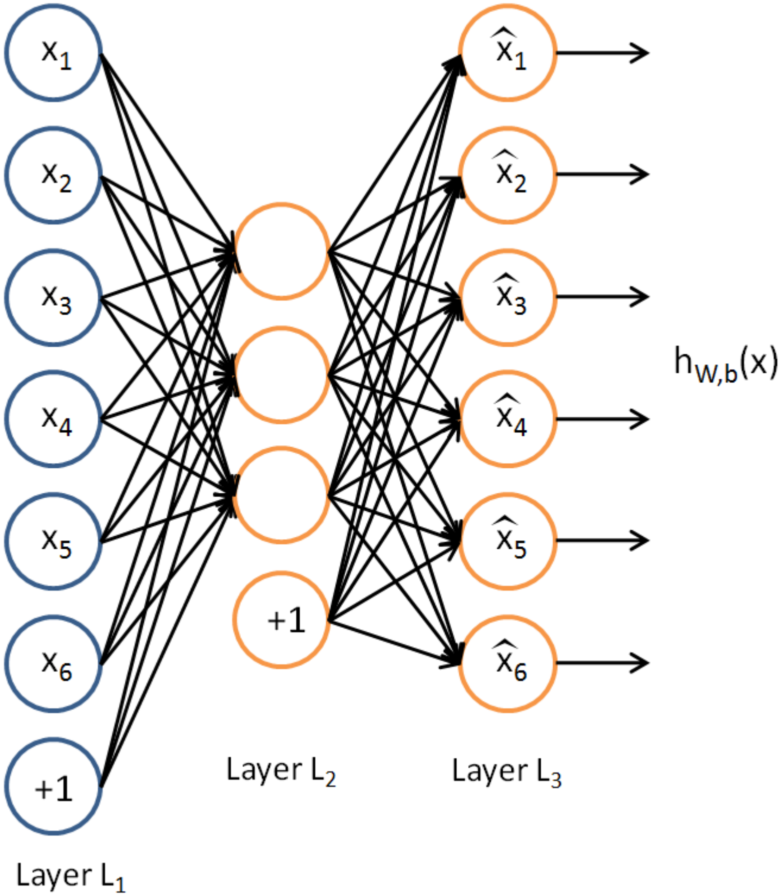
LSTMs help preserve the error that can be back-propagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely

Unsupervised Learning

Autoencoders

Deep Autoencoders

Autoencoders



An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

$$y^{(i)} = x^{(i)}$$

The input-to-hidden part corresponds to an encoder

The hidden-to-output part corresponds to a decoder.

Deep Autoencoders

- A deep Autoencoder is constructed by extending the encoder and decoder of autoencoder with multiple hidden layers.
- Gradient vanishing problem: the gradient becomes too small as it passes back through many layers

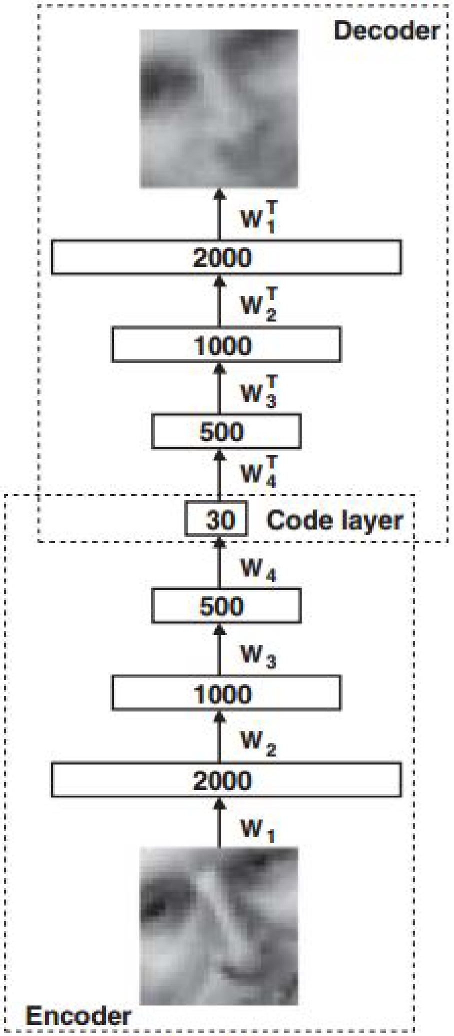


Diagram from (Hinton and Salakhutdinov, 2006)

Training Deep Autoencoders

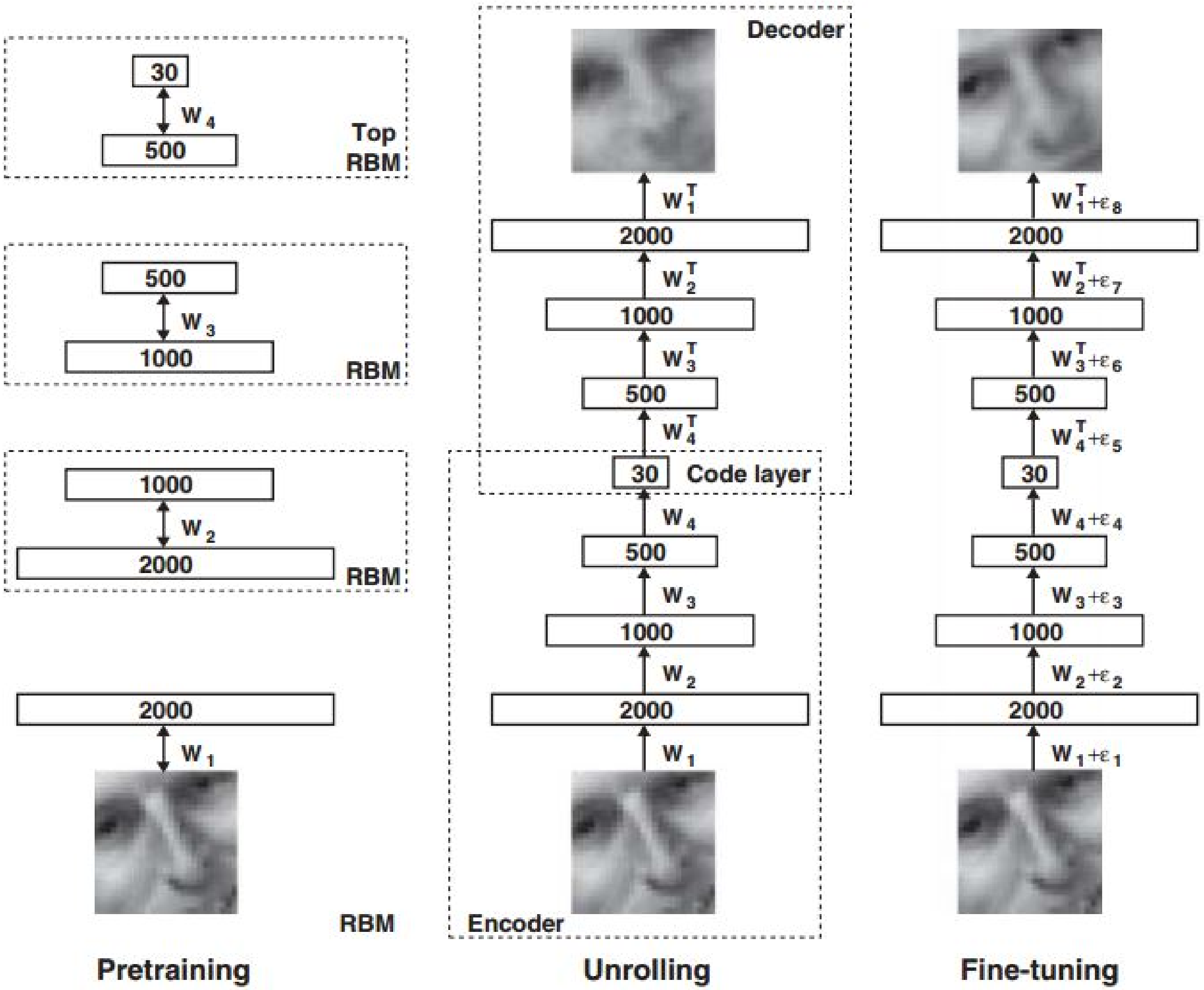
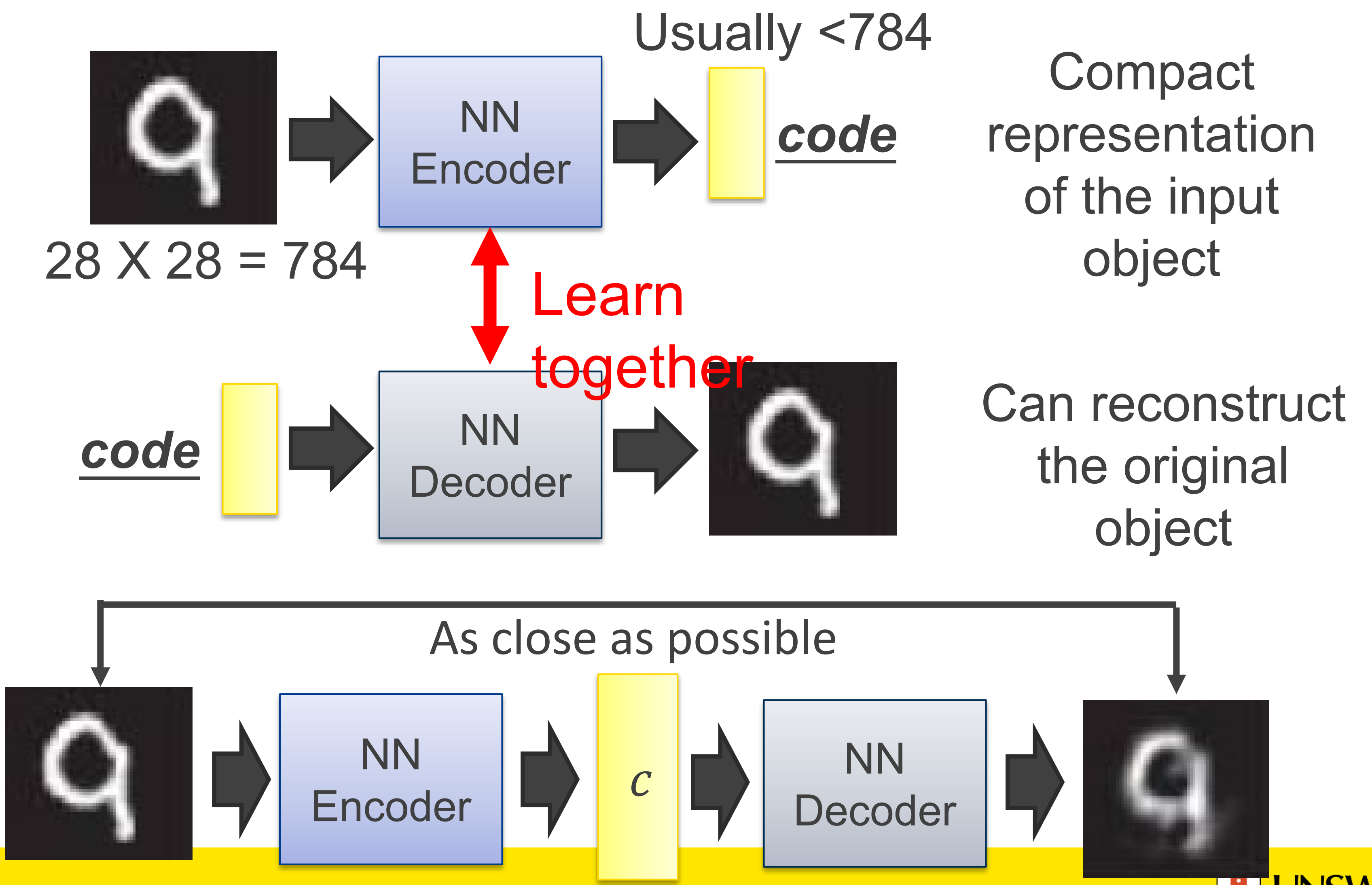


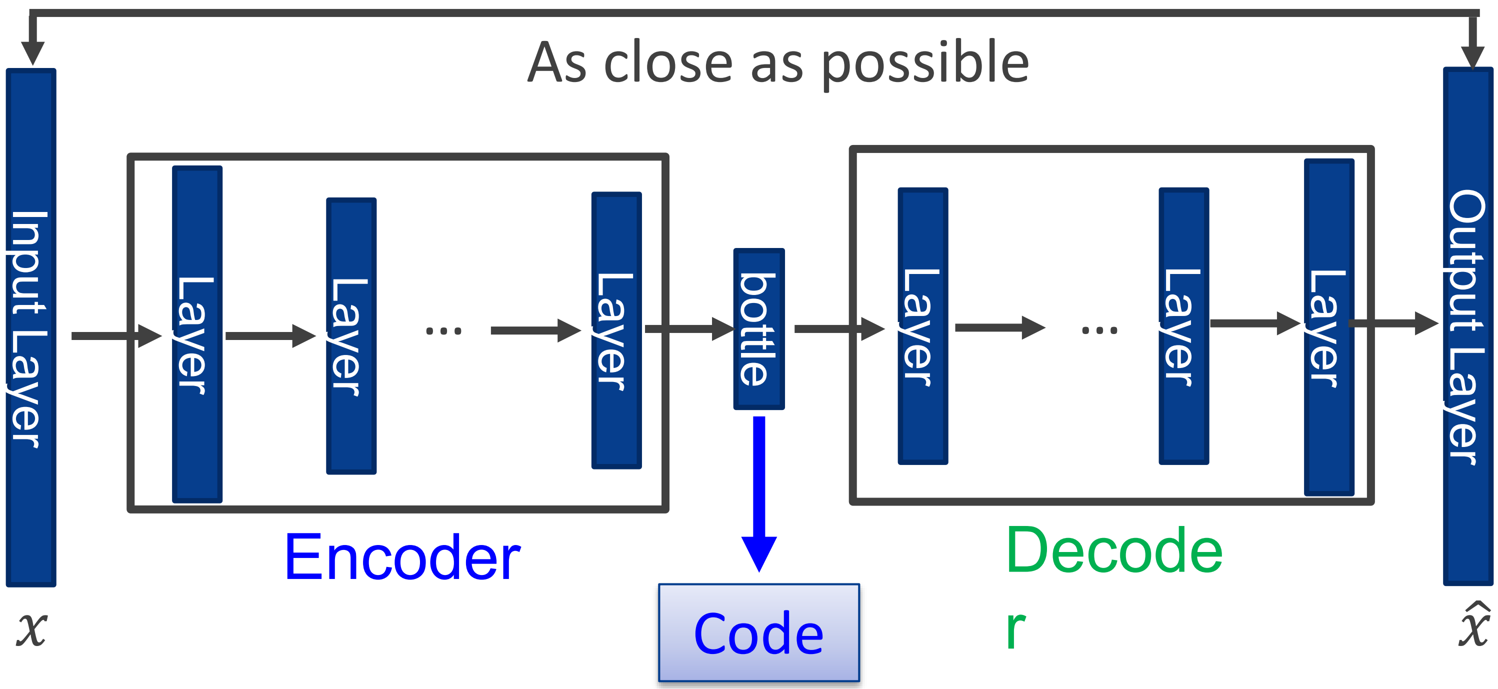
Diagram from (Hinton and Salakhutdinov, 2006)

Auto-encoder



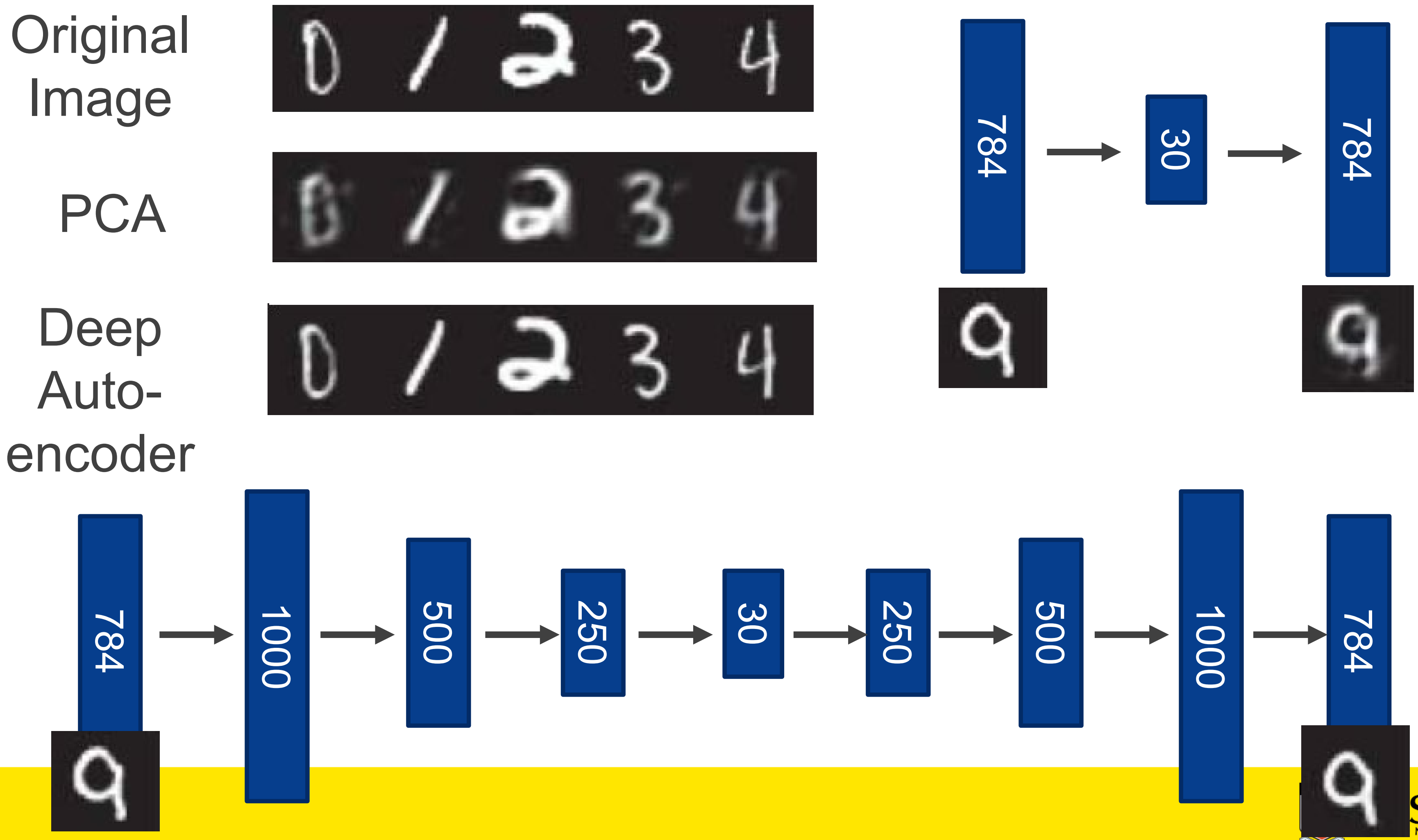
Deep Auto-encoder

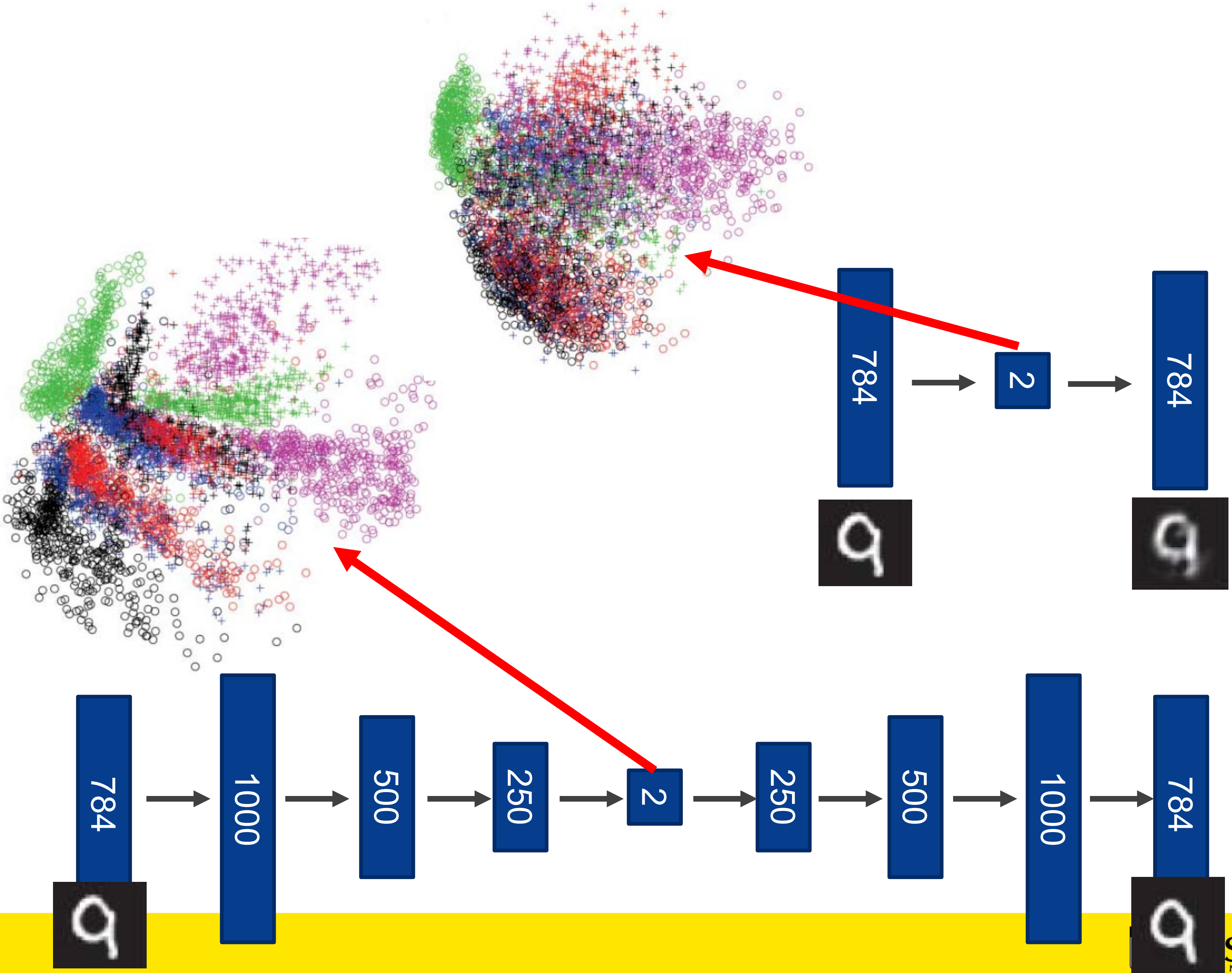
NN encoder + NN decoder = a deep network



Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

Deep Auto-encoder



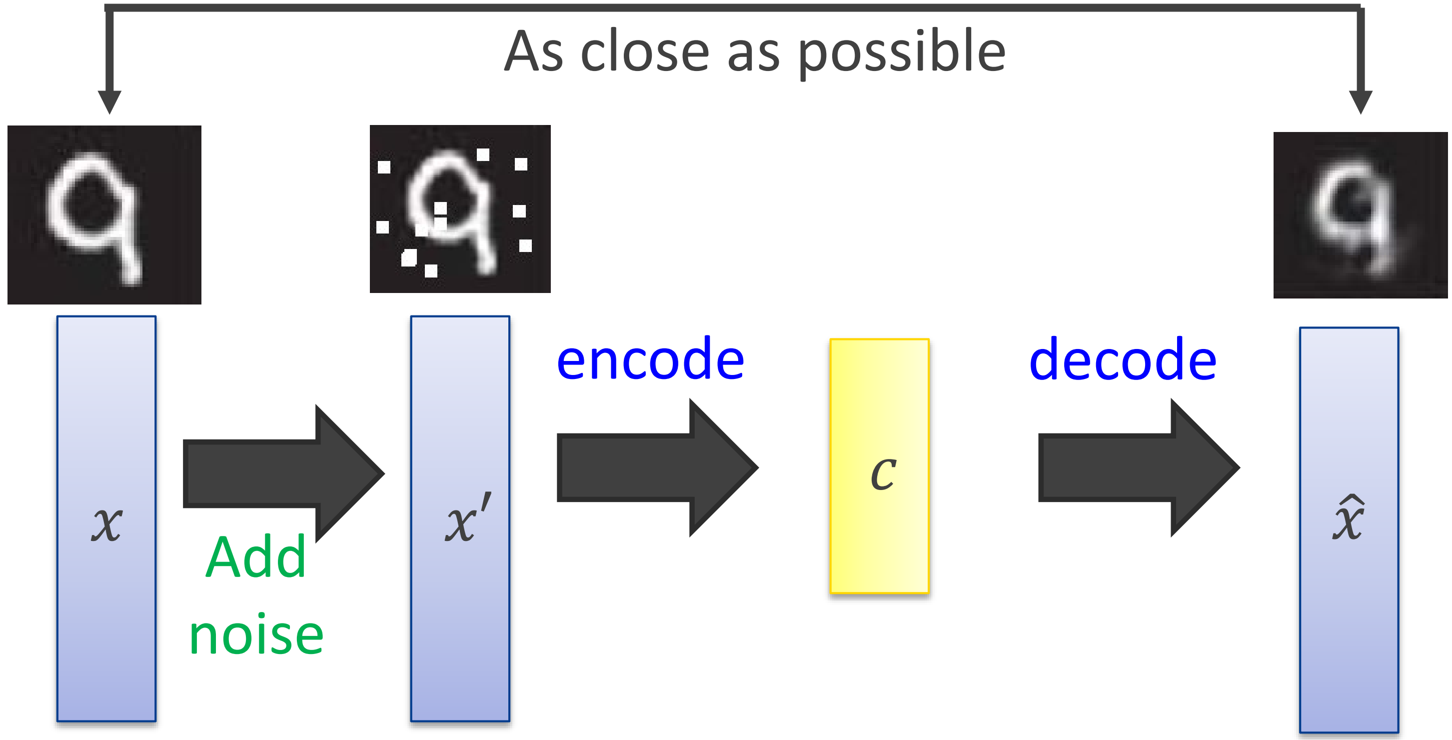

784
2
784
784
1000
500
250
2
250
500
1000
784

Auto-encoder

De-noising auto-encoder

More: Contractive auto-encoder

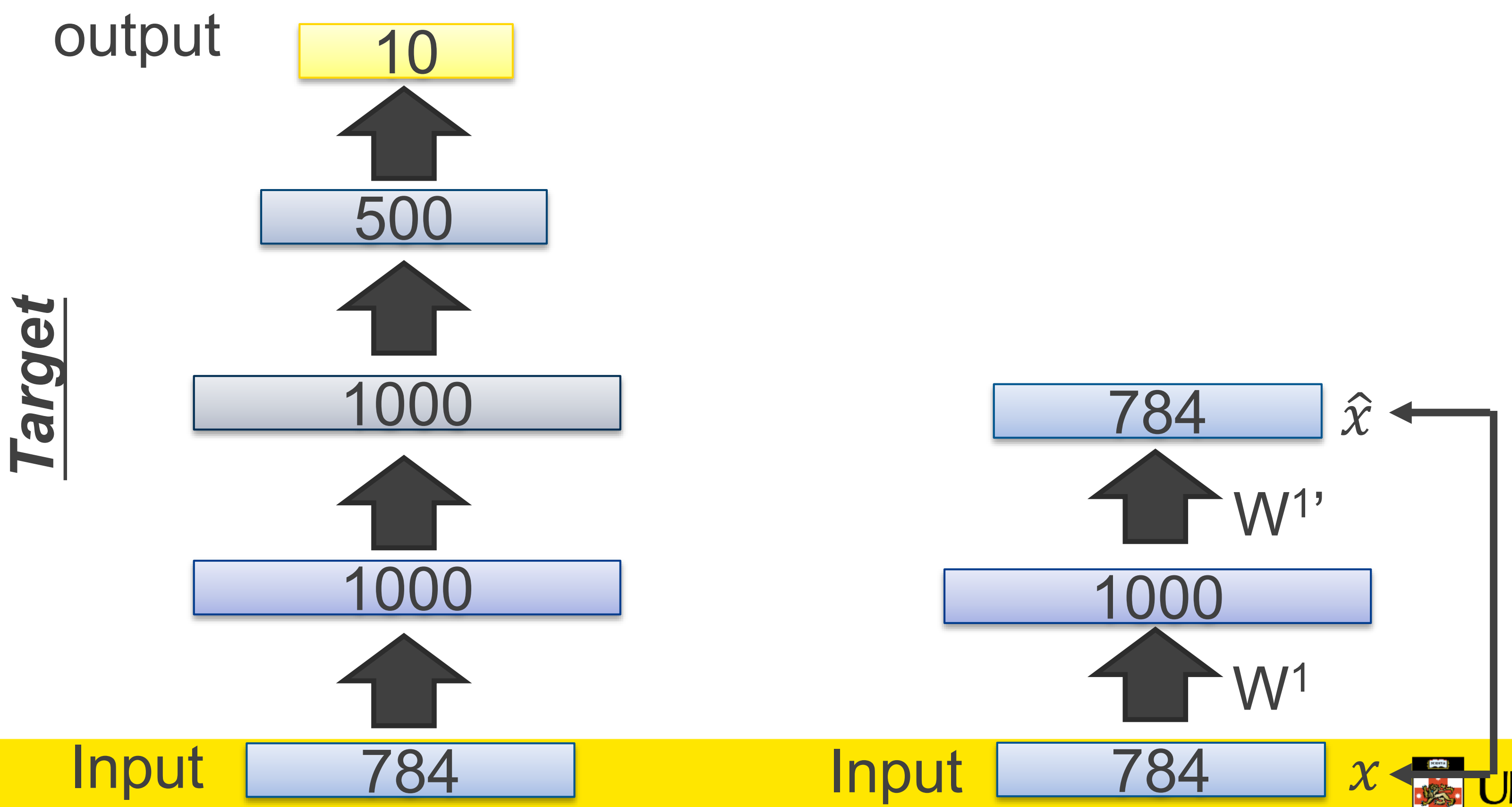
Ref: Rifai, Salah, et al. "Contractive auto-encoders: Explicit invariance during feature extraction.“ *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.



Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML*, 2008.

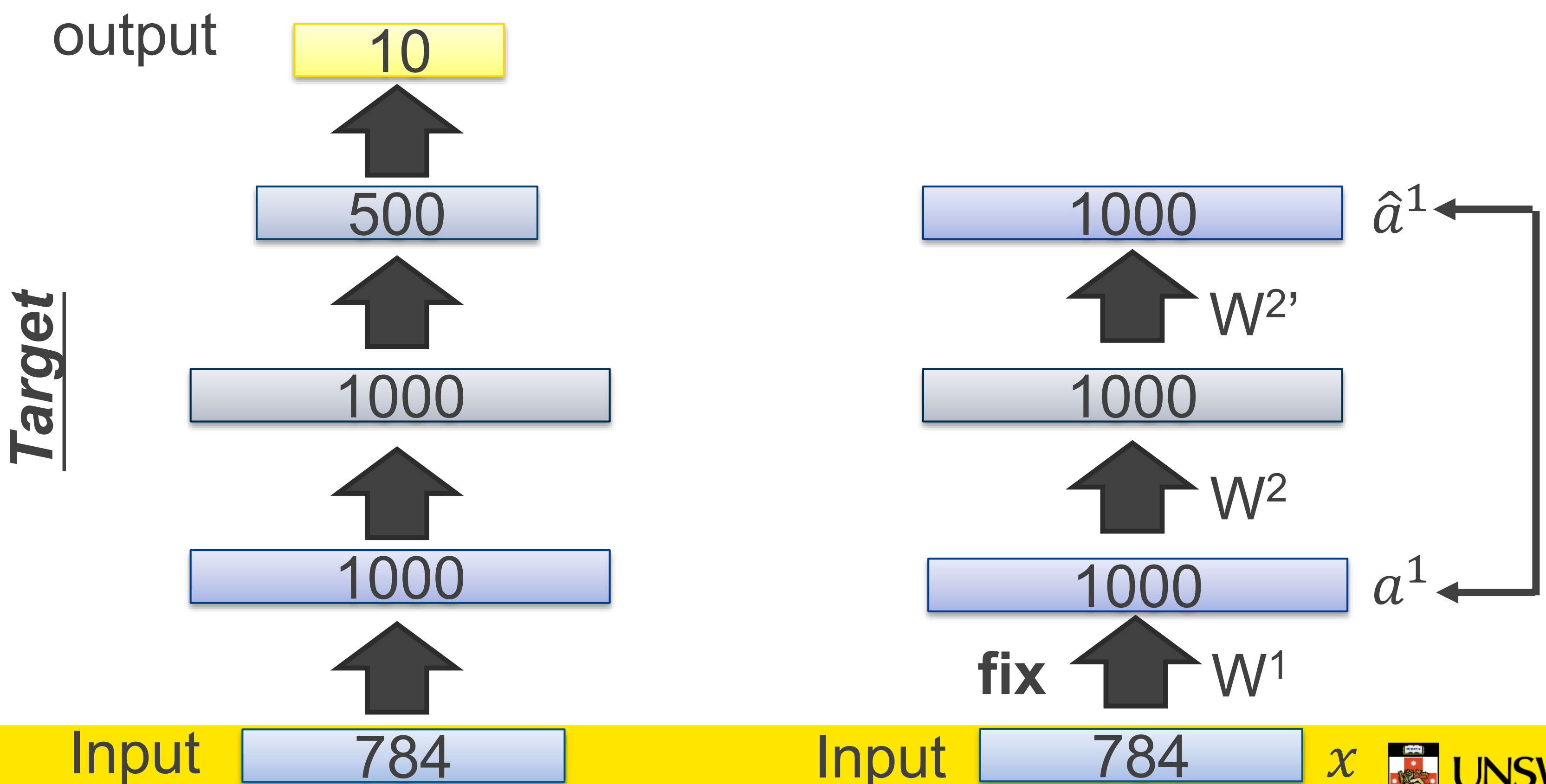
Auto-encoder – Pre-training DNN

Greedy Layer-wise Pre-training *again*



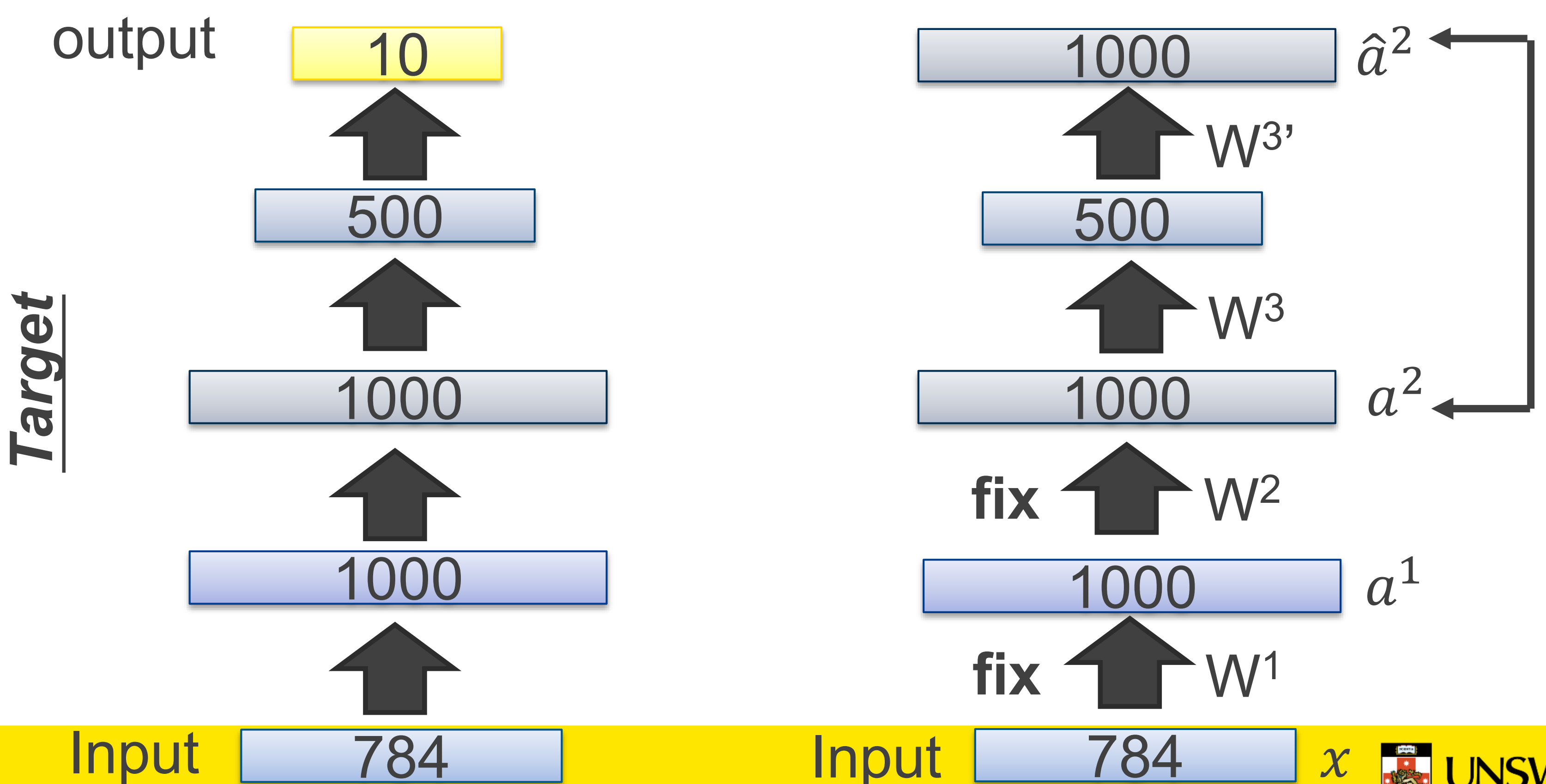
Auto-encoder – Pre-training DNN

Greedy Layer-wise Pre-training *again*



Auto-encoder – Pre-training DNN

Greedy Layer-wise Pre-training *again*



Auto-encoder – Pre-training DNN

Greedy Layer-wise Pre-training *again*