Carnegie Mellon University
Heinz College

Unstructured Data Analysis for Policy

Last lecture: Image analysis with CNNs, time series analysis with RNNs, deep learning & course wrap-up

George Chen

(Last Time) Neural Net as Function Approximation

Given `input`, learn a computer program that computes `output`

Multinomial logistic regression:

```
output = softmax(np.dot(input, W) + b)
```

Multilayer perceptron:

```
intermediate = relu(np.dot(input, W1) + b1)

output = softmax(np.dot(intermediate, W2) + b2)
```

Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)

(Last Time) Convolution

Very commonly used for:

- Blurring an image

*

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

- Finding edges

*

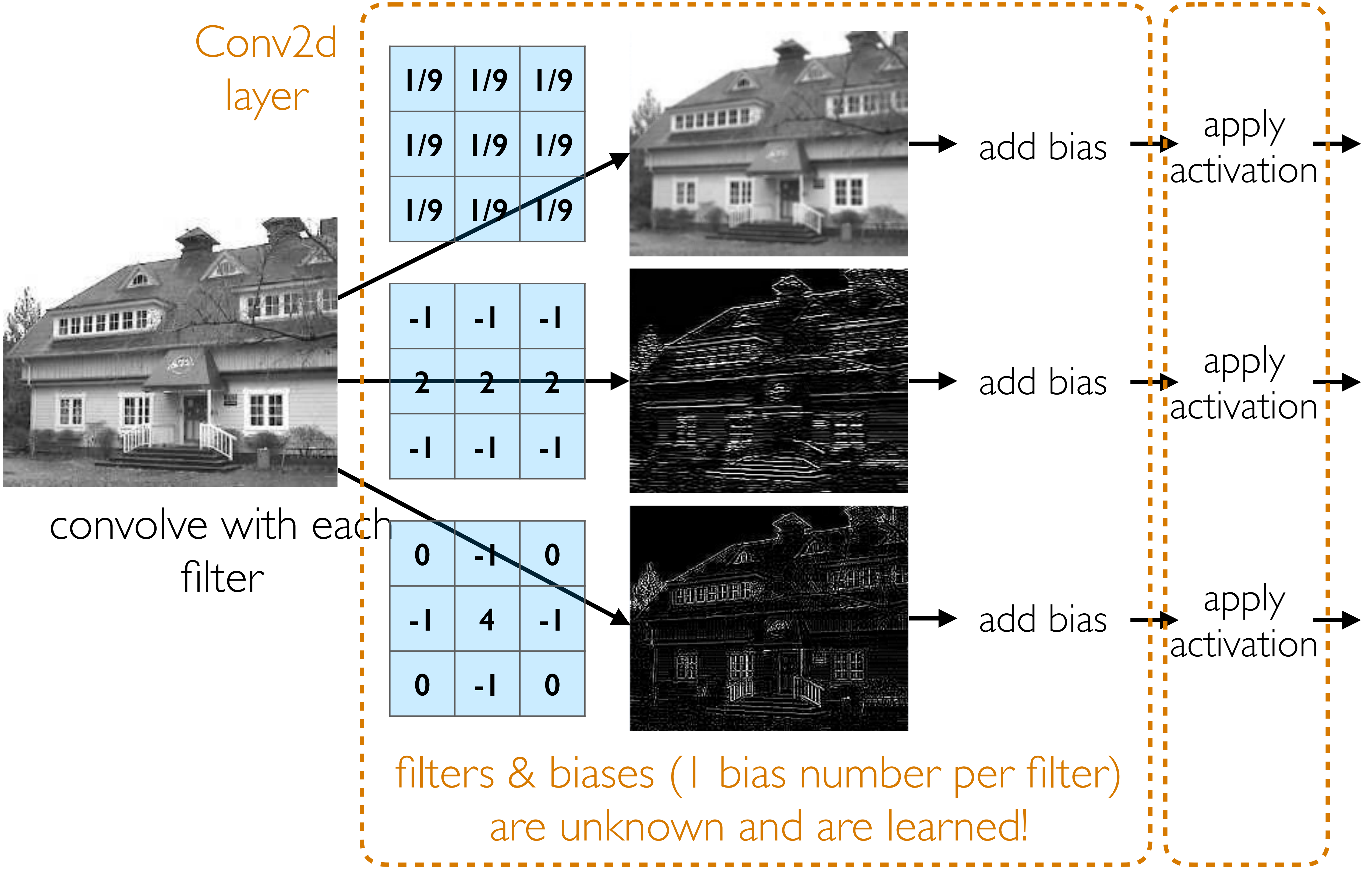
-1	-1	-1
2	2	2
-1	-1	-1

=

(this example finds horizontal edges)

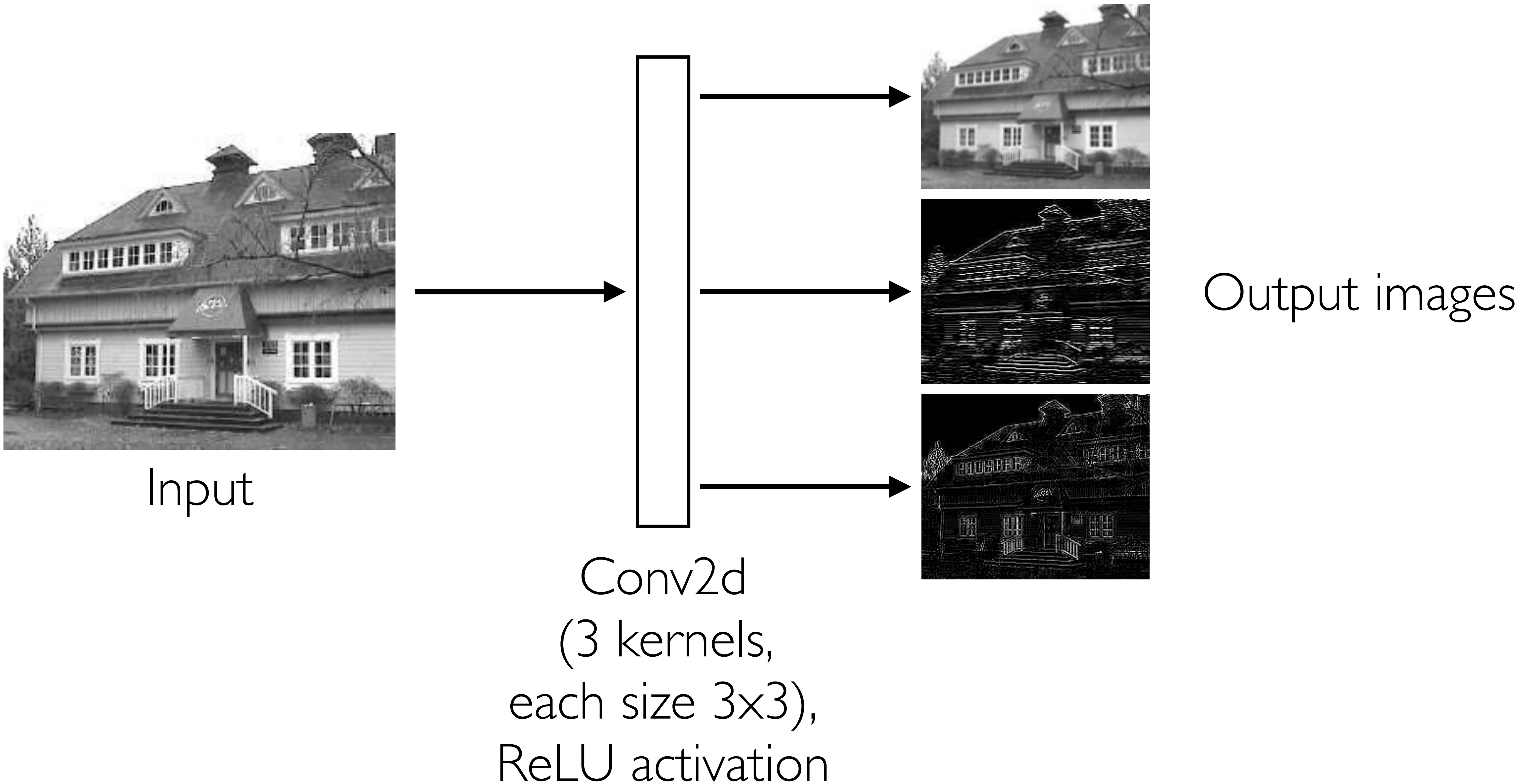
Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input

dimensions:
1 (# channels),
height,
width

Conv2d
(3 kernels,
each size 3x3),
ReLU activation

Stack output images into a single "output feature map"

dimensions:
3,
height-2,
width-2

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input

dimensions:
1 (# channels),
height,
width

Conv2d
(***k*** kernels
each size 3x3),
ReLU activation

Stack output images into a single "output feature map"

dimensions:
k,
height-2,
width-2

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input

dimensions:
d (# channels)
height,
width

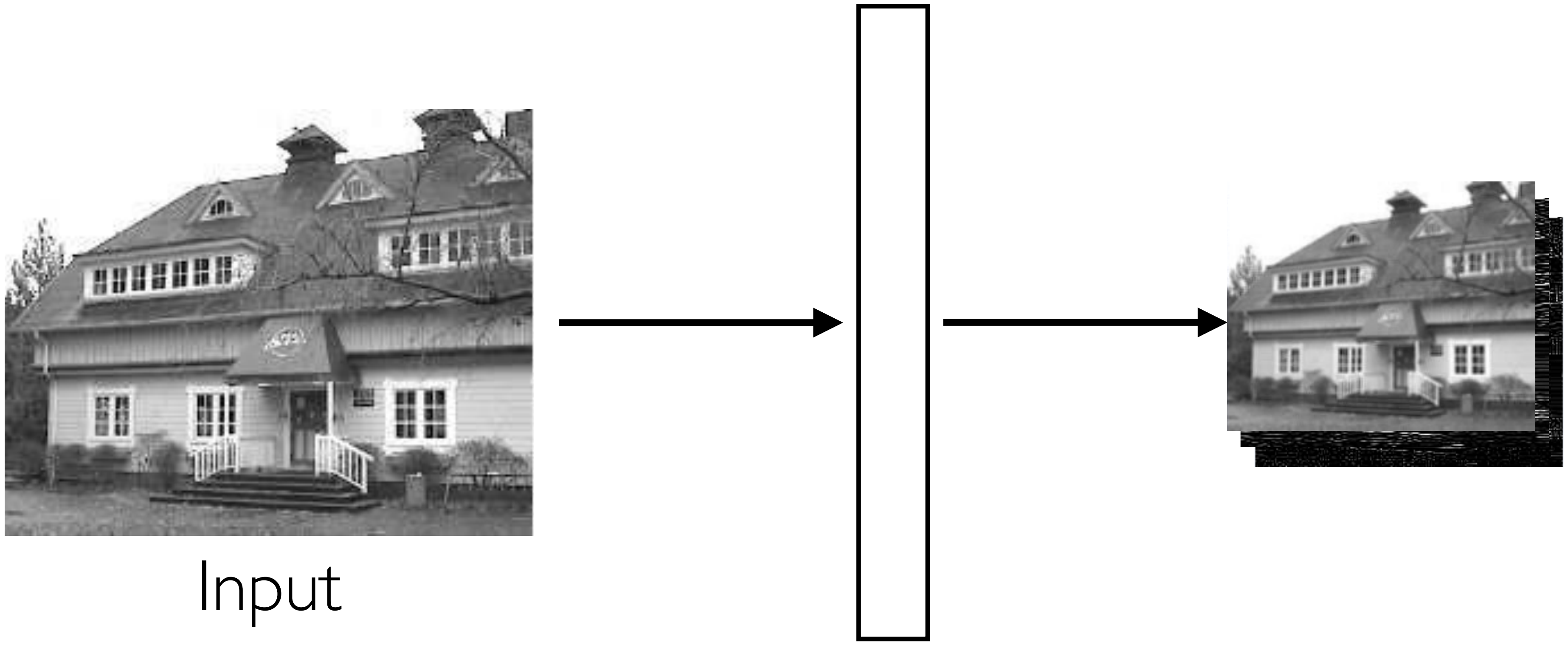
Conv2d
(***k*** kernels
each size ***d***x3x3),
ReLU activation

Stack output images into a single "output feature map"

dimensions:
k,
height-2,
width-2

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Input

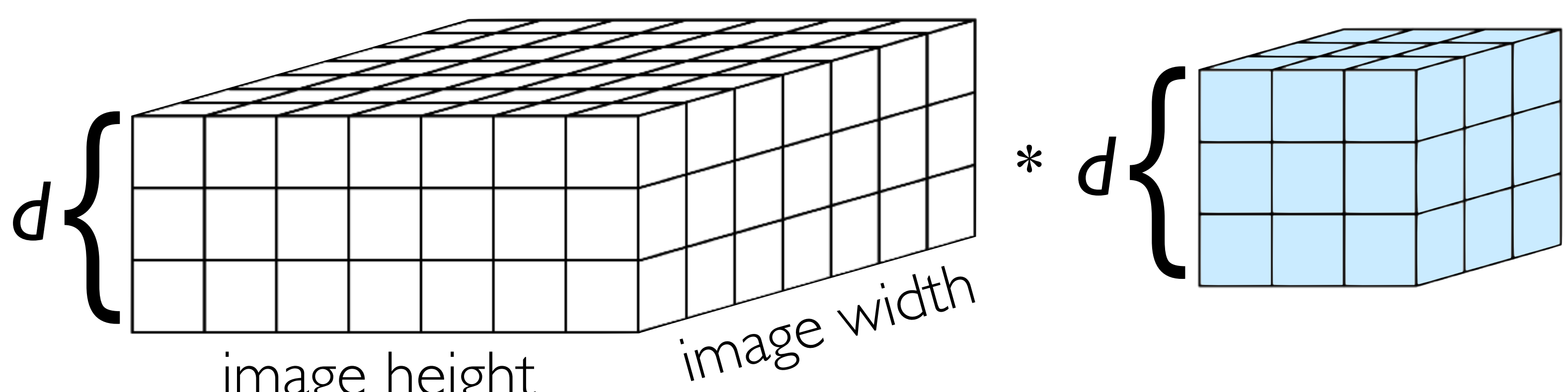
dimensions:
d (# channels)
height,
width

Conv2d
(***k*** kernels
each size ***d***x3x3),
ReLU activation

Stack output images into a single "output feature map"

dimensions:
k,
height-2,
width-2

Each filter:



Pooling

- Produces smaller image summarizing original larger image
- To produce this smaller image, need to aggregate or "pool" together information
- If "object" in input image shifts by a little bit, want output to stay the same

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

$$\begin{bmatrix} 0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0\\ 0&1&1&1&1&1&0\\ 0&1&1&1&0&0&0\\ 0&1&1&1&1&1&0\\ 0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0 \end{bmatrix} * \begin{bmatrix} -1&-1&-1\\ 2&2&2\\ -1&-1&-1 \end{bmatrix} = \begin{bmatrix} 0&1&3&1&0\\ 1&1&1&3&3\\ 0&0&-2&-4&-4\\ 1&1&1&3&3\\ 0&1&3&1&0 \end{bmatrix}$$

Input

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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

Input

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

0	1	3	1	0
1	1	1	3	3
0	0	-2	-4	-4
1	1	1	3	3
0	1	3	1	0

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

Output after max pooling

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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

Input

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

0	1	3	1	0
1	1	1	3	3
0	0	-2	-4	-4
1	1	1	3	3
0	1	3	1	0

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	

Output after max pooling

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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

Input

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

0	1	3	1	0
1	1	1	3	3
0	0	-2	-4	-4
1	1	1	3	3
0	1	3	1	0

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3

Output after max pooling

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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

Input

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

0	1	3	1	0
1	1	1	3	3
0	0	-2	-4	-4
1	1	1	3	3
0	1	3	1	0

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3
1	

Output after max pooling

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
\mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\
\mathbf{2} & \mathbf{2} & \mathbf{2} \\
\mathbf{-1} & \mathbf{-1} & \mathbf{-1}
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0
\end{bmatrix}
$$

Input

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3
1	3

Output after max pooling

Max Pooling

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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

Input

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

0	1	3	1	0
1	1	1	3	3
0	0	-2	-4	-4
1	1	1	3	3
0	1	3	1	0

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

What numbers were involved in computing this 1?

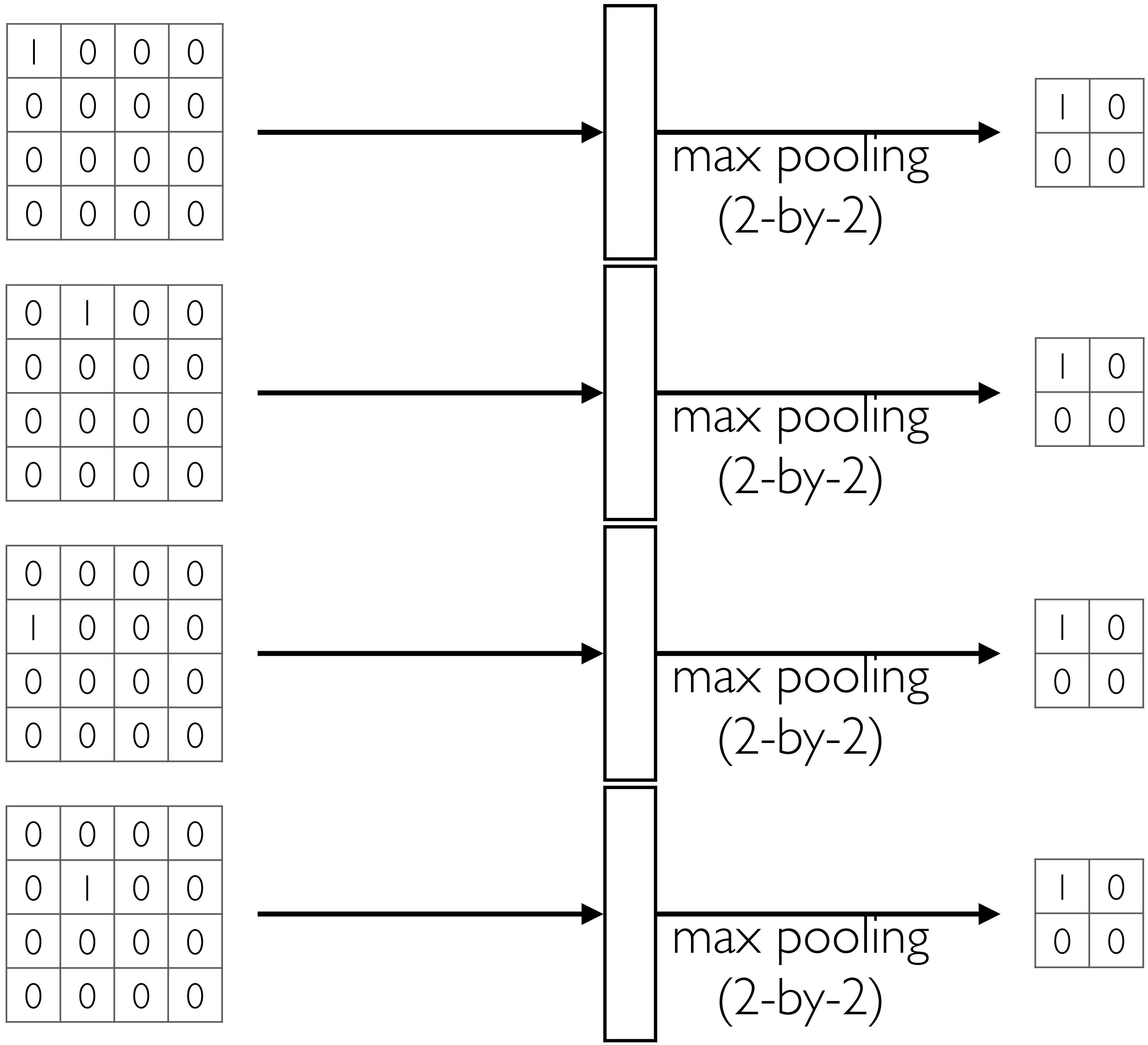
1	3
1	3

Output after max pooling

In this example: 1 pixel in max pooling output captures information from 16 input pixels!

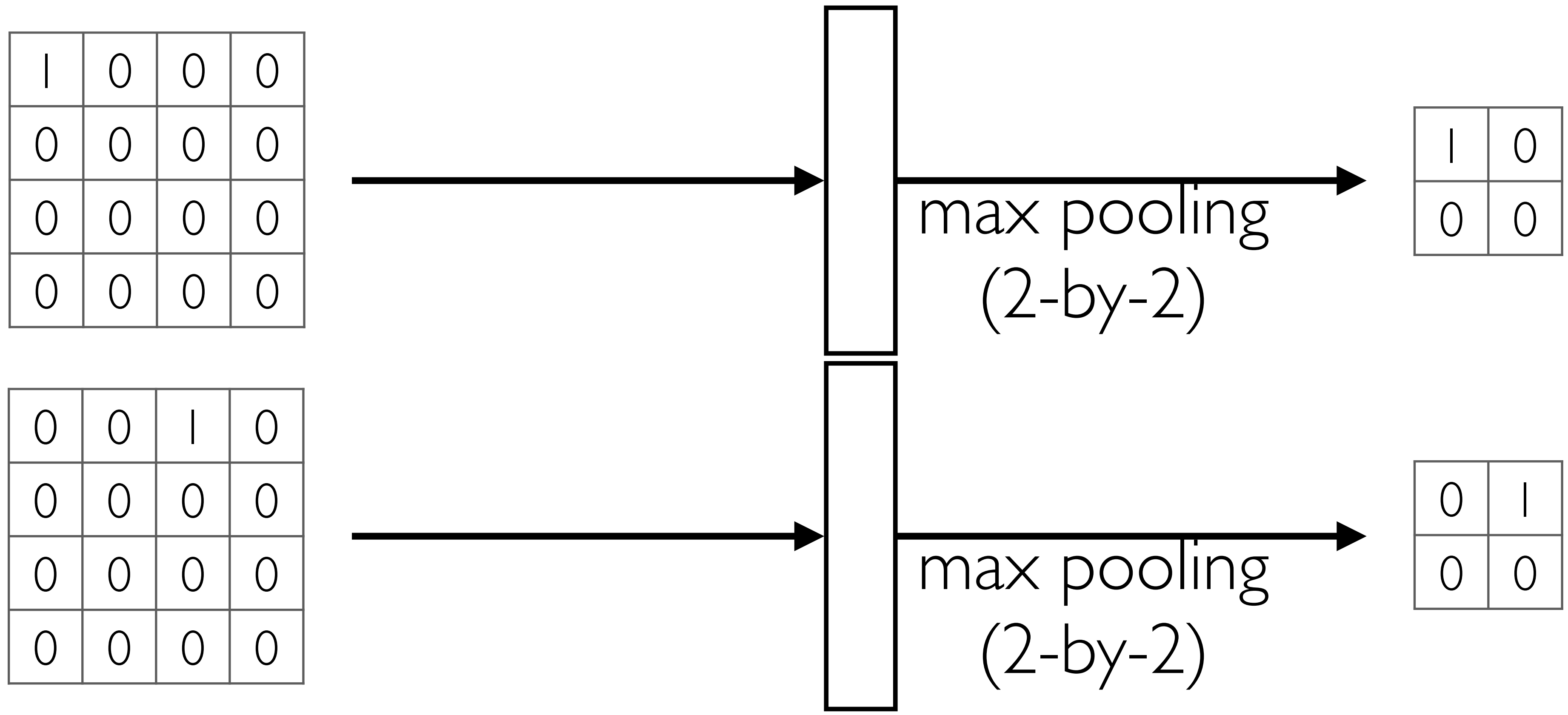
Example: applying max pooling again results in a single pixel that captures info from entire input image!

Small Shifts & Max Pooling



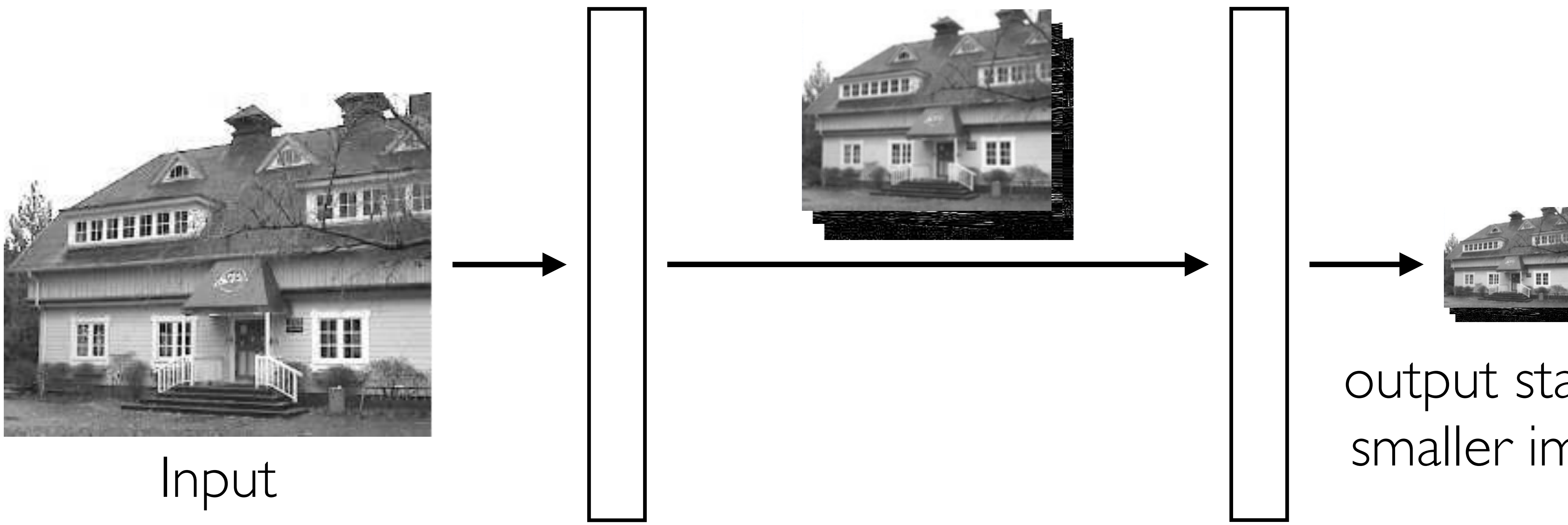
Small shift in input object of interest results in same output

Small Shifts & Max Pooling



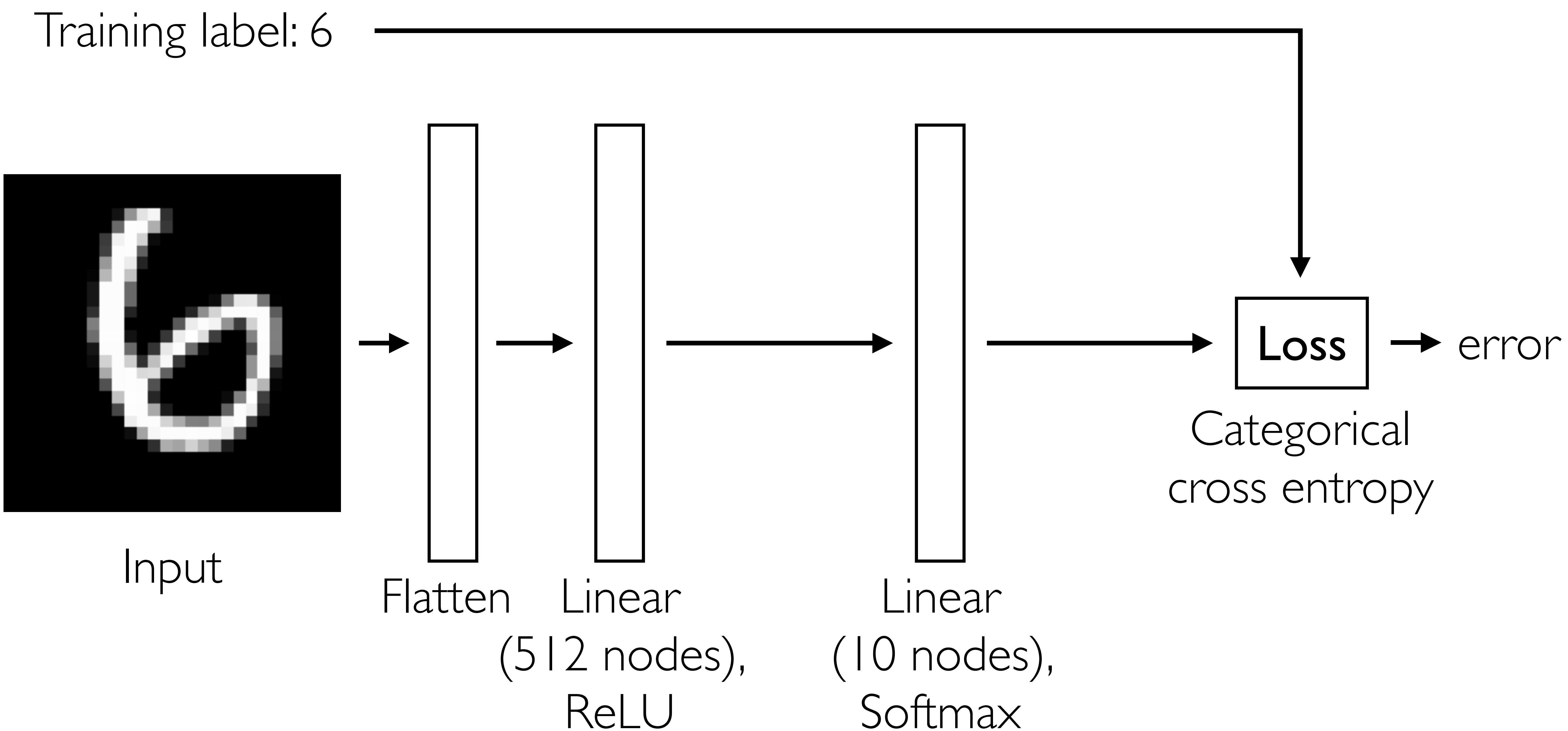
A bigger shift in the input
results in a different output

Basic Building Block of CNNs

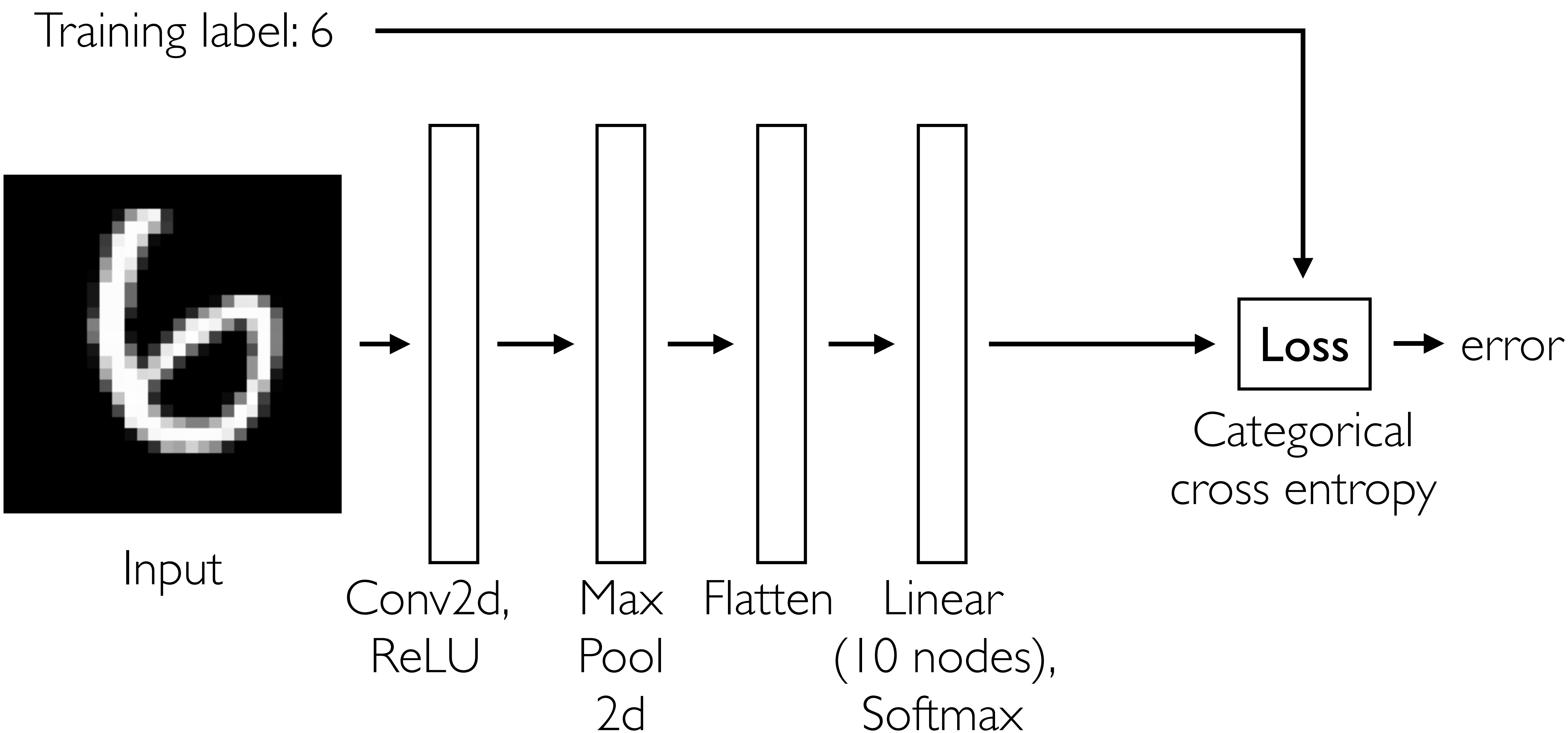


Images from: http://aishack.in/tutorials/image-convolution-examples/

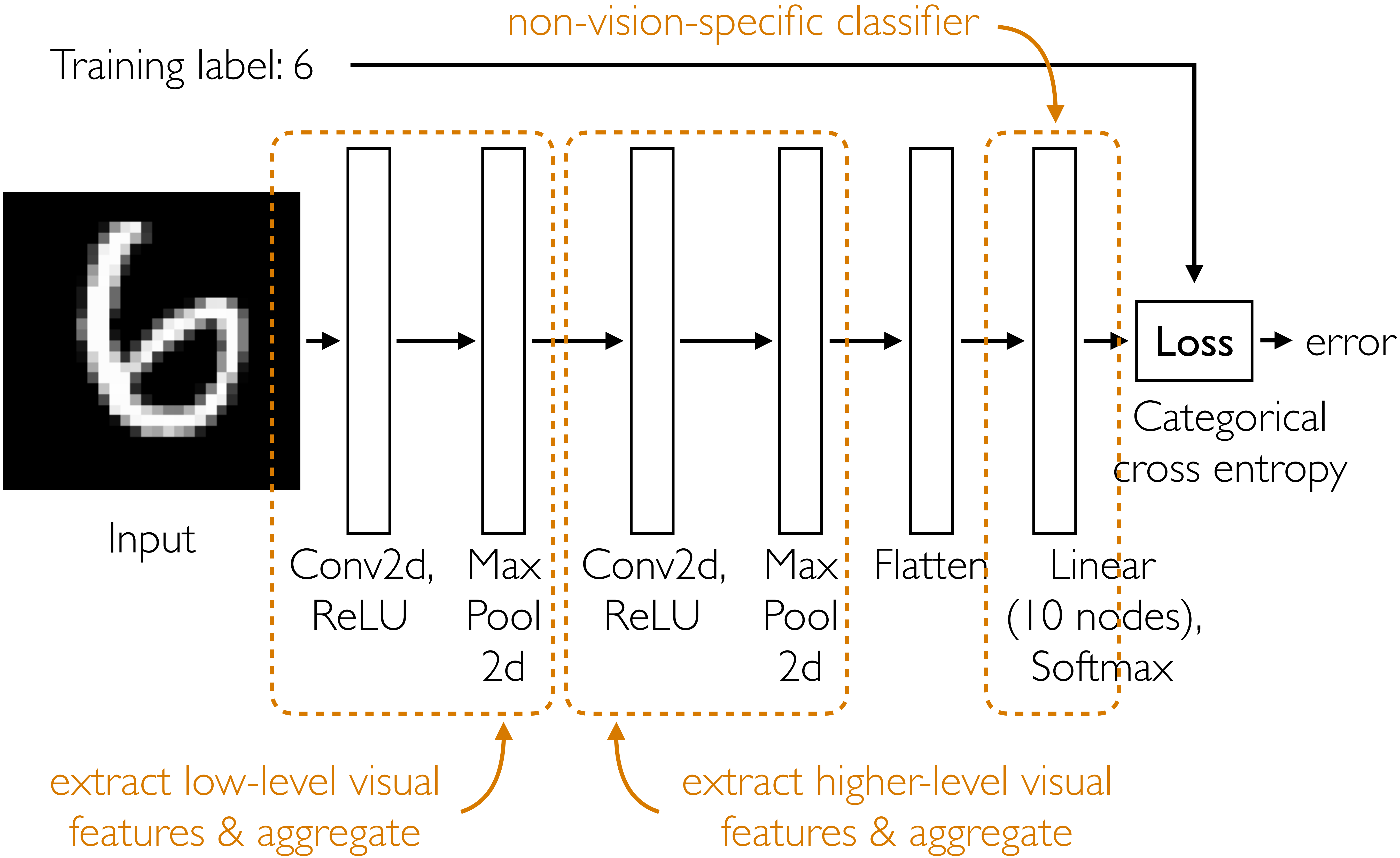
Handwritten Digit Recognition



Handwritten Digit Recognition



Handwritten Digit Recognition



CNNs

Demo

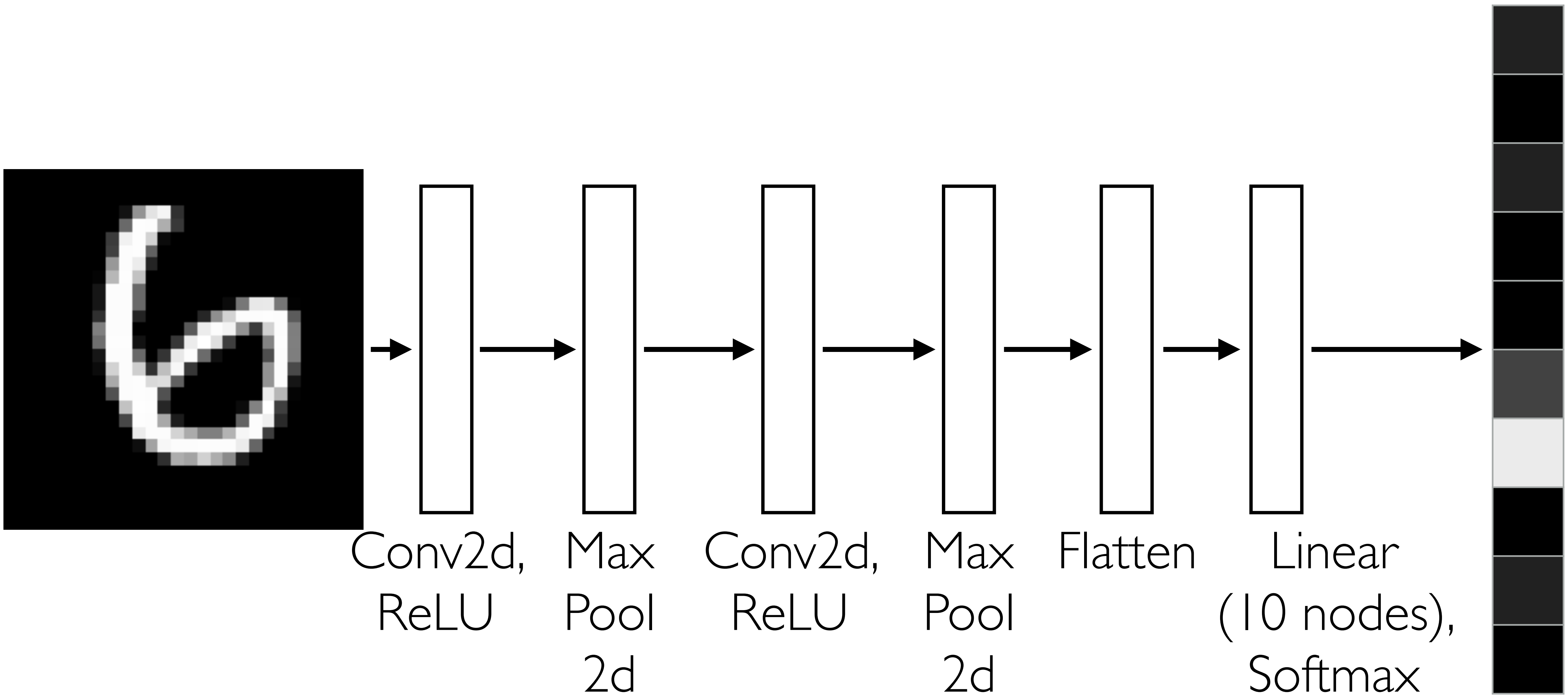
CNNs

- Learn convolution filters for extracting simple features

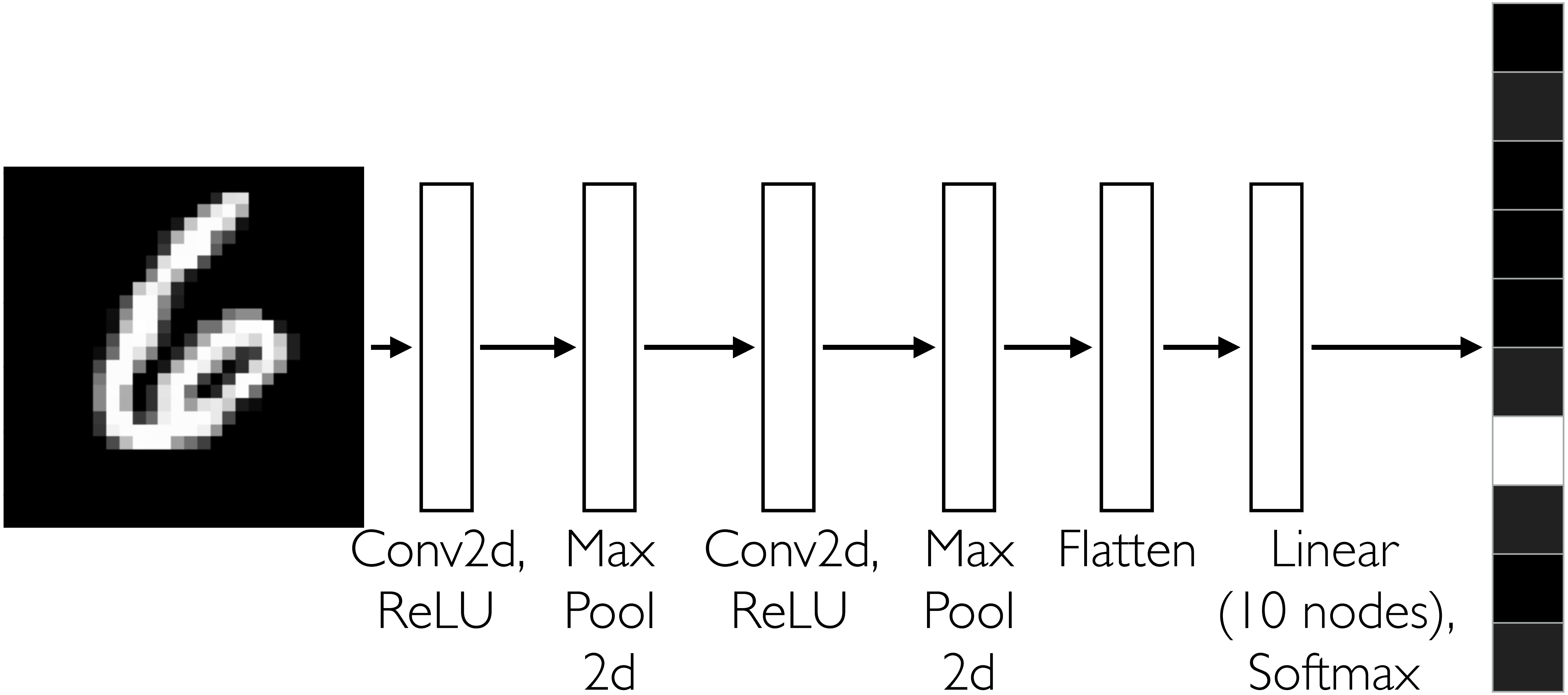
- Max pooling produces a *smaller* summary output and is somewhat invariant to small shifts in input "objects"
  - For examples where max pooling fails to achieve this and for a better way to do pooling, see Richard Zhang's fix for max pooling linked on the course webpage

- Repeat convolution→activation→pooling to learn increasingly higher-level features

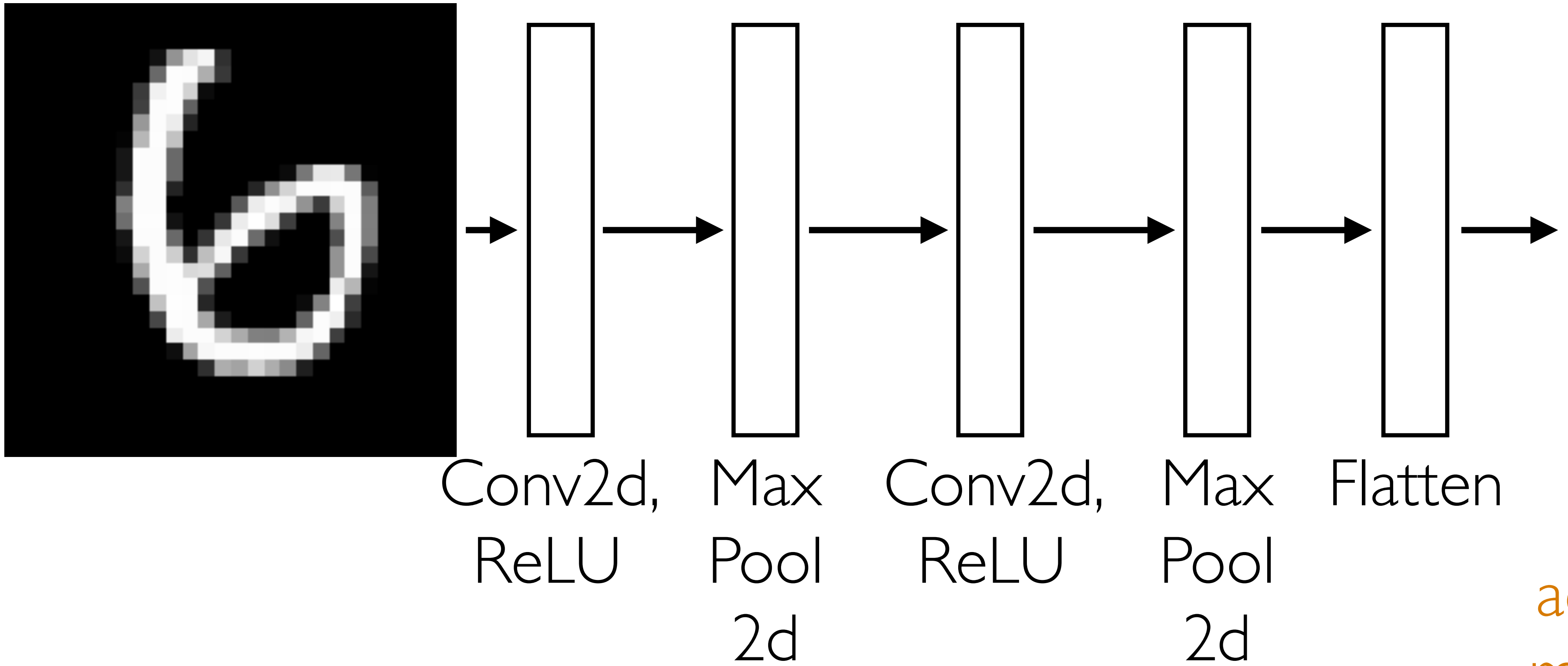
CNNs Encode Semantic Structure for Images



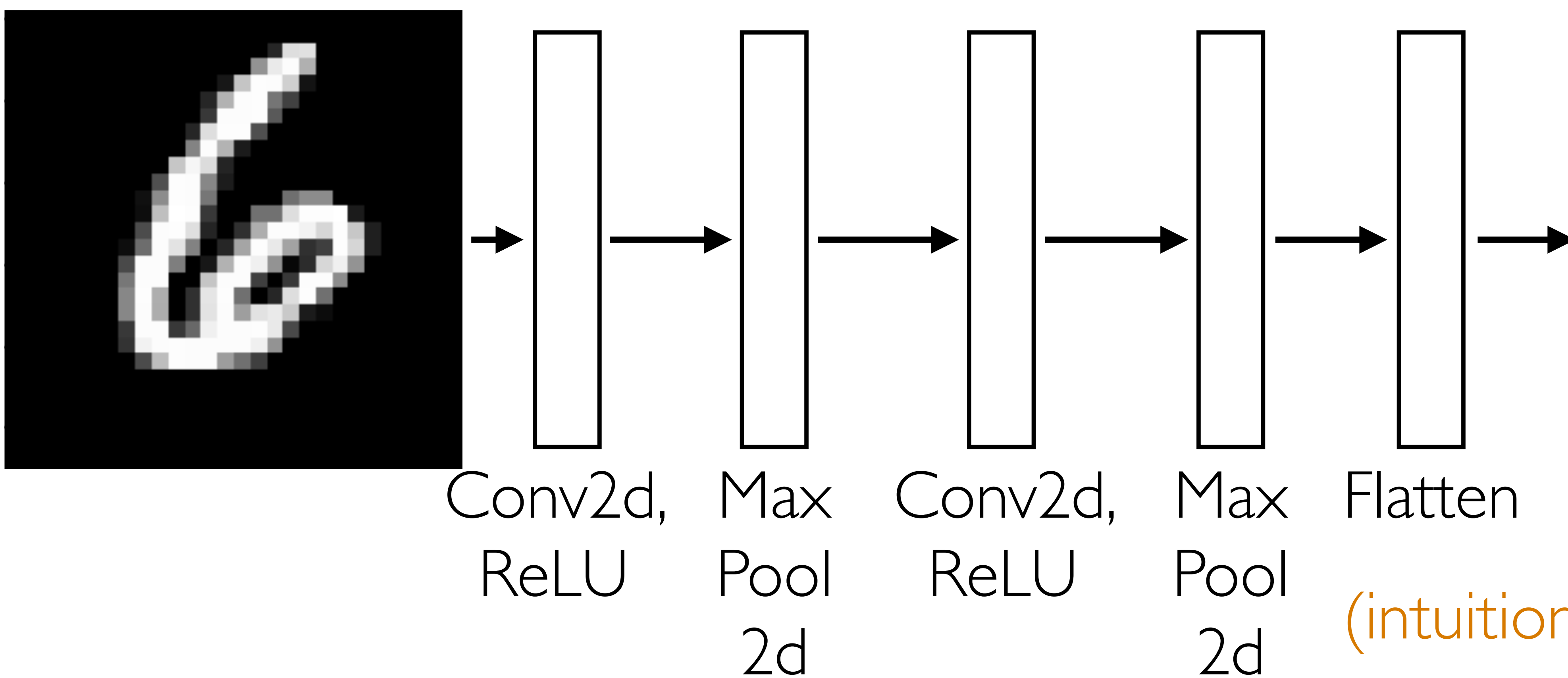
CNNs Encode Semantic Structure for Images



final output for different
input 6's is similar



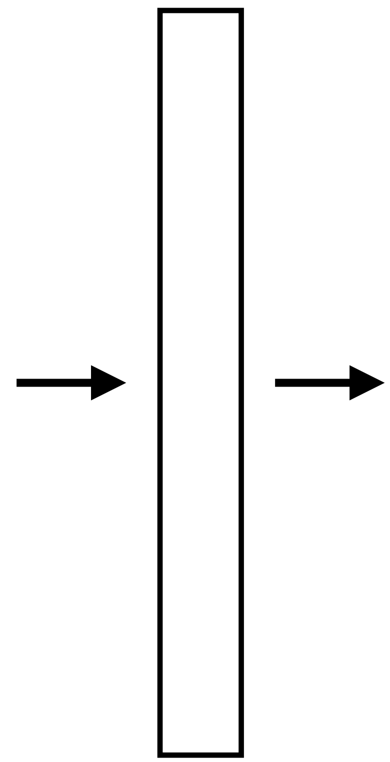
actually, intermediate representations close to the last layer are also similar!



(intuition: recall the crumpled paper analogy!)

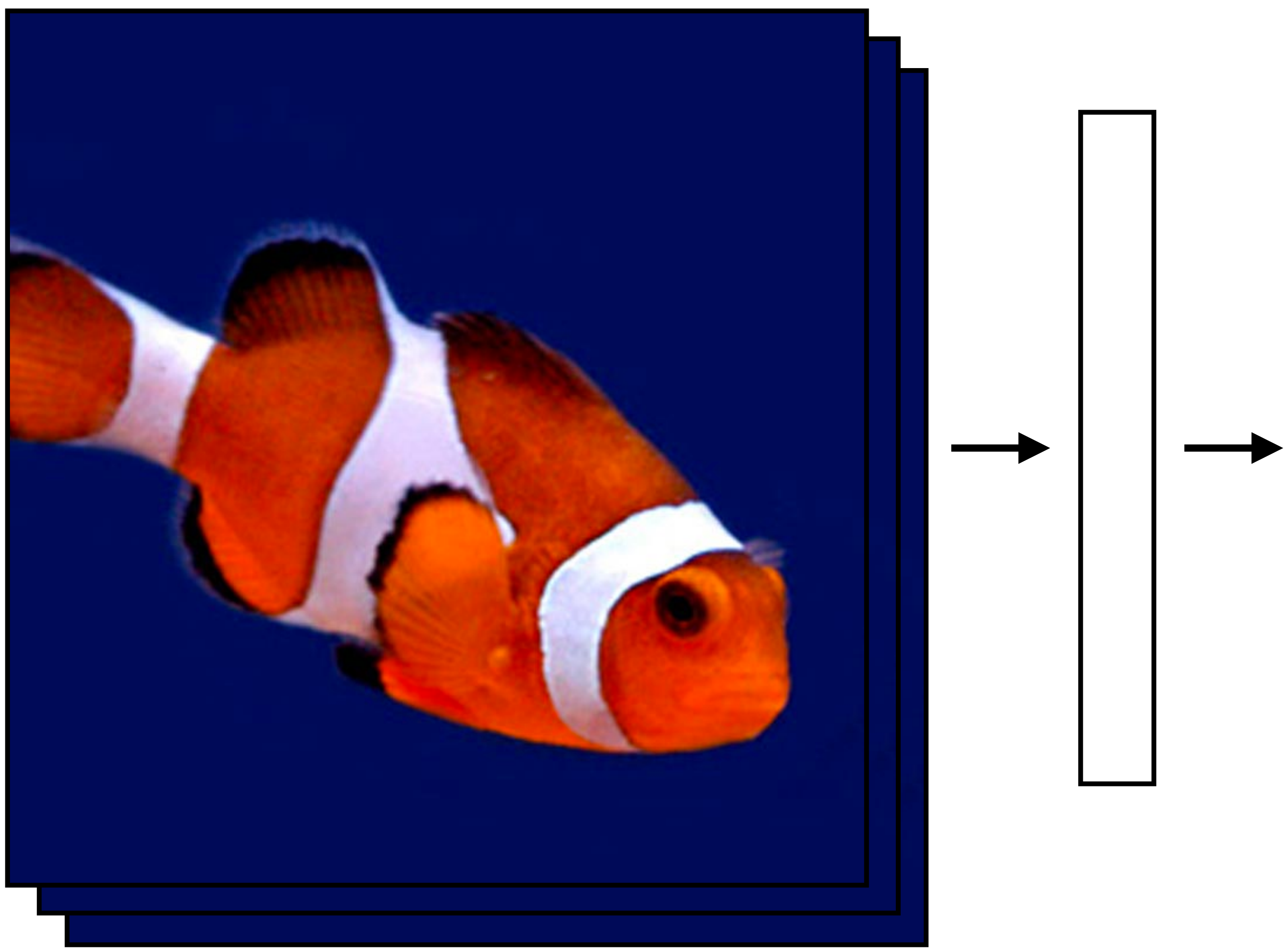
Time Series Data

What we've seen so far are "feedforward" NNs

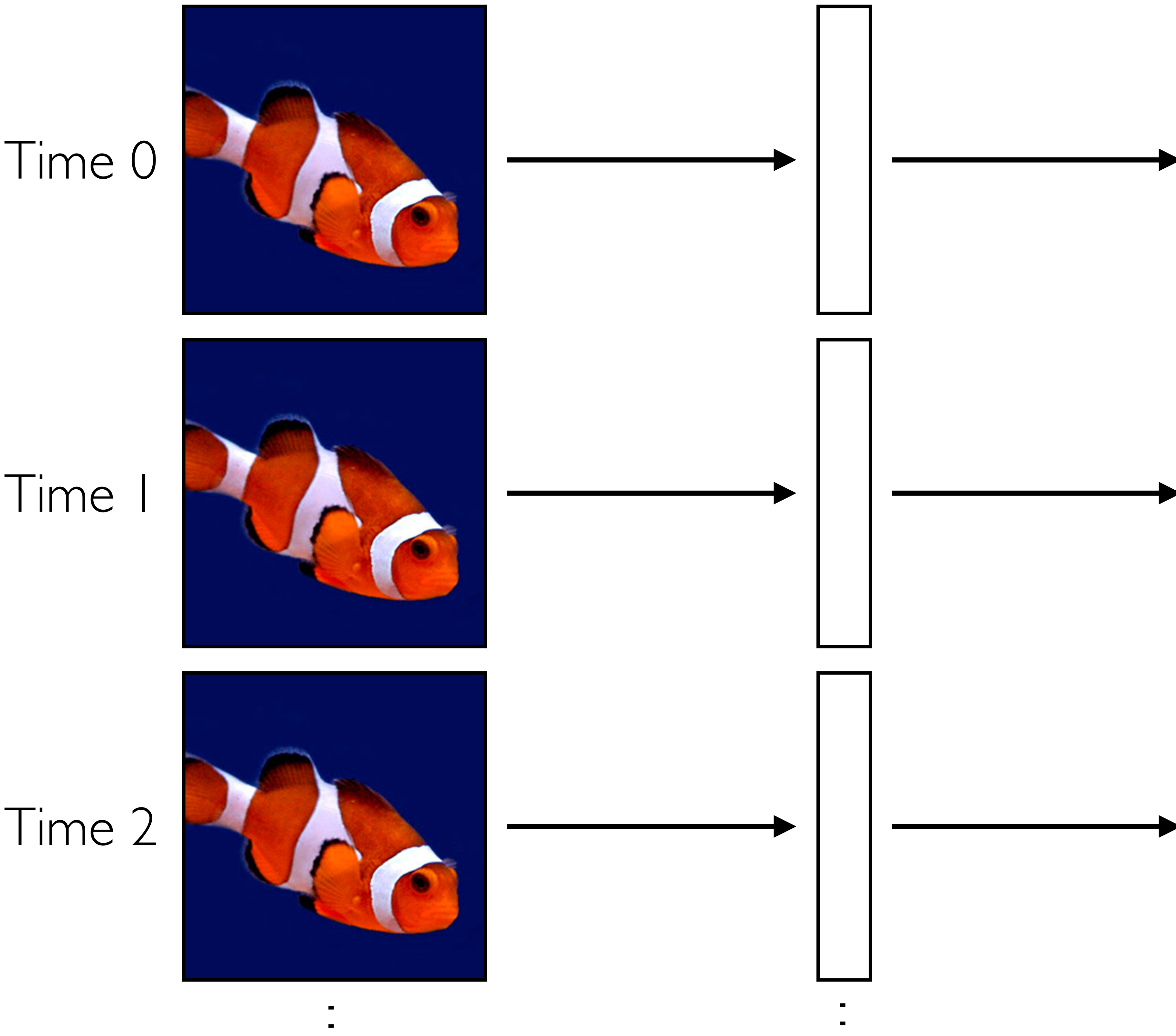


Time Series Data

What we've seen so far are "feedforward" NNs

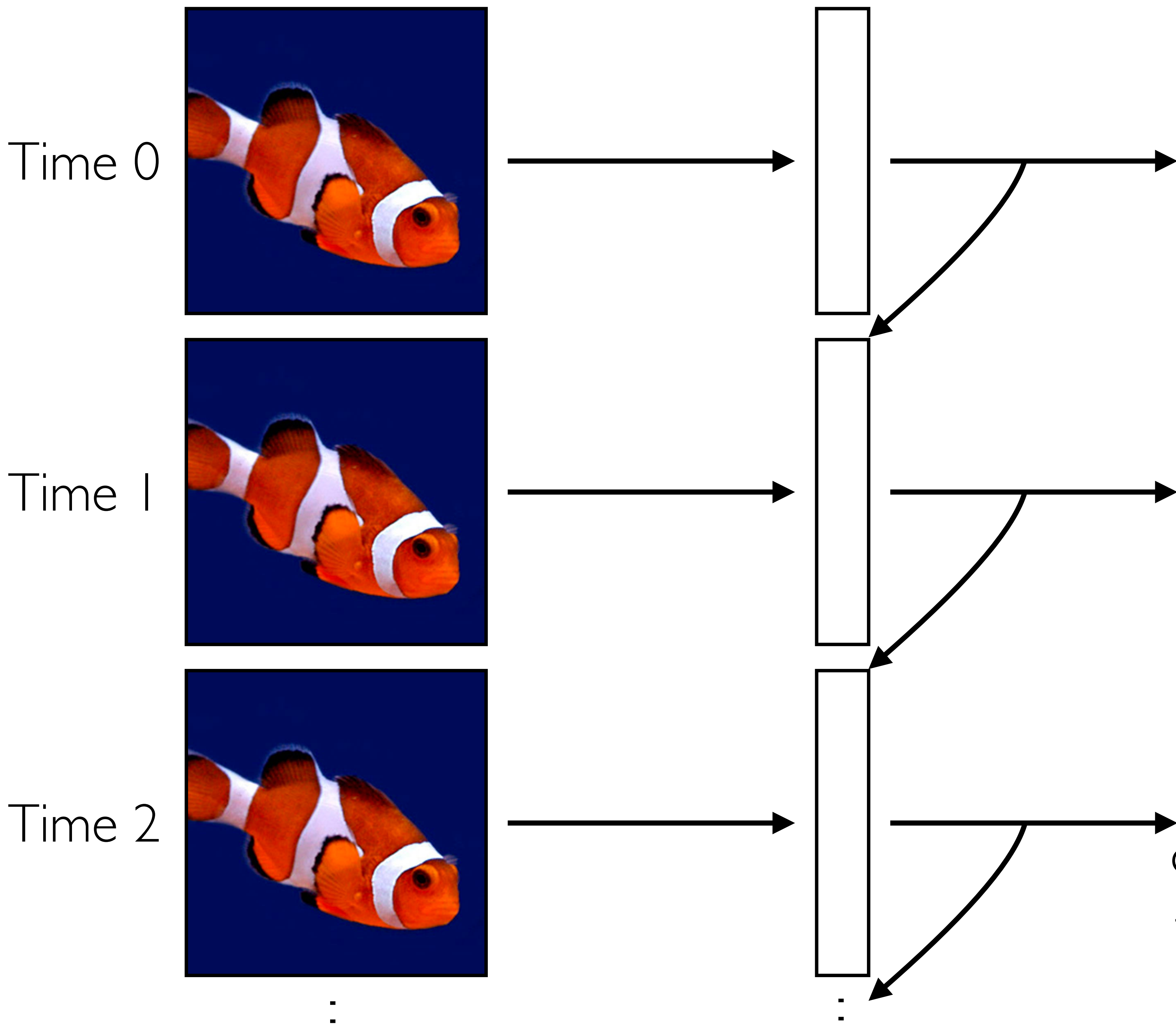


What if we had a video?



Feedforward NN's:
treat each video frame
separately

Recurrent Neural Nets

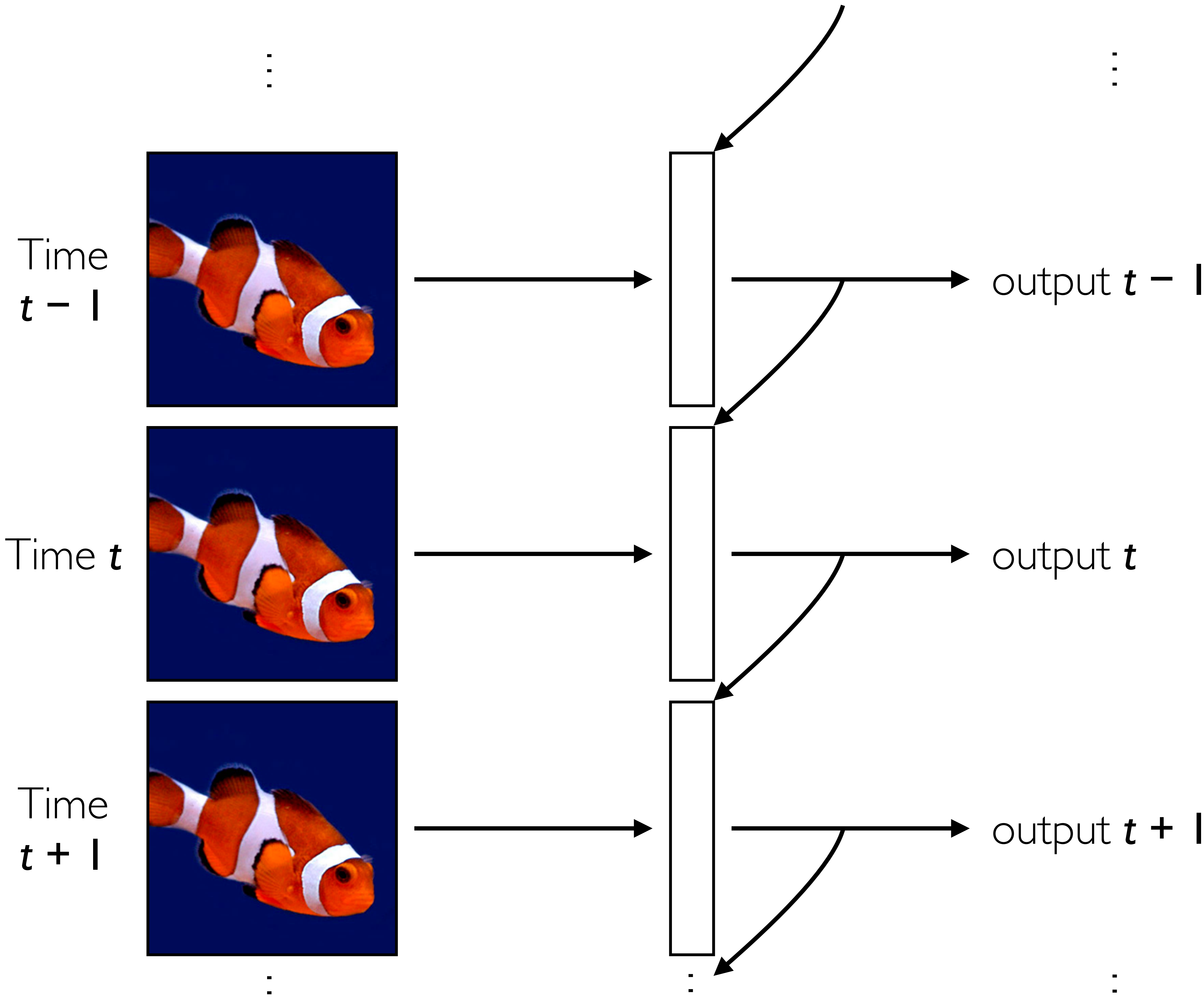


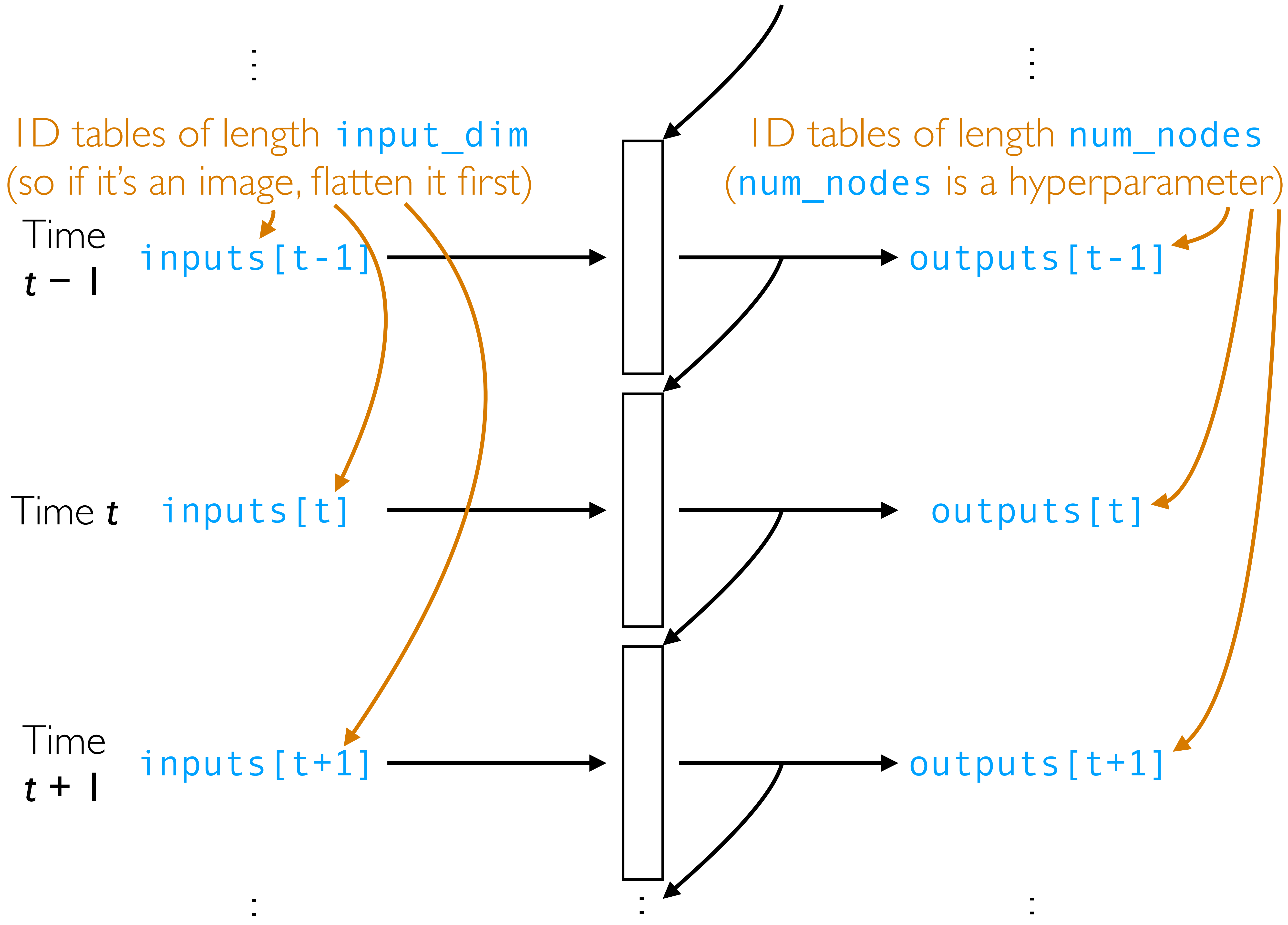
Feedforward NN's:
treat each video frame separately

RNNs:
feed output at previous time step as input to RNN layer at current time step

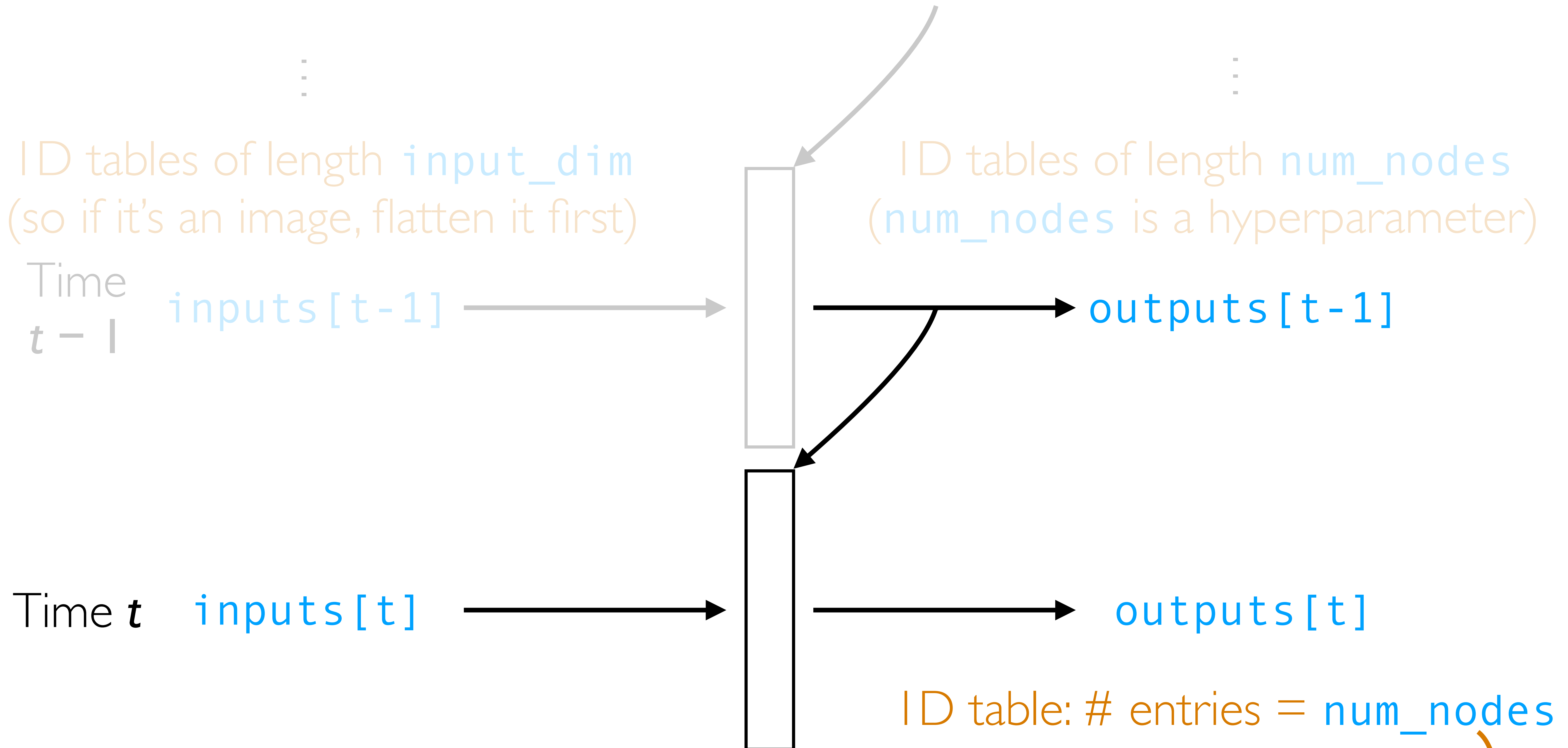
There exist many different kinds of RNNs!

Today: we mostly talk about *vanilla RNNs*


Time
$t-1$
Time t
Time
$t+1$
output $t-1$
output t
output $t+1$



1D tables of length input_dim
(so if it's an image, flatten it first)
1D tables of length num_nodes
(num_nodes is a hyperparameter)
Time t − 1
inputs[t-1]
outputs[t-1]
Time t
inputs[t]
outputs[t]
Time t + 1
inputs[t+1]
outputs[t+1]




```
linear = np.dot(inputs[t], W) + np.dot(outputs[t-1], U) + b
outputs[t] = np.maximum(0, linear)  # ReLU
```

2D table: # rows = input_dim
cols = num_nodes

2D table: # rows = num_nodes
cols = num_nodes

Vanilla RNN with ReLU Activation

list of 1D tables, each with `input_dim` entries

1D table: # entries = `num_nodes`

```
def f(inputs):

    output = np.zeros(num_nodes)
    for input in inputs:
        linear = np.dot(input, W) + np.dot(output, U) + b
        output = np.maximum(0, linear)  # ReLU

    return output
```

2D table: # rows = `input_dim`
cols = `num_nodes`

2D table: # rows = `num_nodes`
cols = `num_nodes`

Parameters: weight matrices `W` & `U`, and bias vector `b`

The vanilla RNN is basically tracking how `output` changes over time

Vanilla RNN with ReLU Activation

list of 1D tables, each with `input_dim` entries

1D table: # entries = `num_nodes`

```
def f(inputs):
    outputs = []
    output = np.zeros(num_nodes)
    for input in inputs:
        linear = np.dot(input, W) + np.dot(output, U) + b
        output = np.maximum(0, linear)  # ReLU
        outputs.append(output)
    return output
    # alternatively, could return `outputs` instead
```

2D table: # rows = `input_dim`
cols = `num_nodes`

2D table: # rows = `num_nodes`
cols = `num_nodes`

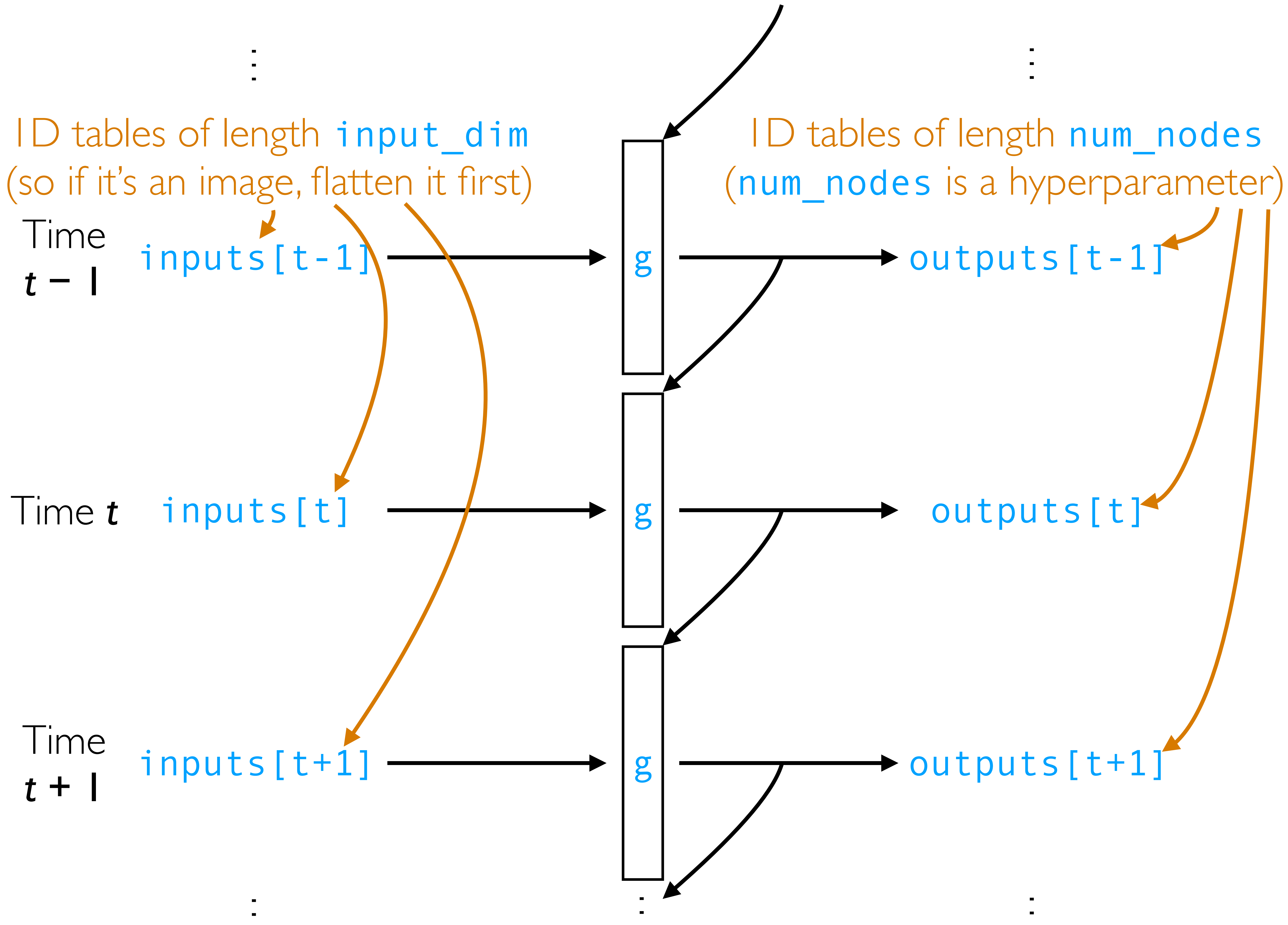
Parameters: weight matrices `W` & `U`, and bias vector `b`

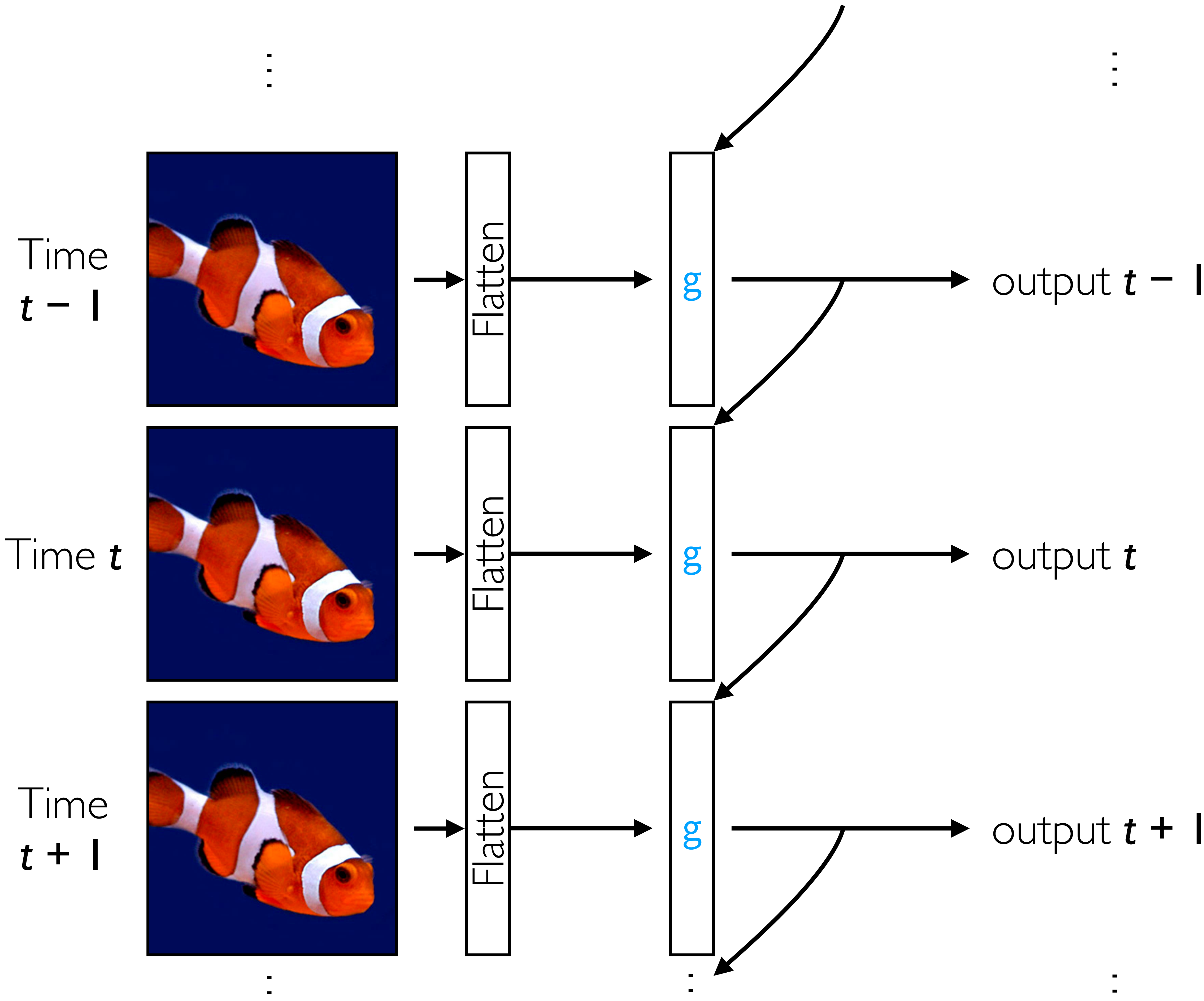
The vanilla RNN is basically tracking how `output` changes over time

Vanilla RNN with ReLU Activation

```
def g(input, prev_output):
    linear = np.dot(input, W) + np.dot(prev_output, U) + b
    output = np.maximum(0, linear)  # ReLU
    return output
```

```
def f(inputs):
    outputs = []
    output = np.zeros(num_nodes)
    for input in inputs:
        output = g(input, output)
        outputs.append(output)
    return output
    # alternatively, could return `outputs`
```

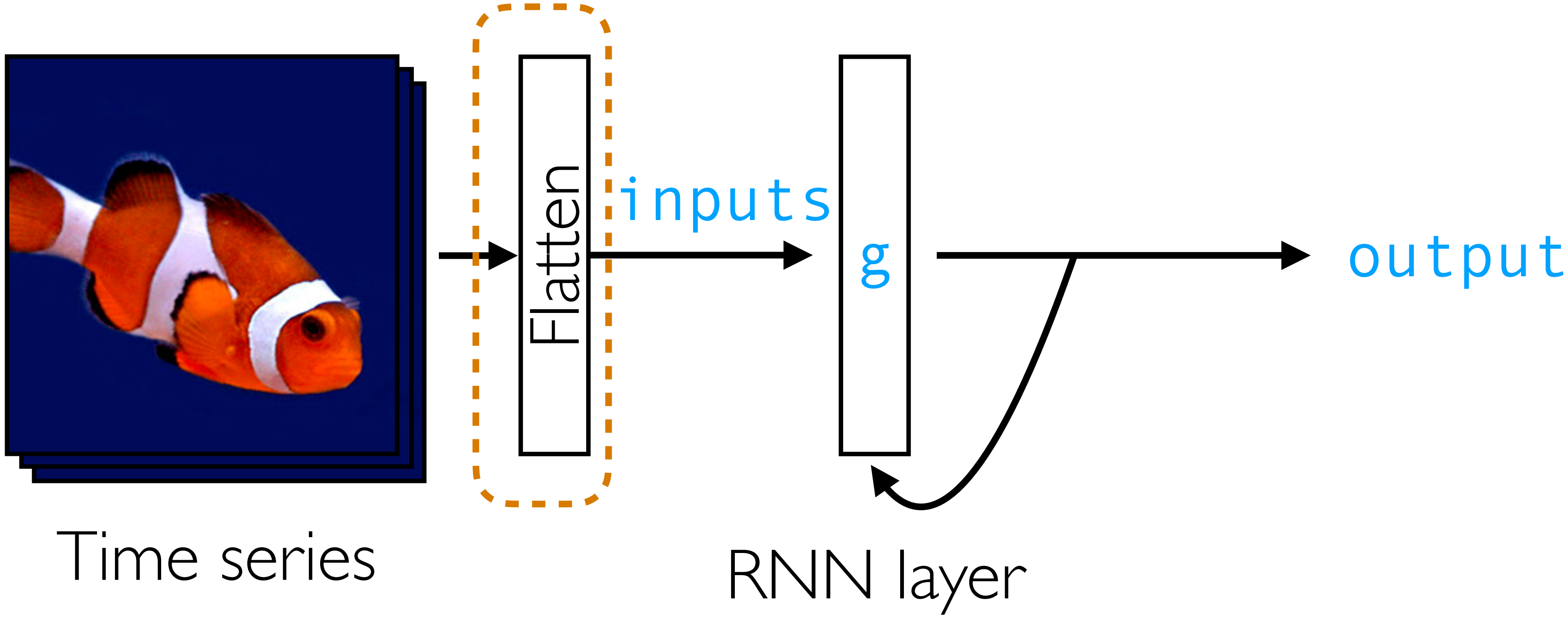

1D tables of length input_dim
(so if it's an image, flatten it first)
1D tables of length num_nodes
(num_nodes is a hyperparameter)
Time
t − 1
inputs[t-1]
g
outputs[t-1]
Time t
inputs[t]
g
outputs[t]
Time
t + 1
inputs[t+1]
g
outputs[t+1]


Time
$t - 1$
Time t
Time
$t + 1$
Flatten
Flatten
Flatten
g
g
g
output $t - 1$
output t
output $t + 1$

Recurrent Neural Nets

Key idea: combine RNN layer with other neural net layers!

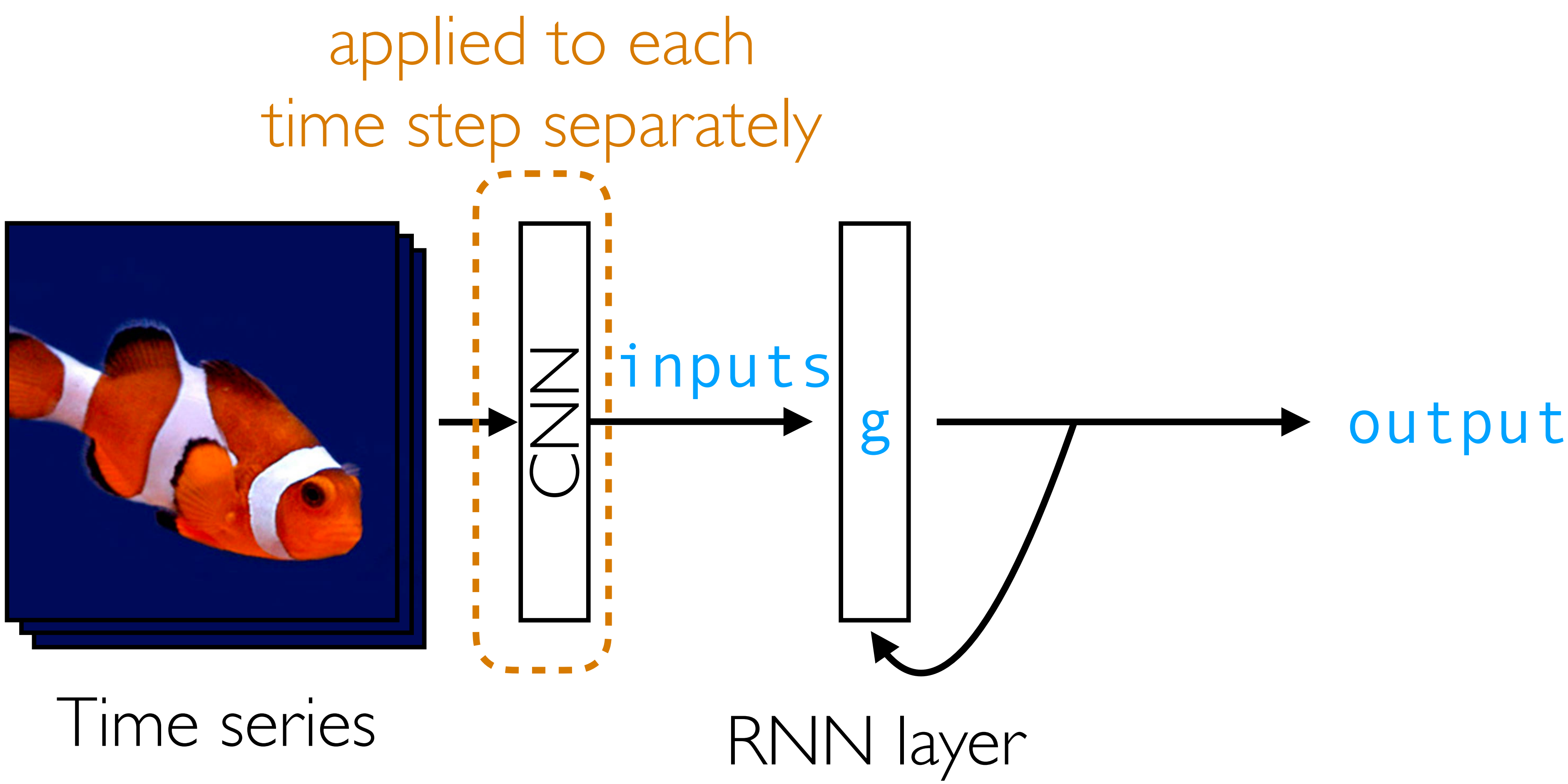
applied to each
time step separately



RNN layer itself does not actually know image structure!!!

Recurrent Neural Nets

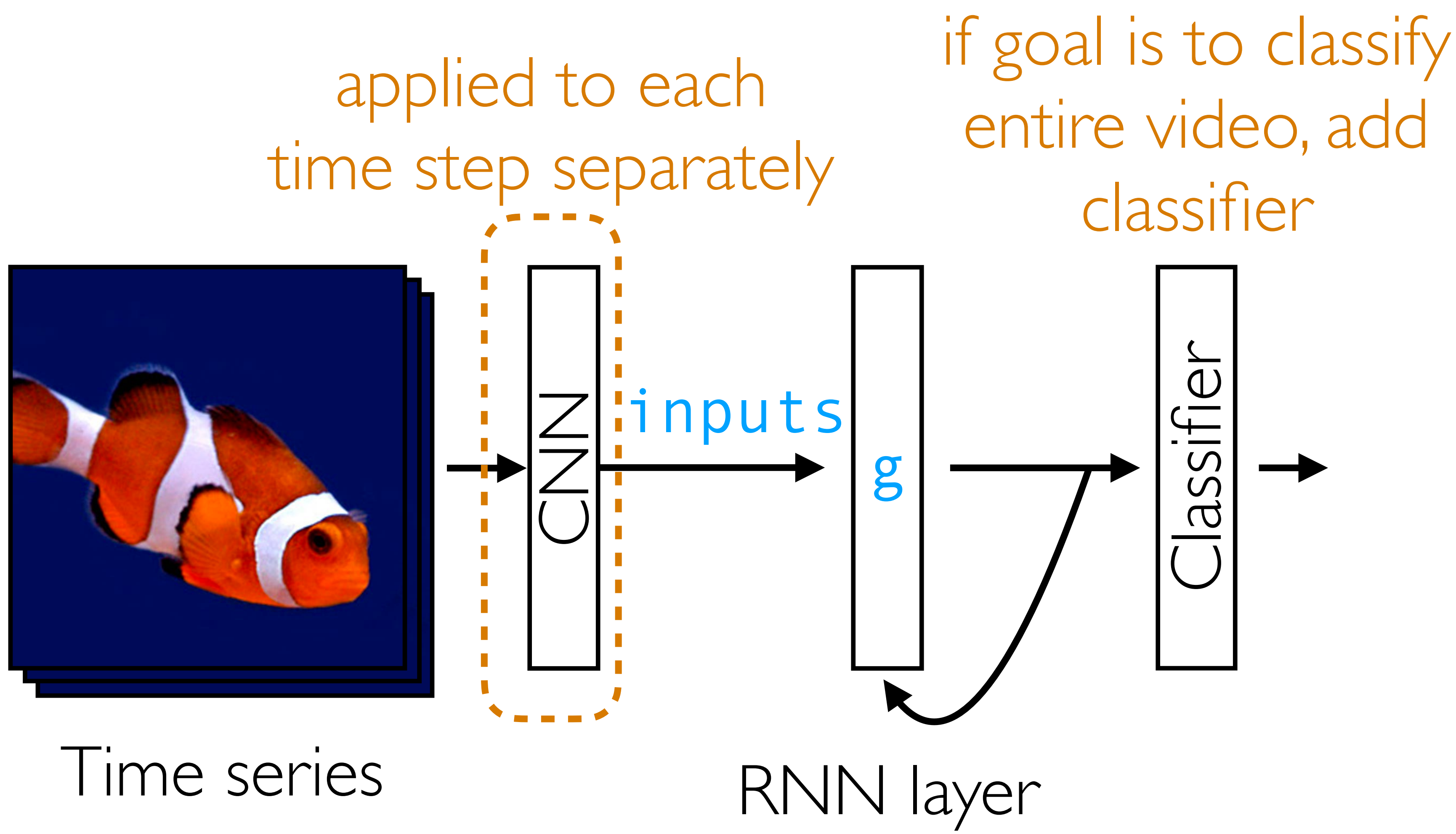
Key idea: combine RNN layer with other neural net layers!



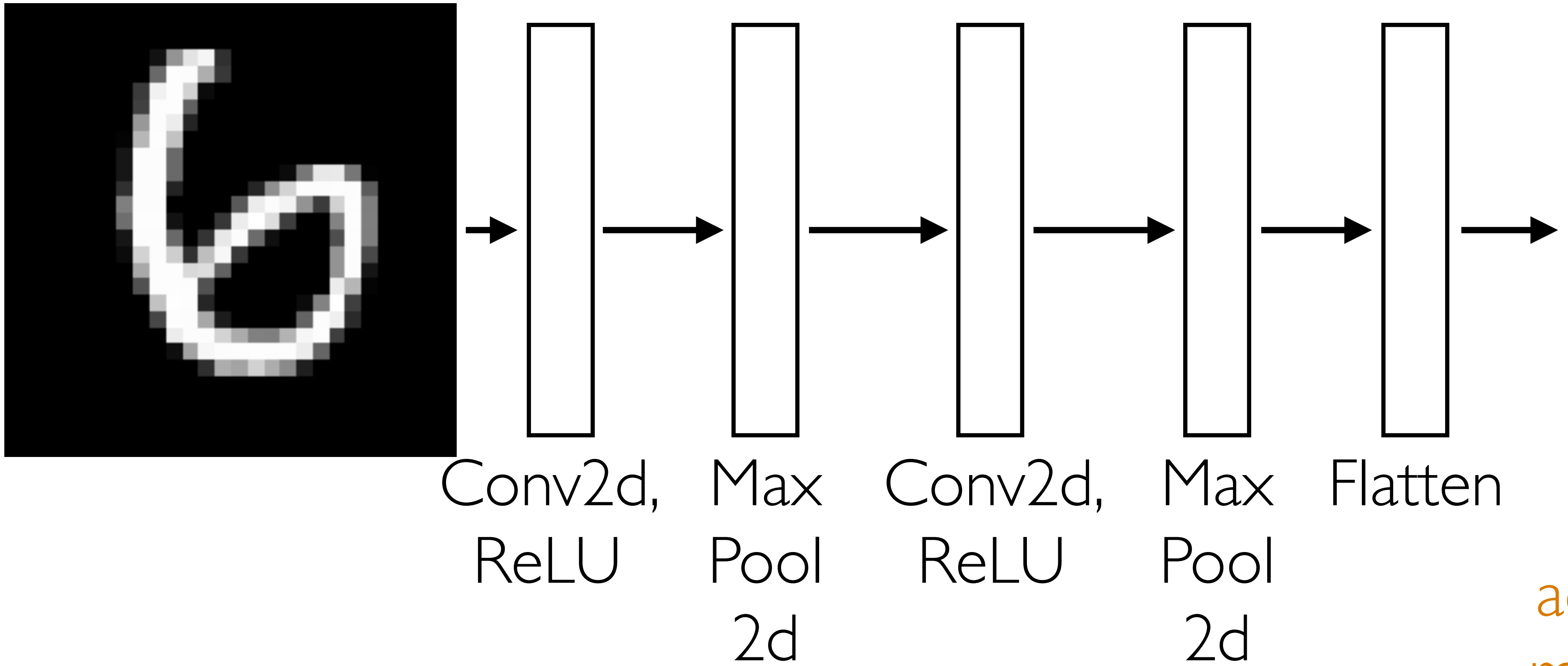
RNN layer itself does not actually know image structure!!!

Recurrent Neural Nets

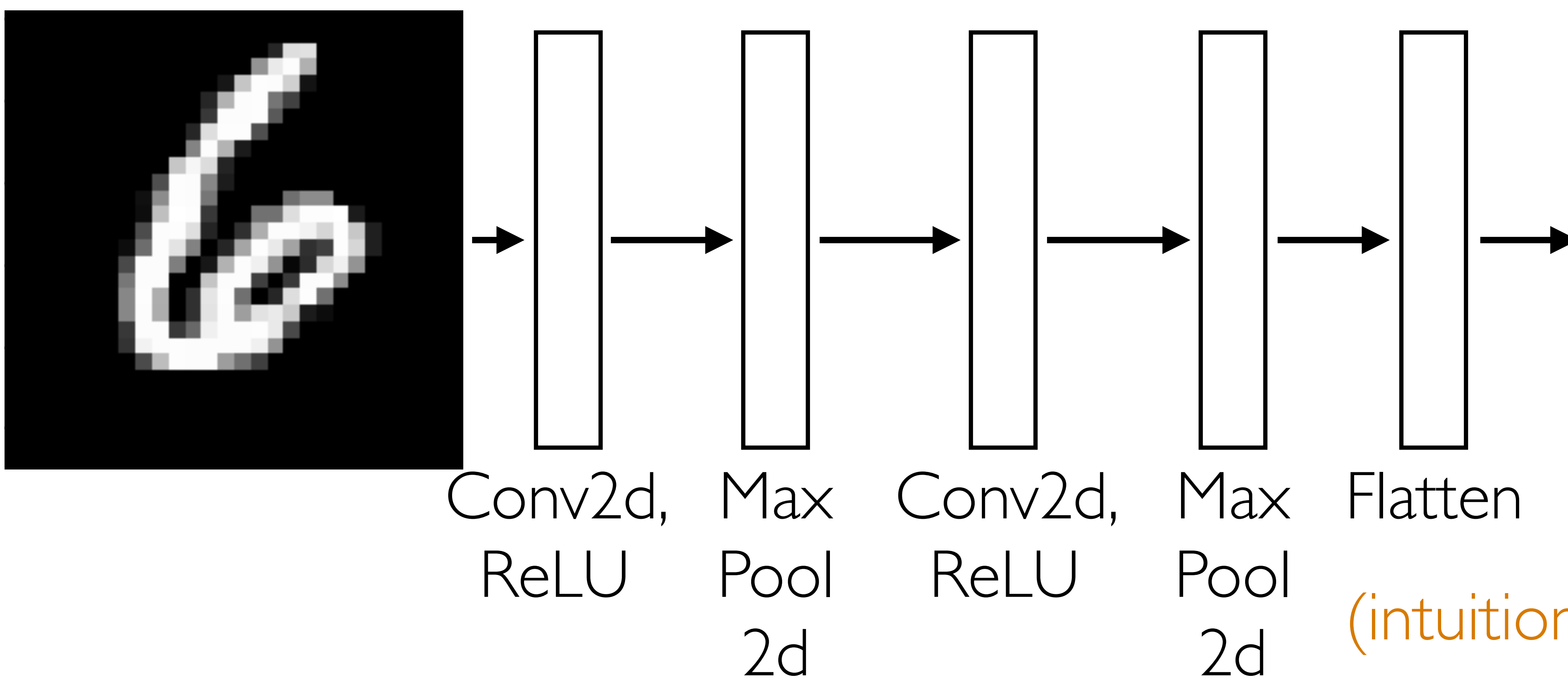
Key idea: combine RNN layer with other neural net layers!



RNN layer itself does not actually know image structure!!!



actually, intermediate representations close to the last layer are also similar!



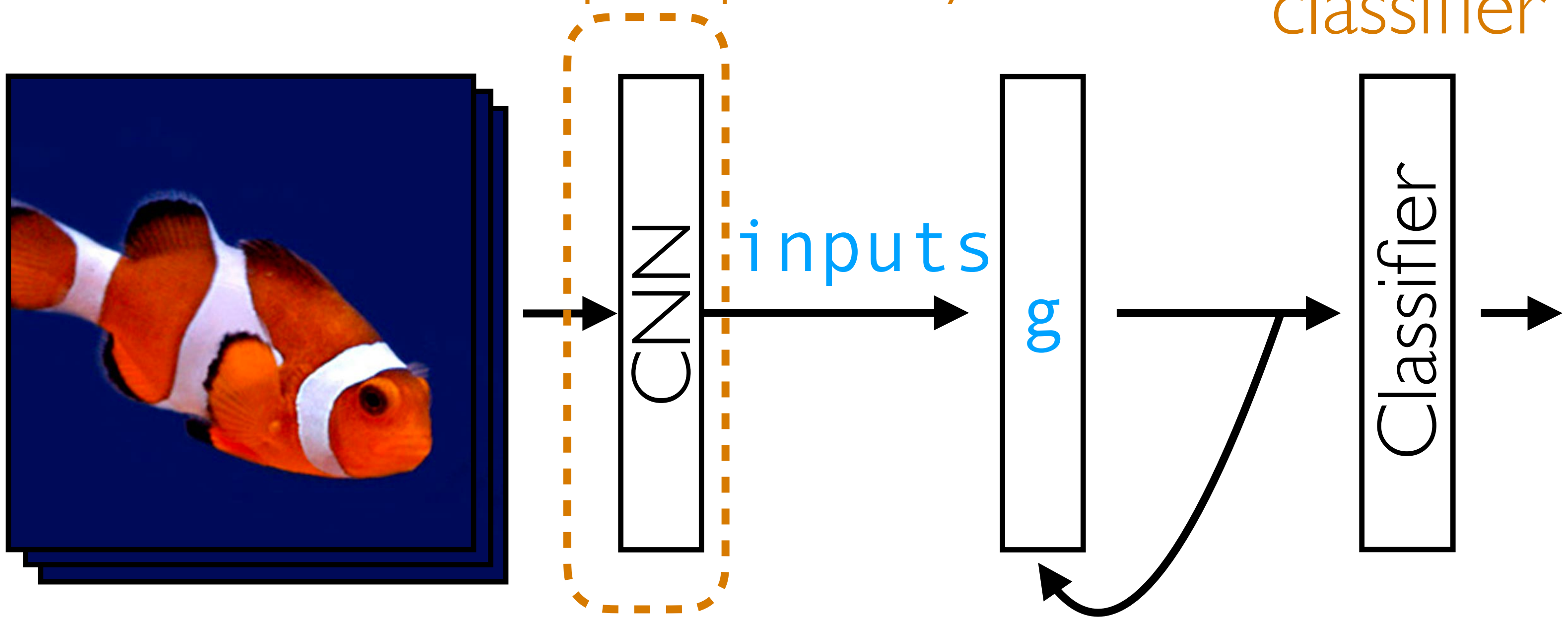
(intuition: recall the crumpled paper analogy!)

Recurrent Neural Nets

Key idea: combine RNN layer with other neural net layers!

applied to each
time step separately

if goal is to classify
entire video, add
classifier

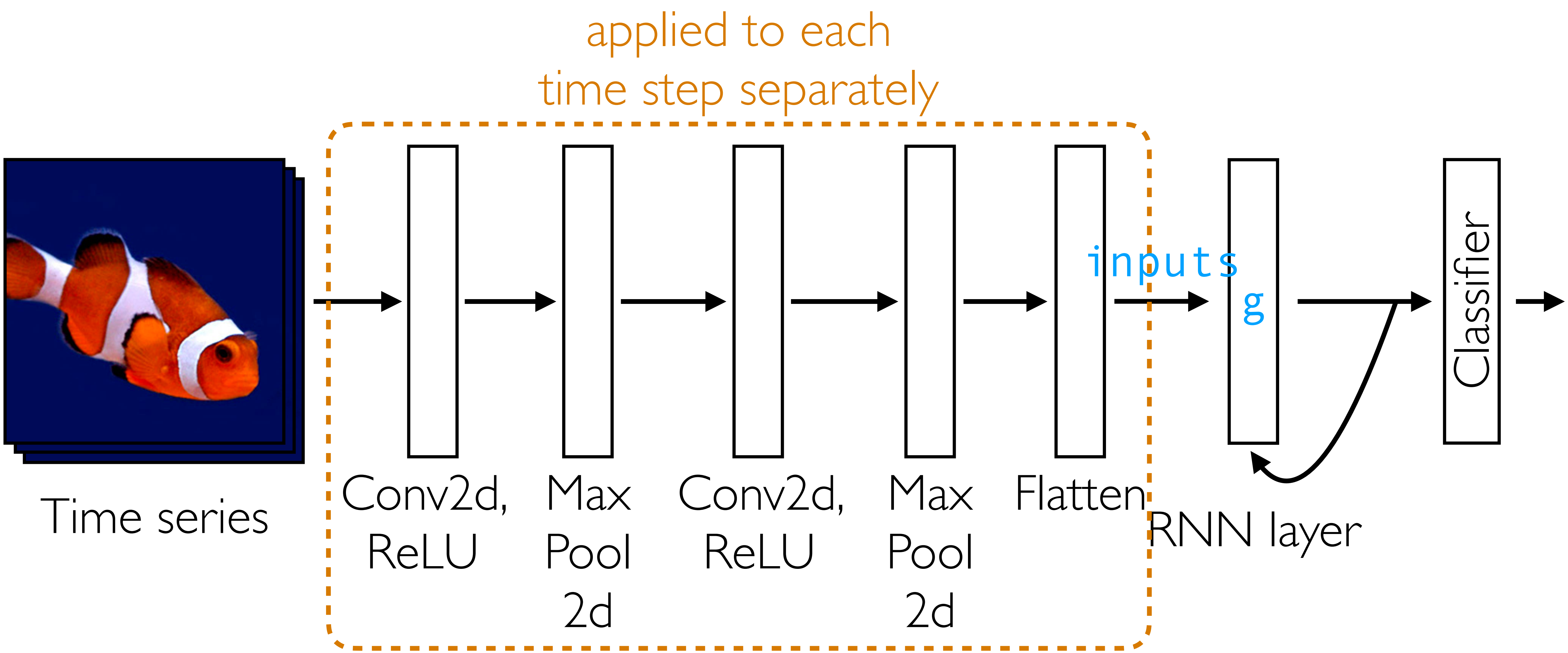


Time series

RNN layer

RNN layer itself does not actually know image structure!!!

Recurrent Neural Nets

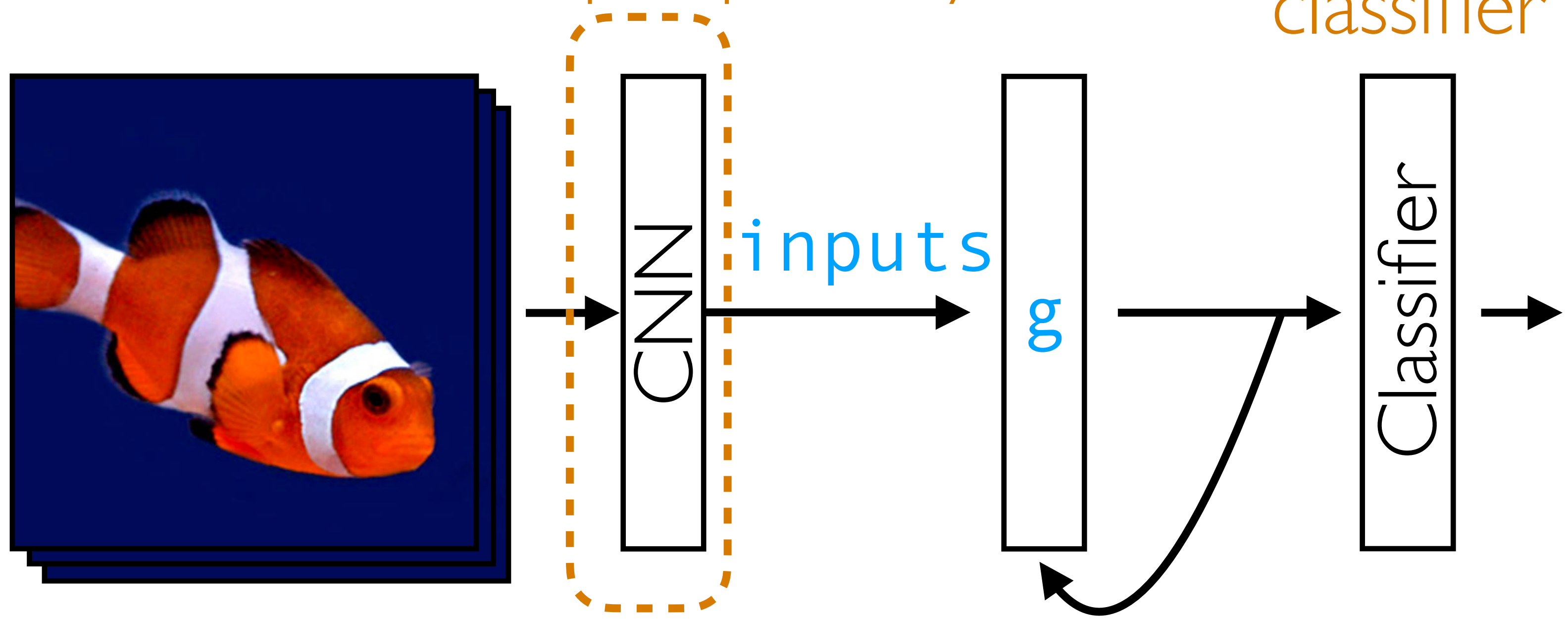


Recurrent Neural Nets

Key idea: combine RNN layer with other neural net layers!

applied to each time step separately

if goal is to classify entire video, add classifier



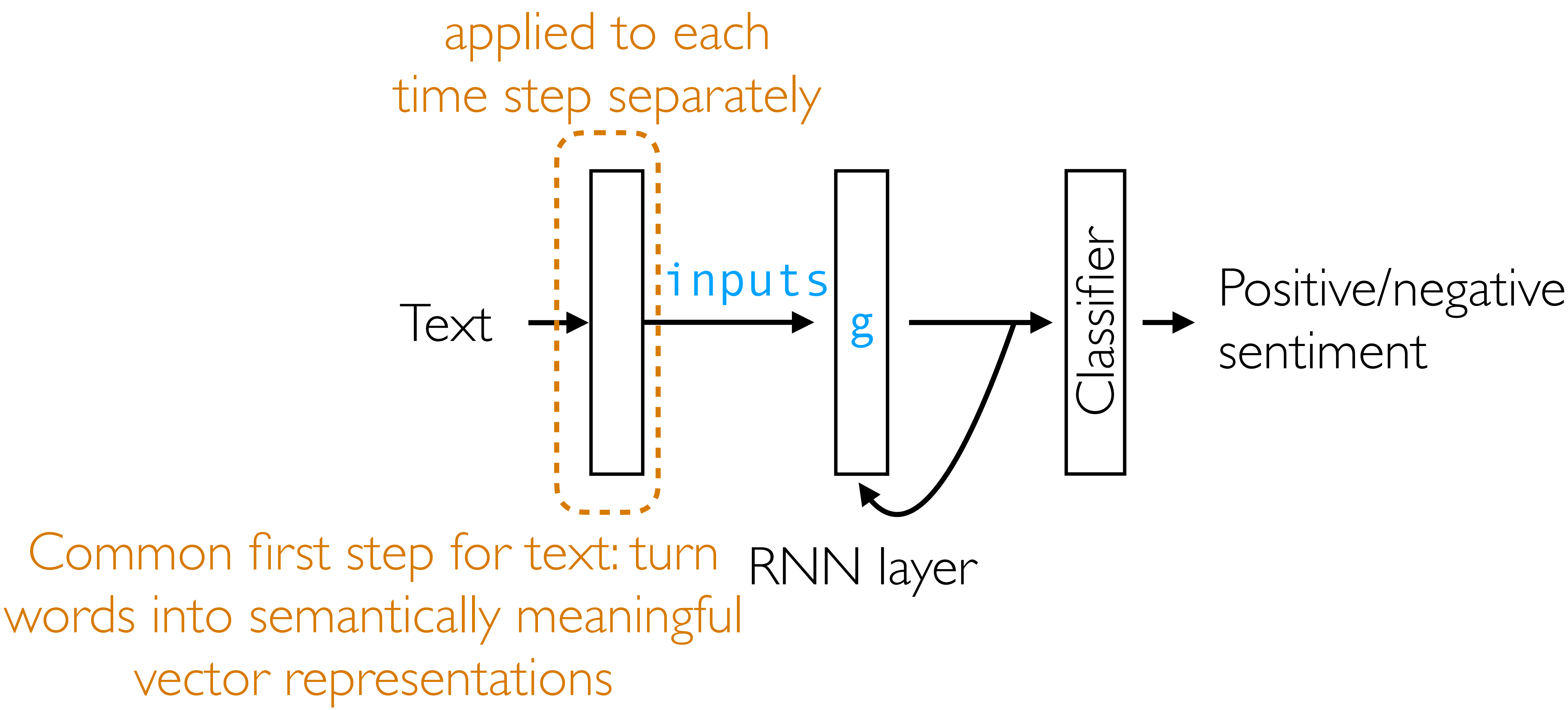
Time series

RNN layer

RNN layer itself does not actually know image structure!!!

Recurrent Neural Nets

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)



(Flashback) Do Data Actually Live on Manifolds?

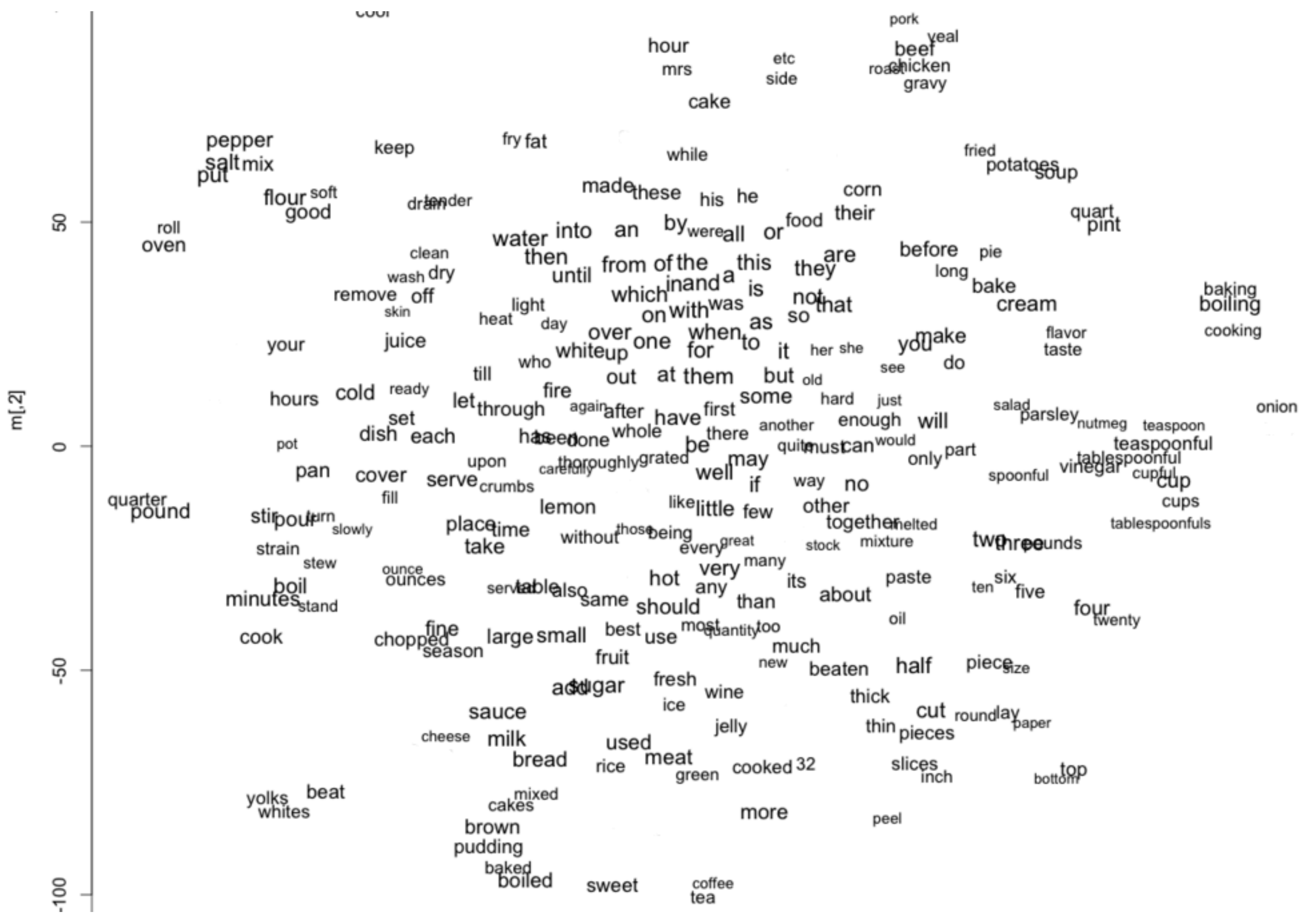
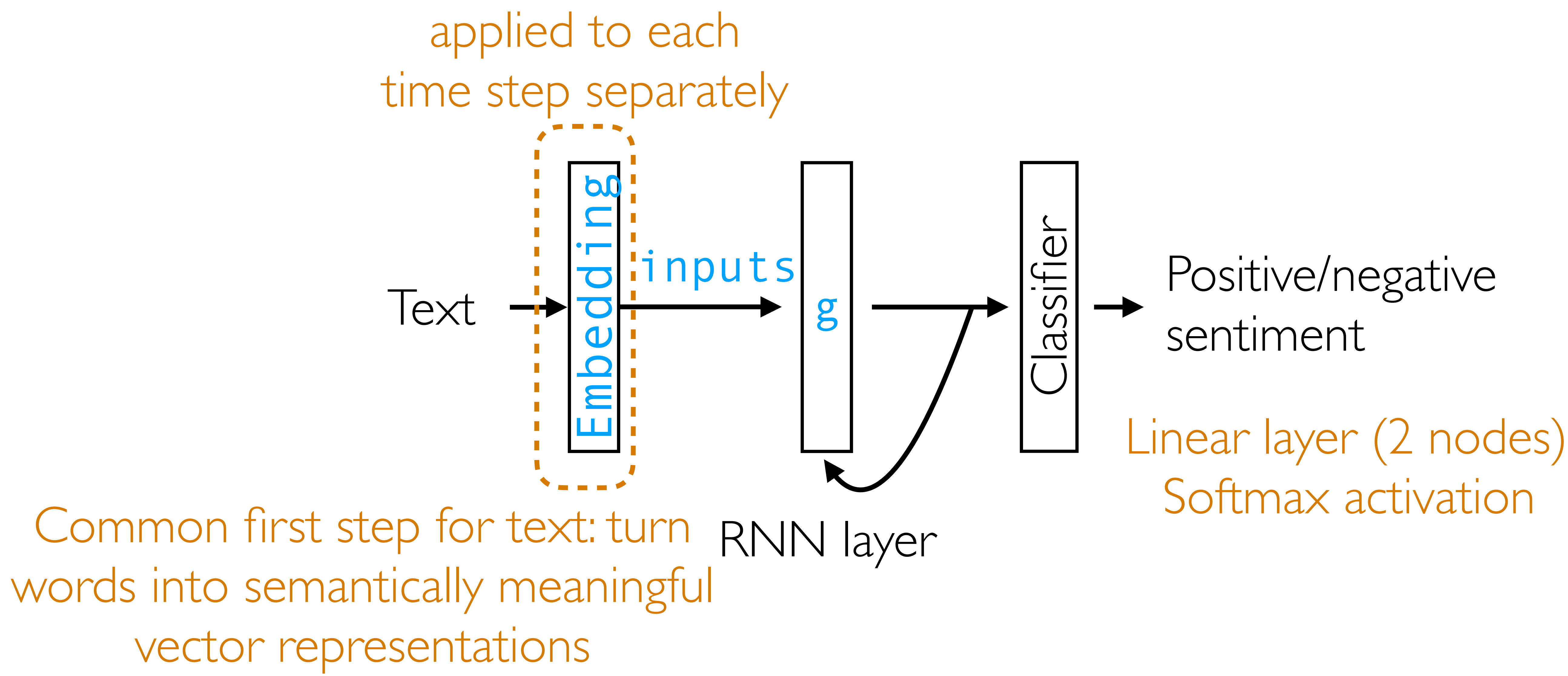


Image source: http://www.adityathakker.com/wp-content/uploads/2017/06/word-embeddings-994x675.png

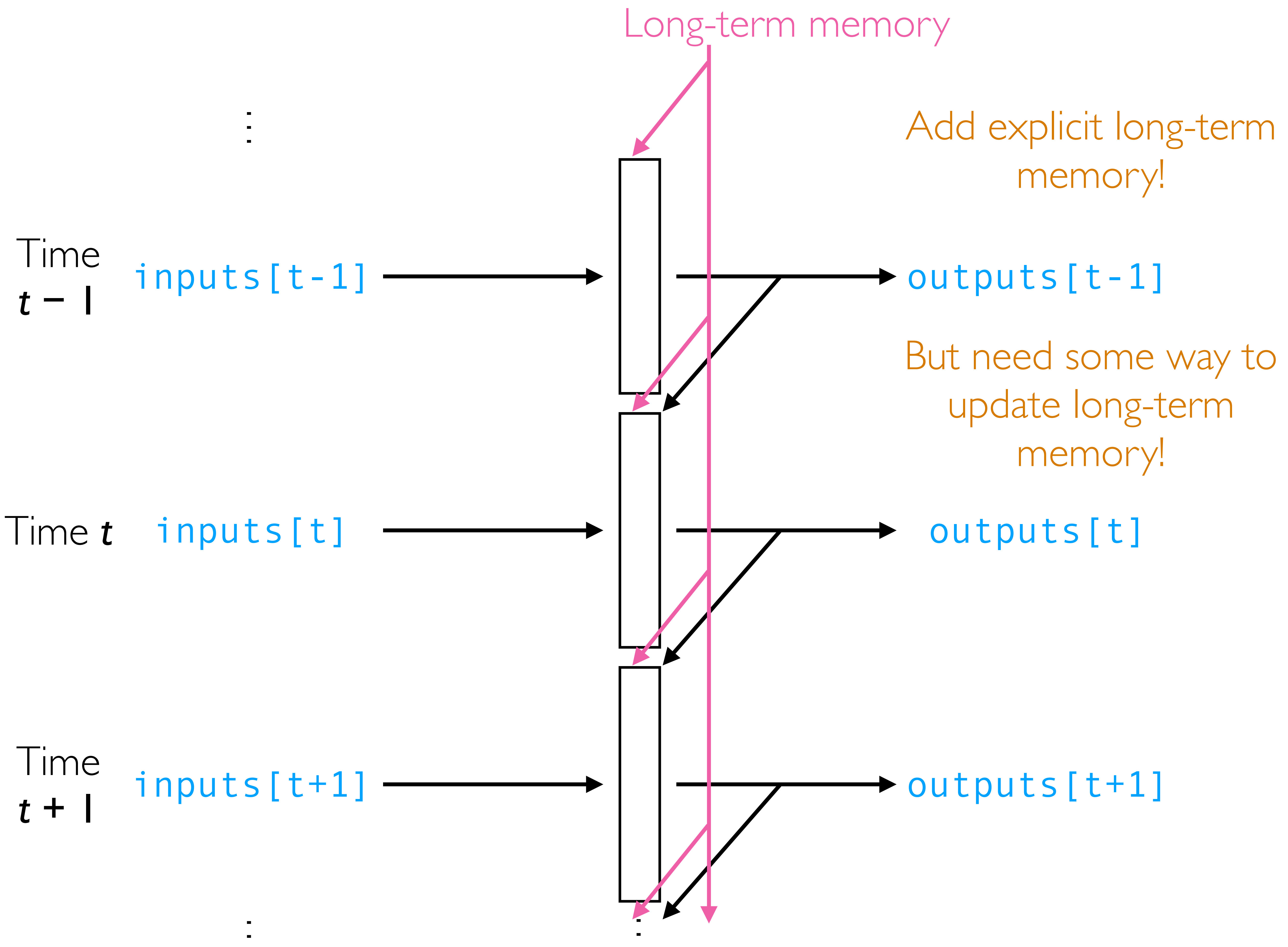
Recurrent Neural Nets

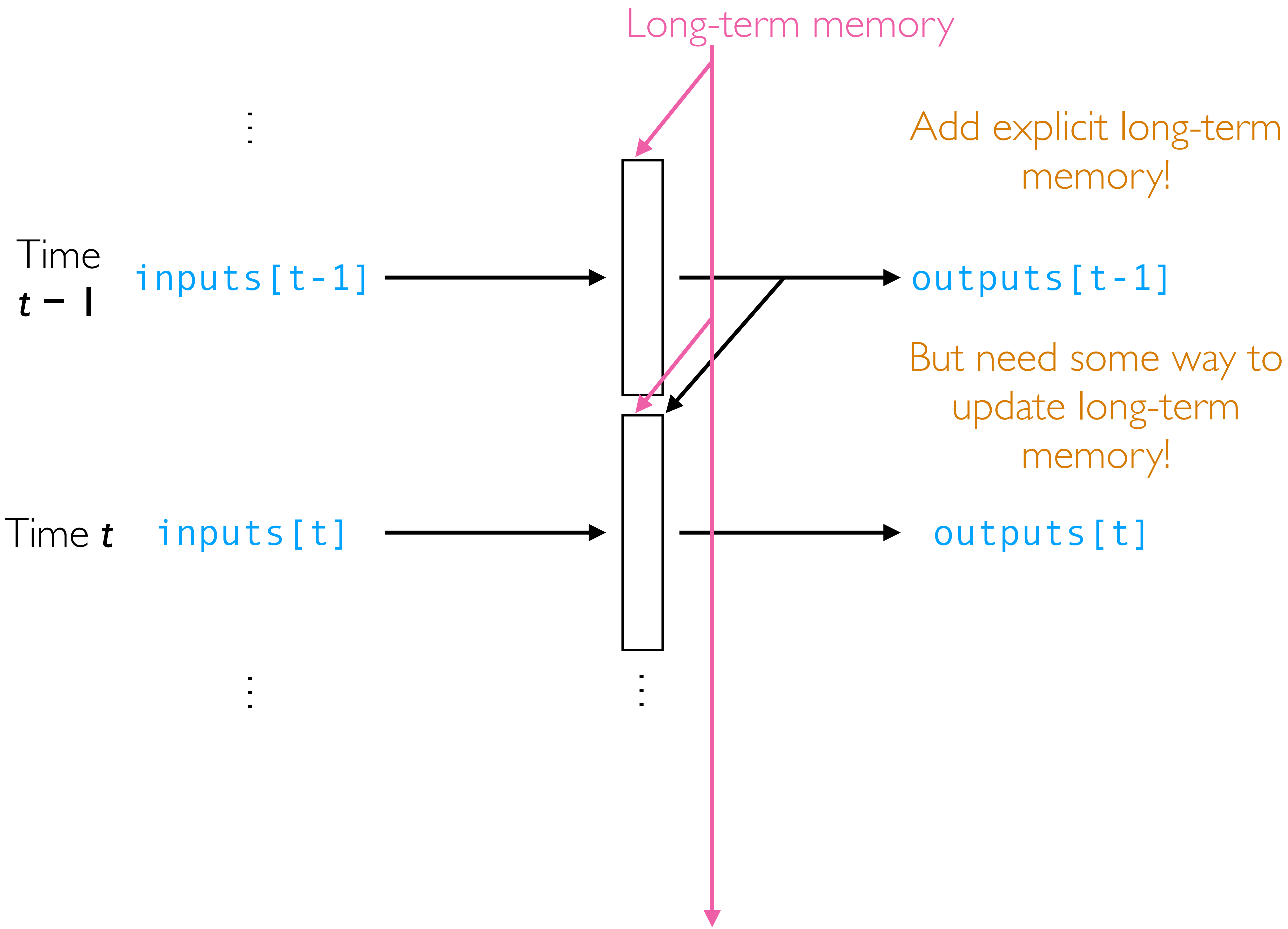
Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

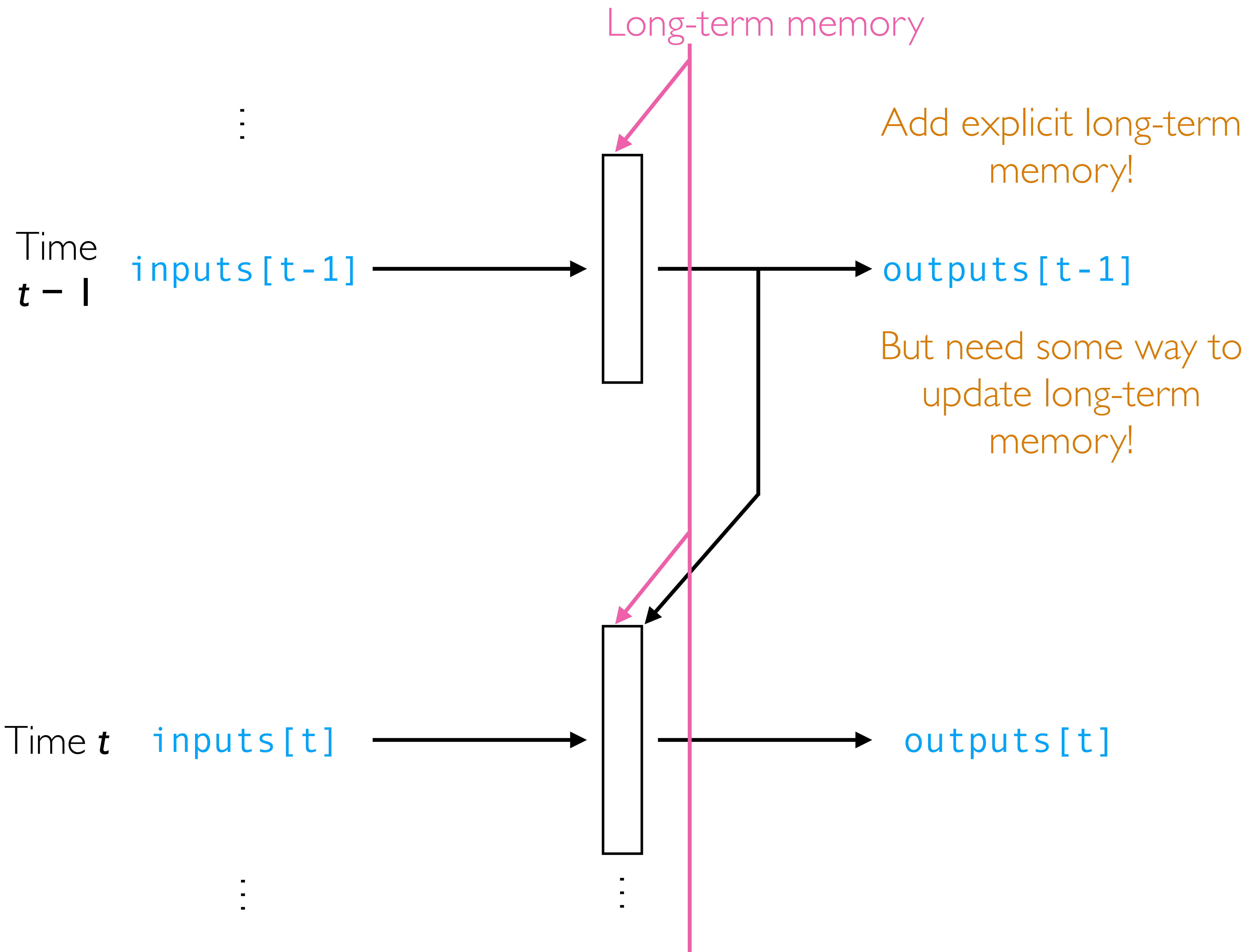


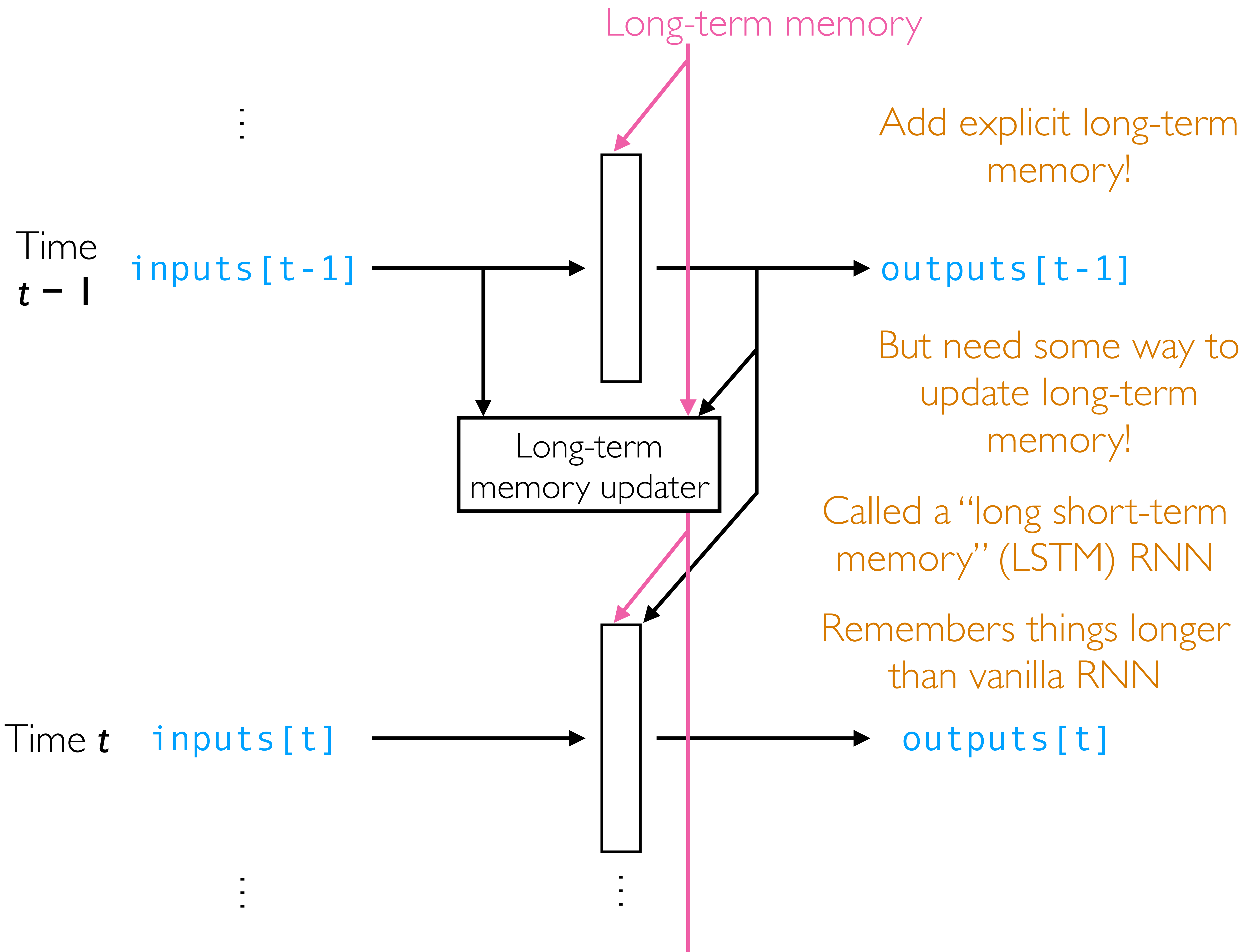
In PyTorch, use the `Embedding` layer and load in pre-trained word embeddings

Vanilla RNNs tend to have gold fish memory and forget things very quickly


Long-term memory
Add explicit long-term memory!
But need some way to update long-term memory!
Time $t-1$
inputs[t-1]
outputs[t-1]
Time t
inputs[t]
outputs[t]
Time $t+1$
inputs[t+1]
outputs[t+1]


Long-term memory
Add explicit long-term memory!
Time $t-1$
inputs[t-1]
outputs[t-1]
But need some way to update long-term memory!
Time t
inputs[t]
outputs[t]


Long-term memory
Add explicit long-term memory!
Time $t - 1$
inputs[t-1]
outputs[t-1]
But need some way to update long-term memory!
Time t
inputs[t]
outputs[t]


Long-term memory
Add explicit long-term memory!
Time $t-1$
inputs[t-1]
outputs[t-1]
But need some way to update long-term memory!
Long-term memory updater
Called a "long short-term memory" (LSTM) RNN
Remembers things longer than vanilla RNN
Time t
inputs[t]
outputs[t]

Recap/Important Reminder

Neural nets are *not* doing magic; **incorporating structure is very important to state-of-the-art deep learning systems**

- An RNN tracks how what's stored in memory changes over time — **an RNN's job is made easier if the memory is a semantically meaningful representation**
  - Word embeddings encode semantic structure—words with similar meaning are mapped to nearby Euclidean points
  - CNNs encode semantic structure for images—images that are "similar" are mapped to nearby Euclidean points
- Vanilla RNNs do not explicitly track long-term memory and tends to forget things
  - LSTMs explicitly incorporate long-term memory and learn when to update long-term memory

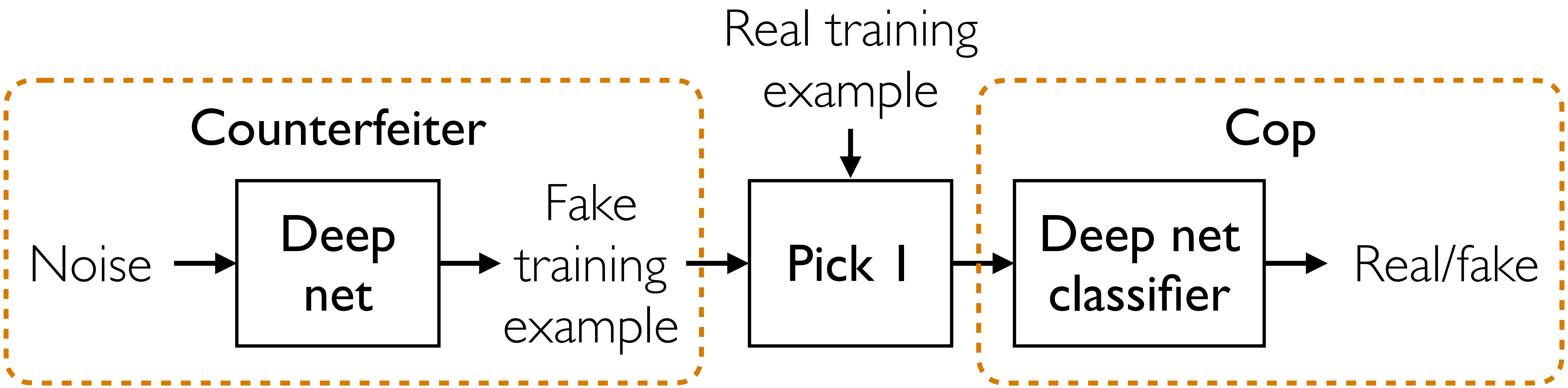
We barely saw deep learning in this class!
(At this point, there are multiple semester-long courses on specific deep learning concepts!)

Let me go over one key topic that I think is relevant to policy…

Generate Fake Data that Look Real

Unsupervised approach: generate data that look like training data

Example: Generative Adversarial Network (GAN)



Counterfeiter tries to get better at tricking the cop

Cop tries to get better at telling which examples are real vs fake

Terminology: counterfeiter is the **generator**, cop is the **discriminator**

Other approaches: variational autoencoders, pixelRNNs/pixelCNNs

Generate Fake Data that Look Real

Fake celebrities generated by NVIDIA using GANs
(Karras et al Oct 27, 2017)

Google DeepMind's WaveNet makes fake audio that sounds like whoever you want using pixelRNNs (Oord et al 2016)

Generate Fake Data that Look Real

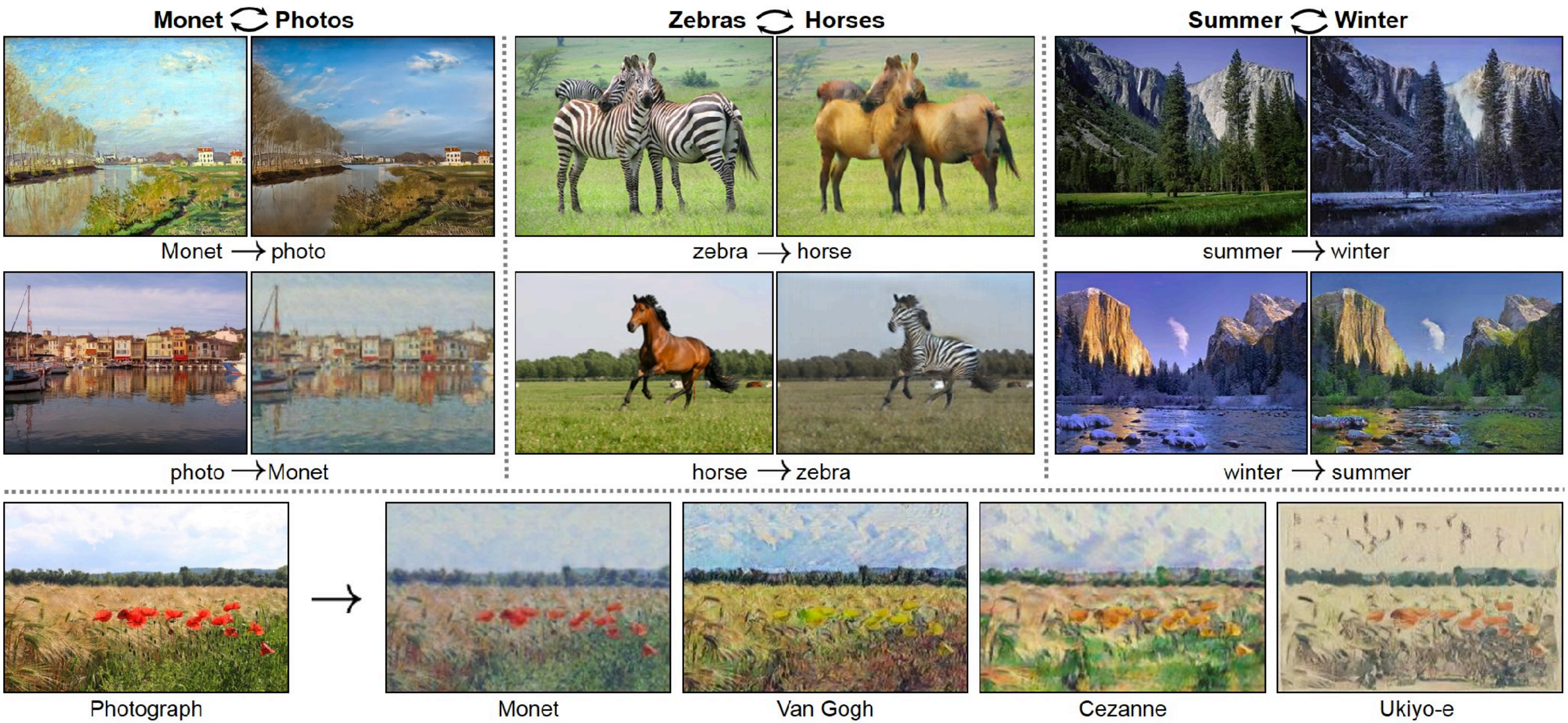


Image-to-image translation results from UC Berkeley using GANs (Isola et al 2017, Zhu et al 2017)

The technology or generating fake images/video/ audio that look real *is getting a lot better over time & I think will lead to serious societal problems…*

What if we simply can no longer tell what is fake vs real news anymore?

What if governments take advantage of better and better AI technologies to generate fake news to make their citizens think a certain way?

The Future of Deep Learning

- Deep learning learns computer programs
  - We have only seen simple examples of these computer programs in this class, but the programs that can be learned are becoming increasingly sophisticated
- All the best ideas that lead to amazing prediction results incorporate problem-specific structure
- How do we automatically discover important problem structure?
- How do we do lifelong learning?
- How do we reason about causality?

Some Parting Thoughts

- Remember to **visualize steps of your data analysis pipeline**
  - Helpful in debugging & interpreting intermediate/final outputs
- Very often there are *tons* of models/design choices to try
  - Try to come up with **quantitative metrics** that make sense for your problem, and use these metrics to **evaluate models (think about how we chose hyperparameters!)**
  - But don't blindly rely on metrics without **interpreting results in the context of your original problem!**
- Often times you won't have labels! If you really want labels:
  - Manually obtain labels (either you do it or crowdsource)
- There is a *lot* we did not cover — **keep learning!**

Want to Learn More?

Some courses at CMU:

- Natural language processing (analyze text): 11-611
- Computer vision (analyze images): 16-720
- Deep learning: 11-785, 10-707
- Deep reinforcement learning: 10-703
- Math for machine learning: 10-606, 10-607
- Intro to machine learning at different levels of math: 10-601, 10-701, 10-715
- Machine learning with large datasets: 10-605

This list isn't exhaustive and there are courses not just at CMU (e.g., other schools, Coursera, edX, Udacity)!