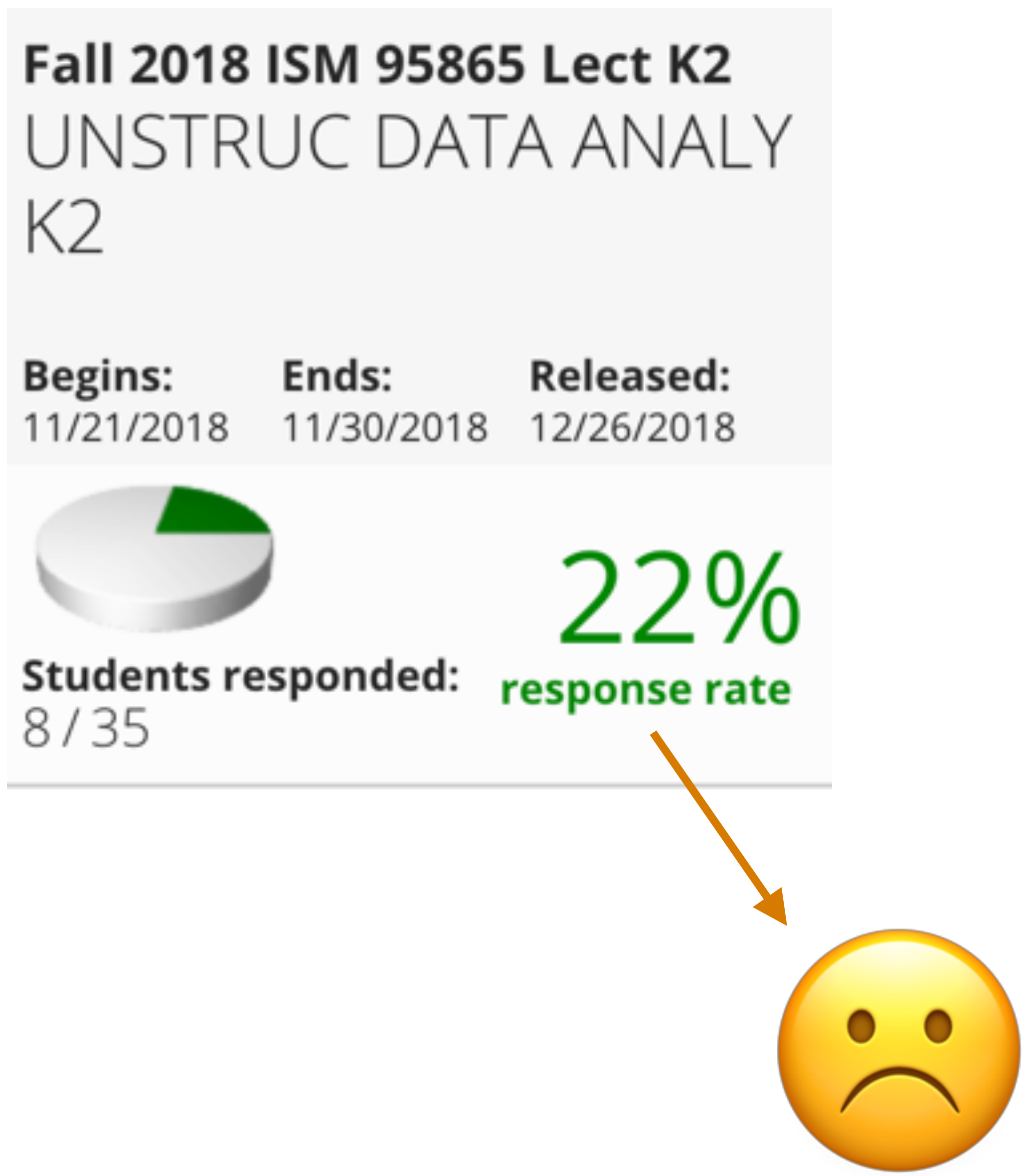
Carnegie Mellon University
Heinz College

95-865 Australia Lecture 6: CNNs, RNNs, Deep Learning and Course Wrap-up

George Chen

Faculty Course Evaluations

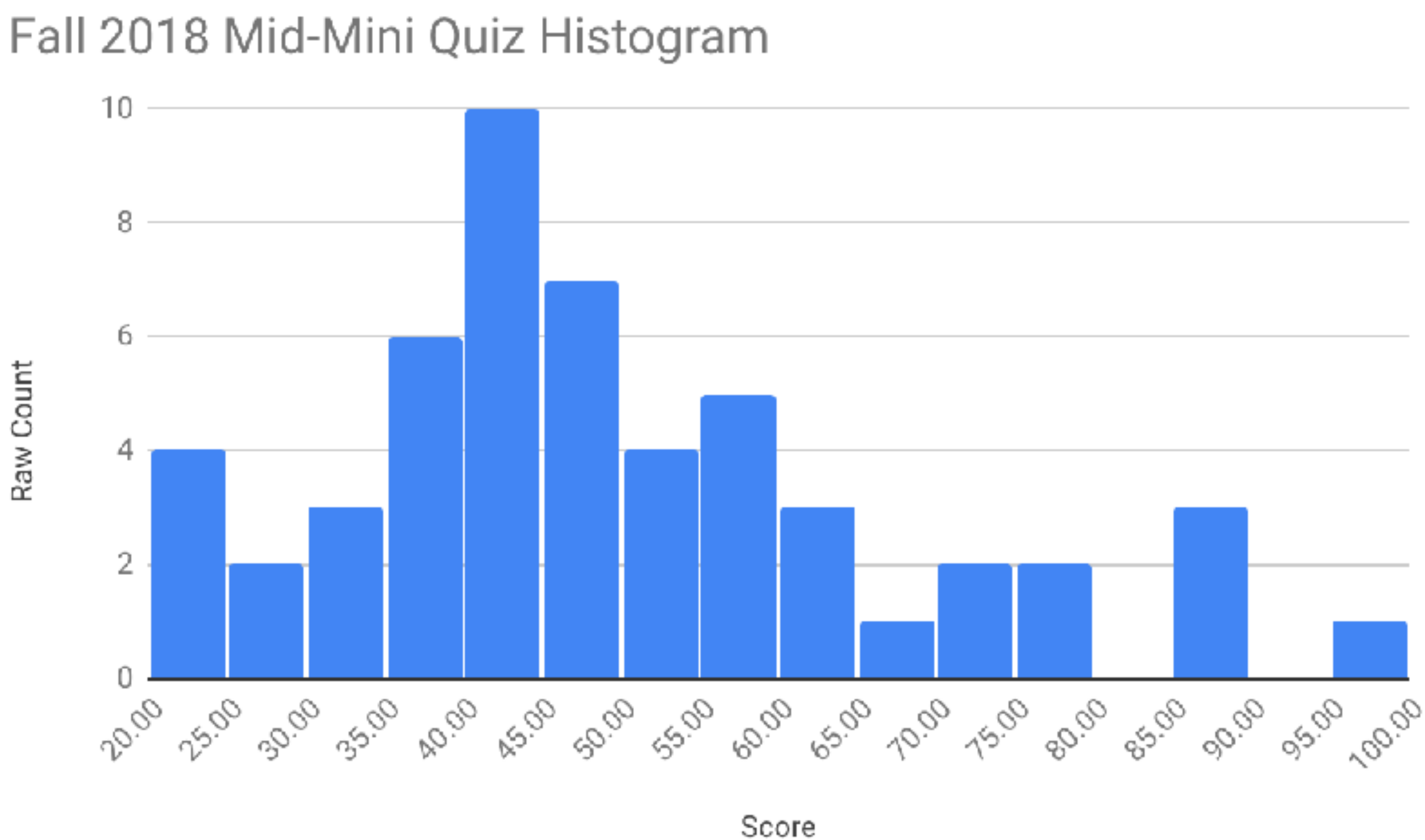
Please provide valuable feedback/vent your frustration



If you're not sure what to write about:

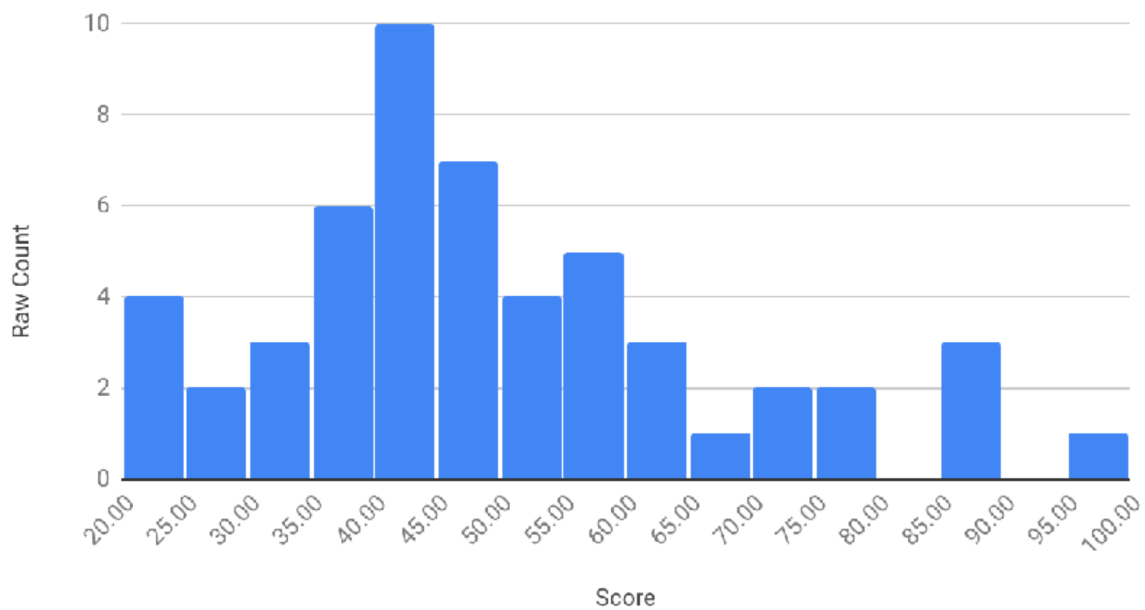
- what additional Python prep *prior* to taking the course would have been helpful?
- what Python review *during* the course would have been helpful?
- most/least favorite parts of the course?

Quiz Results



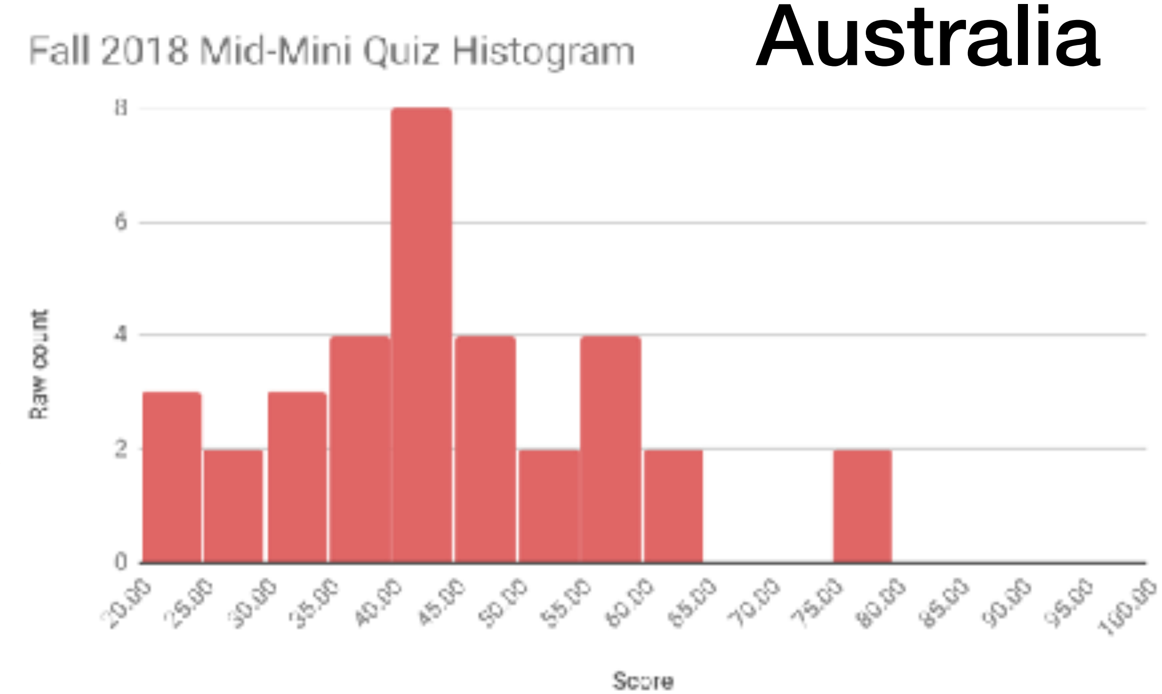
Quiz Results

Combined



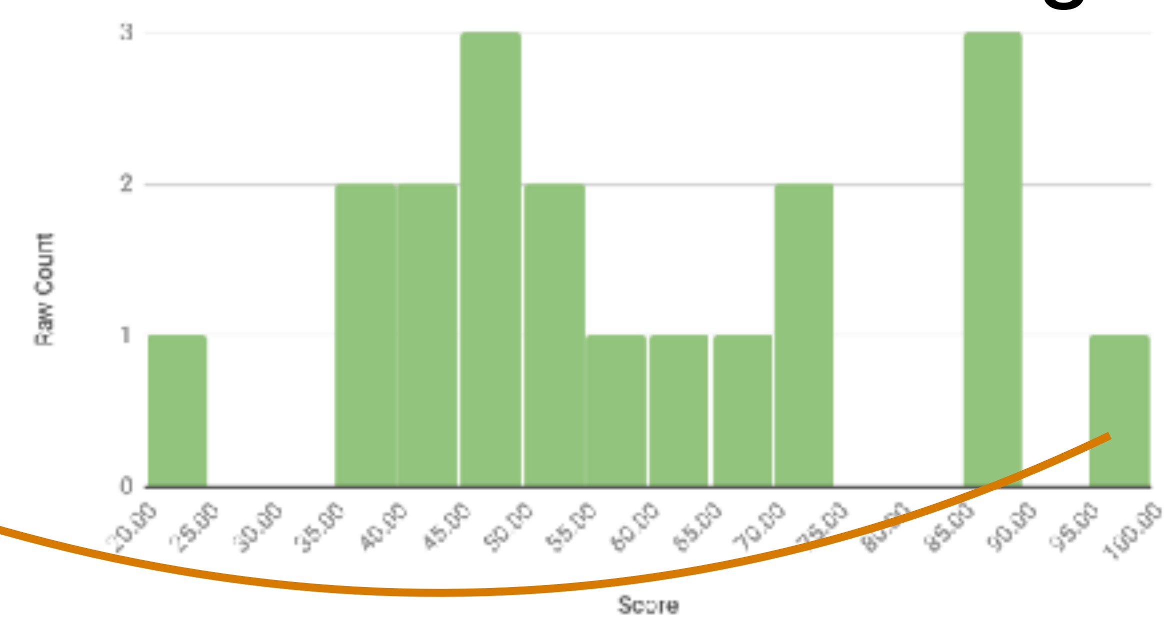
Mean: 49.3, std dev: 17.8

Australia



Mean: 44.1, std dev: 13.6

Pittsburgh



Mean: 58.6, std dev: 20.5

Max score achieved: 98

Quiz Results

1. Don’t panic

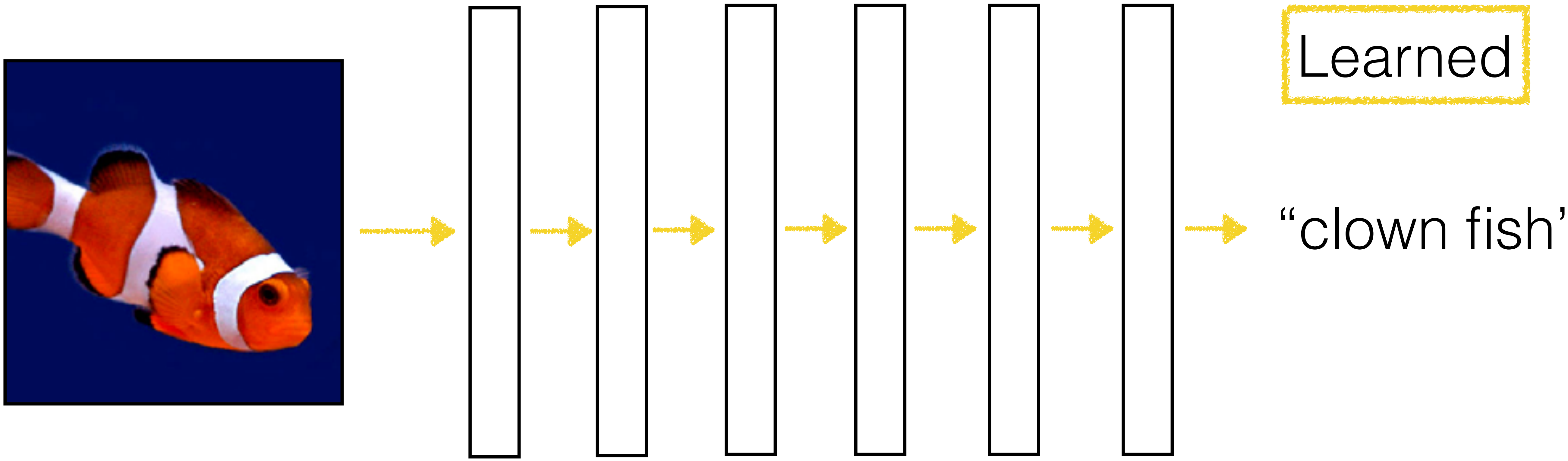
2. Quiz regrade requests due Friday 11/30 (email me and *be specific* about what you think was incorrectly graded)

3. There’s still the final exam next Thursday

Today

- Recap on some key neural net ideas
- Image analysis with convolutional neural nets
- Time series analysis with recurrent neural nets
- Roughly how learning a neural net works
- Overview of some deep learning topics we didn't get to
- Course wrap-up

Deep Learning



- Inspired by biological neural nets *but otherwise not the same at all* (biological neural nets do *not* work like deep nets)
- Learns a layered representation
    - Tries to get rid of manual feature engineering
    - Need to design constraints for what features are learned to account for structure in data (e.g., images, text, …)

Learning a neural net amounts to curve fitting

We’re just estimating a function

Neural Net as Function Approximation

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

this is a **function**

Single-layer neural net example:

```
def f(input):
    output = softmax(np.dot(W, input) + b)
    return output
```

the only things that we are learning
(we fix their dimensions in advance)

We are fixing what the function `f` looks like in code
and are only adjusting `W` and `b`!!!

Neural Net as Function Approximation

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

Single-layer neural net example:

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

Two-layer neural net example:

```
layer1_output = relu(np.dot(W1, input) + b1)

output = softmax(np.dot(W2, layer1_output) + b2)
```

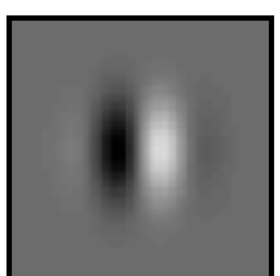
Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)

Architecting Neural Nets

- Increasing number of layers (depth) makes neural net more complex
  - Can approximate more functions
  - More parameters needed
    - More training data may be needed
- Designing neural net architectures is a bit of an art
  - How to select the number of neurons for intermediate layers?
  - Very common in practice: modify existing architectures that are known to work well (e.g., VGG-16 for computer vision/image processing)

Image analysis with Convolutional Neural Nets (CNNs, also called convnets)

Convolution

filter 

Slide by Phillip Isola

Convolution

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 image

0	0	0
0	1	0
0	0	0

Filter
(also called “kernel”)

Convolution

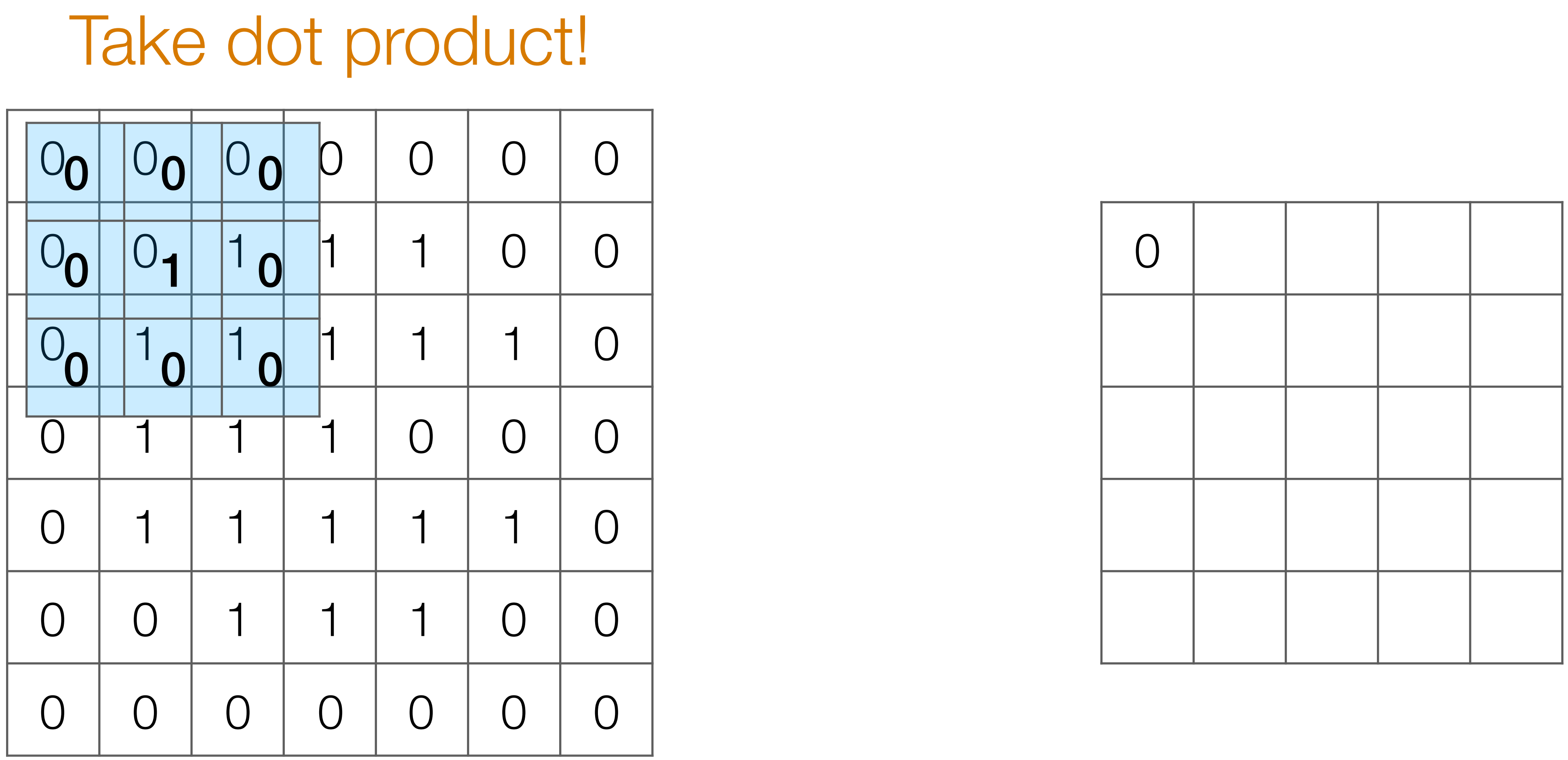
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input image

0	0	0
0	1	0
0	0	0

Filter
(also called “kernel”)

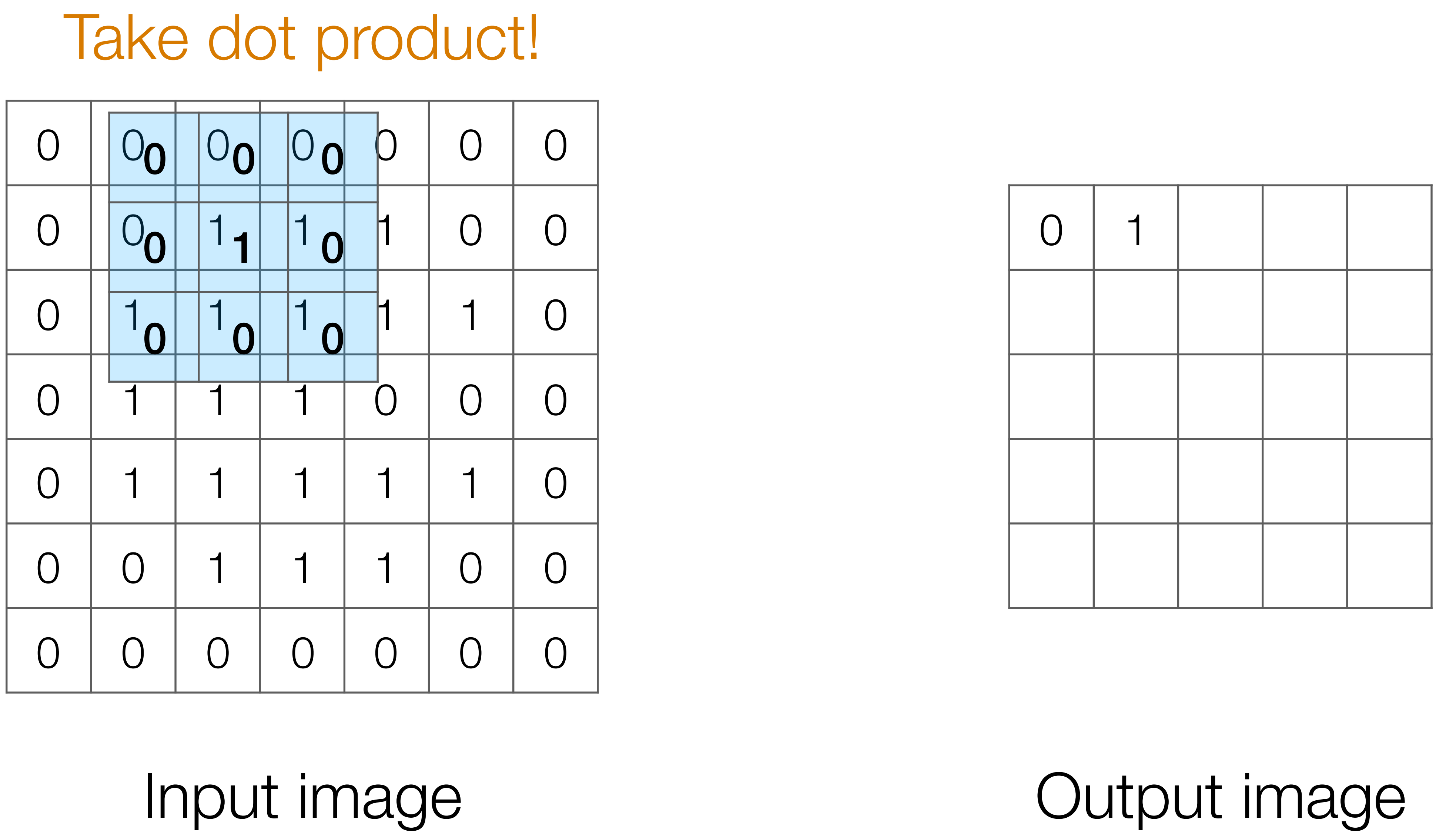
Convolution



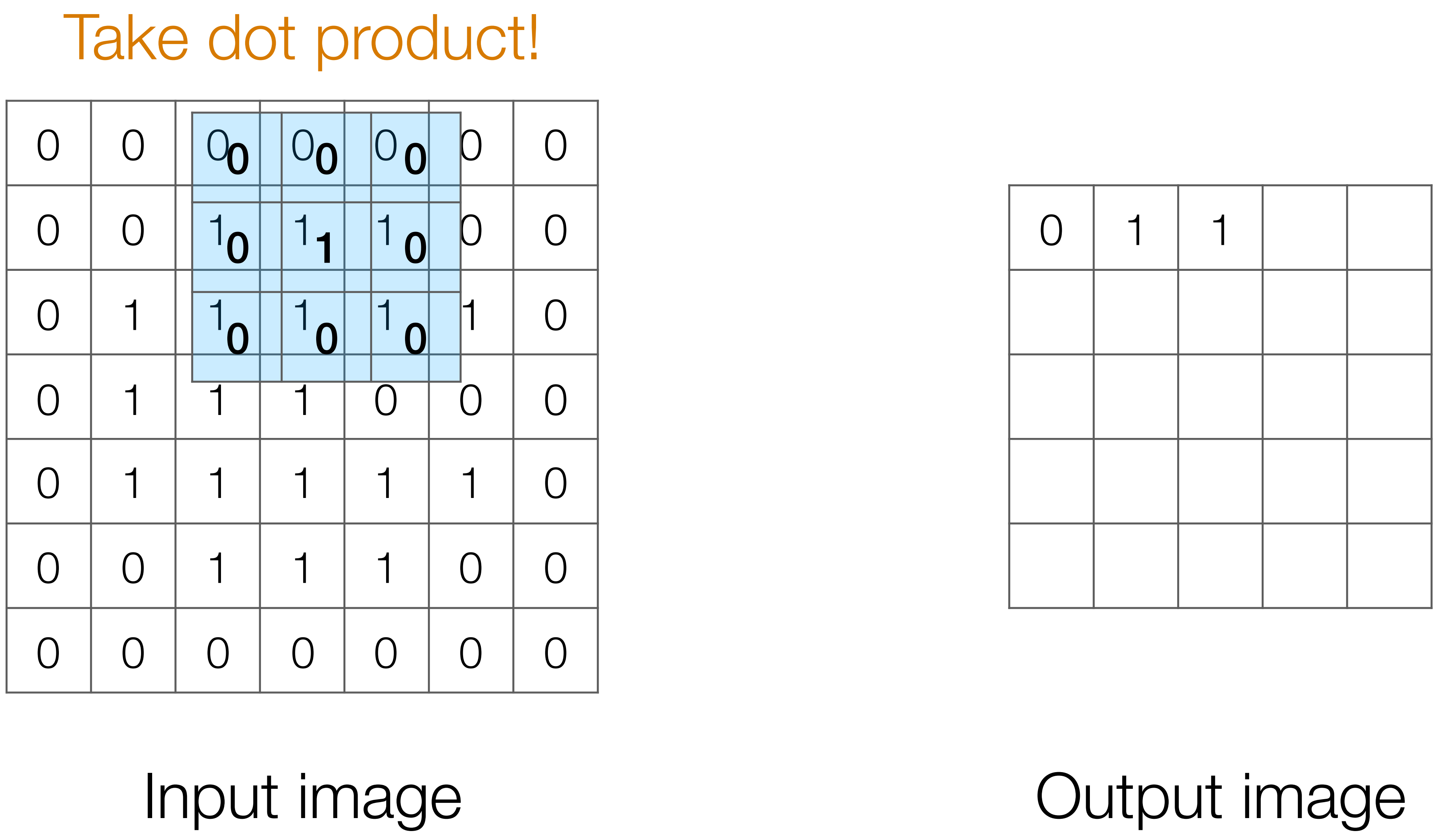
Input image

Output image

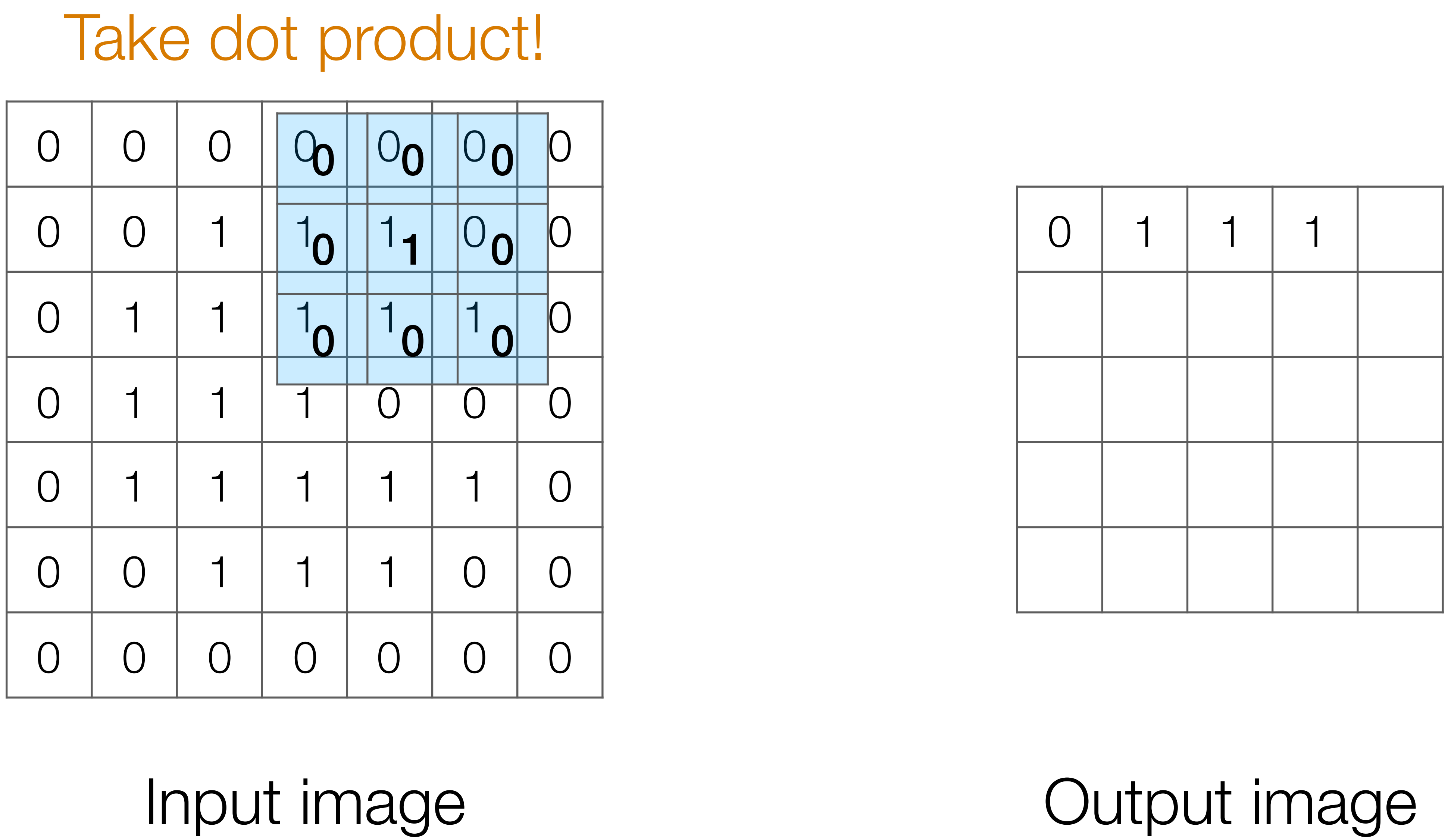
Convolution



Convolution

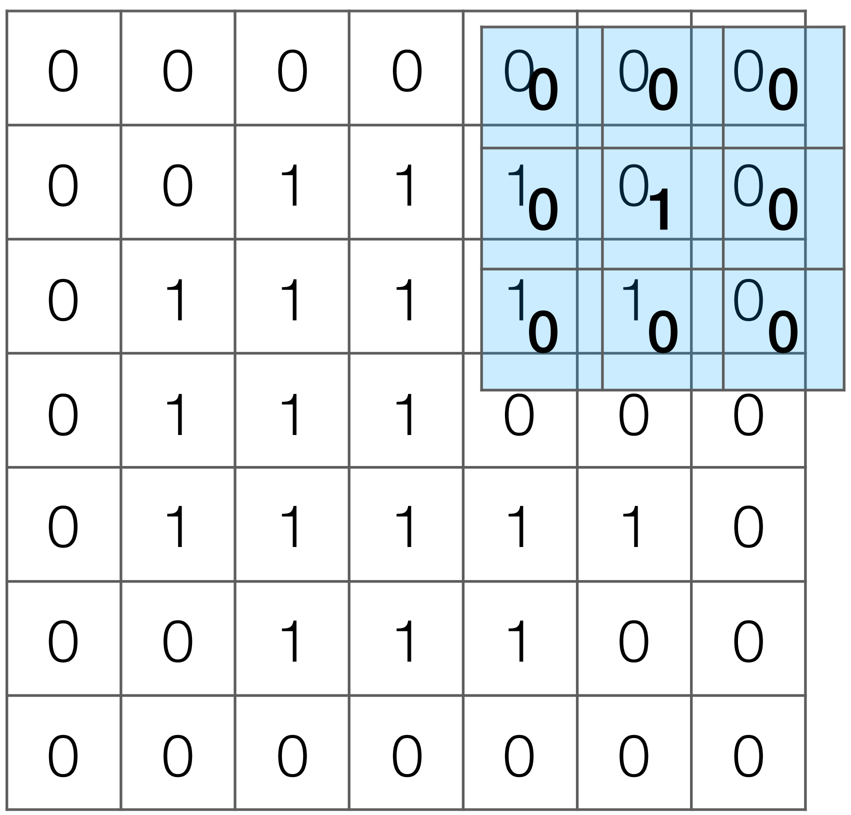


Convolution



Convolution

Take dot product!



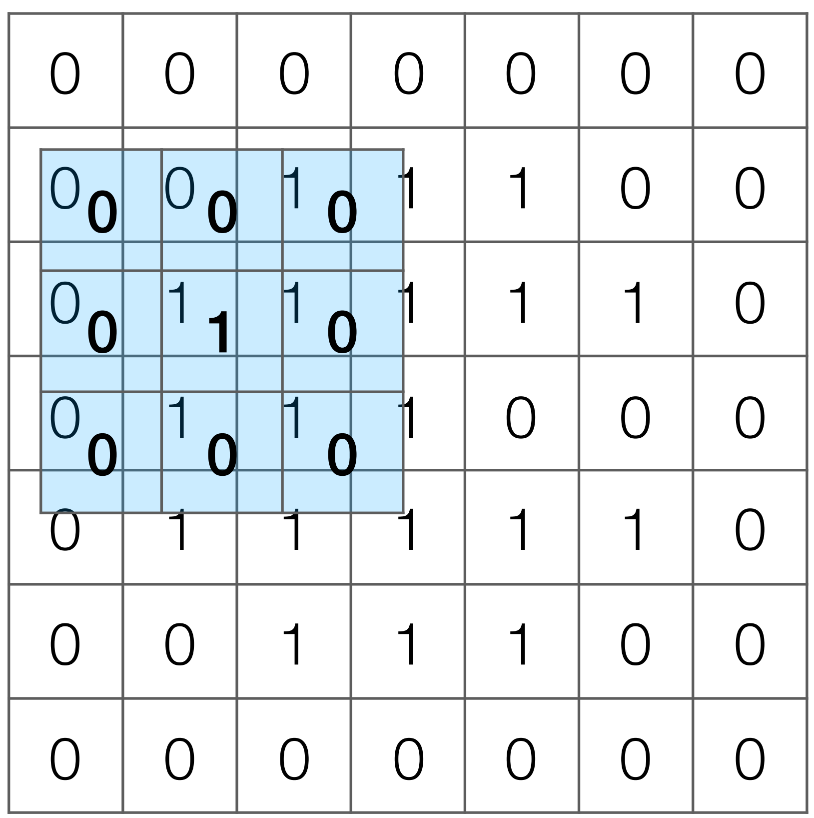
Input image

0	1	1	1	0

Output image

Convolution

Take dot product!



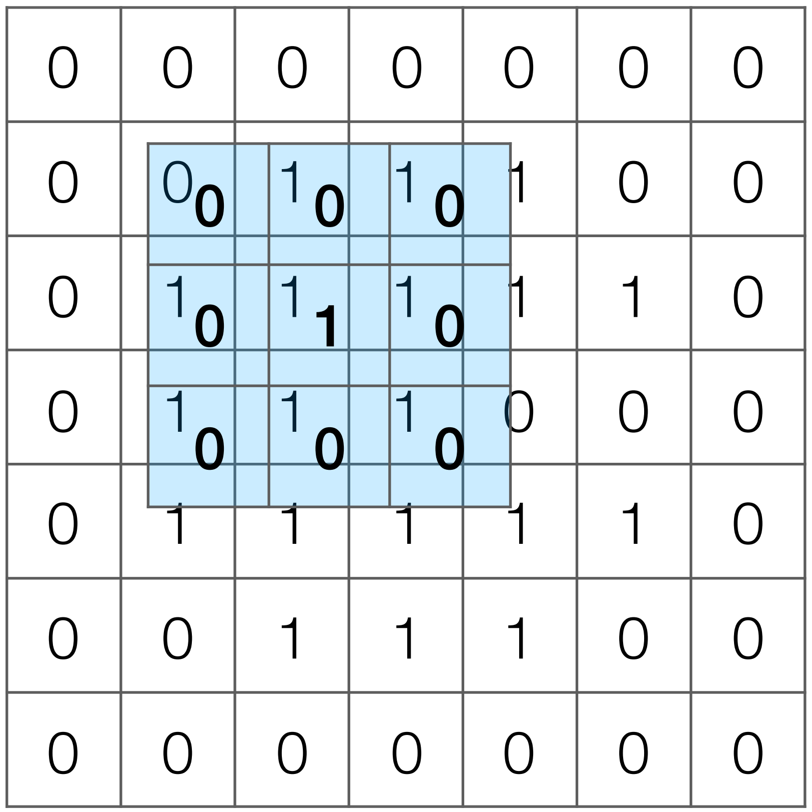
Input image

0	1	1	1	0
1				

Output image

Convolution

Take dot product!



Input image

0	1	1	1	0
1	1			

Output image

Convolution

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Input image

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0's to input

Convolution

$$
\begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{bmatrix}
*
\begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0}
\end{bmatrix}
=
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Input image

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0's to input

Convolution

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0}
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix}
$$

Input image

Output image

Convolution

$$
\begin{bmatrix} 0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0\\ 0&1&1&1&1&1&0\\ 0&1&1&1&0&0&0\\ 0&1&1&1&1&1&0\\ 0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0 \end{bmatrix} * \frac{1}{9}\begin{bmatrix} 1&1&1\\ 1&1&1\\ 1&1&1 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 3&5&6&5&3\\ 5&8&8&6&3\\ 6&9&8&7&4\\ 5&8&8&6&3\\ 3&5&6&5&3 \end{bmatrix}
$$

Input image

Output image

Convolution

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

*

-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

Input image

Output image

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

*

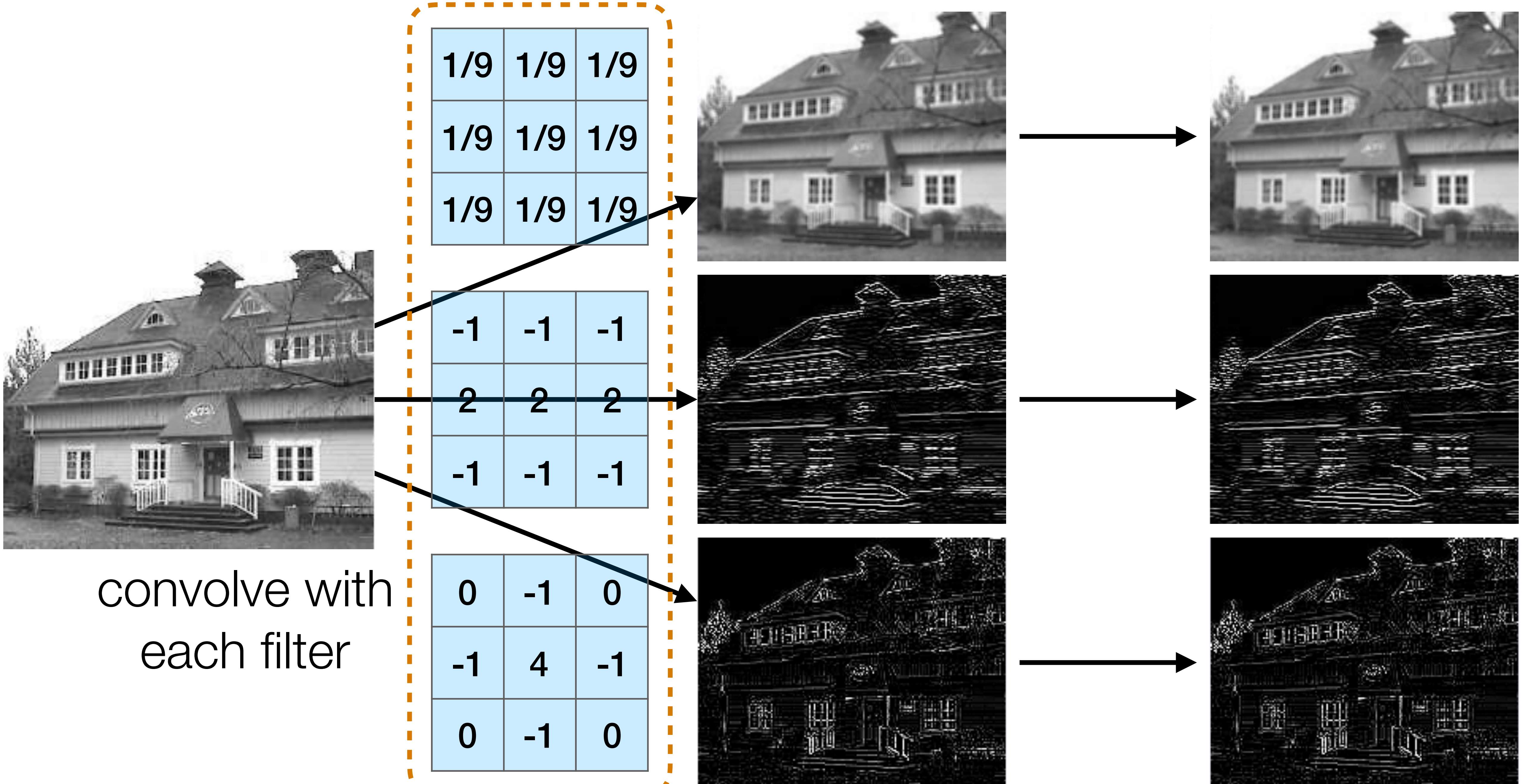
-1	-1	-1
2	2	2
-1	-1	-1

=

(this example finds horizontal edges)

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

Convolution Layer

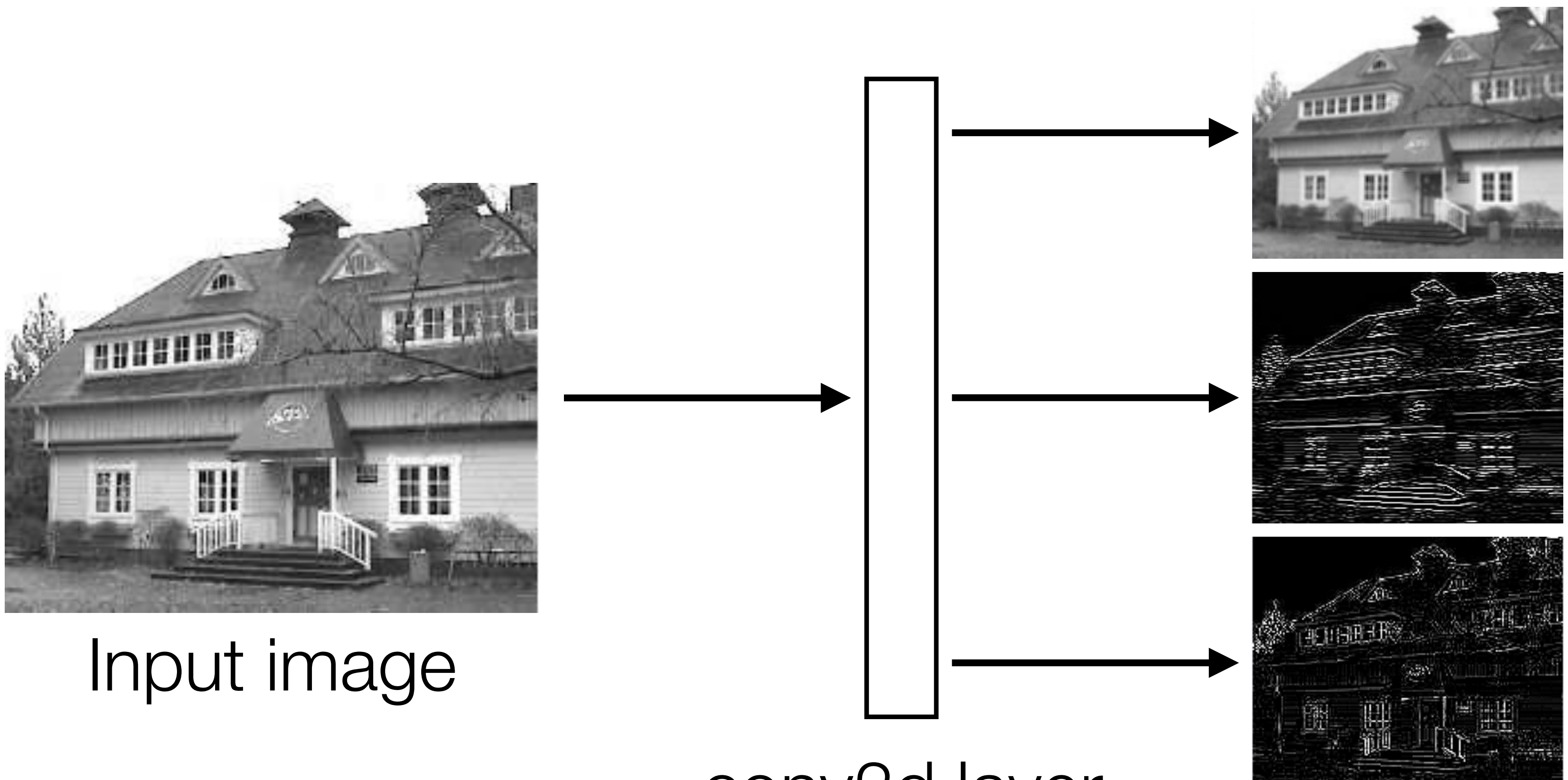


filters are actually unknown and are learned!

activation (e.g., ReLU)

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

Convolution Layer



Input image

conv2d layer
with ReLu activation
and three 3x3 kernels

Output images

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

Convolution Layer

Input image

dimensions:
height,
width

conv2d layer
with ReLu activation
and three 3x3 kernels

Stack output images into a single "output feature map"

dimensions:
height-2,
width-2,
number of kernels
(3 in this case)

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

Convolution Layer

Input image

dimensions:
height,
width

conv2d layer
with ReLu activation
and *k* 3x3 kernels

Stack output images into a single "output feature map"

dimensions:
height-2,
width-2,
k

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

Convolution Layer

Input image

dimensions:
height,
width,
depth *d* (# channels)

conv2d layer
with ReLu activation
and *k* 3x3x*d* kernels

technical detail: there's also a bias vector

Stack output images into a single "output feature map"

dimensions:
height-2,
width-2,
k

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

Pooling

- Aggregate local information

- Produces a smaller image
(each resulting pixel captures some “global” information)

- If object in input image shifts a little, output is the same

Max Pooling

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 image

*

-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

Max Pooling

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

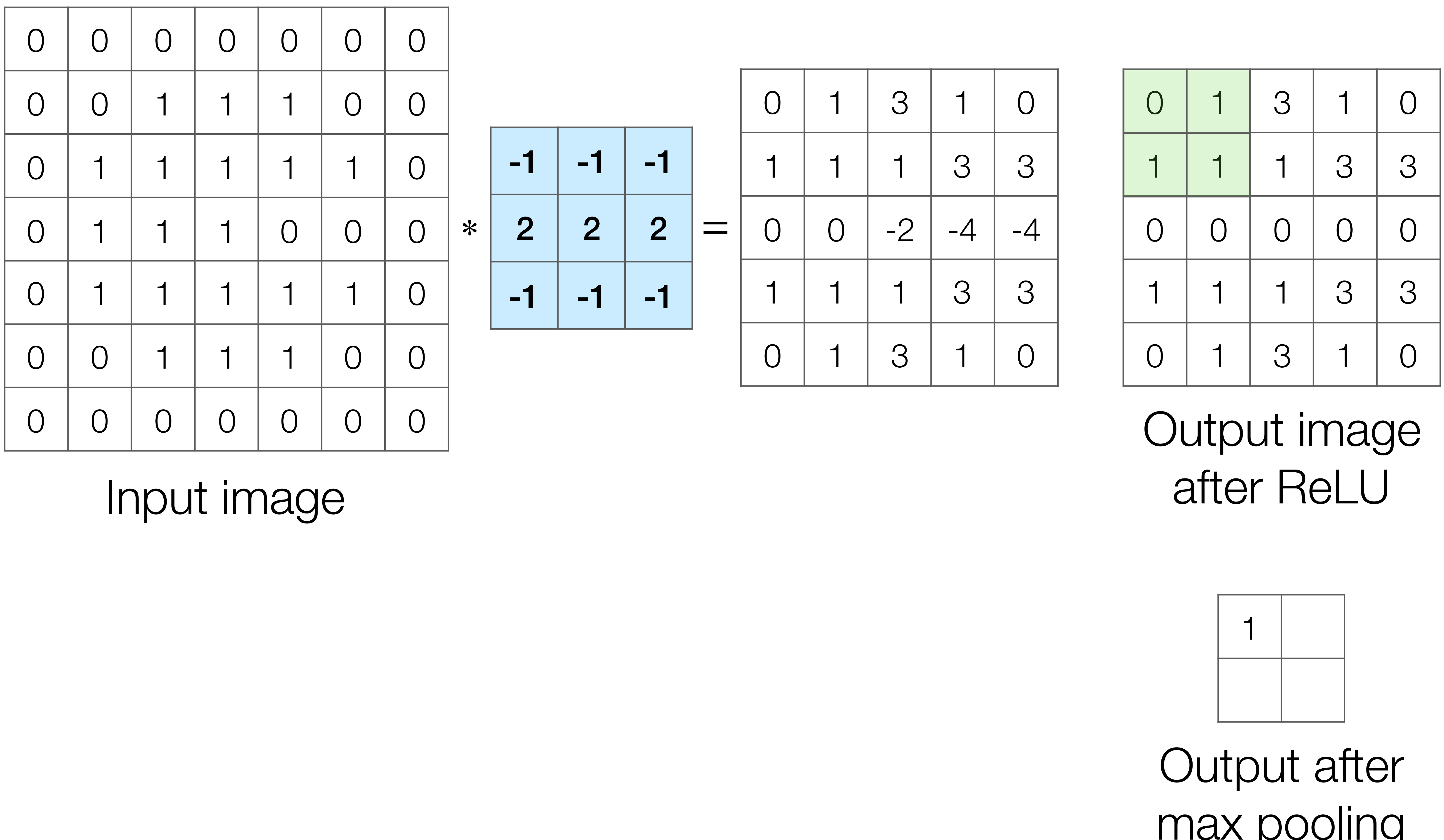
Input image

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


Input image
Output image
after ReLU
Output after
max pooling

Max Pooling

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

Input image

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

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 image

*

-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

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

Input image

$$
\begin{bmatrix}
0&1&3&1&0\\
1&1&1&3&3\\
0&0&\boxed{0}&\boxed{0}&0\\
1&1&\boxed{1}&\boxed{3}&3\\
0&1&3&1&0
\end{bmatrix}
$$

Output image after ReLU

$$
\begin{bmatrix}
1&3\\
1&3
\end{bmatrix}
$$

Output after max pooling

Max Pooling

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

Input image

$$\begin{bmatrix} 0 & 1 & 3 & 1 & 0 \\ 1 & 1 & 1 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 3 & 3 \\ 0 & 1 & 3 & 1 & 0 \end{bmatrix}$$

Output image after ReLU

What numbers were involved in computing this 1?

$$\begin{bmatrix} 1 & 3 \\ 1 & 3 \end{bmatrix}$$

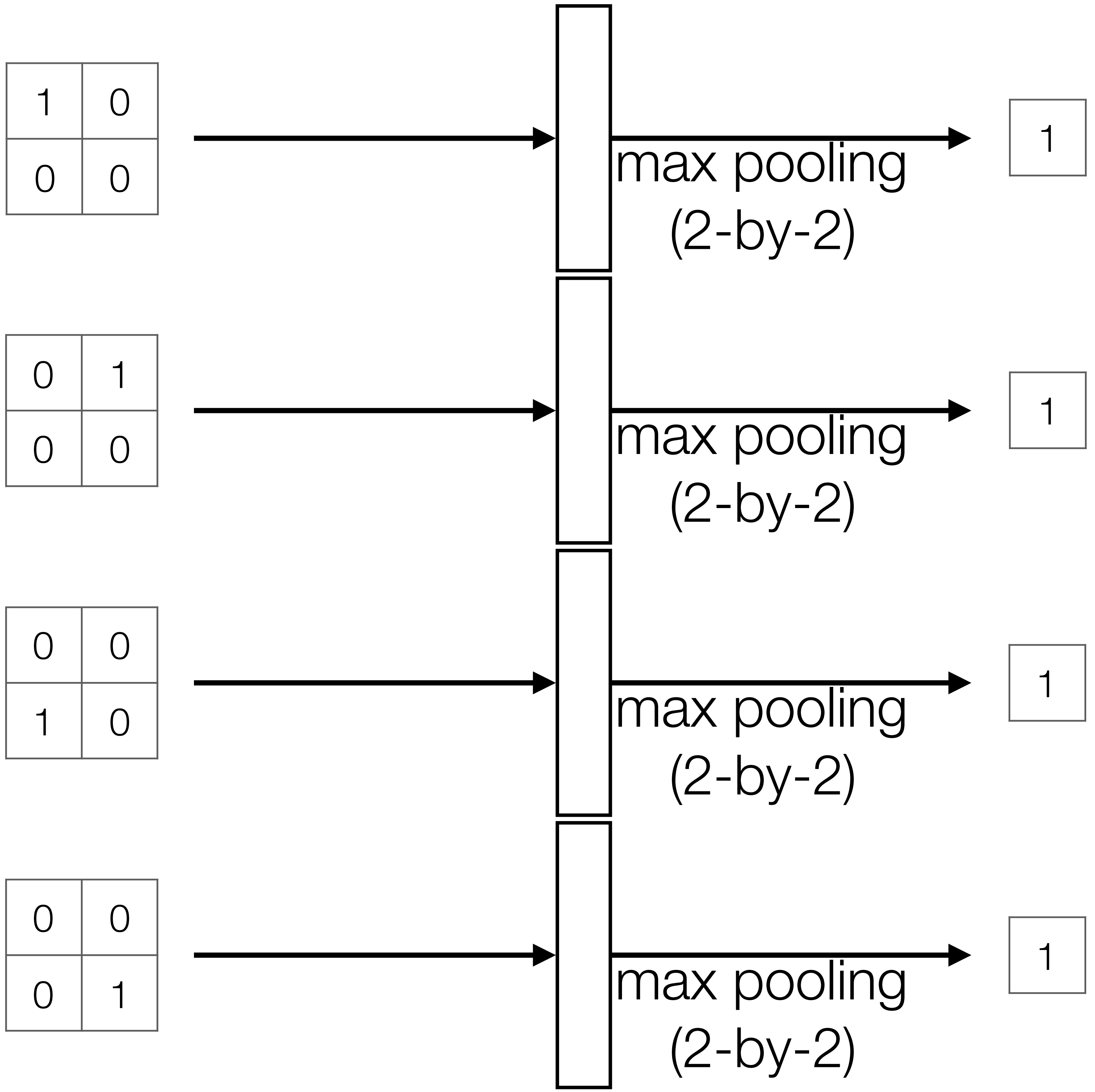
Output after max pooling

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

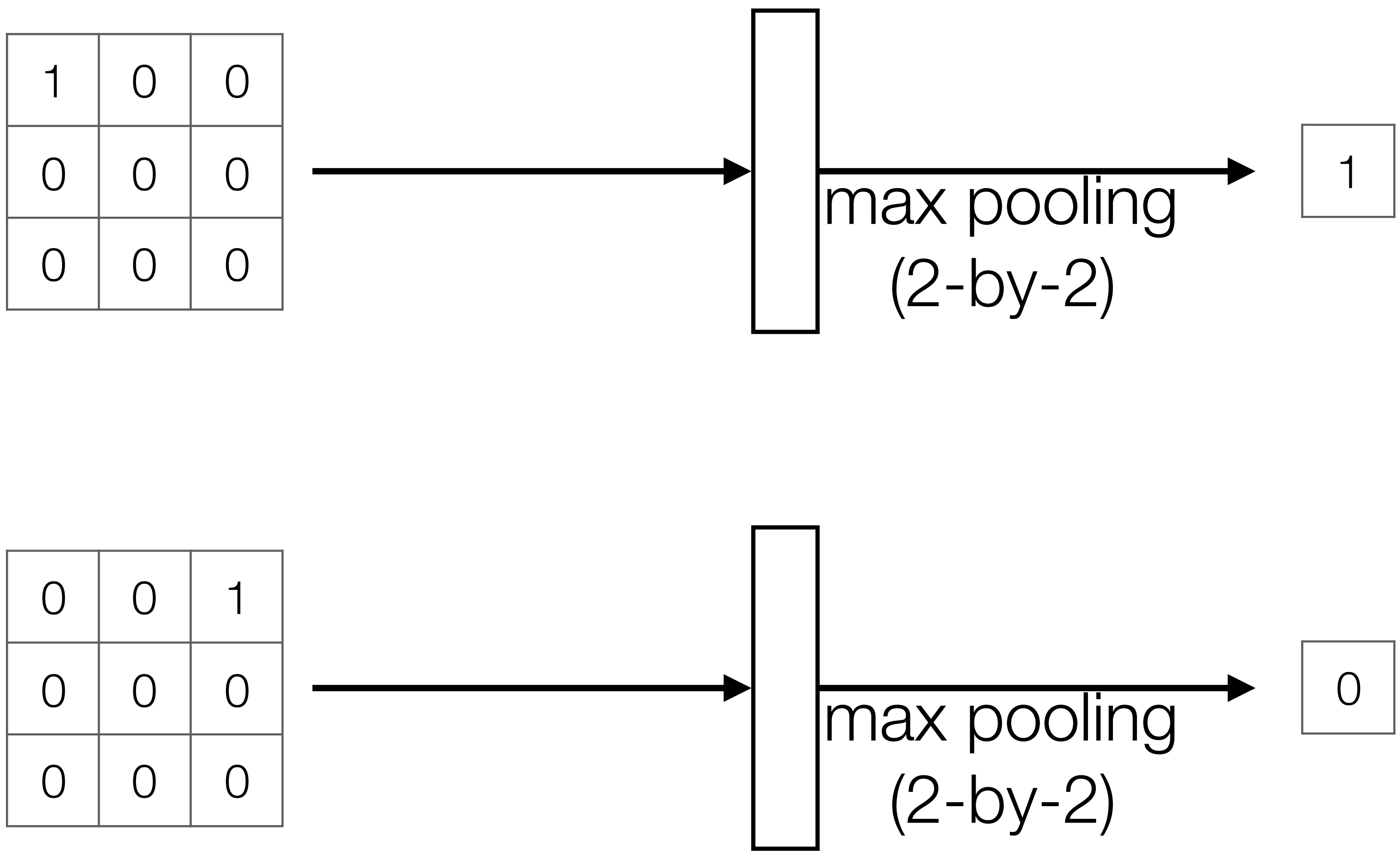
Example: applying max pooling again results in a single pixel that captures info from entire input image!

Max Pooling and (Slight) Shift Invariance

Small shift of object in input image results in same output

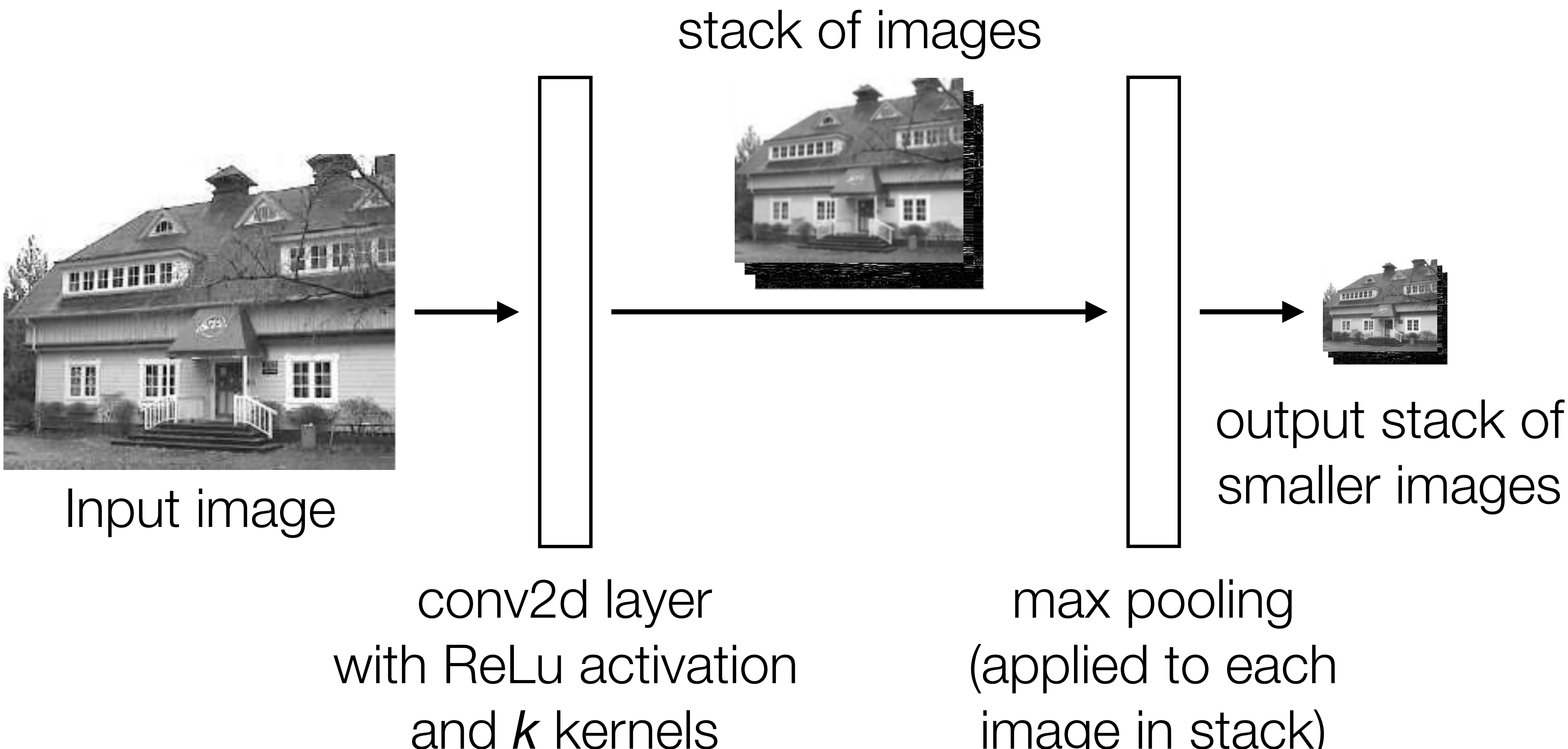


Max Pooling and (Slight) Shift Invariance



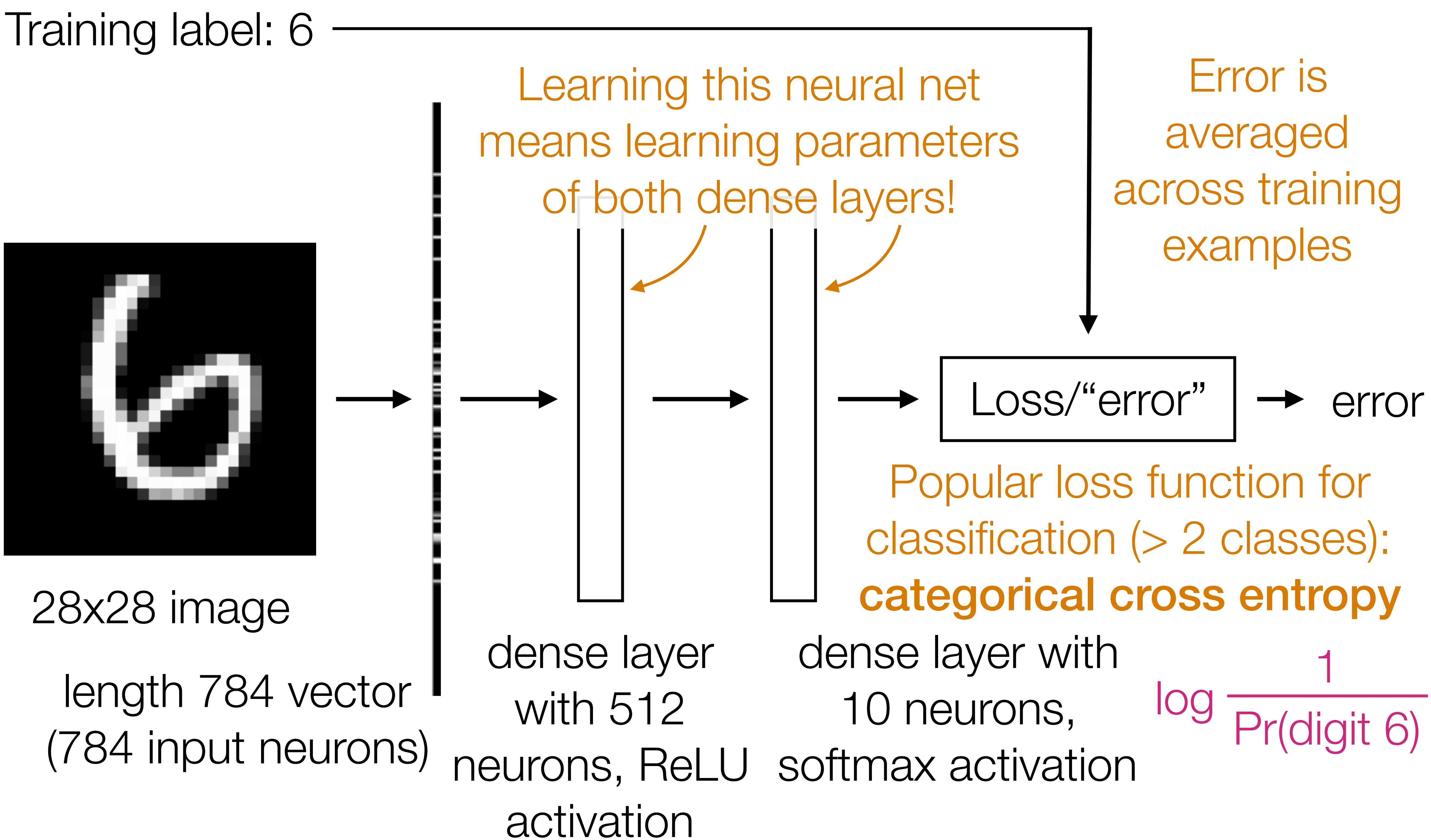
Big shift in input can still change output

Basic Building Block of CNN’s

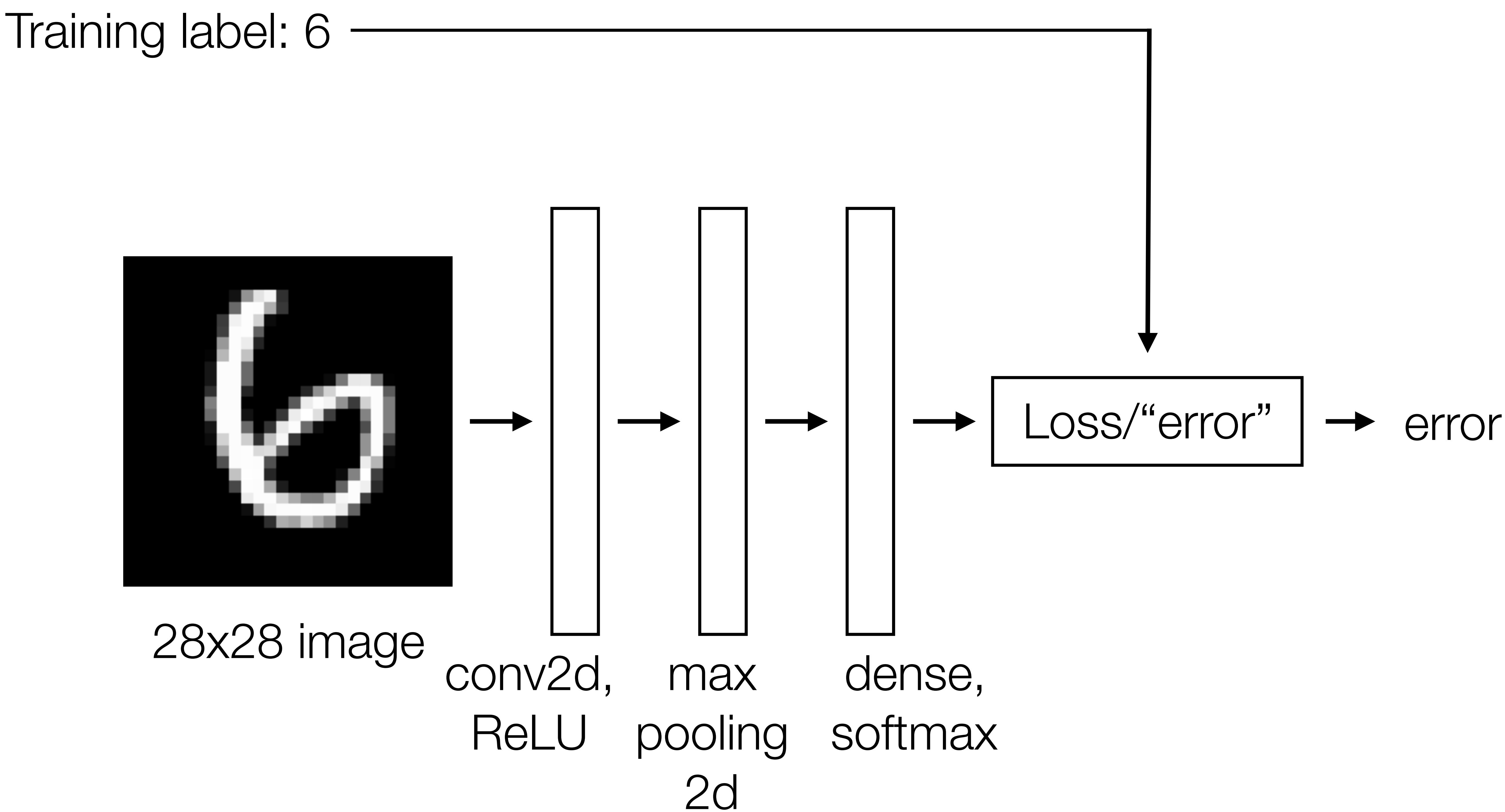


Images from: http://aishack.in/tutorials/image-convolution-examples/

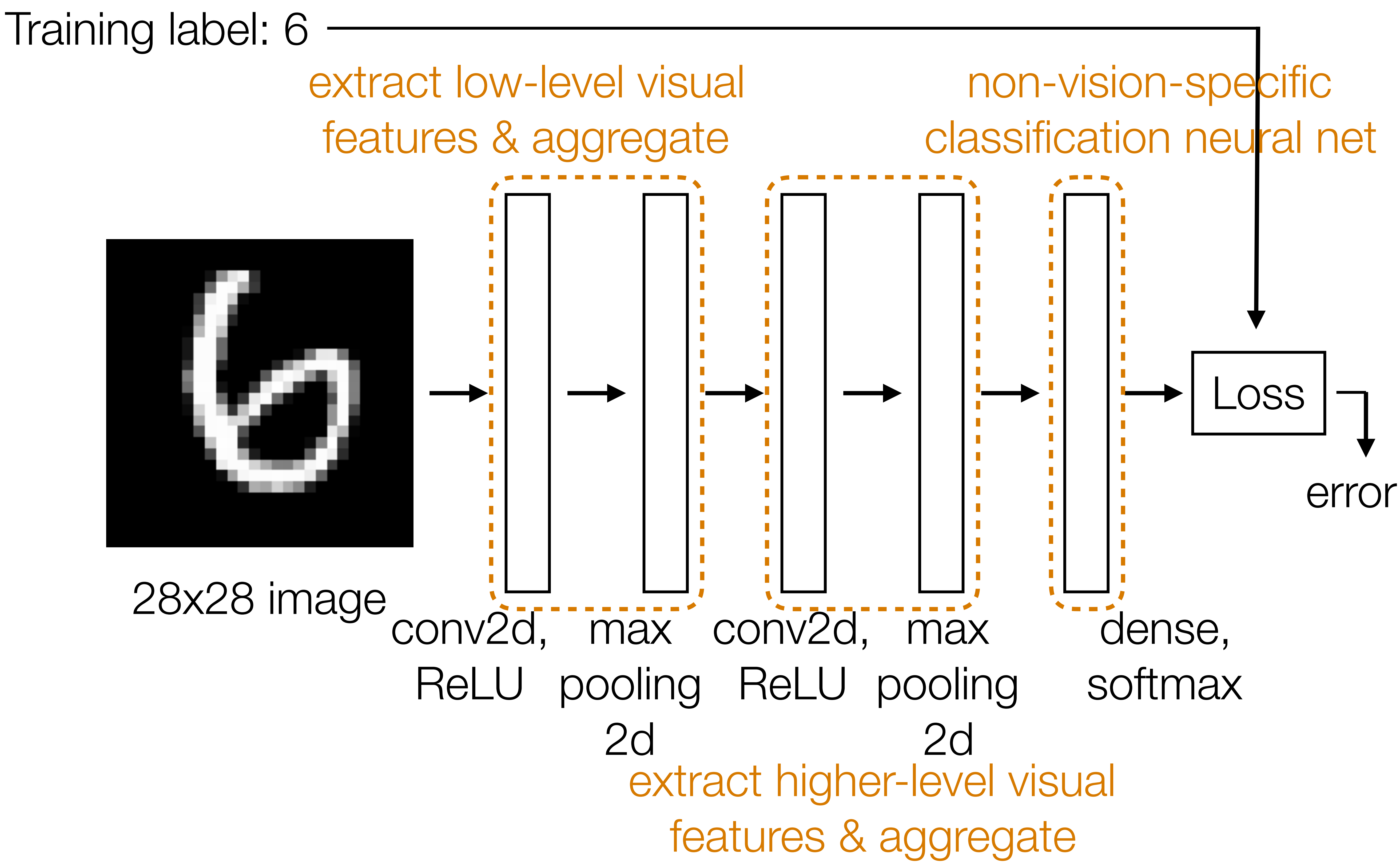
Handwritten Digit Recognition



Handwritten Digit Recognition



Handwritten Digit Recognition



CNN Demo

CNN’s

- Learn convolution filters for extracting simple features

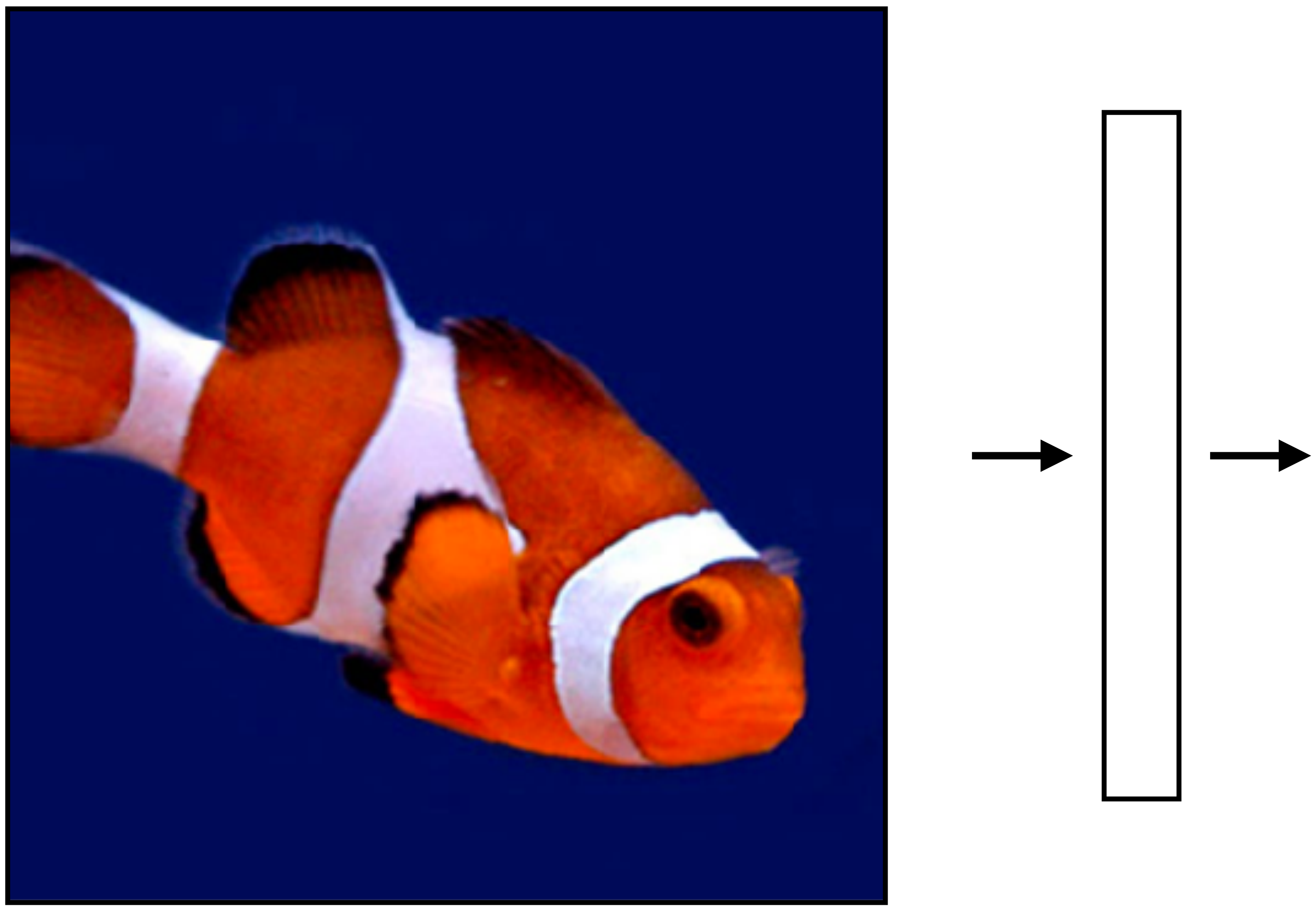
- Max pooling summarizes information and produces a *smaller* output and is invariant to small shifts in input objects

- Can then repeat the above two layers to learn features from increasingly higher-level representations

Time series analysis with Recurrent Neural Networks (RNNs)

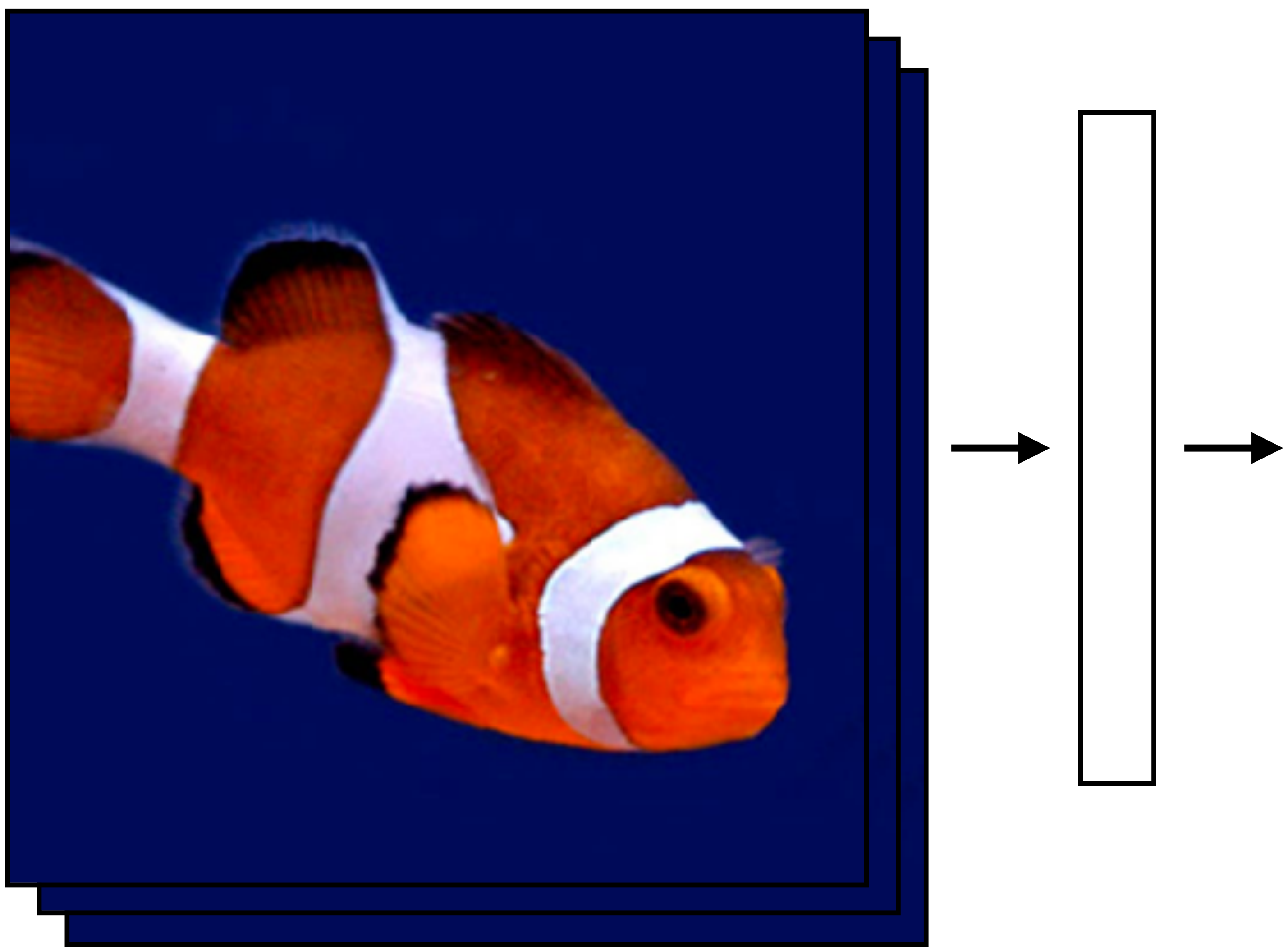
RNNs

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



RNNs

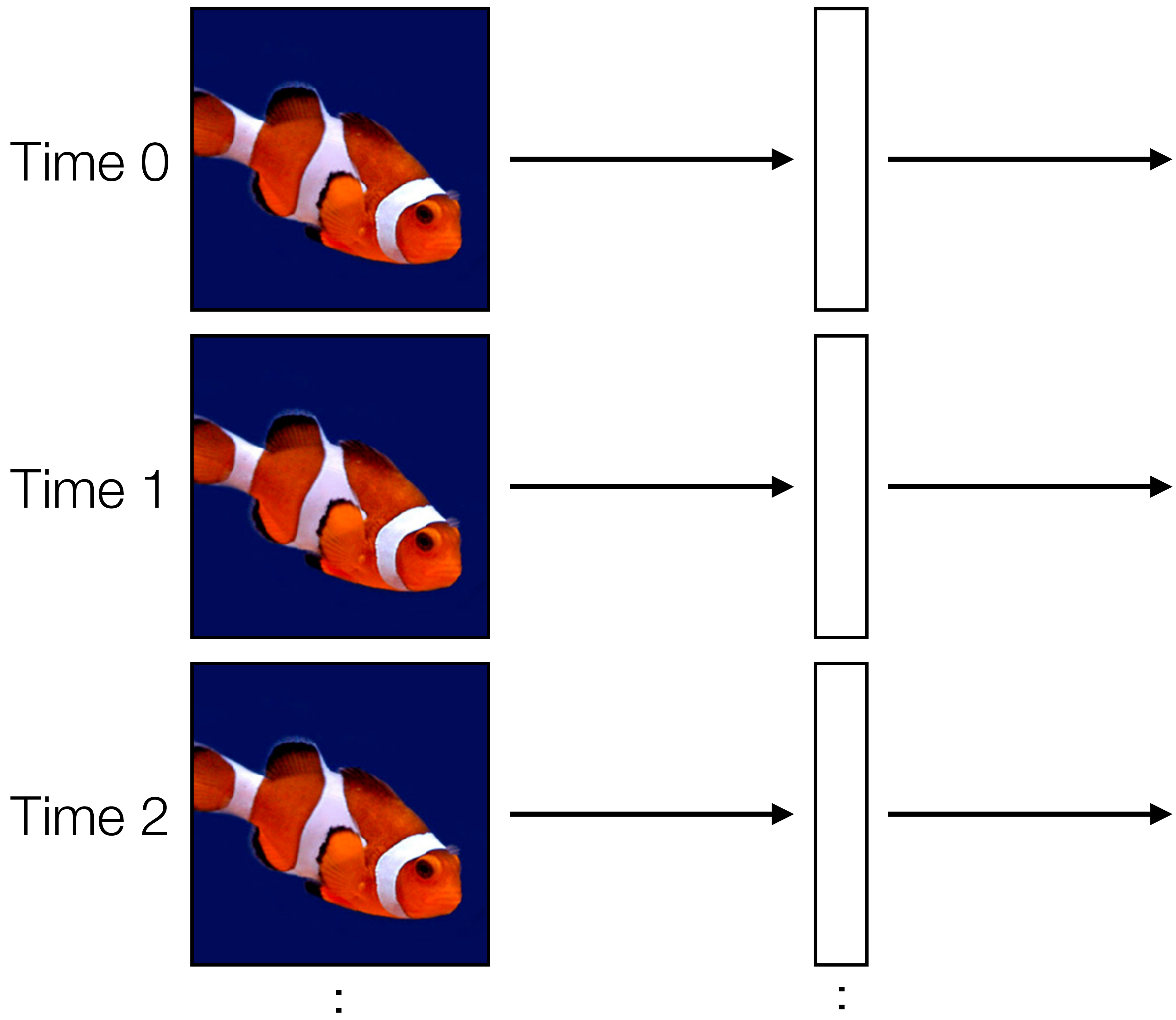
What we've seen so far are "feedforward" NNs



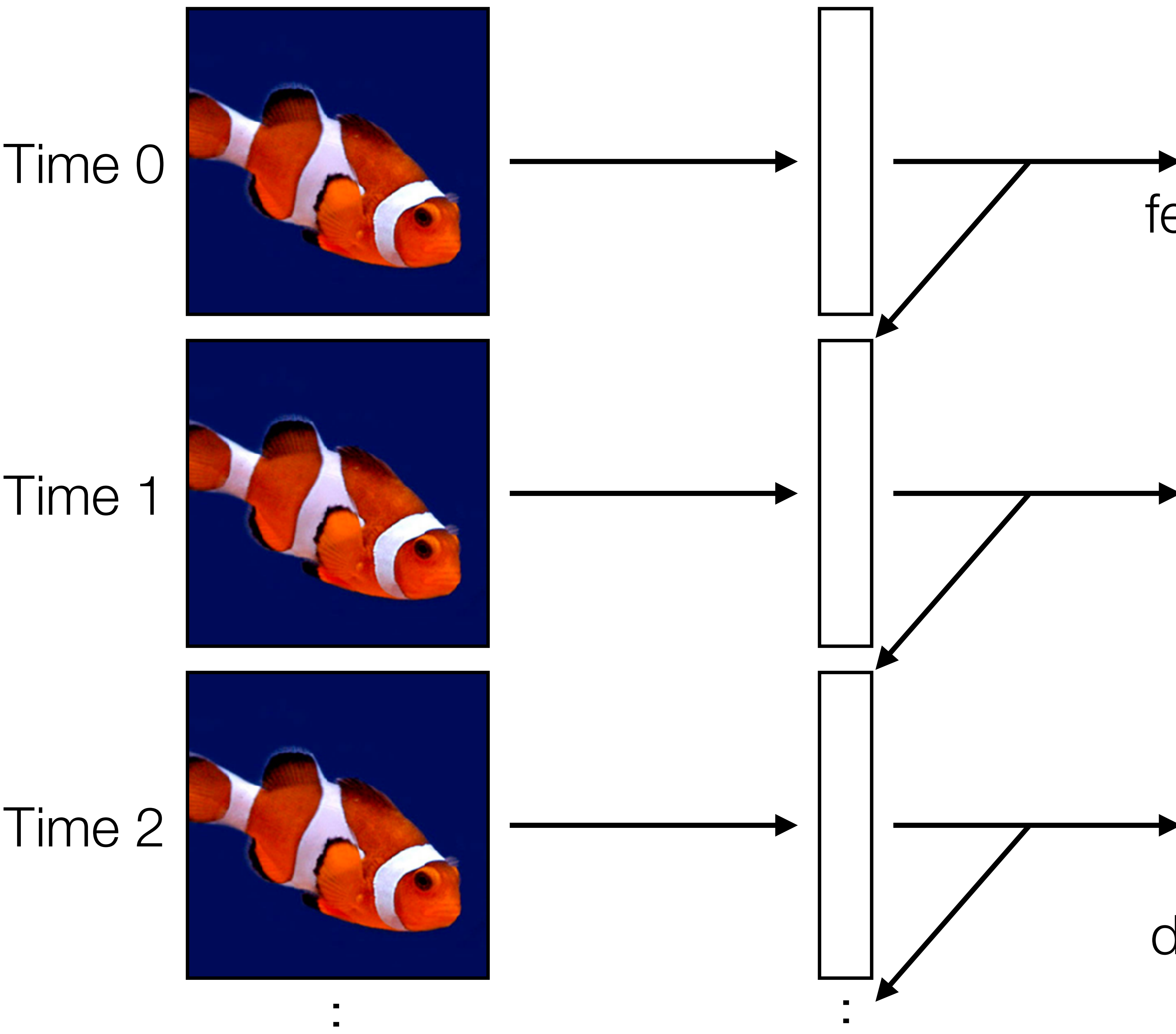
What if we had a video?

RNNs

Feedforward NN's:
treat each video frame
separately



RNNs



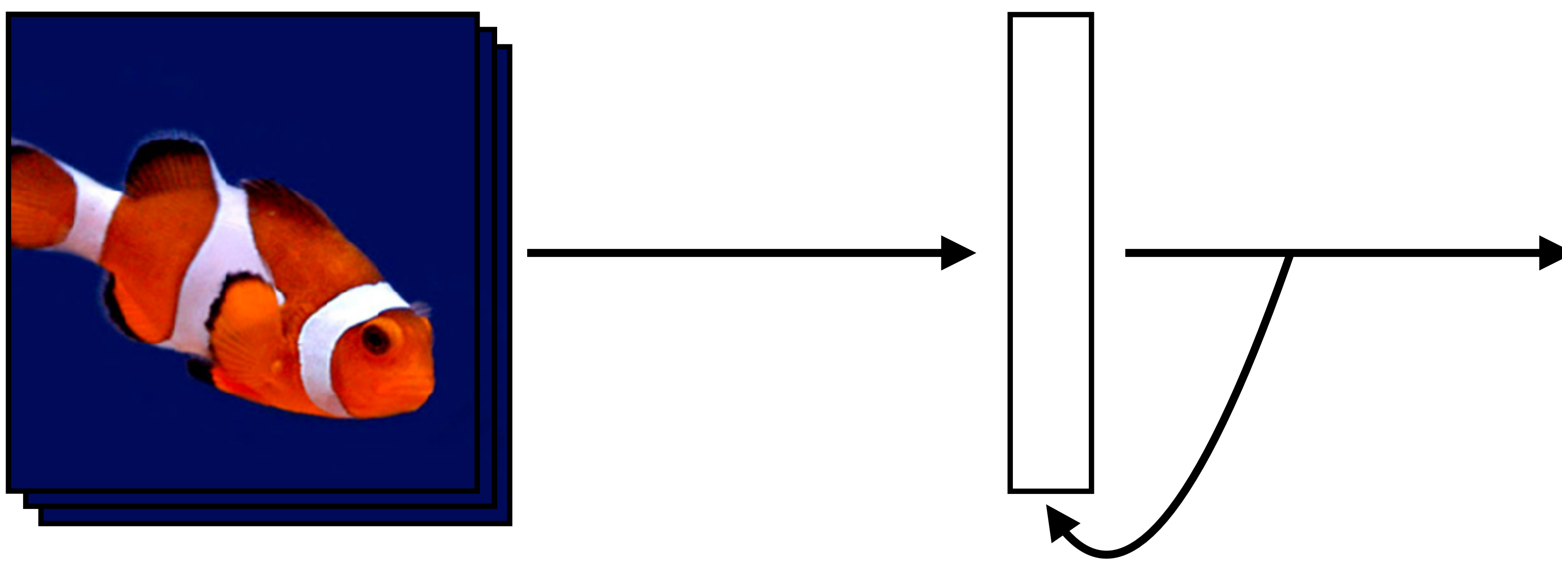
Feedforward NN's:
treat each video frame separately

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

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`, `GRU`

Recommendation:
don't use `SimpleRNN`

RNNs



Time series

RNN layer

Feedforward NN’s:
treat each video frame separately

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

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`, `GRU`

Recommendation:
don’t use `SimpleRNN`

Under the Hood

```
current_state = 0
for input in input_sequence:
  output = g(input, current_state)
  current_state = output
```

Different functions g correspond to different RNNs

Example: SimpleRNN

memory stored in `current_state` variable!

```
current_state = 0

for input in input_sequence:

  output = activation(np.dot(W, input)
                      + np.dot(U, current_state)
                      + b)

  current_state = output
```

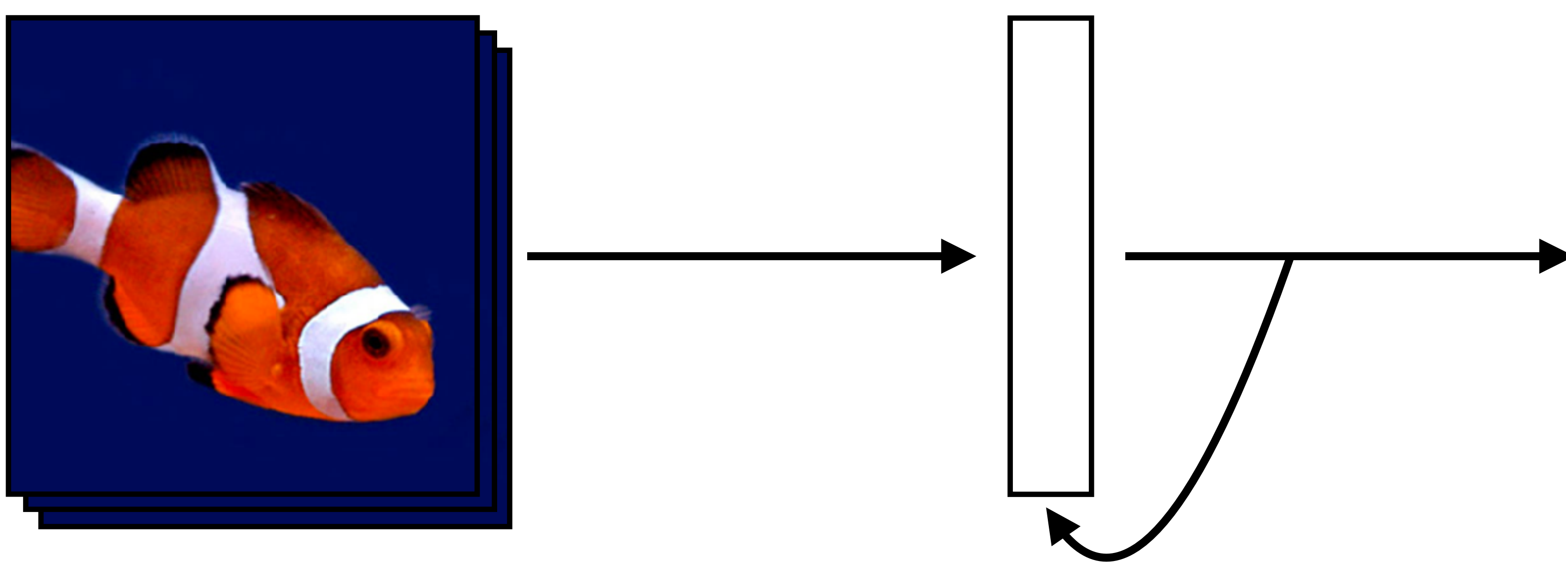
Activation function could, for instance, be ReLU

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

Key idea: **it's like a dense layer in a `for` loop with some memory!**

RNNs

readily chains together with other neural net layers



Time series

RNN layer

like a dense layer that has memory

Feedforward NN's:
treat each video frame separately

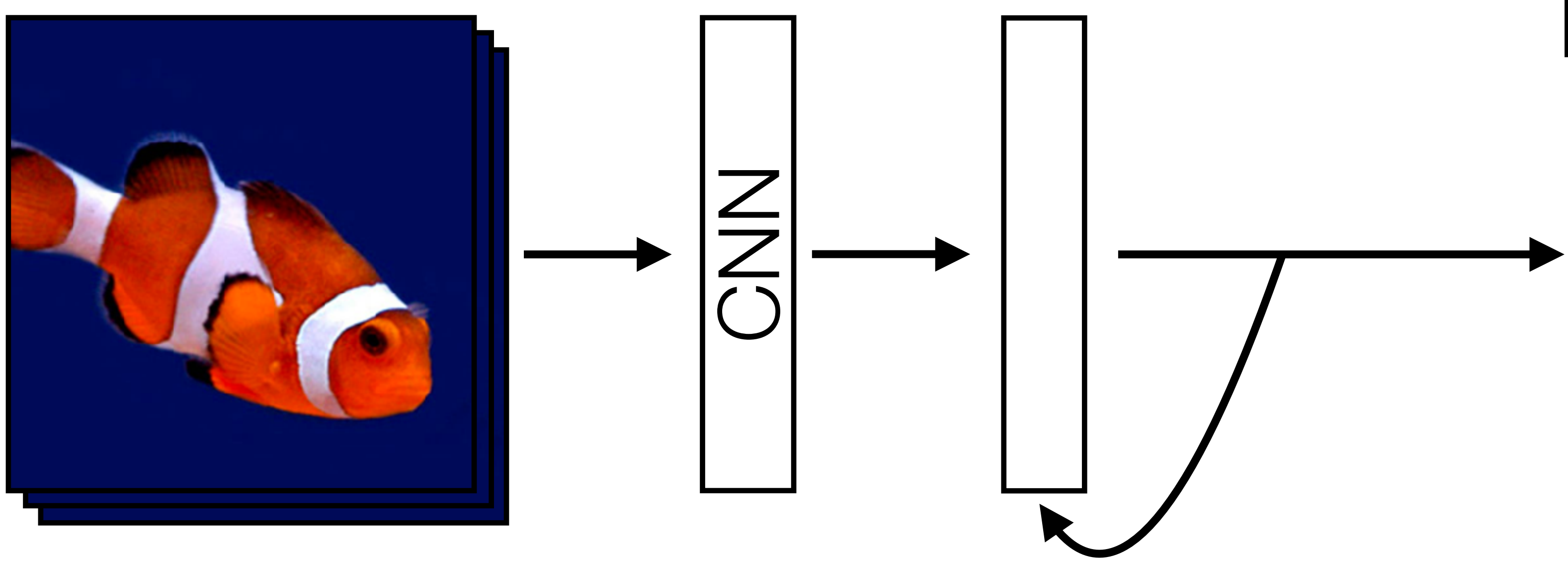
RNN's:
feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`, `GRU`

Recommendation:
don't use `SimpleRNN`

RNNs

readily chains together with other neural net layers



Time series

RNN layer

like a dense layer that has memory

Feedforward NN's: treat each video frame separately

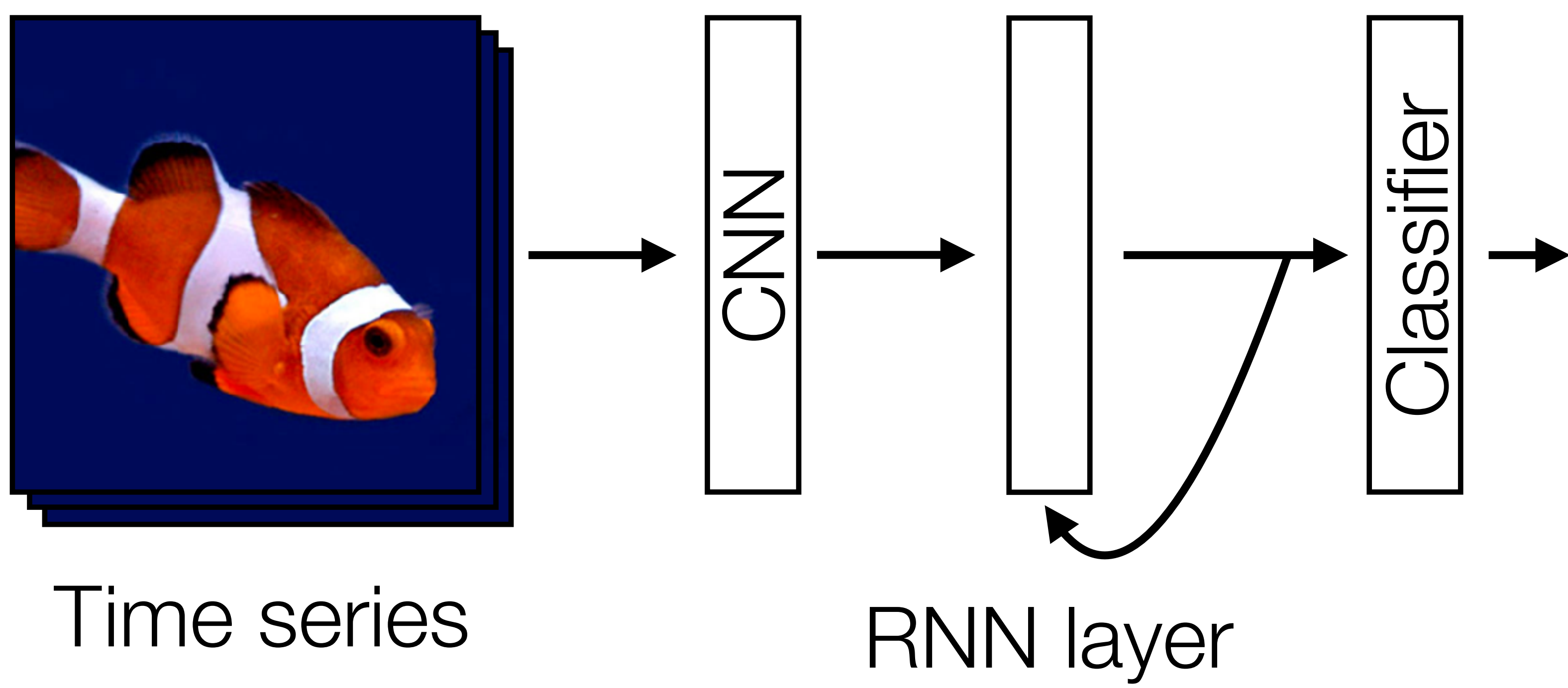
RNN's: feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options: `SimpleRNN`, `LSTM`, `GRU`

Recommendation: don't use `SimpleRNN`

RNNs

readily chains together with other neural net layers



like a dense layer that has memory

Feedforward NN's:
treat each video frame separately

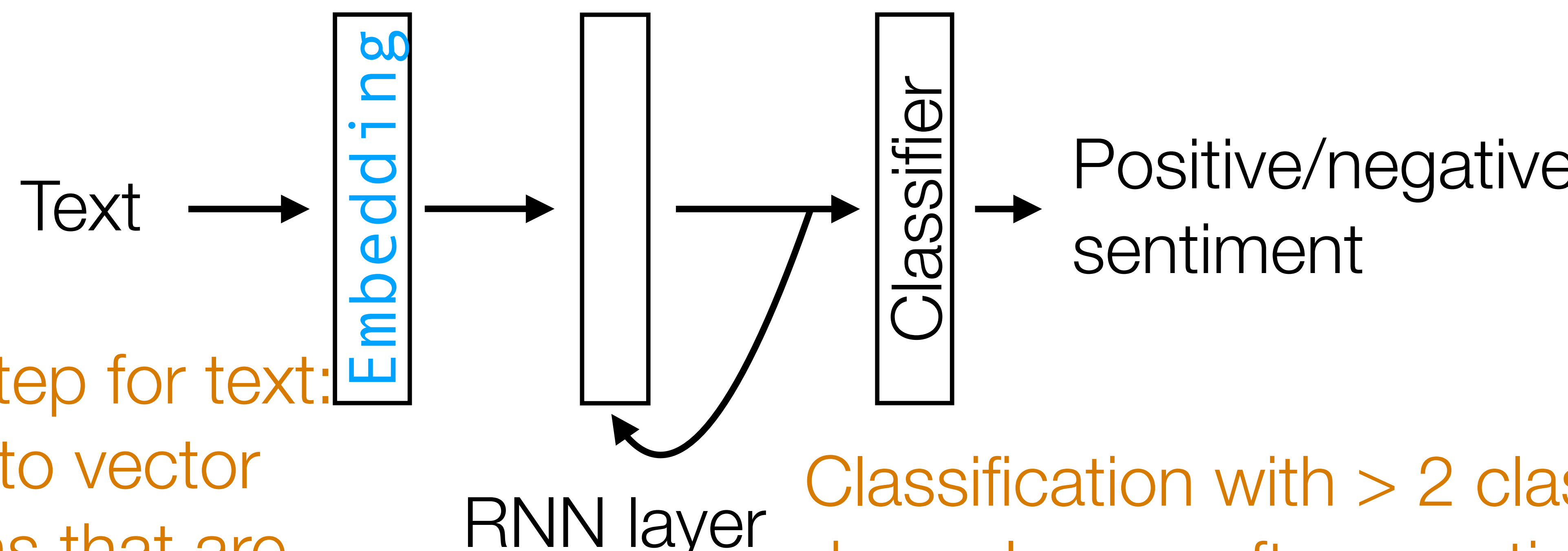
RNN's:
feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`, `GRU`

Recommendation:
don't use `SimpleRNN`

RNNs

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



Common first step for text: turn words into vector representations that are semantically meaningful

In `keras`, use the `Embedding` layer

Classification with > 2 classes: dense layer, softmax activation

Classification with 2 classes: dense layer with 1 neuron, sigmoid activation

RNNs

Demo

RNNs

- Neatly handles time series in which there is some sort of global structure, so memory helps
    - If time series doesn't have global structure, RNN performance might not be much better than 1D CNN

- An RNN layer by itself doesn't take advantage of image/text structure!
    - For images: combine with convolution layer(s)
    - For text: combine with embedding layer

A Little Bit More Detail

Simple RNN: has trouble remembering things from long ago…

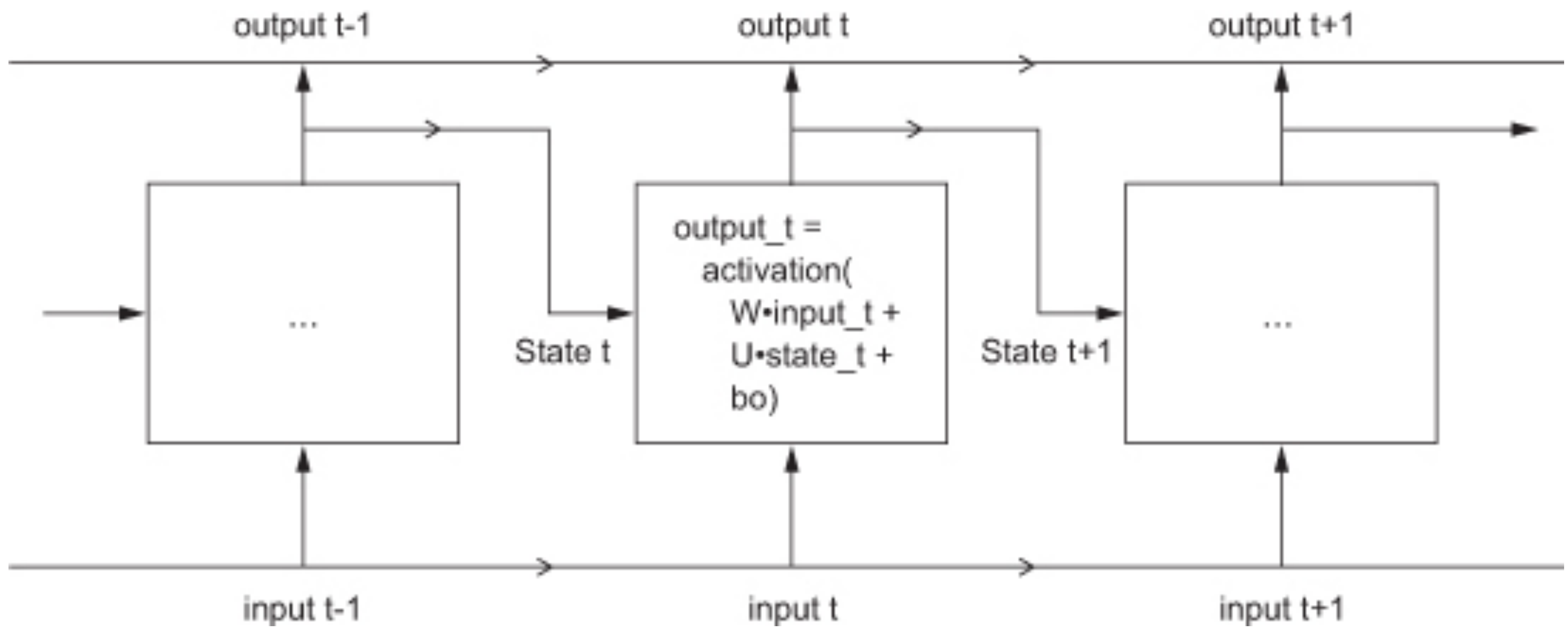


Figure 6.13 from Francois Chollet's book *Deep Learning with Python*

A Little Bit More Detail

Introduce a “carry” state for tracking longer term memory

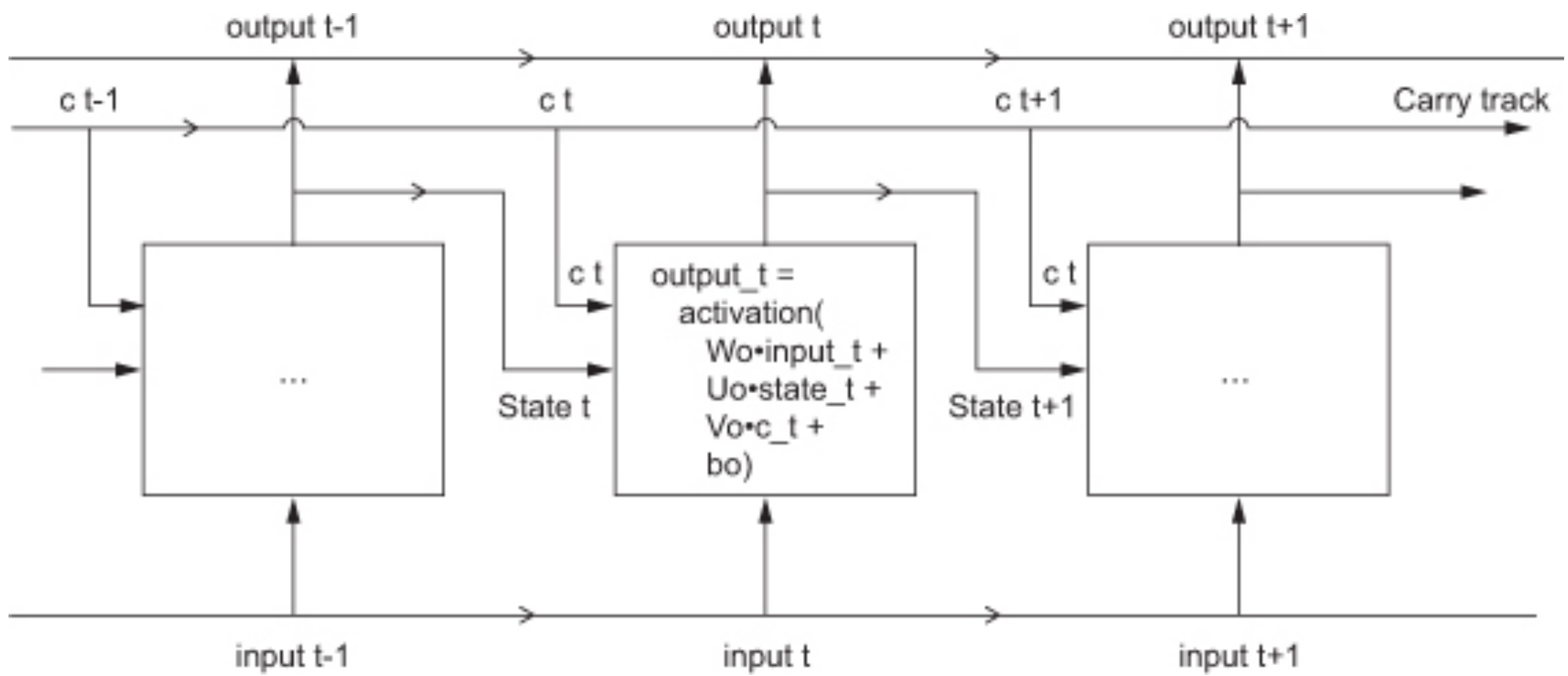


Figure 6.14 from Francois Chollet’s book *Deep Learning with Python*

A Little Bit More Detail

LSTM: figure out how to update "carry" state

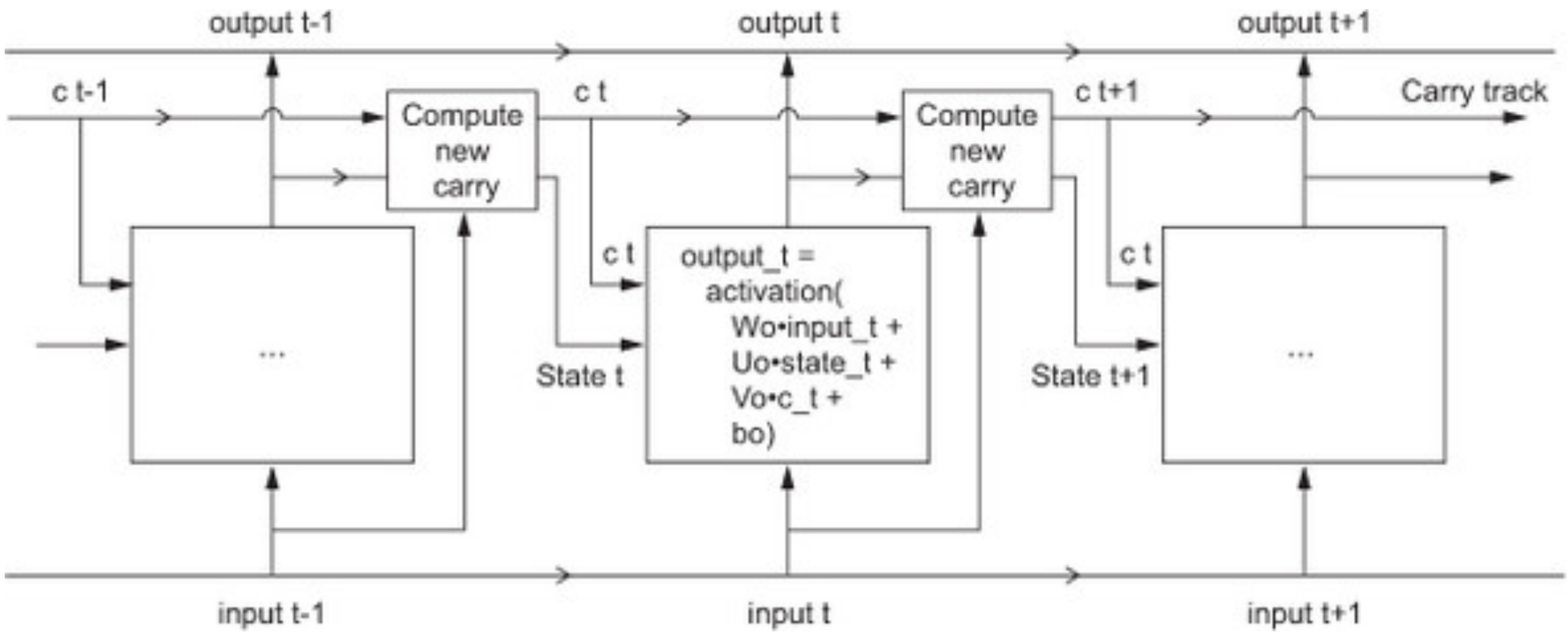


Figure 6.15 from Francois Chollet's book *Deep Learning with Python*

Learning a Deep Net

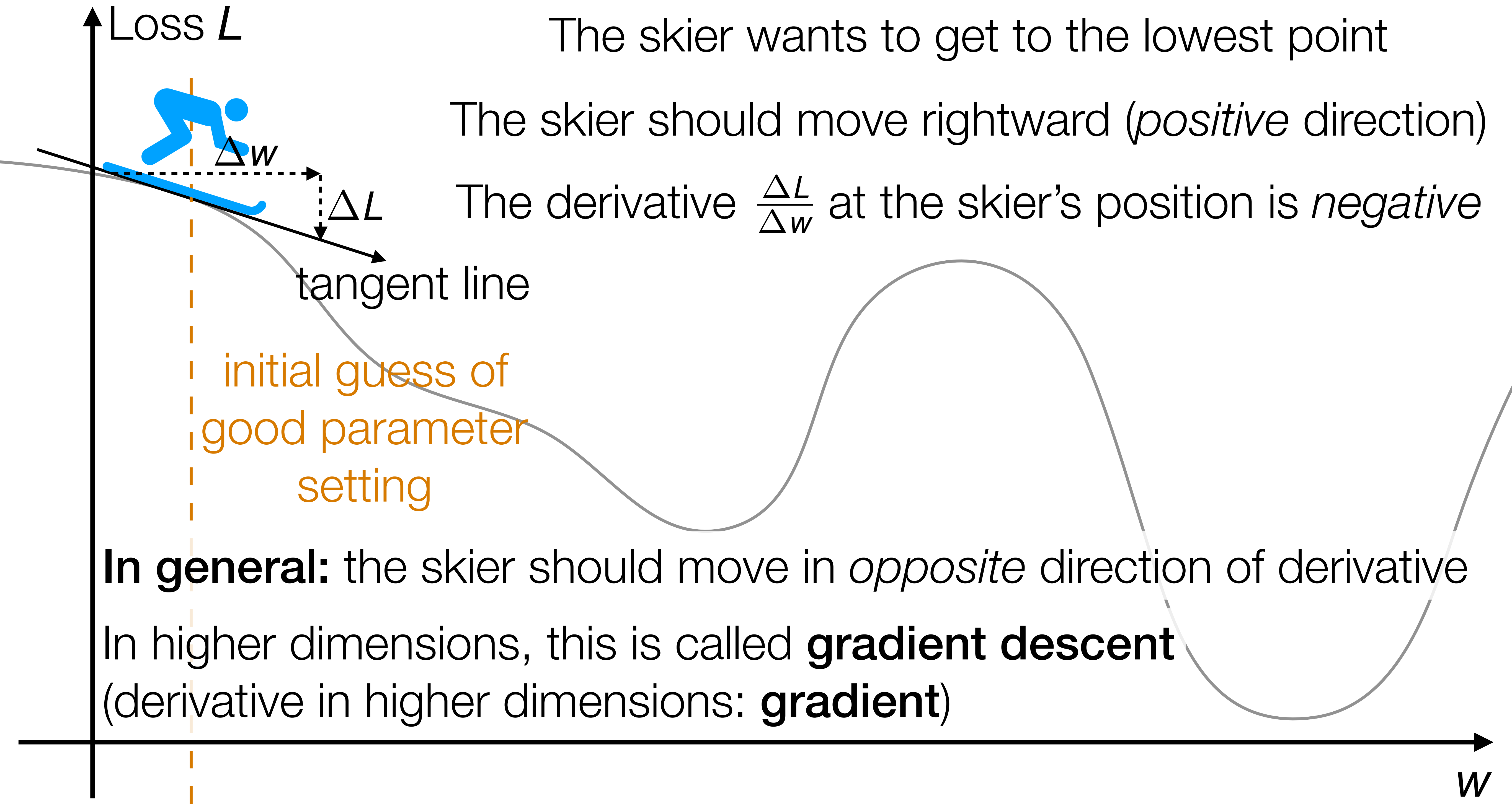
Gradient Descent

Suppose the neural network has a single real number parameter *w*

The skier wants to get to the lowest point

The skier should move rightward (*positive* direction)

The derivative $\frac{\Delta L}{\Delta w}$ at the skier's position is *negative*

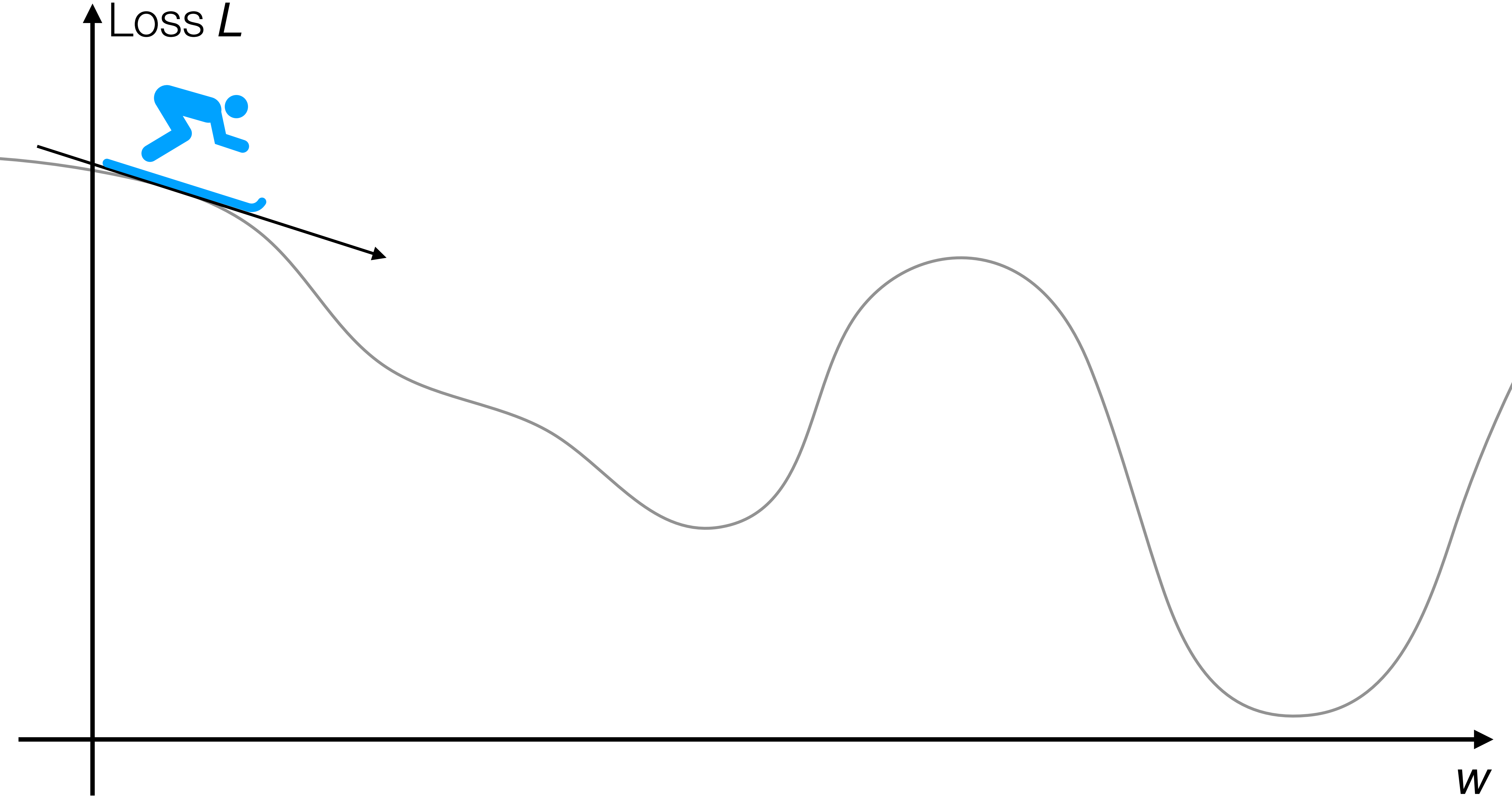


In general: the skier should move in *opposite* direction of derivative

In higher dimensions, this is called **gradient descent**
(derivative in higher dimensions: **gradient**)

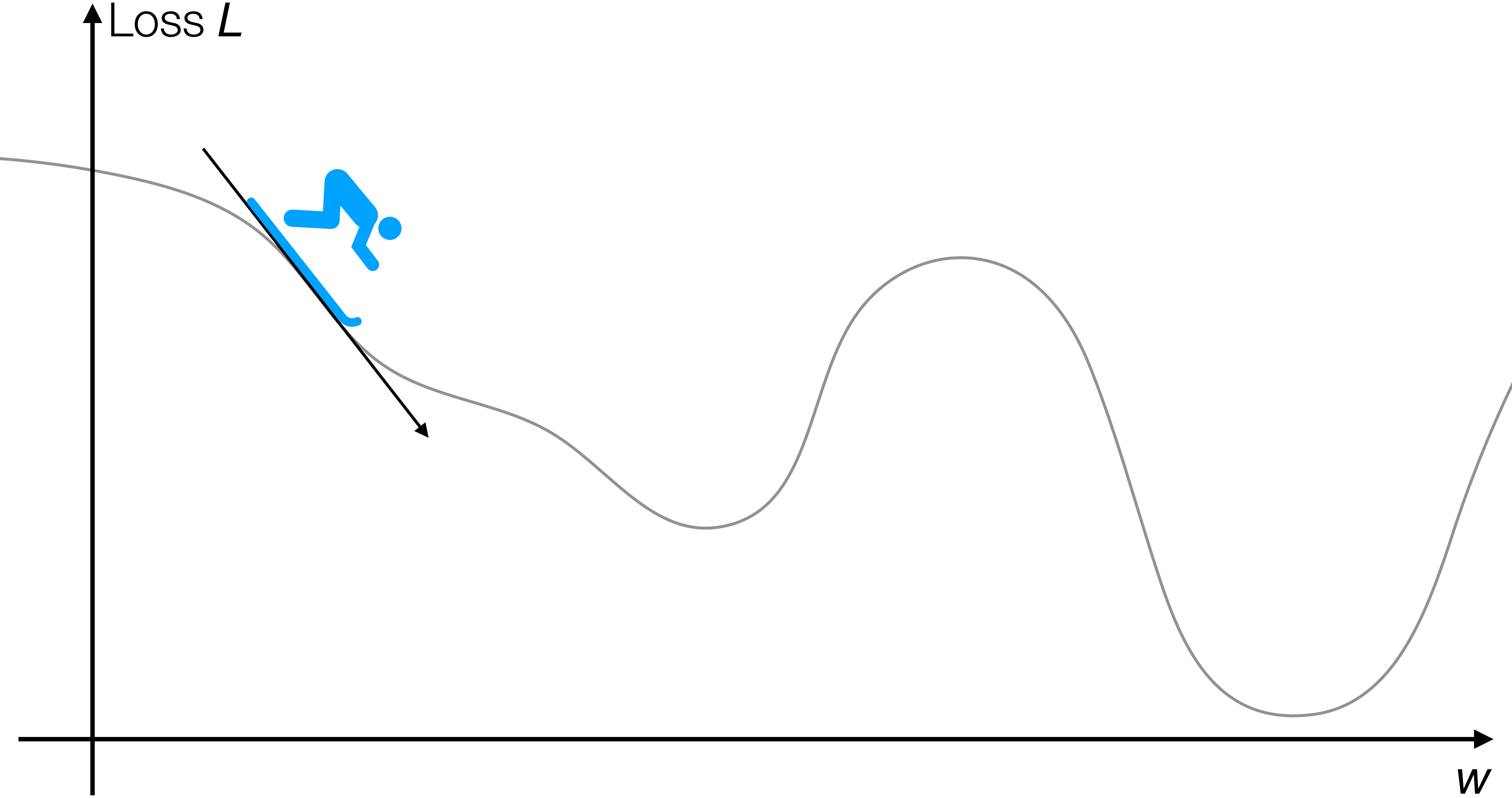
Gradient Descent

Suppose the neural network has a single real number parameter w



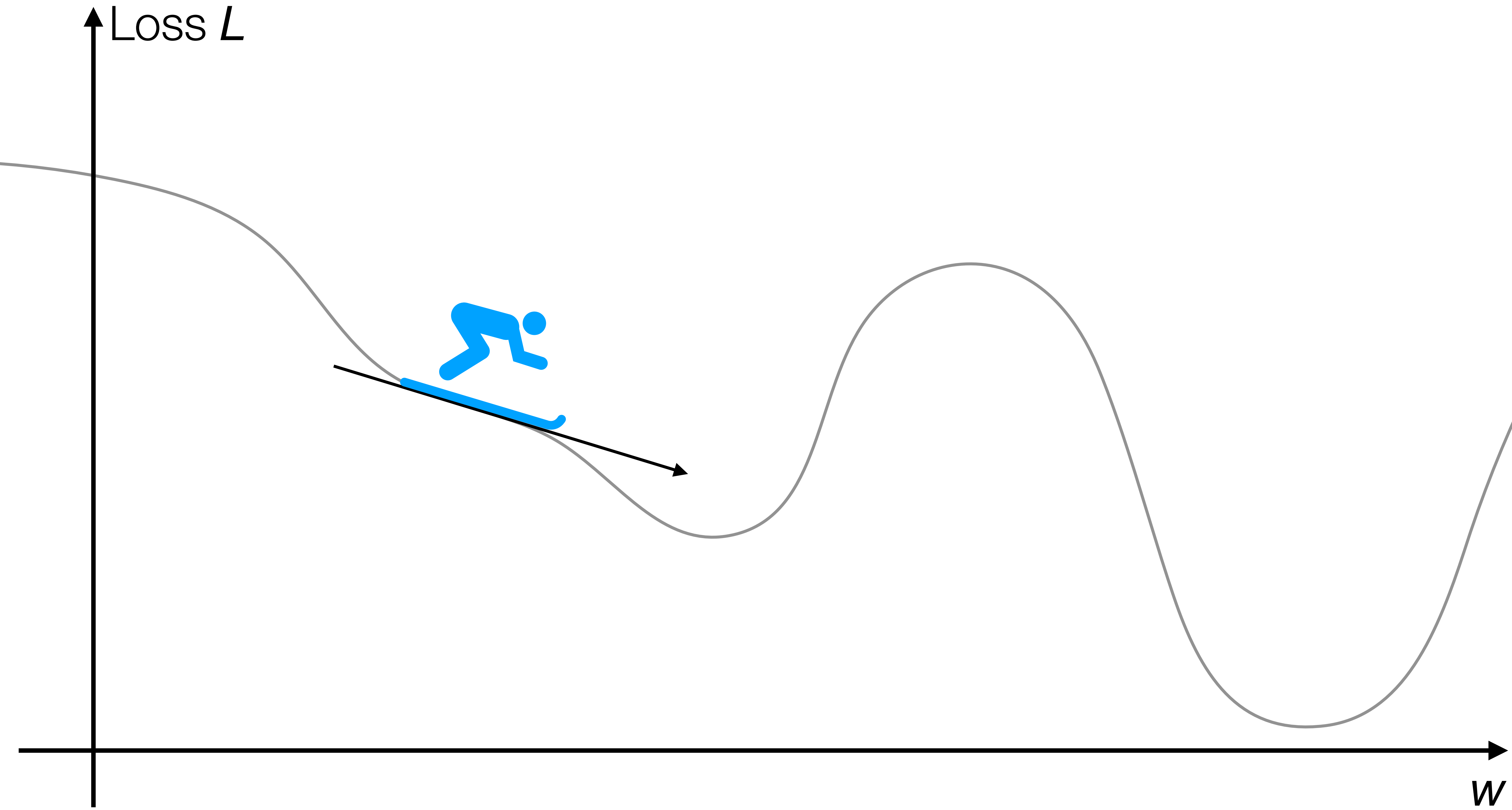
Gradient Descent

Suppose the neural network has a single real number parameter *w*



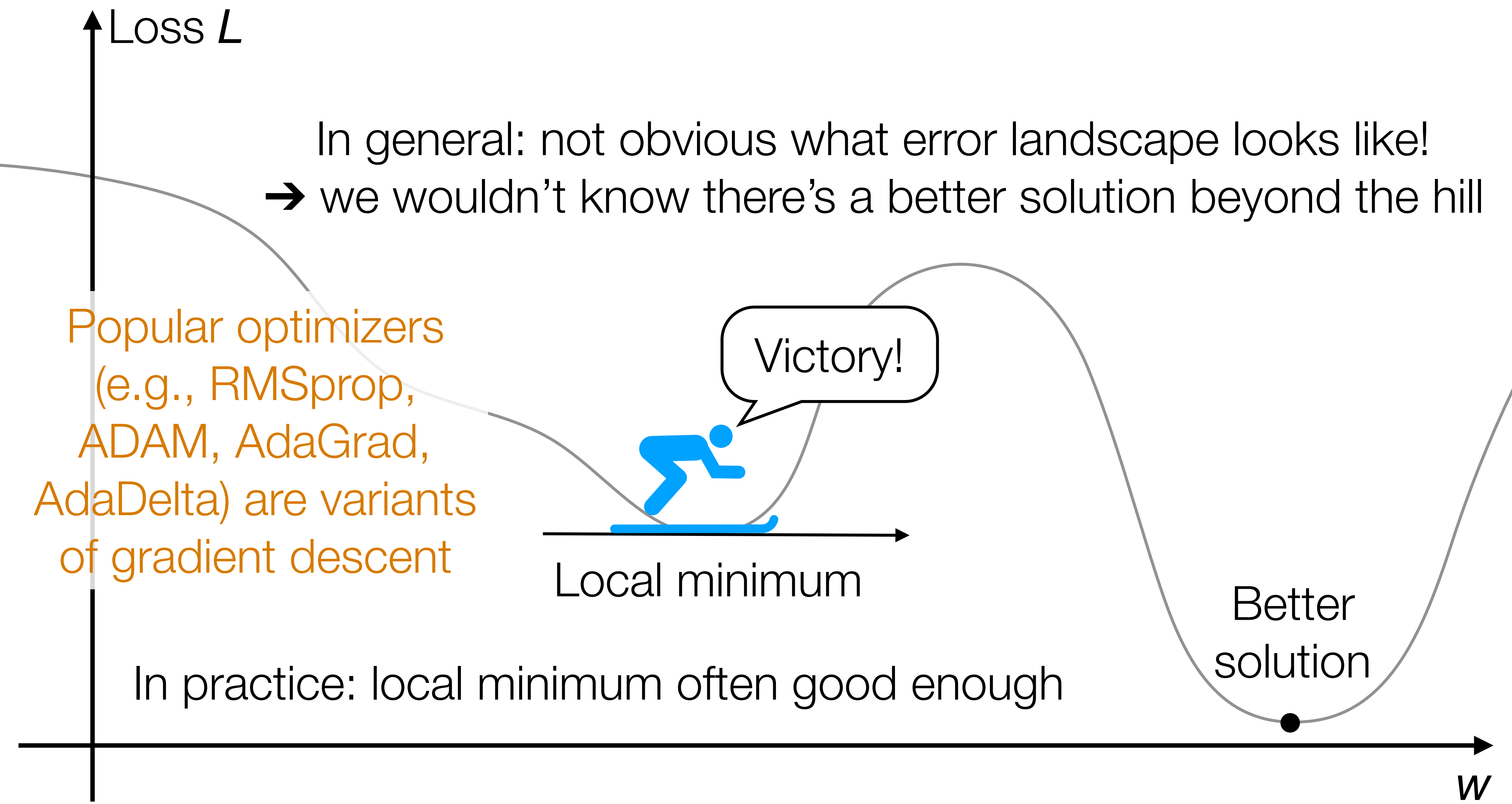
Gradient Descent

Suppose the neural network has a single real number parameter w



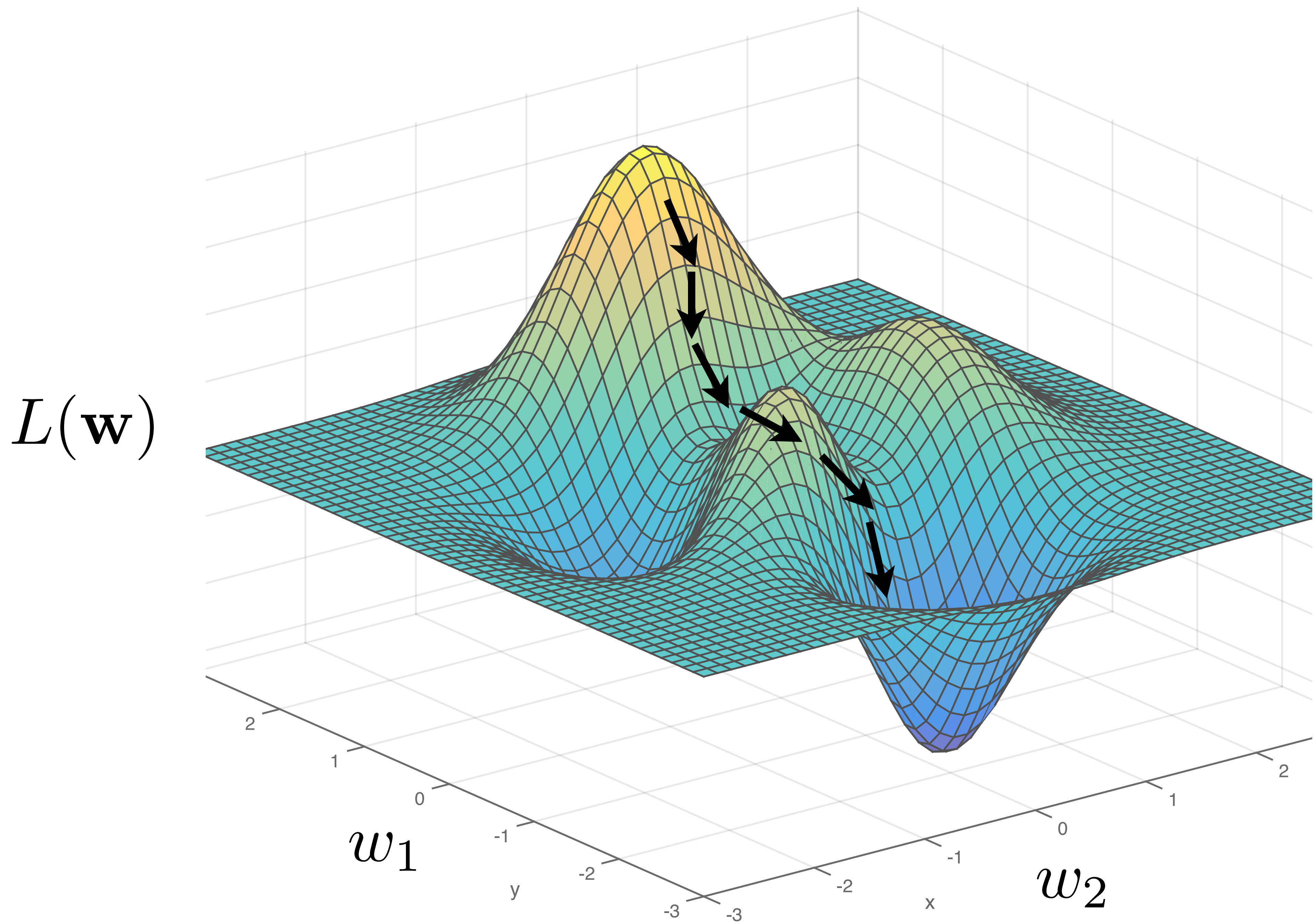
Gradient Descent

Suppose the neural network has a single real number parameter *w*



Gradient Descent

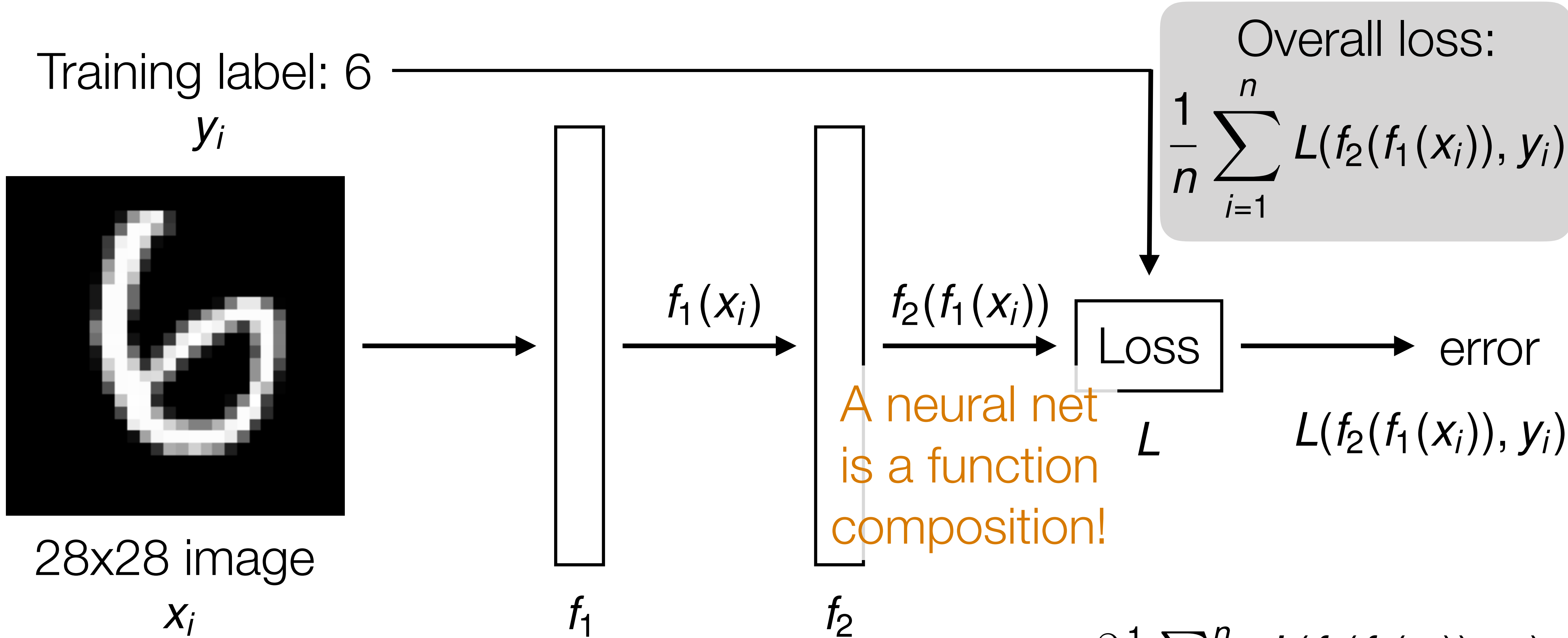
2D example



Slide by Phillip Isola

Remark: In practice, deep nets often have > *millions* of parameters, so *very* high-dimensional gradient descent

Handwritten Digit Recognition



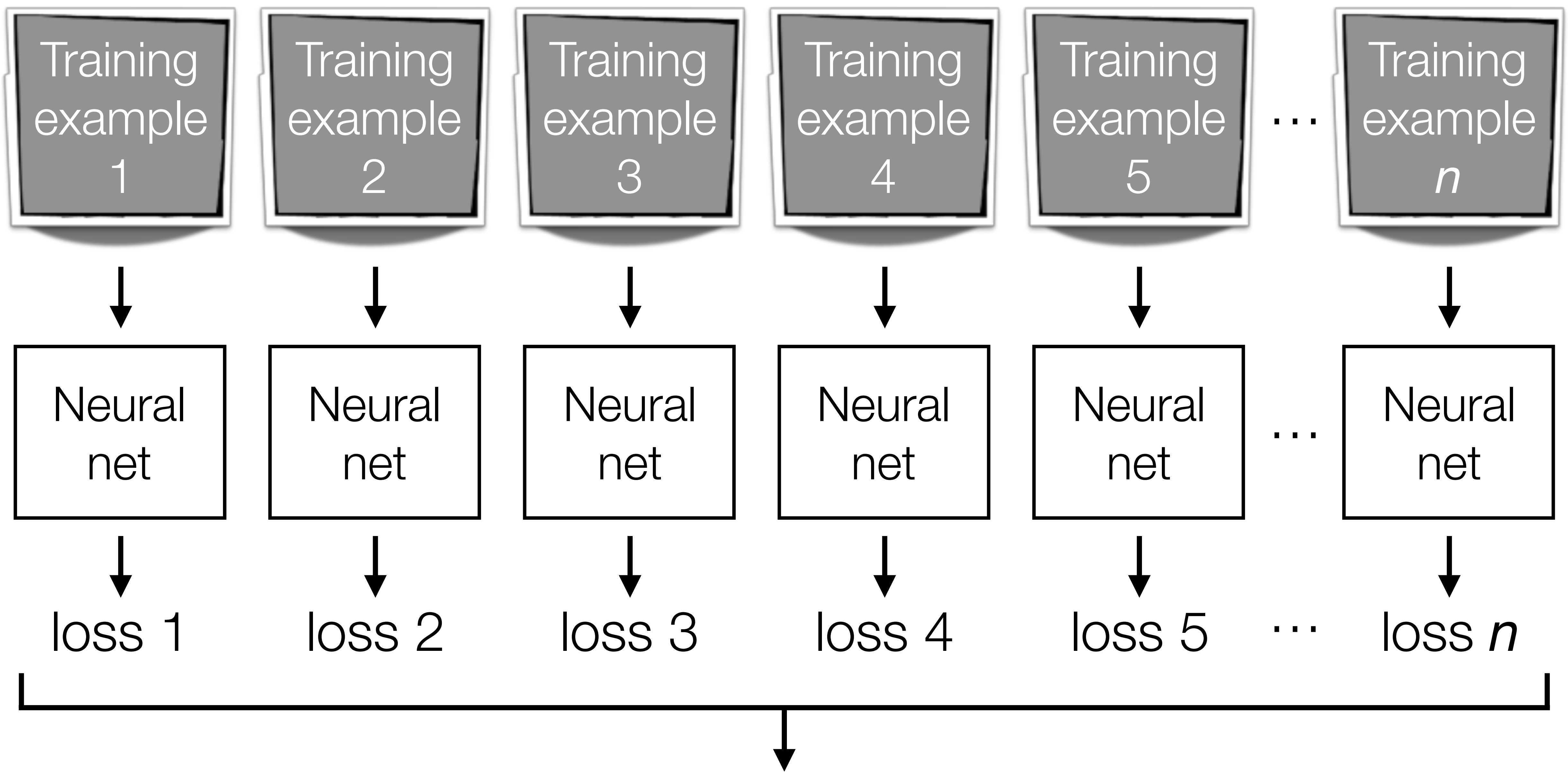
All parameters: θ

Gradient: $\dfrac{\partial \frac{1}{n}\sum_{i=1}^{n} L(f_2(f_1(x_i)), y_i)}{\partial \theta}$

Automatic differentiation is crucial in learning deep nets!

Careful derivative chain rule calculation: **back-propagation**

Gradient Descent



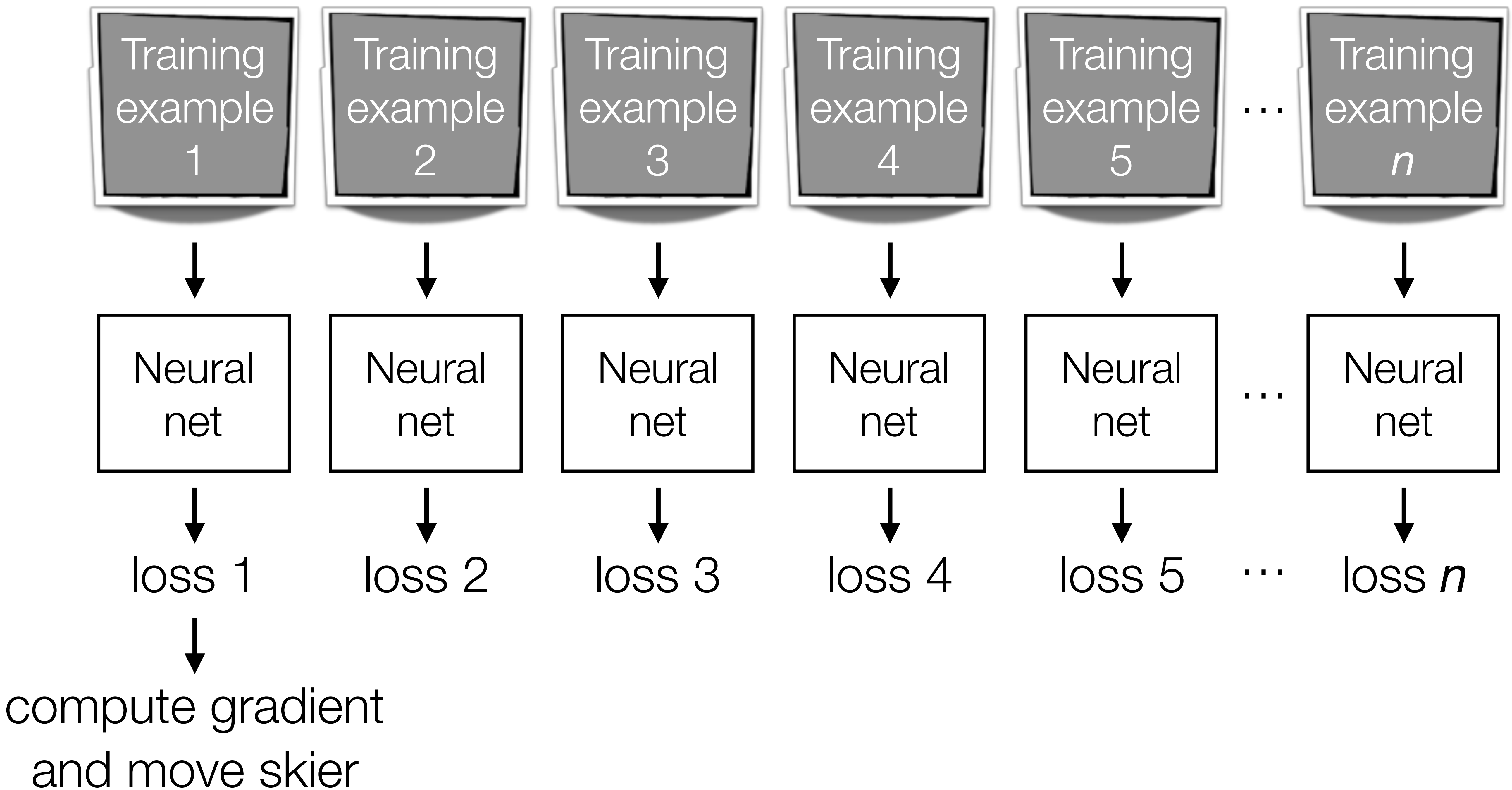
average loss

compute gradient and move skier

We have to compute lots of gradients to help the skier know where to go!

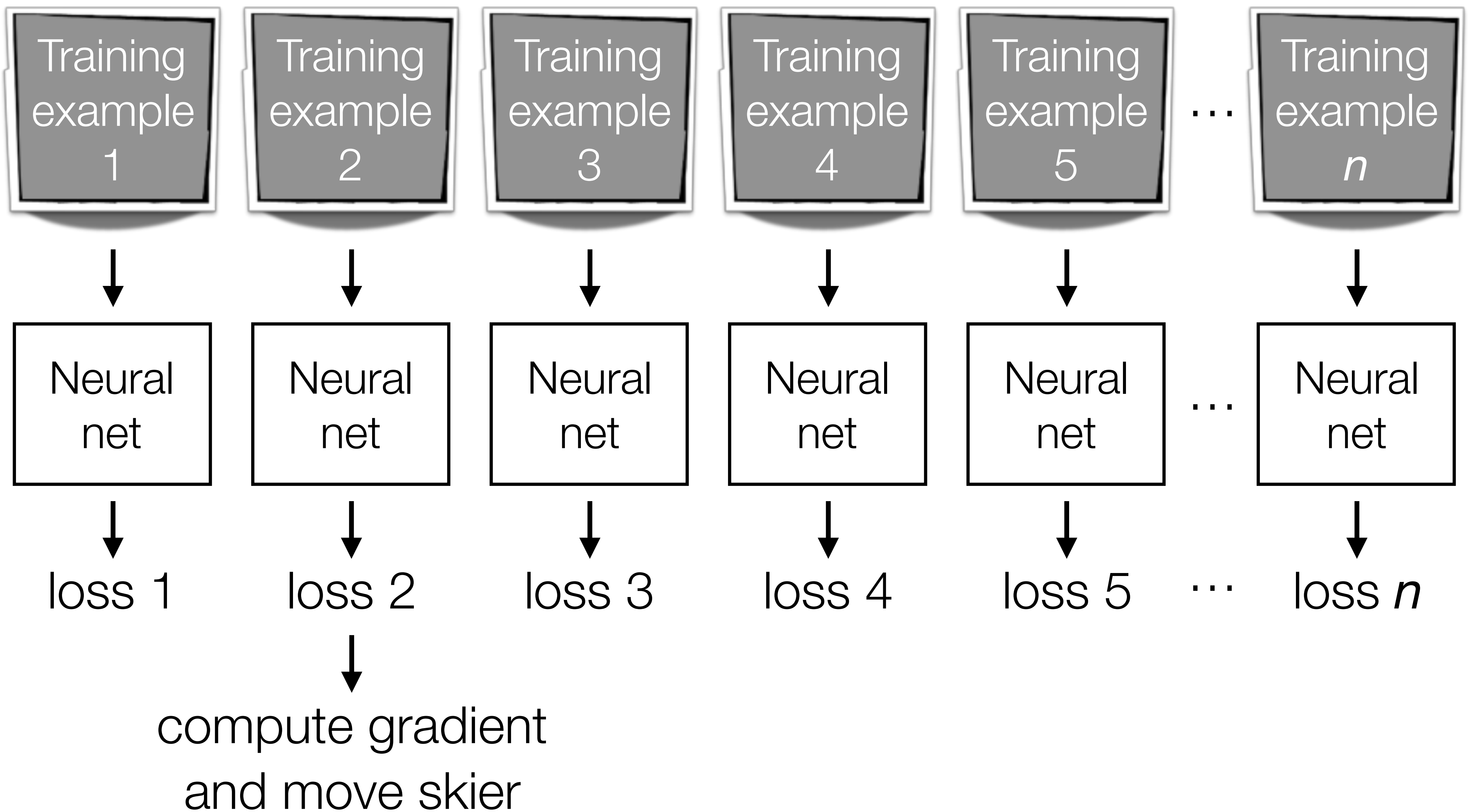
Computing gradients using all the training data seems really expensive!

Stochastic Gradient Descent (SGD)



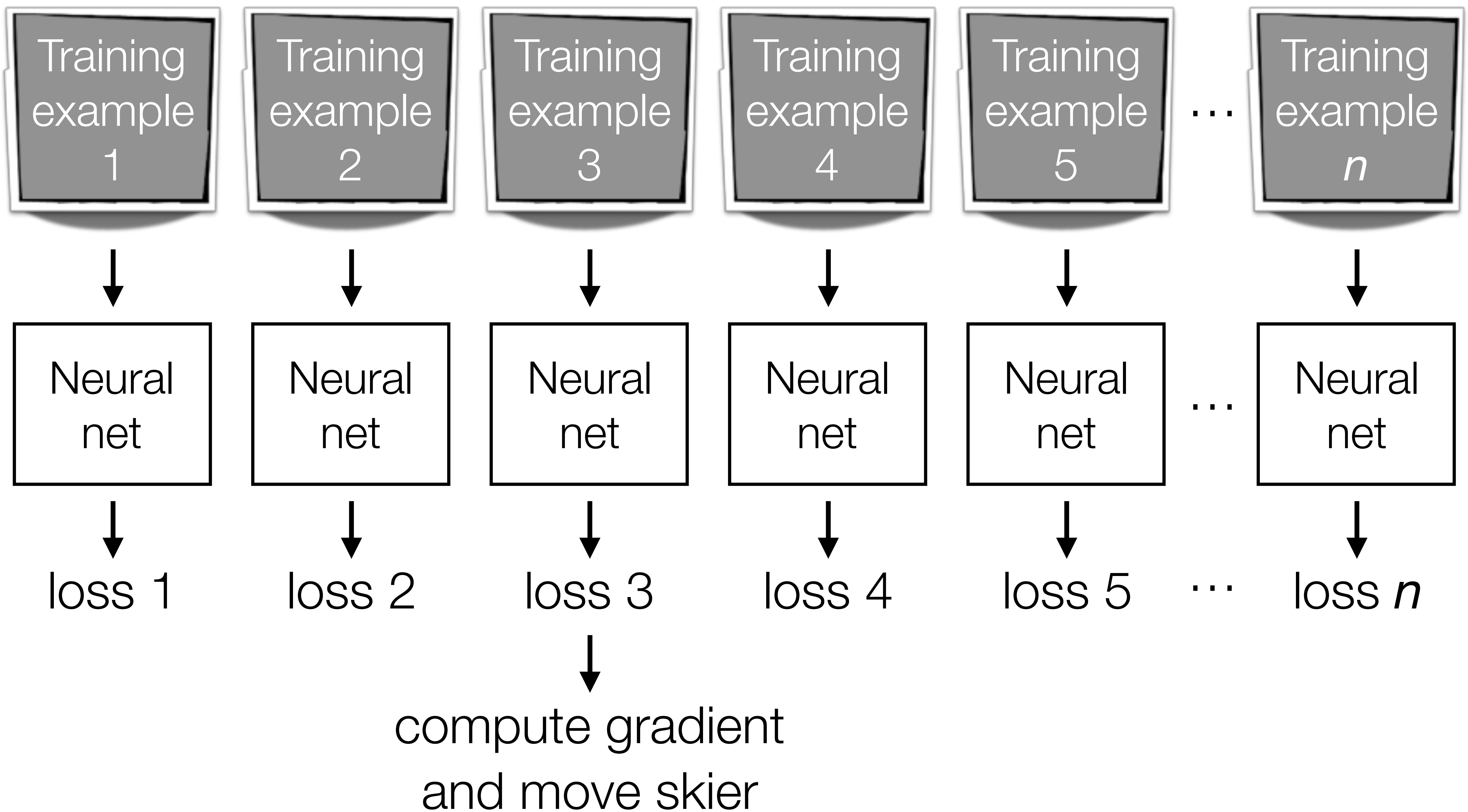
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the “full” gradient)

Stochastic Gradient Descent (SGD)



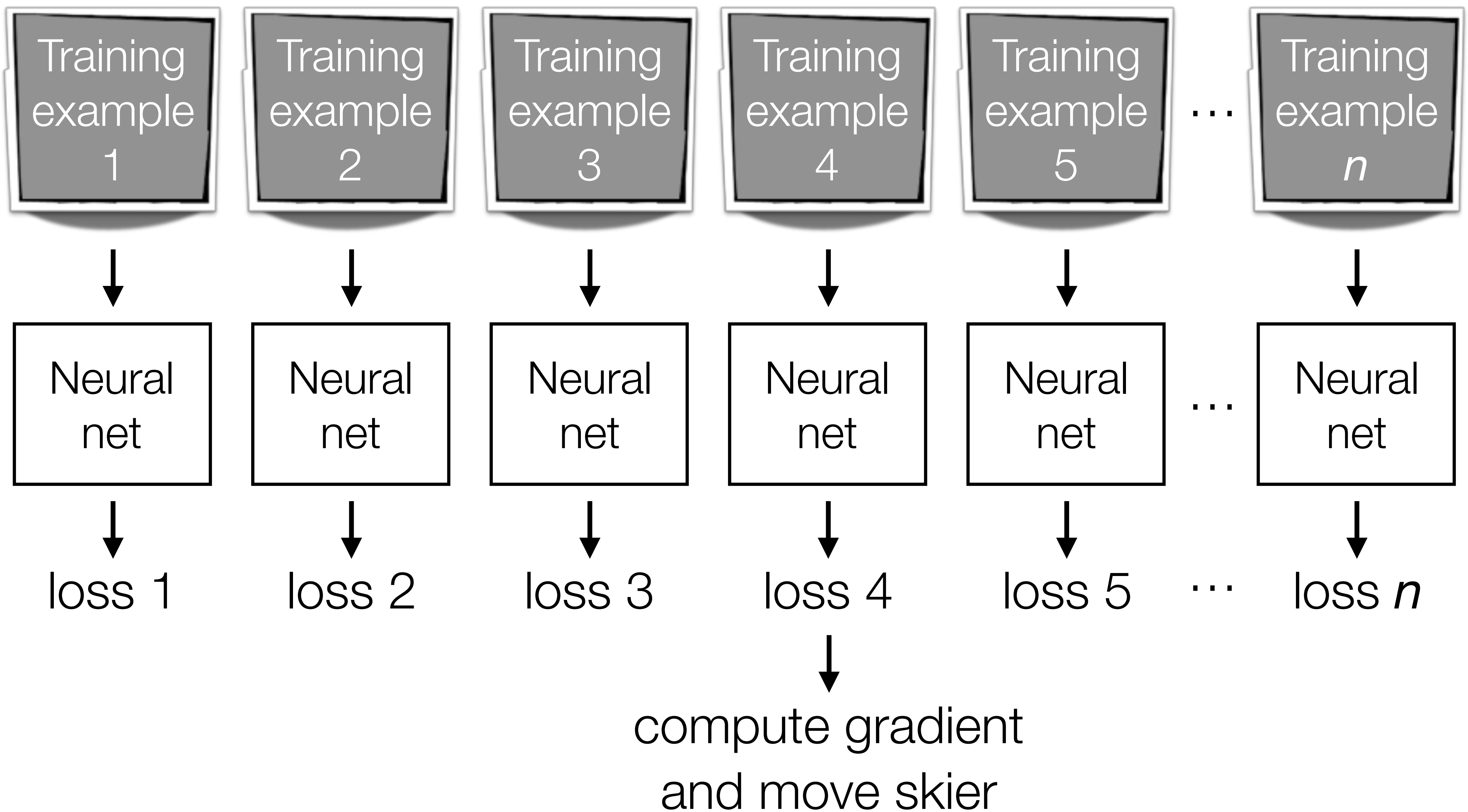
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the “full” gradient)

Stochastic Gradient Descent (SGD)



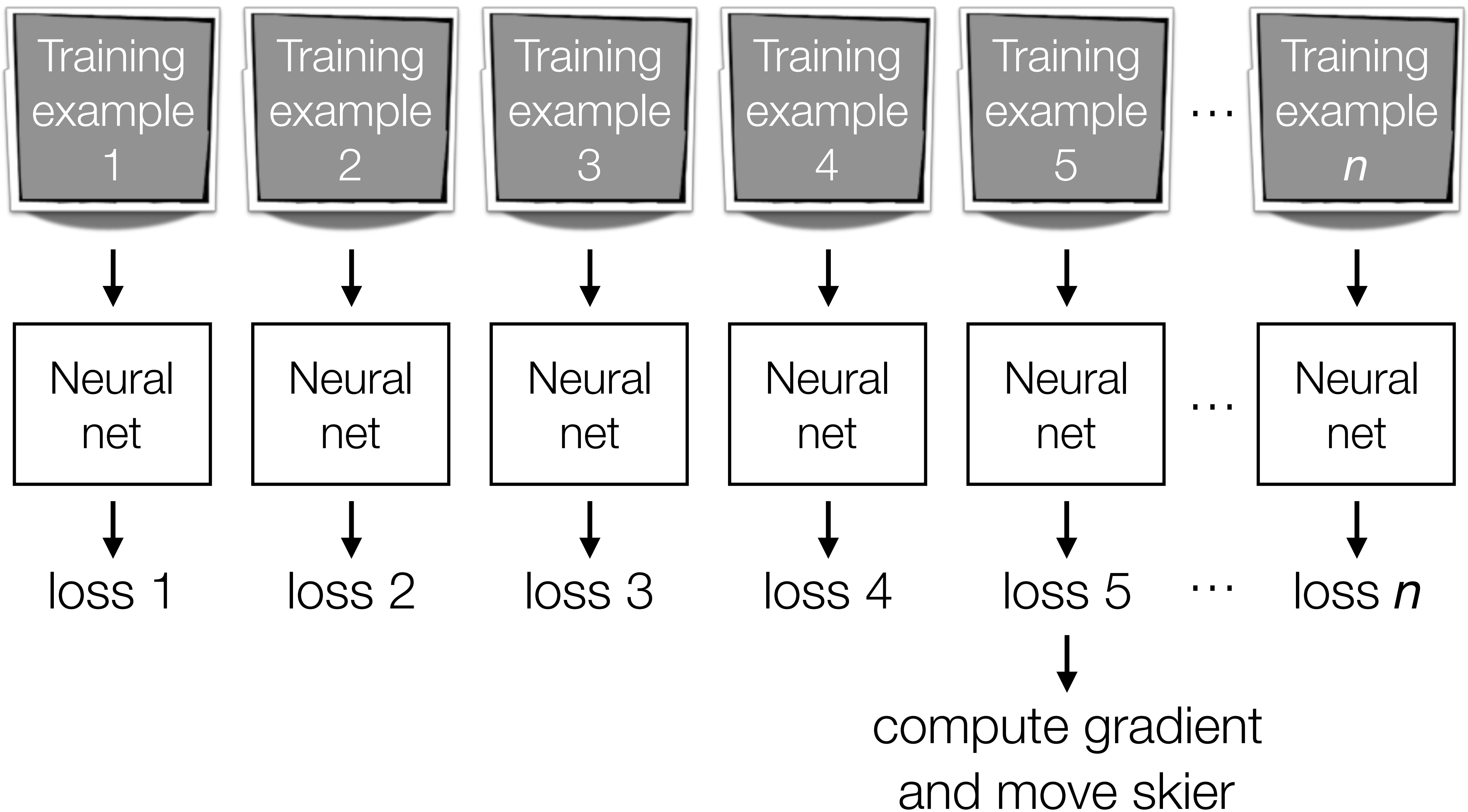
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



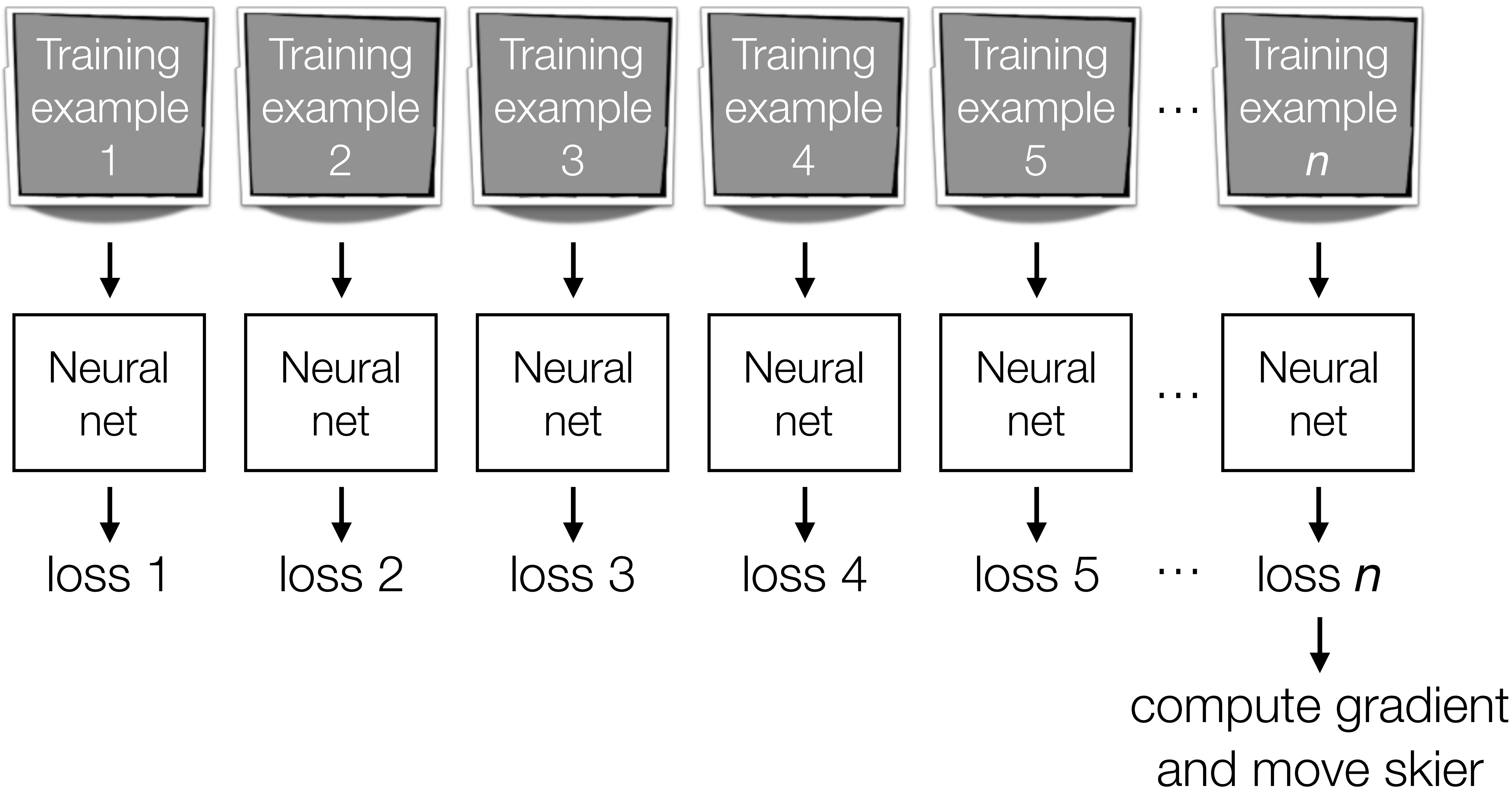
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the “full” gradient)

Stochastic Gradient Descent (SGD)



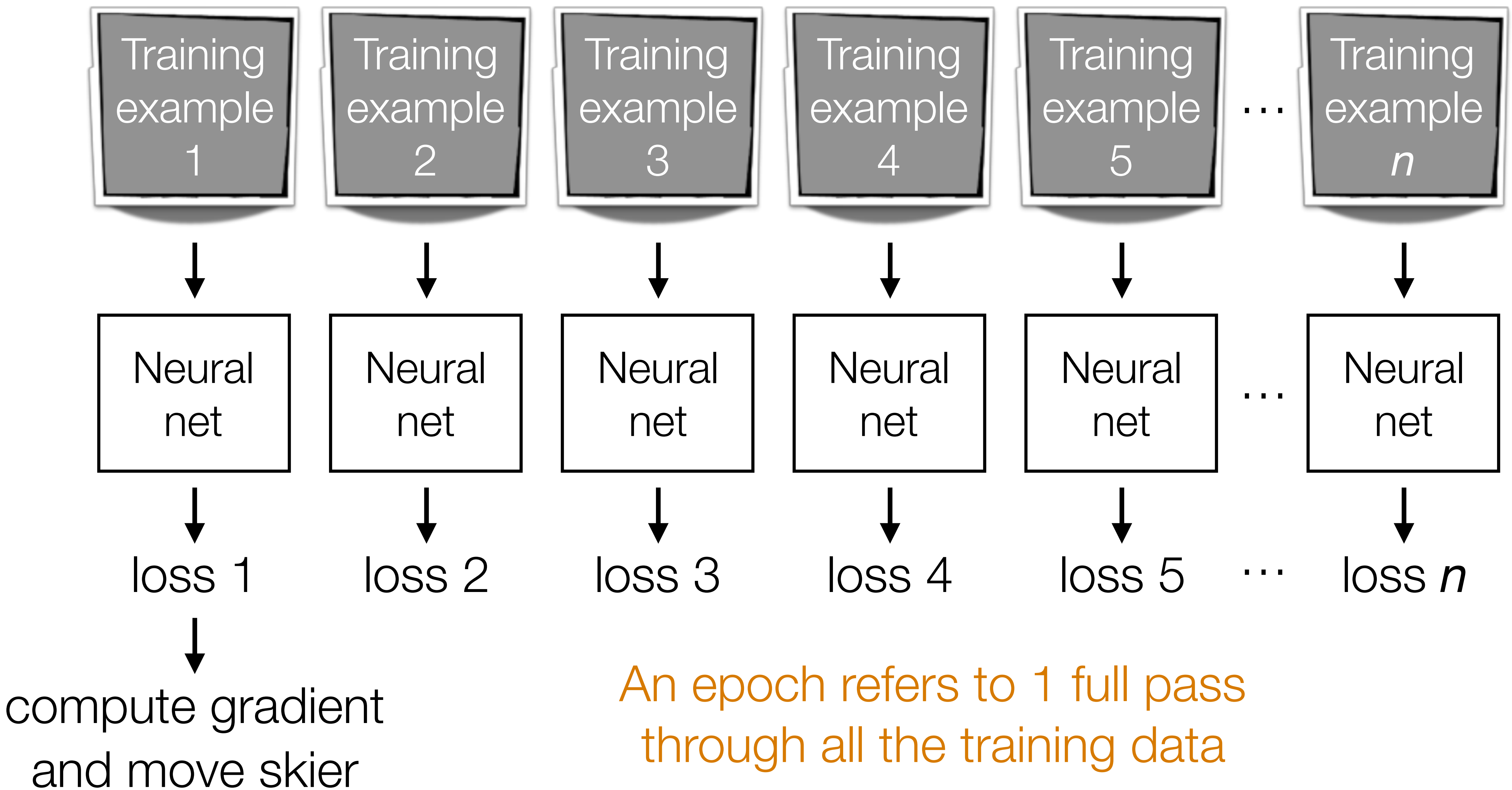
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

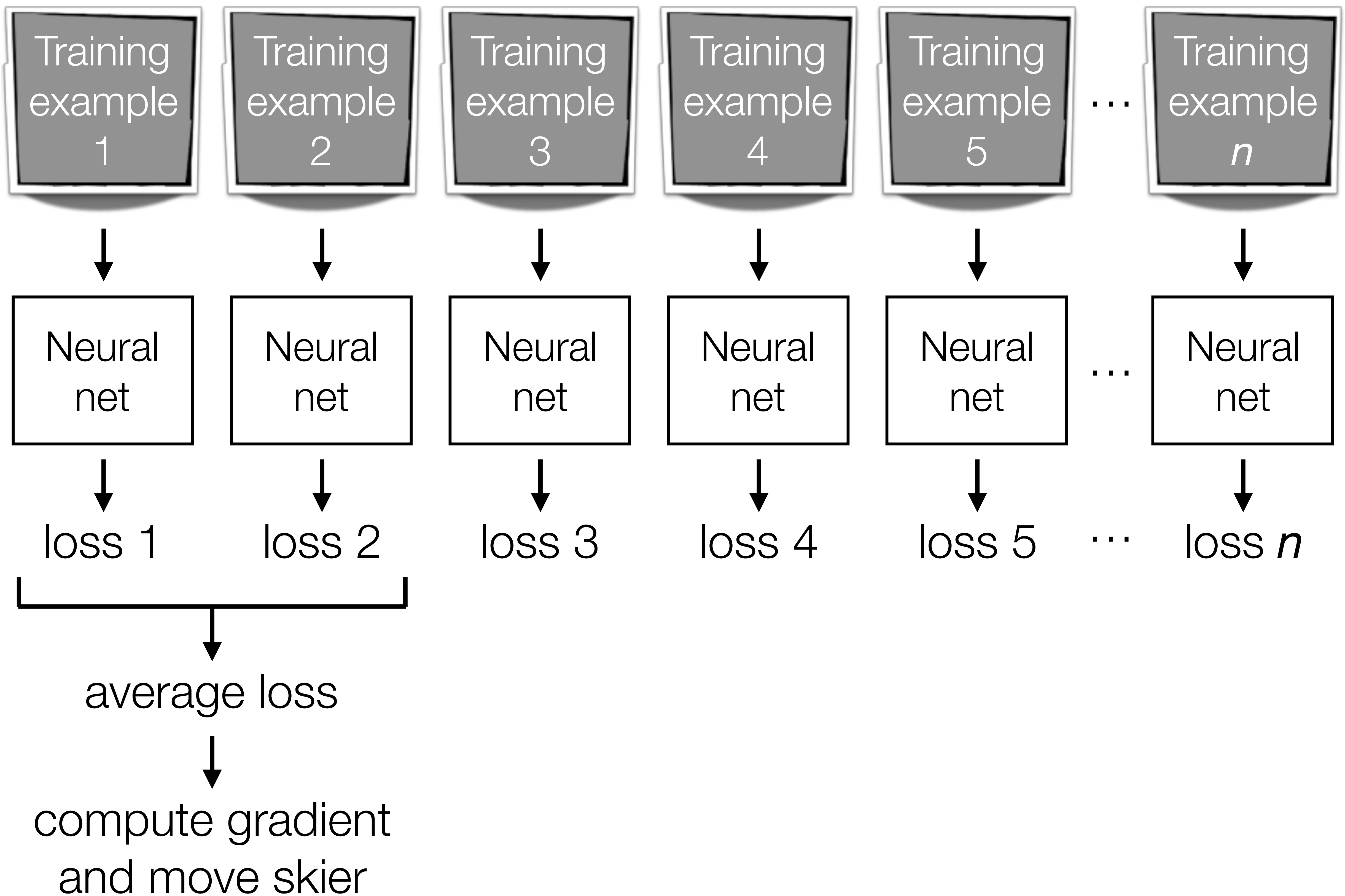
Stochastic Gradient Descent (SGD)



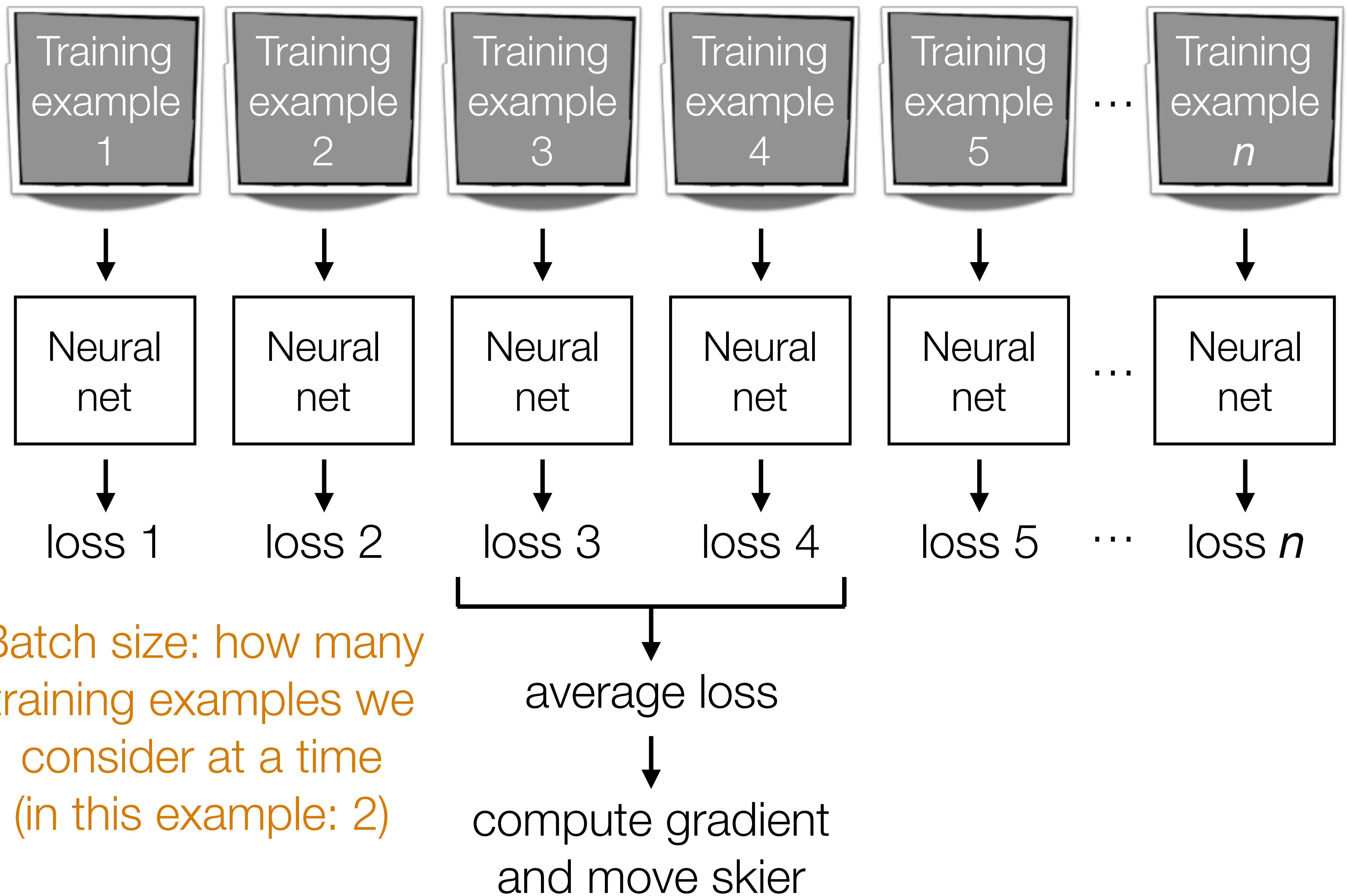
An epoch refers to 1 full pass through all the training data

SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the “full” gradient)

Mini-Batch Gradient Descent



Mini-Batch Gradient Descent



Batch size: how many training examples we consider at a time (in this example: 2)

Best variant of SGD to use?
Best # of epochs? Best batch size?

Active area of research

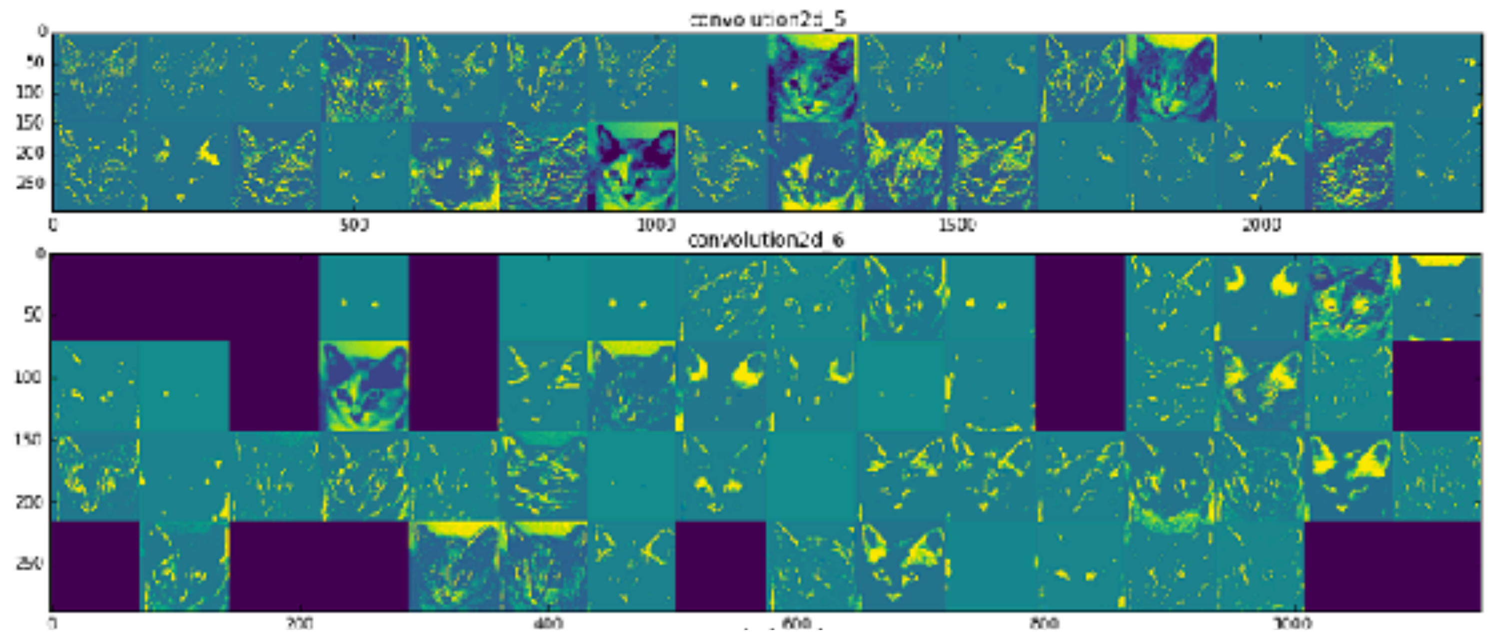
Depends on problem, data, hardware, etc

Example: even with a GPU, you can get slow learning (slower than CPU!) if you choose # epochs/batch size poorly!!!

There's a lot more to deep learning that we didn't cover

Visualizing What a Deep Net Learned

- Very straight-forward for CNNs
  - Plot filter outputs at different layers



  - Plot regions that maximally activate an output neuron

Images: Francois Chollet's "Deep Learning with Python" Chapter 5

Example: Wolves vs Huskies

(a) Husky classified as wolf (b) Explanation

Turns out the deep net learned that wolves are wolves because of snow…

➔ visualization is crucial!

Source: Ribeiro et al. "Why should I trust you? Explaining the predictions of any classifier." KDD 2016.

Dealing with Small Datasets

Data augmentation: generate perturbed versions of your training data to get larger training dataset

Training image
Training label: cat

Mirrored
Still a cat!

Rotated & translated
Still a cat!

We just turned 1 training example in 3 training examples

Allowable perturbations depend on data (e.g., for handwritten digits, rotating by 180 degrees would be bad: confuse 6's and 9's)

Dealing with Small Datasets

Fine tuning: if there's an existing pre-trained neural net, you could modify it for your problem that has a small dataset

Example: classify between Tesla's and Toyota's

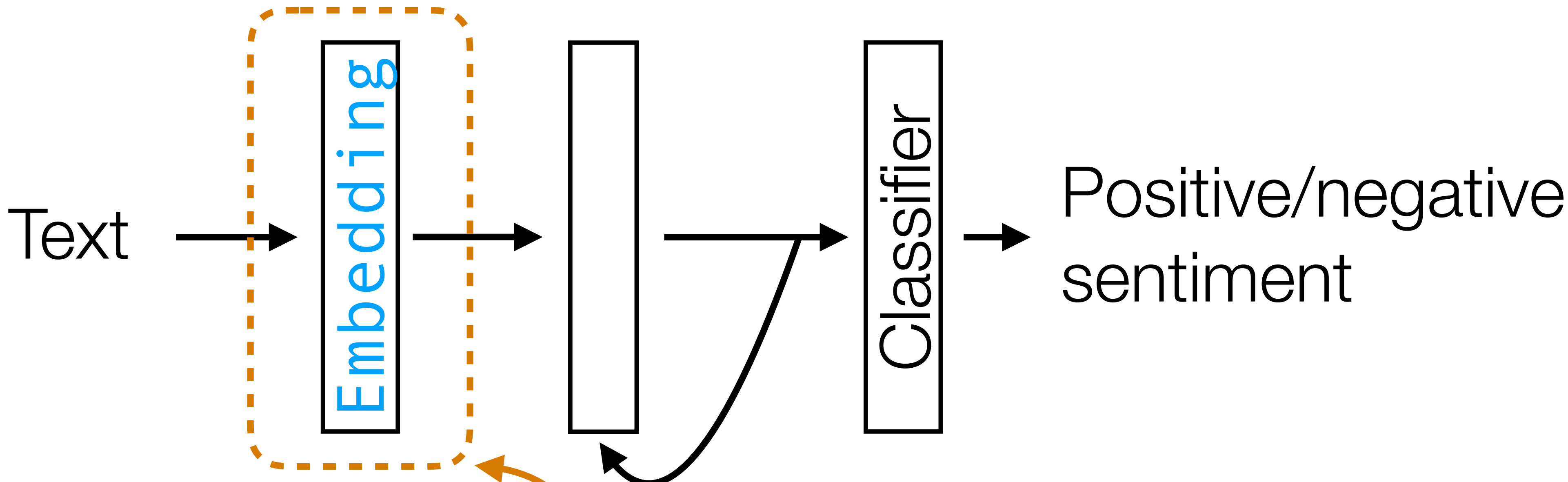
You collect photos from the internet of both, but your dataset size is small, on the order of 1000 images

Strategy: take existing pre-trained CNN for ImageNet classification and change final layer to do classification between Tesla's and Toyota's rather than classifying into 1000 objects

Dealing with Small Datasets

Fine tuning: if there's an existing pre-trained neural net, you could modify it for your problem that has a small dataset

Example: sentiment analysis RNN demo



We fixed the weights here to come from GloVe and disabled training for this layer!

GloVe vectors pre-trained on massive dataset (Wikipedia + Gigaword)

IMDb review dataset is small in comparison

Self-Supervised Learning

Even without labels, we can set up a prediction task!

Hide part of training data and try to predict what you've hid!

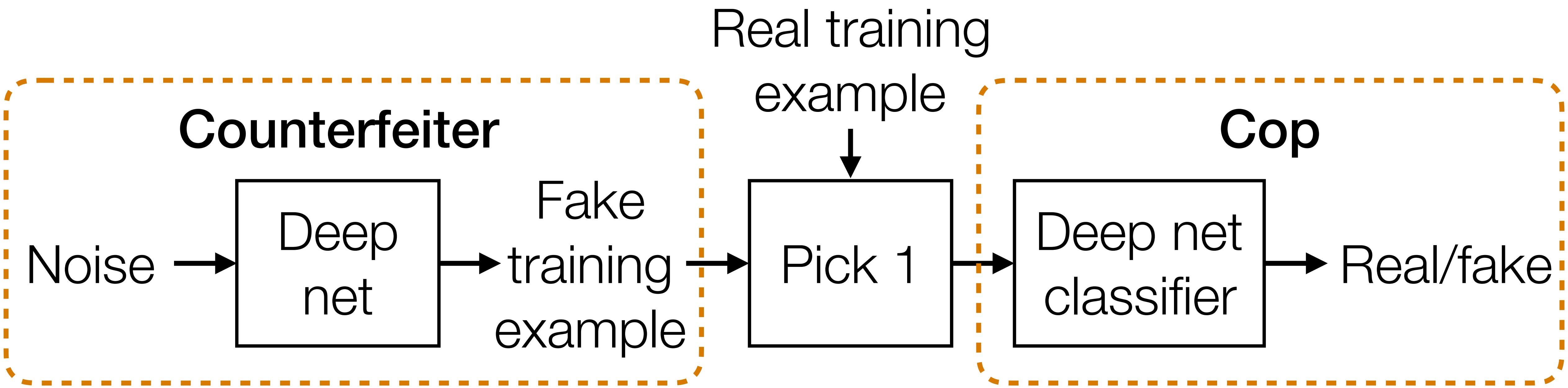
Example: word embeddings like word2vec, GloVe

Word embeddings will be covered in your next recitation
(it's a clever application of predictive data analytics concepts)

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

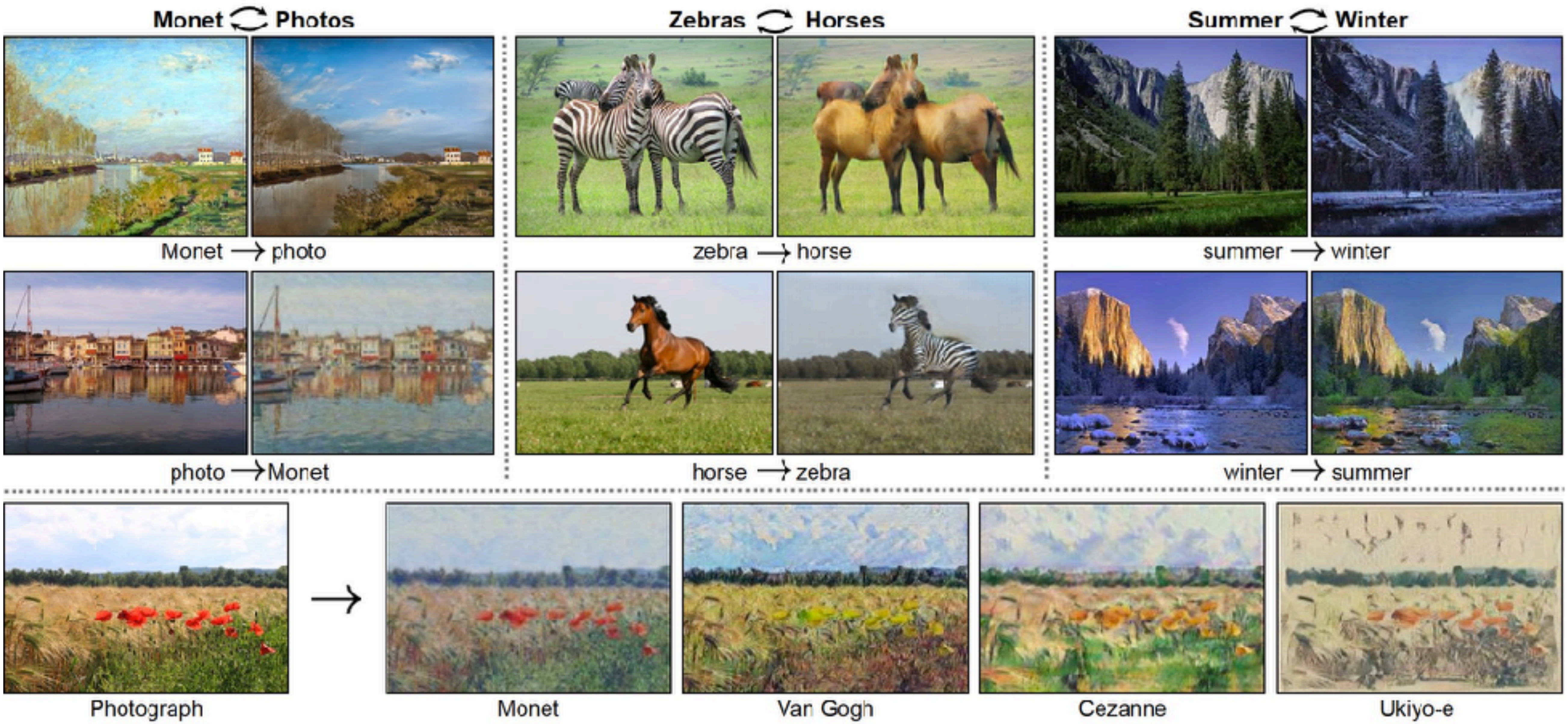
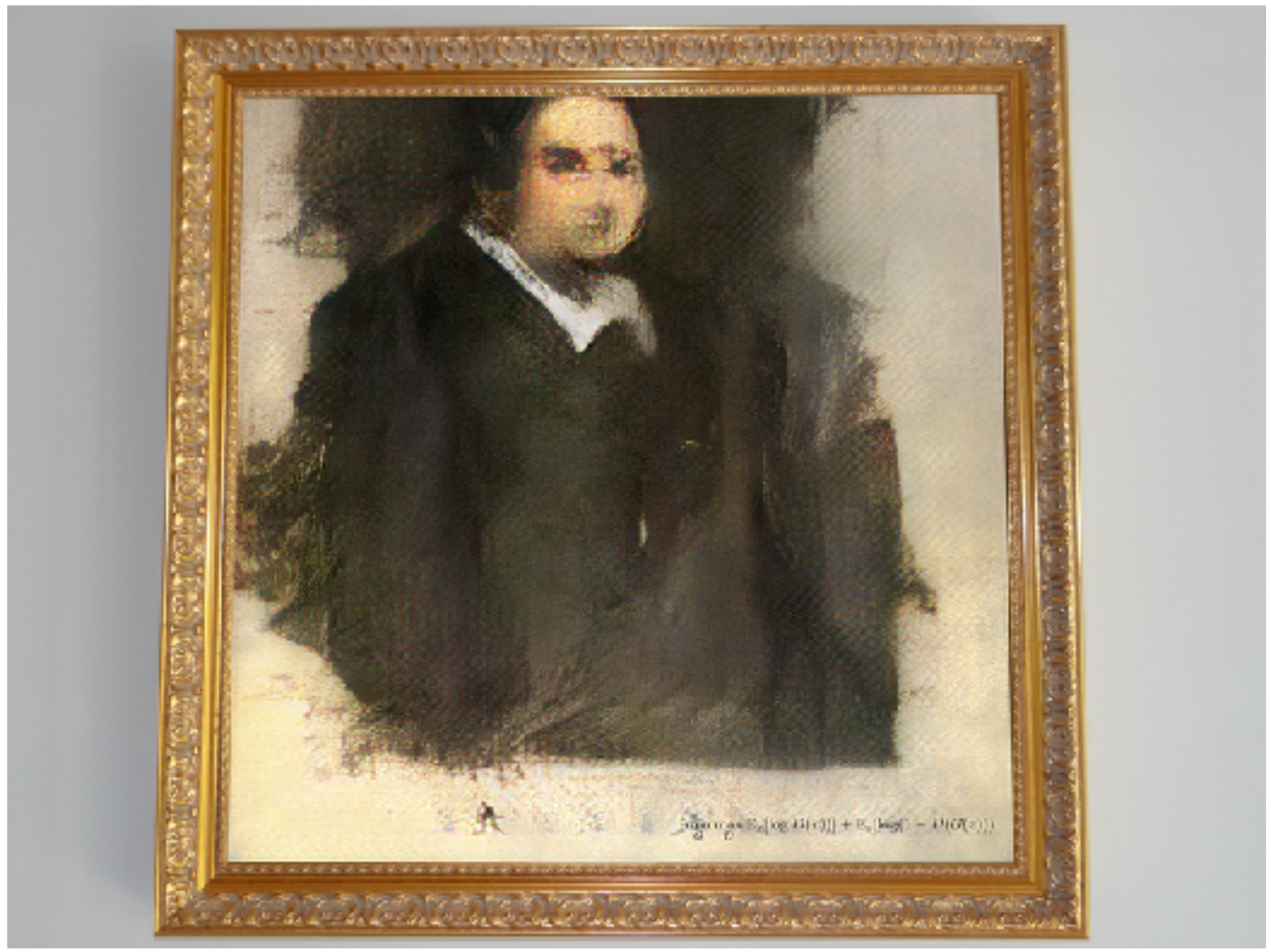


Image-to-image translation results from UC Berkeley using GANs (Isola et al 2017, Zhu et al 2017)

Generate Fake Art



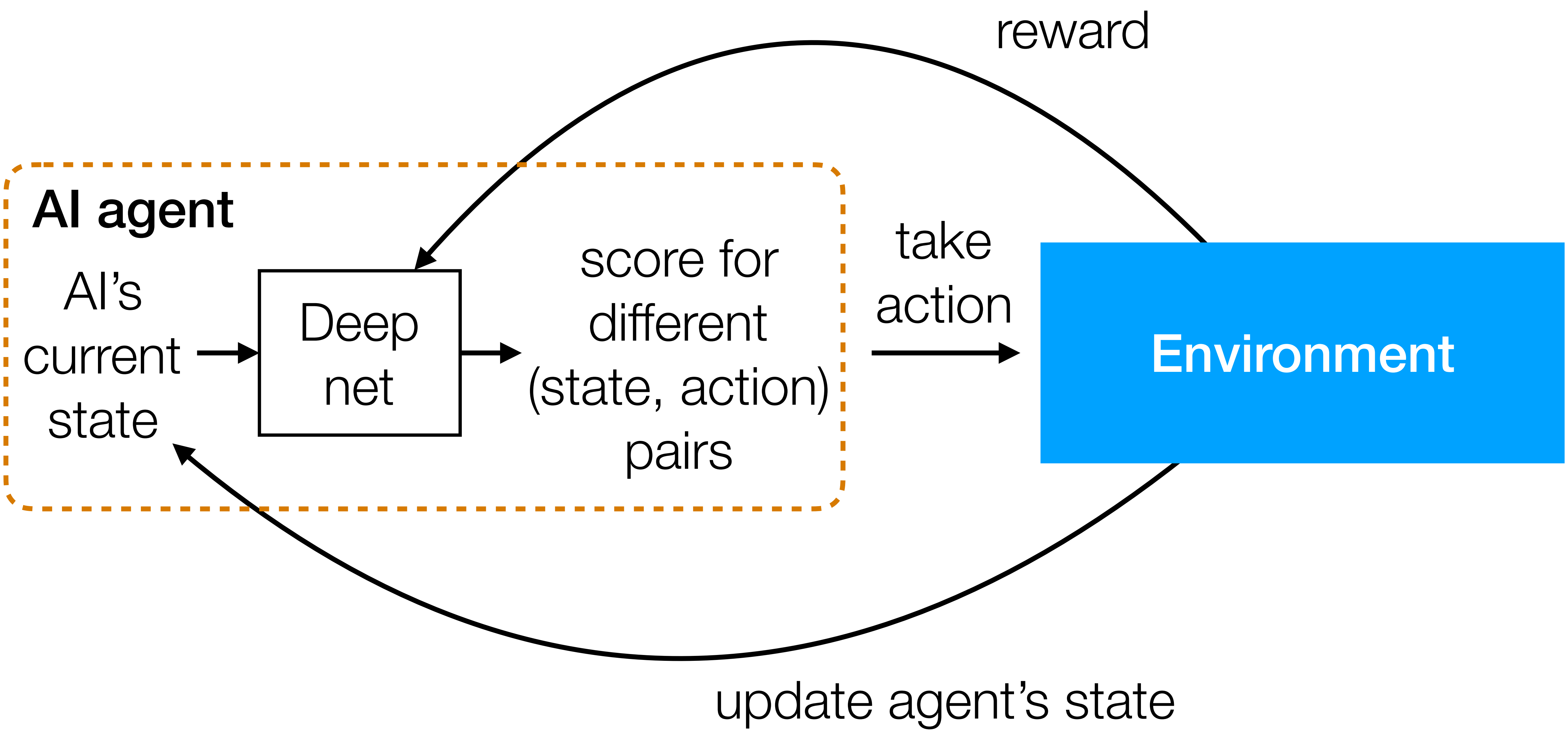
October 2018: estimated to go for $7,000-$10,000

10/25/2018: Sold for $432,500

Source: https://www.npr.org/2018/10/22/659680894/a-i-produced-portrait-will-go-up-for-auction-at-christie-s

Deep Reinforcement Learning

The machinery behind AlphaGo and similar systems

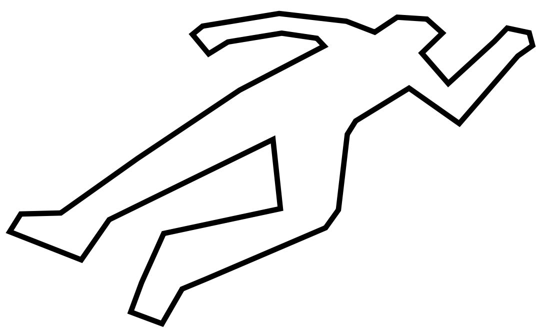


The Future of Deep Learning

- Deep learning currently is still very limited in what it can do — the layers do simple operations and have to be differentiable
    - Adversarial examples at test time remain a problem
    - Basically just doing an elaborate function approximation (curve fitting)
    - The resulting learned function is comprised of a series of basic operations, possibly with a `for` loop (for RNN's)
- Still lots of engineering and expert knowledge used to design some of the best systems (e.g., AlphaGo)
    - How do we get away with using less expert knowledge?
- How do we do lifelong learning?

Unstructured Data Analysis

Question



The dead body

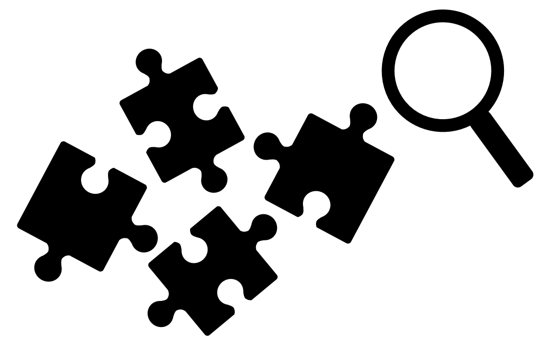
This is provided by a practitioner

Data

The evidence

Some times you have to collect more evidence!

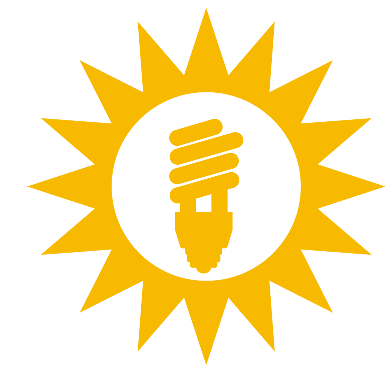
Finding Structure



Puzzle solving, careful analysis

Exploratory data analysis

Insights



When? Where? Why? How? Perpetrator catchable?

Answer original question

<u>There isn't always a follow-up prediction problem to solve</u>

95-865 Some Parting Thoughts

- Remember to **visualize steps of your data analysis pipeline**
  - Helpful for both debugging and interpreting outputs
- Very often there are *tons* of models/design choices to try
  - 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)
  - Set up self-supervised learning task
- There is a *lot* we did not cover — **keep learning!**