RNN Demo

list of length-2 tuples each containing (review, label 0 or 1)

1. Load in training data (25000 IMDb reviews) `train_dataset`
2. Do a 80/20 split of the training data into:
   - proper training data (20000 reviews) `proper_train_dataset`
   - validation data (5000 reviews) `val_dataset`
3. Convert each proper training review into tokens using spaCy **(the demo was updated so that after spaCy's tokenization, we convert each token to lowercase)**

```
"Master cinéaste Alain Resnais likes to work with those actors"
```

(using the new `tokenizer_lowercase` function)

```
['master', 'cinéaste', 'alain', 'resnais', 'likes', 'to', 'work', 'with', 'those', 'actors']
```

4. Build a vocabulary using the proper training reviews `vocab`

behaves like a function (input: list of strings, output: list of integers)

- proper training data (20000 reviews) `proper_train_dataset`
- validation data (5000 reviews) `val_dataset`

3. Convert each proper training review into tokens using spaCy **(the demo was updated so that after spaCy's tokenization, we convert each token to lowercase)**

`"Master cinéaste Alain Resnais likes to work with those actors"`

(using the new `tokenizer_lowercase` function)

`['master', 'cinéaste', 'alain', 'resnais', 'likes', 'to', 'work', 'with', 'those', 'actors']`

4. Build a vocabulary using the proper training reviews `vocab`

behaves like a function (input: list of strings, output: list of integers)

5. Compute each proper training review's encoded version

(using the `vocab` function)

`[1259, 59266, 11261, 16475, 1225, 7, 171, 20, 162, 169]`

`proper_train_dataset_encoded`
`val_dataset_encoded`

list of length-2 tuples each containing (encoded review, label 0 or 1)

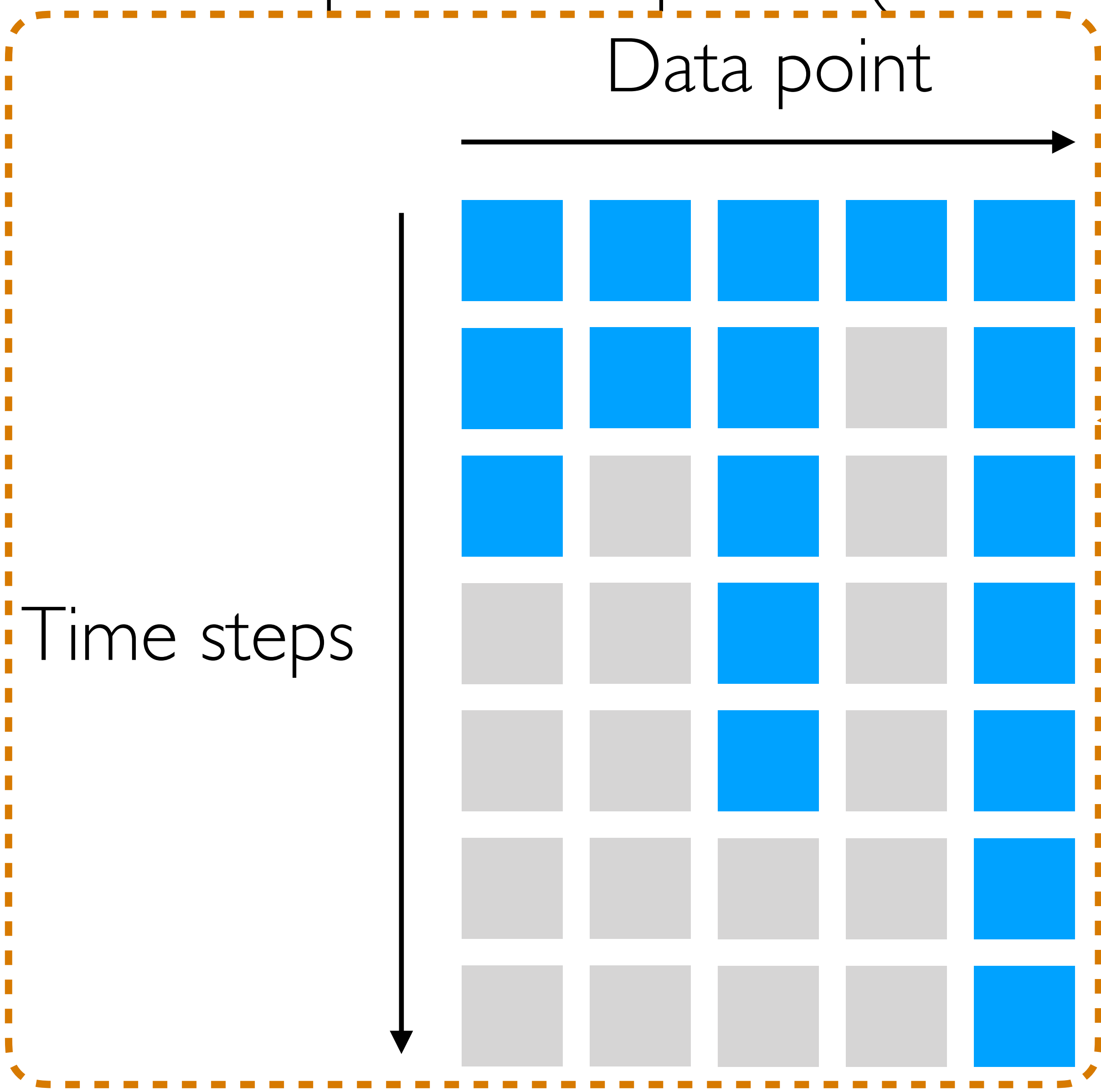
`proper_train_dataset_encoded`
`val_dataset_encoded`

list of length-2 tuples each containing (encoded review, label 0 or 1)

6. Construct neural net (instead of `nn.Sequential`, we make a class that inherits from `nn.module`)

PyTorch convention: the `forward` function specifies how a neural net actually processes a batch of input data

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7

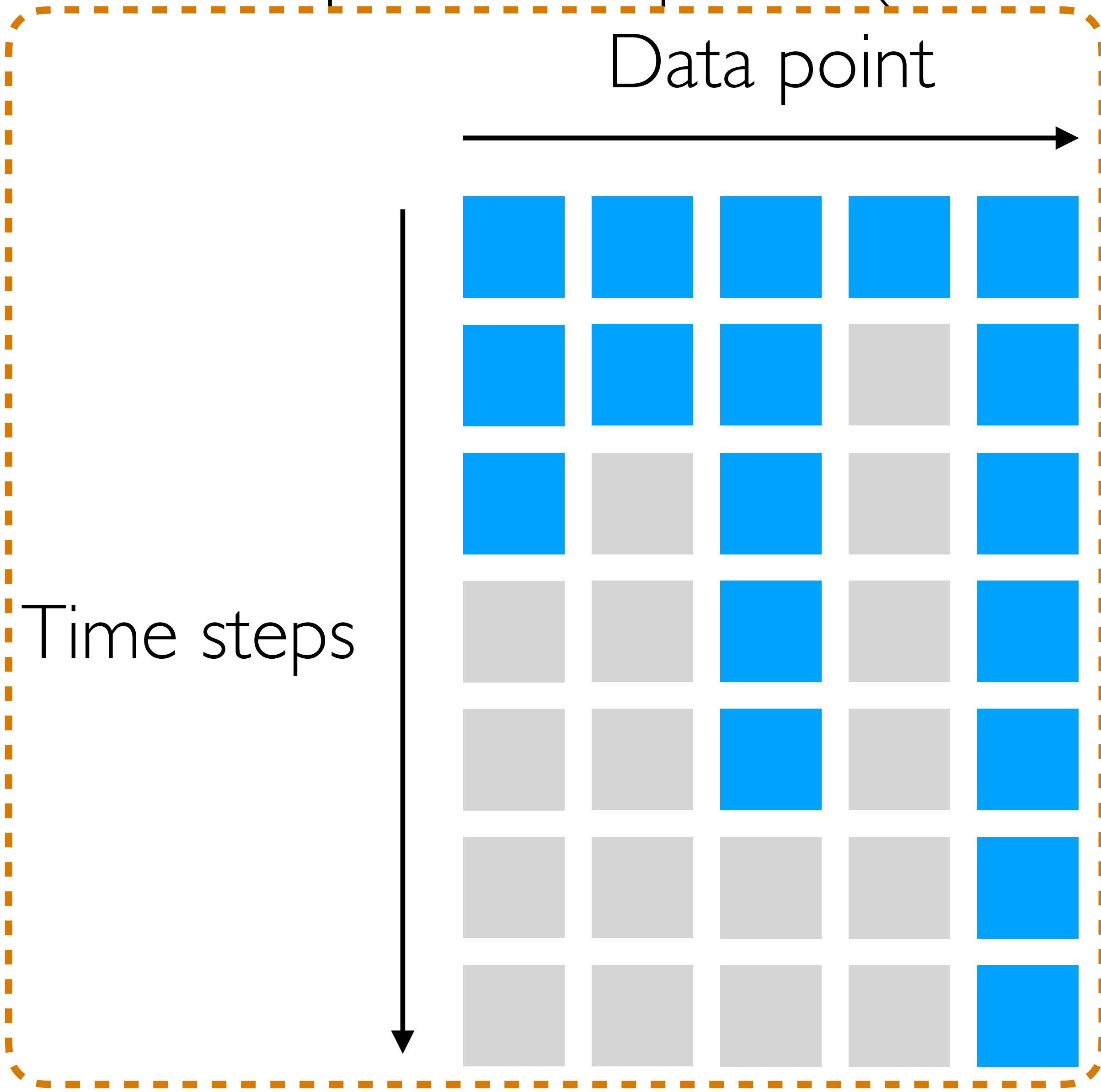


The neural net we constructed has a `forward` function with two inputs:
- a 2D table
  (each column is for 1 data point)
- a 1D table
  (specifies length for each time series)

Blue entries contain actual values from the 5 time series

Gray entries contain padded values (e.g., zeros)

Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7



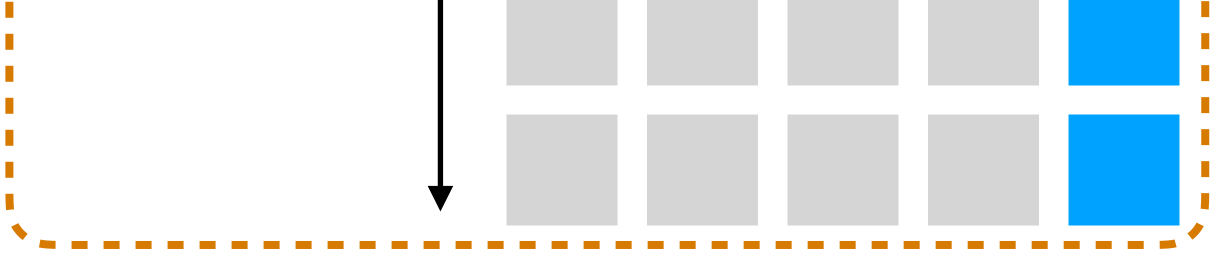
The neural net we constructed has a `forward` function with two inputs:
- a 2D table
  (each column is for 1 data point)
- a 1D table
  (specifies length for each time series)

Blue entries contain actual values from the 5 time series

Gray entries contain padded values (e.g., zeros)

```
In [30]: # example where there are 5 input time series of lengths 3, 2, 5, 1, 7;
         # we specify these time series using a 2D table that is padded and a
         # 1D table of lengths (see lecture slides for details)
         summary(simple_lstm_model,
                 input_data=[torch.zeros((7, 5), dtype=torch.long),
                             torch.tensor([3, 2, 5, 1, 7], dtype=torch.long)])
```

Data types matter in PyTorch (torch.long means these tables store integers)



Gray entries contain
padded values (e.g., zeros)

```
In [30]: # example where there are 5 input time series of lengths 3, 2, 5, 1, 7;
         # we specify these time series using a 2D table that is padded and a
         # 1D table of lengths (see lecture slides for details)
         summary(simple_lstm_model,
                 input_data=[torch.zeros((7, 5), dtype=torch.long),
                             torch.tensor([3, 2, 5, 1, 7], dtype=torch.long)])
```

Data types matter in PyTorch (torch.long means these tables store integers)

7. Train the neural net some max number of epochs

8. Automatically tune on one hyperparameter:
   choose # of epochs to be the one achieving highest validation accuracy

9. Load in the saved neural net from the best # of epochs

10. Finally load in test data, tokenize and convert each test review into
    a list of integers, and use the trained neural net to predict

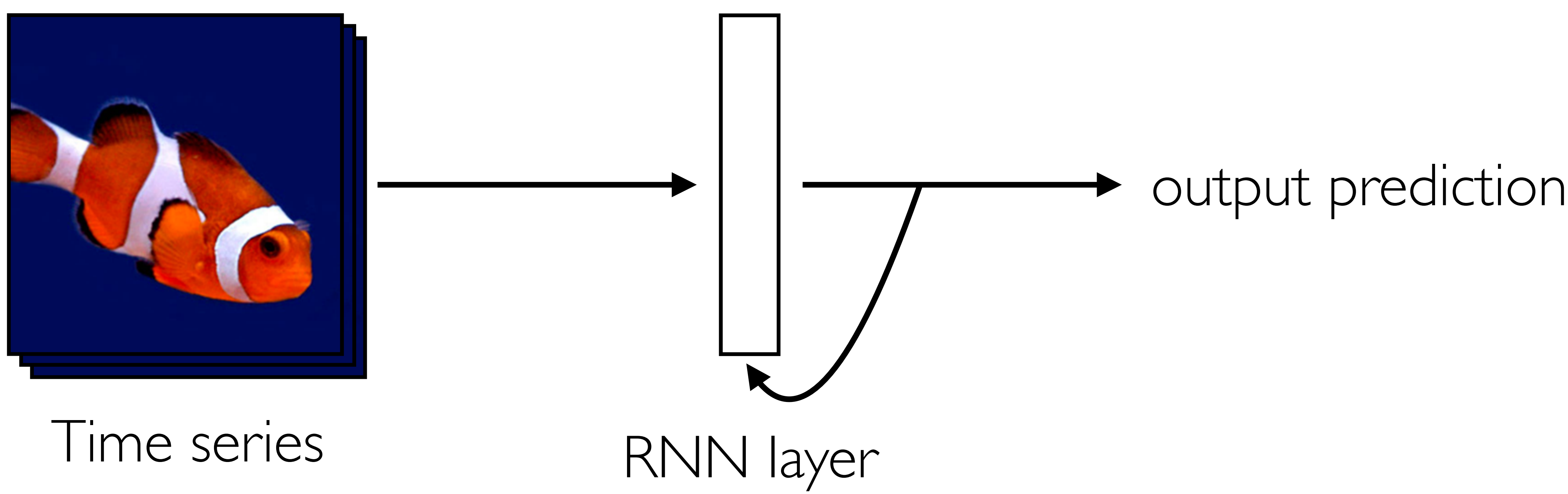
A special kind of RNN: an “LSTM”

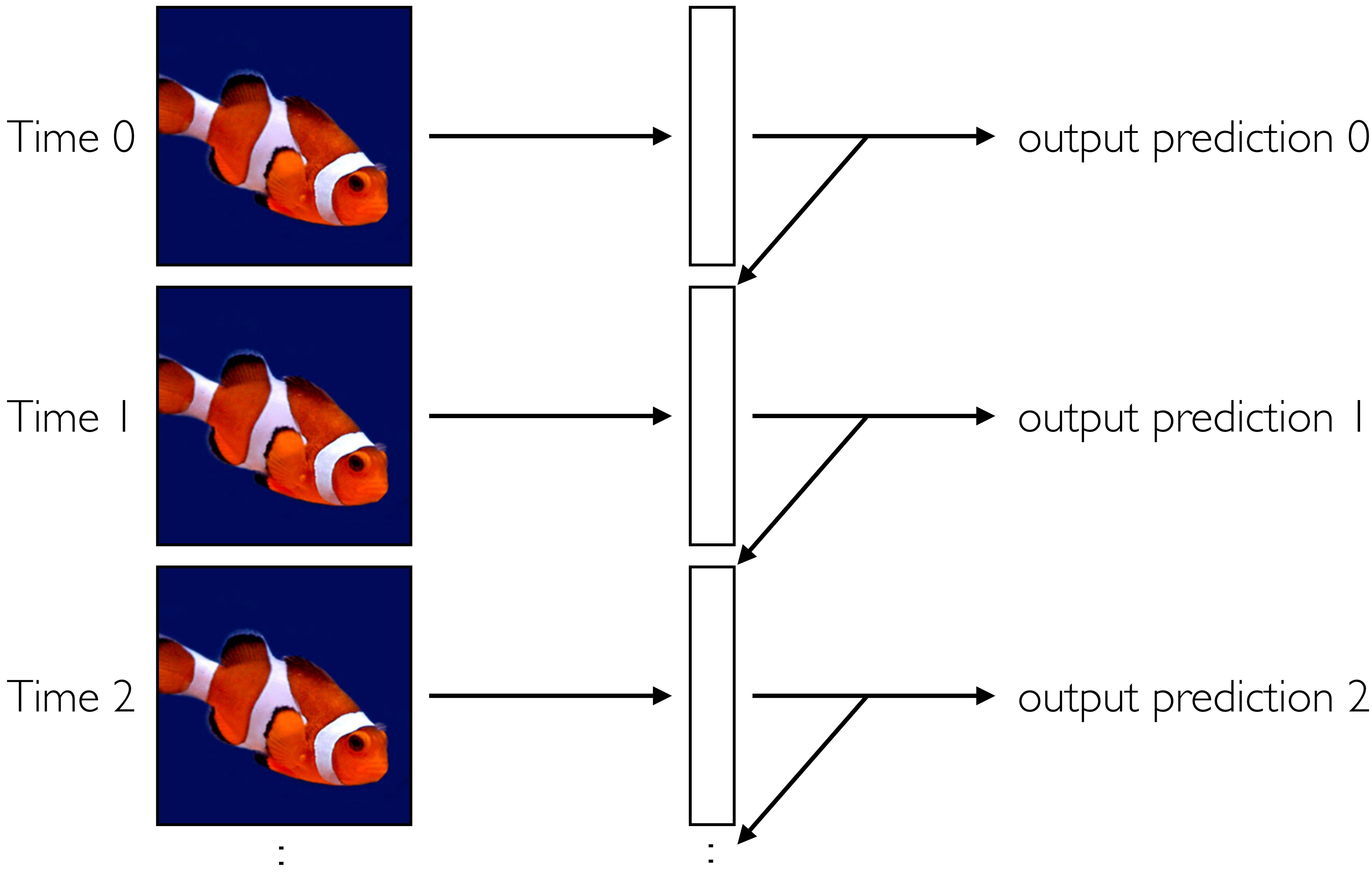
(Flashback) Vanilla ReLU RNN

```
current_state = np.zeros(num_nodes)
outputs = []
for input in input_sequence:
  linear = np.dot(input, W.T) + b    \
           + np.dot(current_state, U.T)
  output = np.maximum(0, linear) # ReLU
  outputs.append(output)
  current_state = output
```

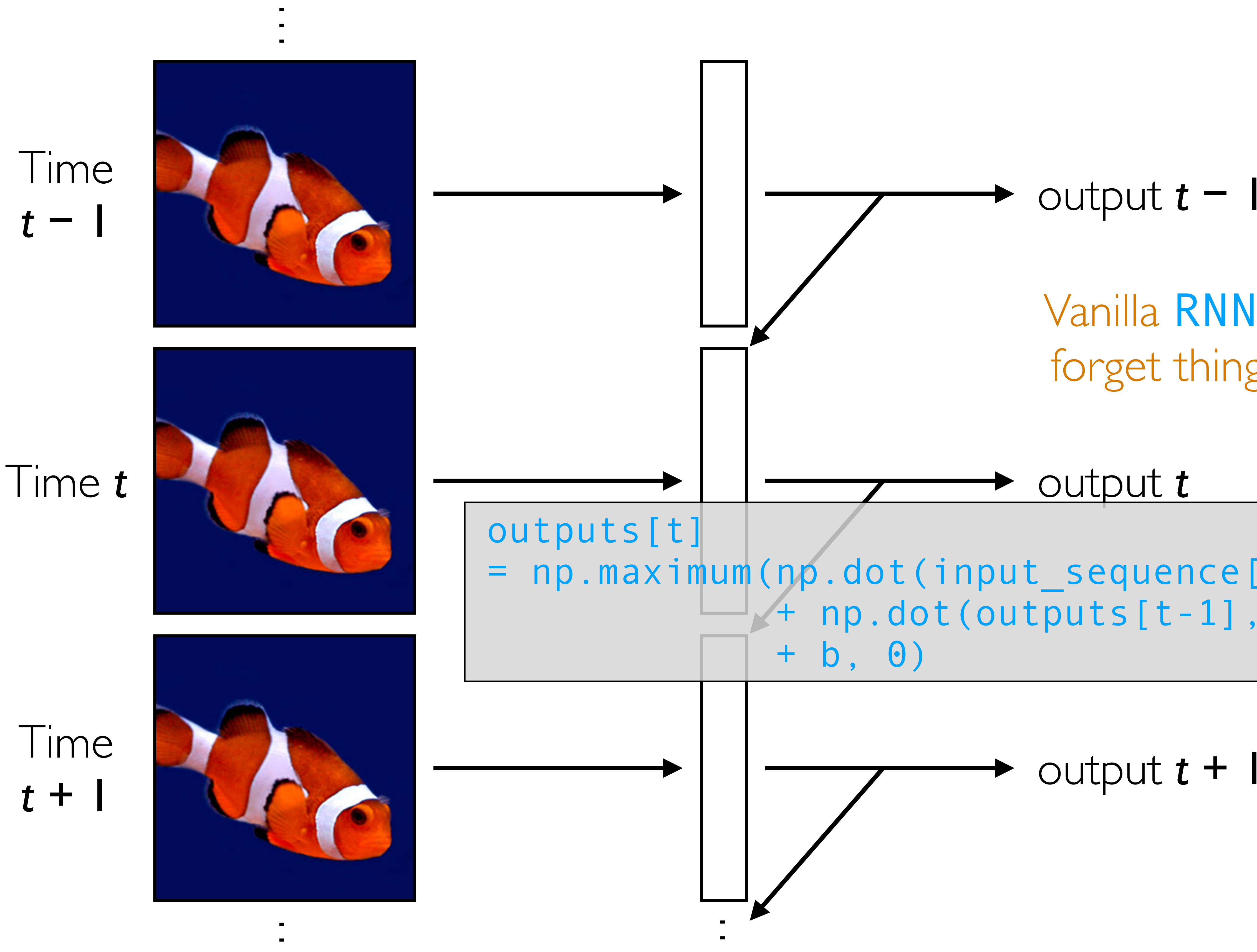
In general: there is an output at every time step

For simplicity, in today's lecture, we only use the very last time step's output


Time series
RNN layer
output prediction

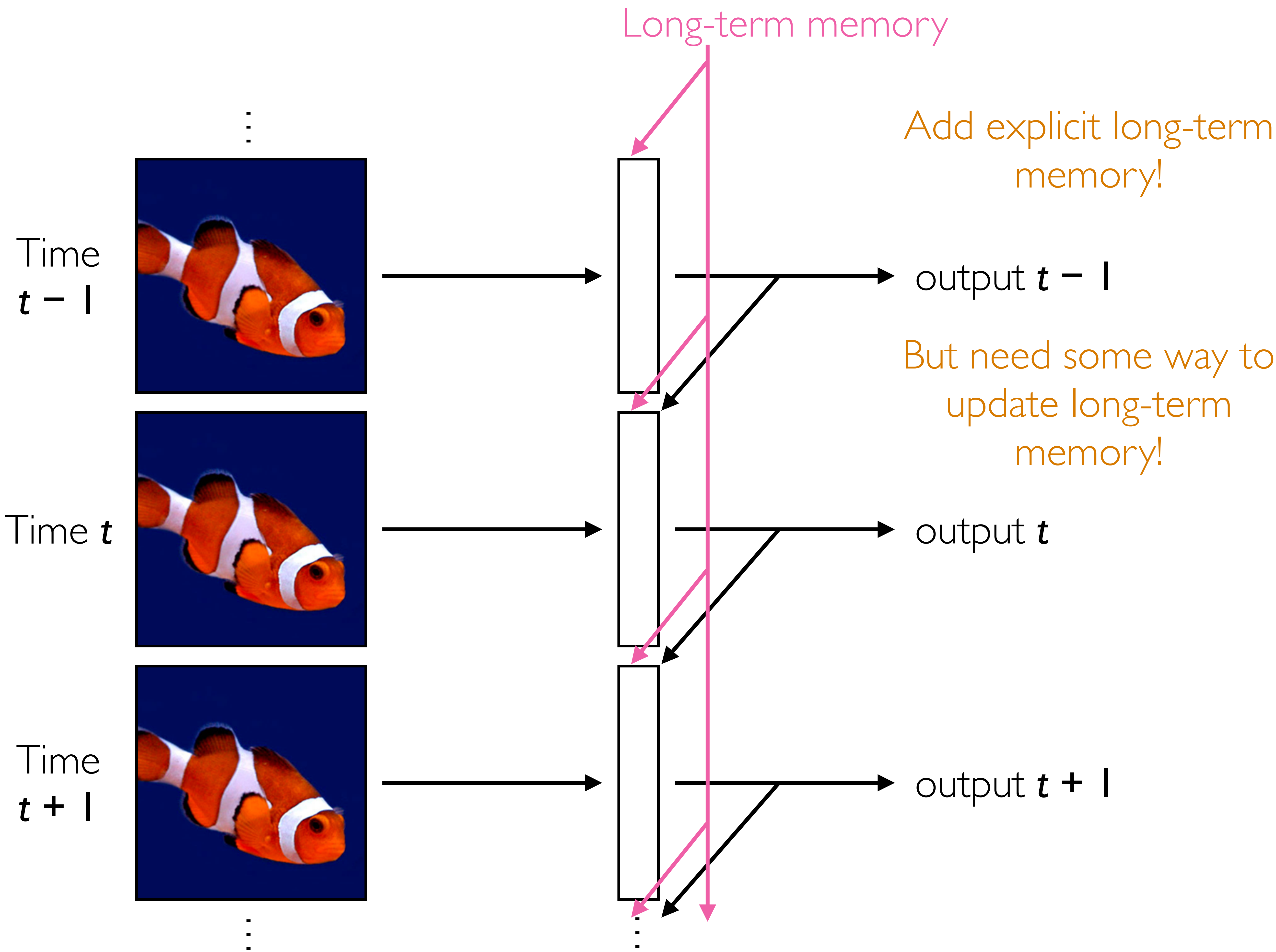


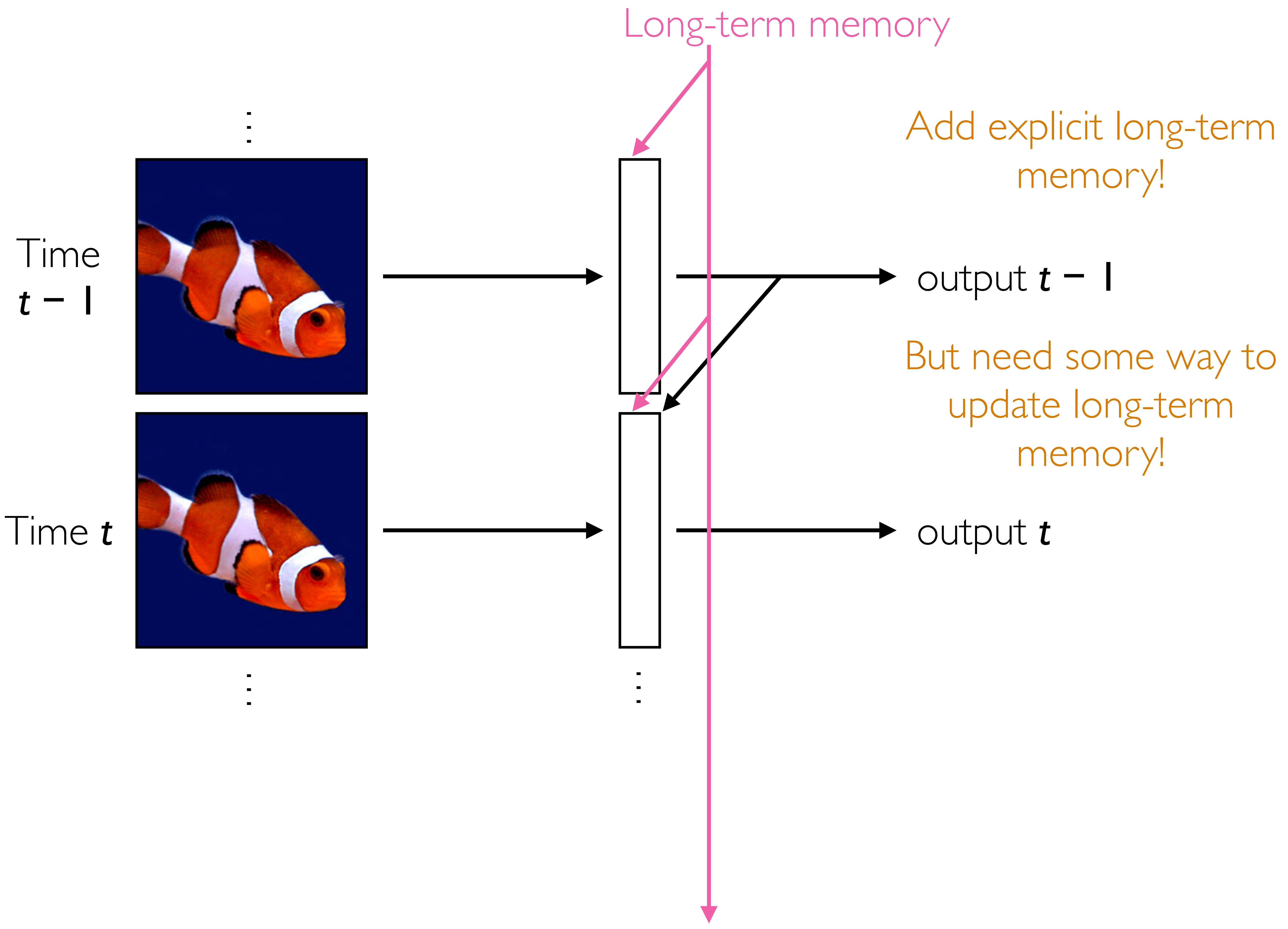
Time 0
Time 1
Time 2
output prediction 0
output prediction 1
output prediction 2



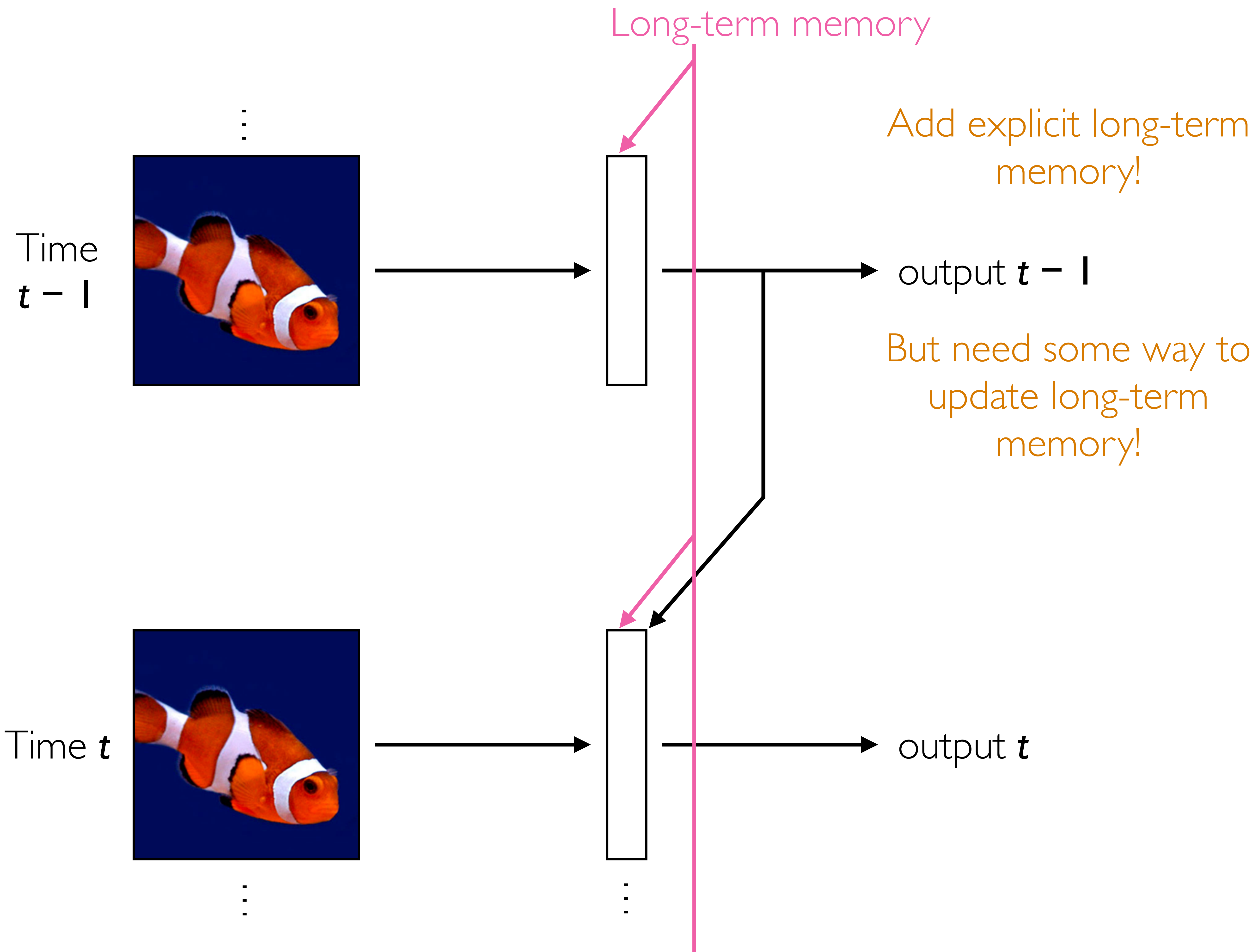
Vanilla RNN tends to forget things quickly

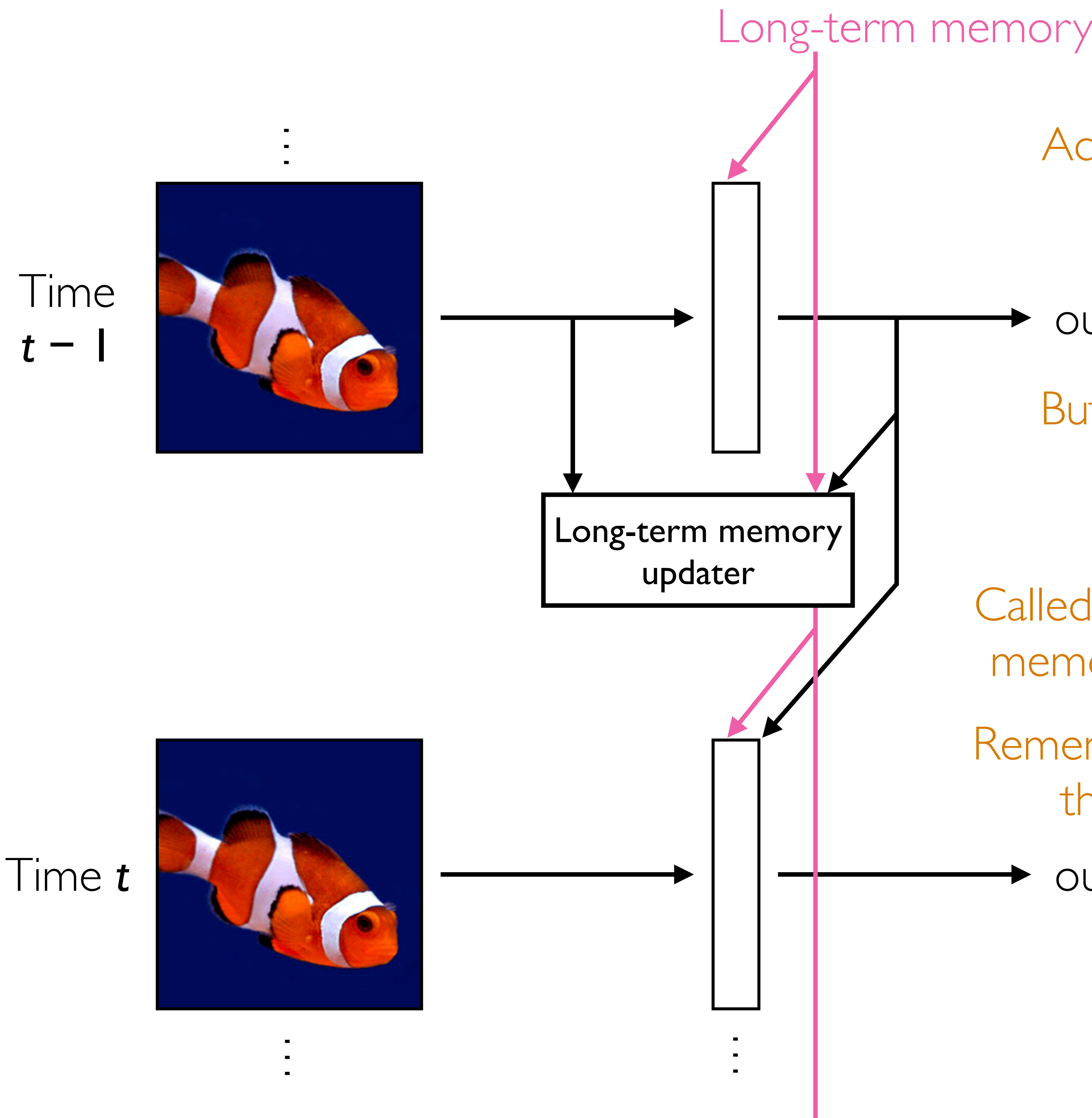
```
outputs[t]
= np.maximum(np.dot(input_sequence[t], W.T)
             + np.dot(outputs[t-1], U.T)
             + b, 0)
```


Long-term memory
Add explicit long-term memory!
But need some way to update long-term memory!
Time $t - 1$
Time t
Time $t + 1$
output $t - 1$
output t
output $t + 1$



Long-term memory
Add explicit long-term memory!
Time $t-1$
output $t-1$
But need some way to update long-term memory!
Time t
output t


Long-term memory
Add explicit long-term memory!
Time $t - 1$
output $t - 1$
But need some way to update long-term memory!
Time t
output t



Add explicit long-term memory!

But need some way to update long-term memory!

Called a "long short-term memory" (LSTM) RNN

Remembers things longer than vanilla RNN

Analyzing Times Series with CNNs

- Think about an image with 1 column, and where the rows index time steps: this is a time series!
- Think about a 2D image where rows index time steps, and the columns index features: this is a multivariate time series (feature vector that changes over time!)
- CNNs can be used to analyze time series *but inherently the size of the filters used say how far back in time we look*
- If your time series data all have the same length (same number of time steps) and do not have long-range dependencies that require long-term memory, CNNs can do well already!

  ⇒ If you need long-term memory or time series with different lengths, use RNNs

- Note: while it is possible to have a CNN take in inputs that vary in size, we did not cover this in lecture

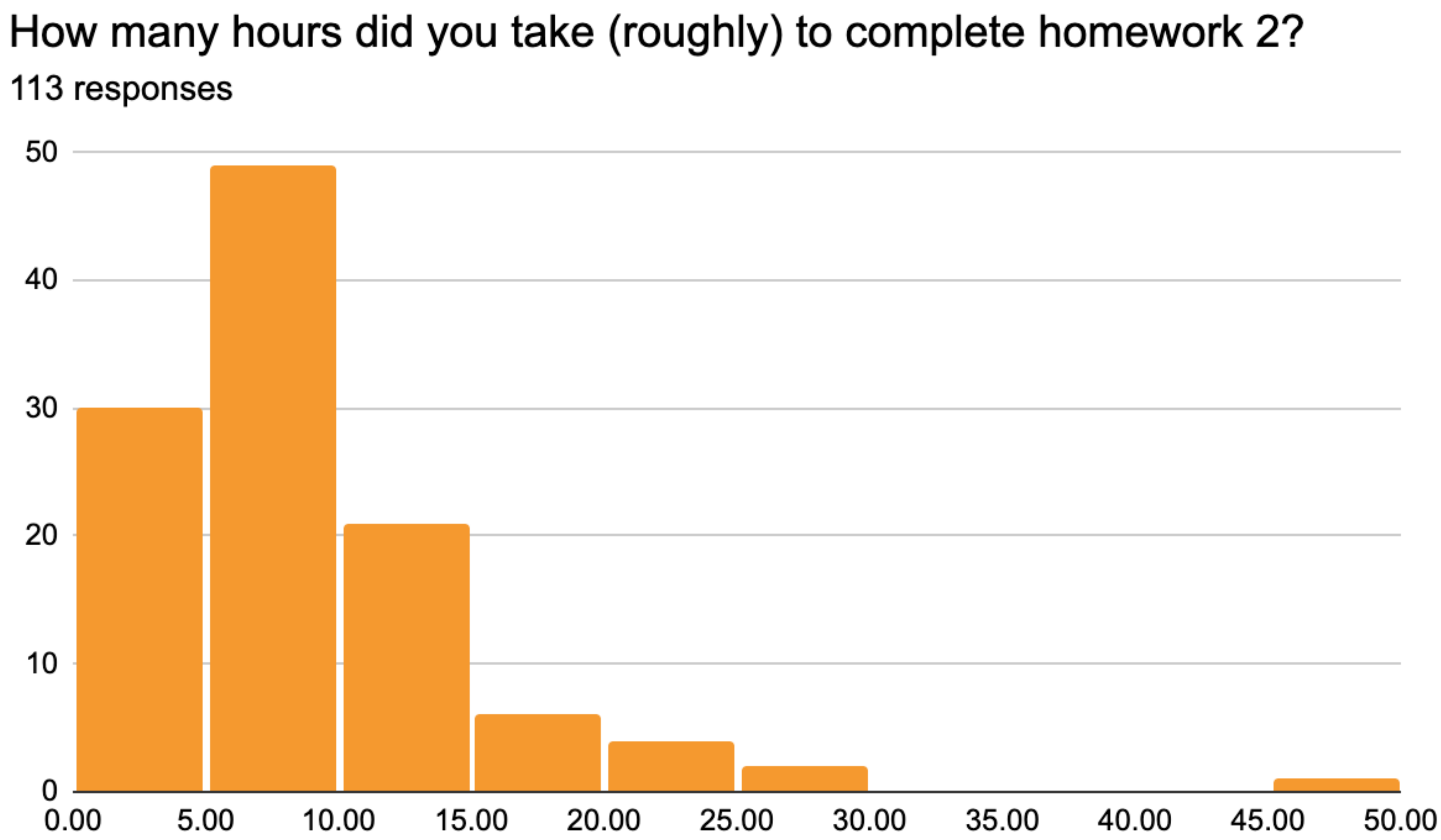
Carnegie Mellon University
Heinz College

95-865 Unstructured Data Analytics

Last lecture: Additional deep learning topics; course wrap-up

Slides by George H. Chen

HW2 Questionnaire (1/3)



HW2 Questionnaire (2/3)

Free response comments/feedback

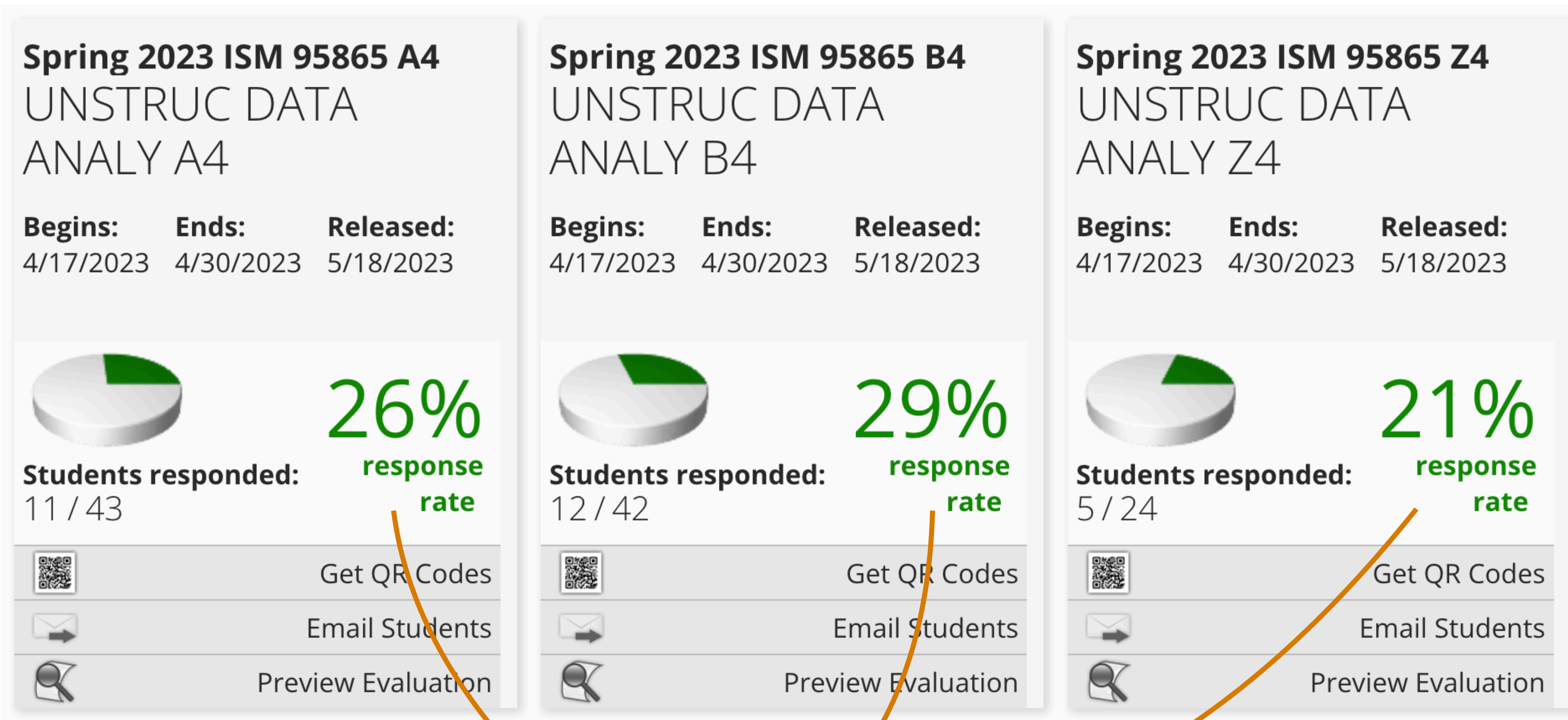
- Reading material and notes:
  - I realize that the current situation is not great (i.e., there's no single textbook/easy to understand resource that covers all the topics of 95-865 at the same level of detail)
  - Many students said they use StatQuest
- There was a comment saying that asking ChatGPT to explain concepts has been very helpful and that, basically, ChatGPT's office hours slots are 24/7 (nice!)
  - Careful! ChatGPT had a high error rate on Quiz 1 Problem 1
- I got several requests from students saying that they wish I provided problems like the ones from your real quizzes
  - This is precisely why we provide many practice quizzes — these <u>are</u> real past quizzes!

HW2 Questionnaire (3/3)

- A number of students are still asking for more demos
  - As I stated in Lecture 11 (in my thoughts on the HW1 questionnaire): it's important that you learn to not only find other demos yourself **but to create your own demos**
    - For example, start with a demo that already exists using a dataset you find interesting, and think about other possible analyses that you can do on the same dataset
      - In fact, a number of demos from my lectures are like this where I have cited the original demo that I modified
  - I think it's important to recognize that getting better at data analysis (unstructured or not) *requires practice*
  - Analogy: it's like learning how to swim
    - Sure, you can watch more and more demonstrations of people swimming, *but to get good yourself, you have to practice*

Faculty Course Evaluations

Please fill out faculty course evaluations to provide feedback on the course!



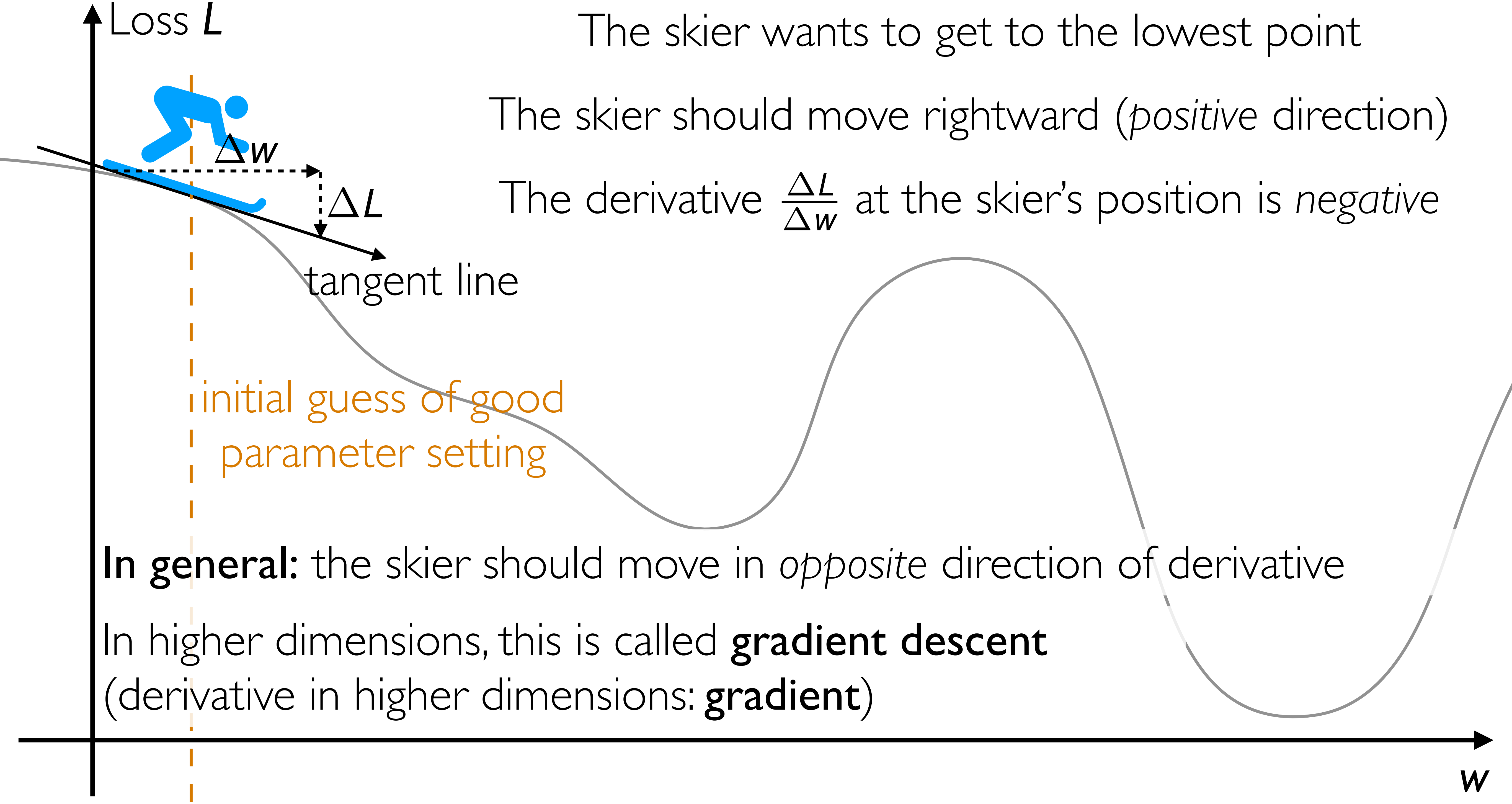
Let's get these response rates higher! 😃

Outline

- How learning a deep net roughly works
- Dealing with small datasets
    - Data augmentation
    - Fine-tuning
- Self-supervised learning (word embeddings are a special case)
- Some other deep learning topics that are good to know about
- Course wrap-up

Learning a Deep Net

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)

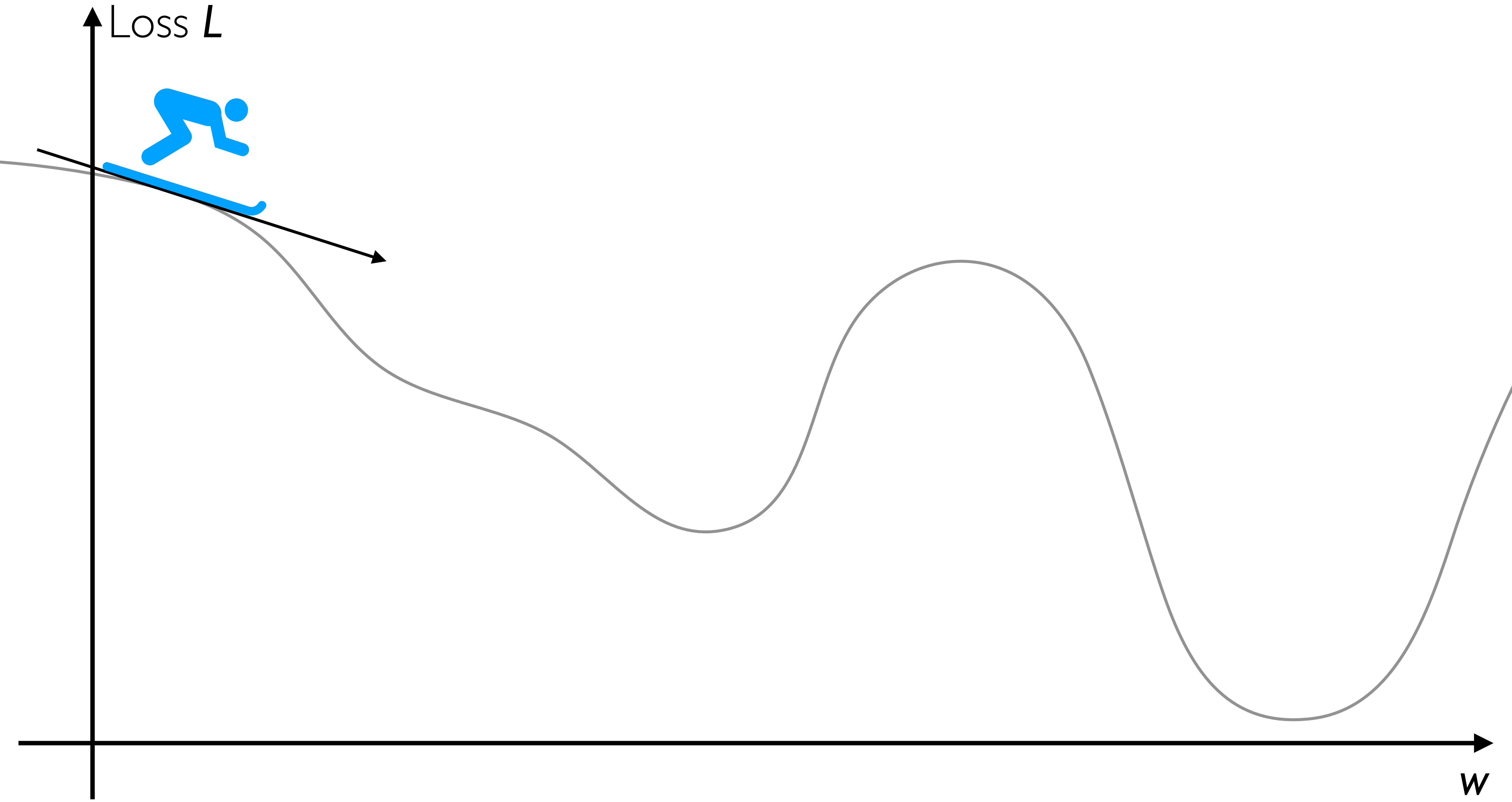
The derivative $\frac{\Delta L}{\Delta w}$ at the skier's position is *negative*

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

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

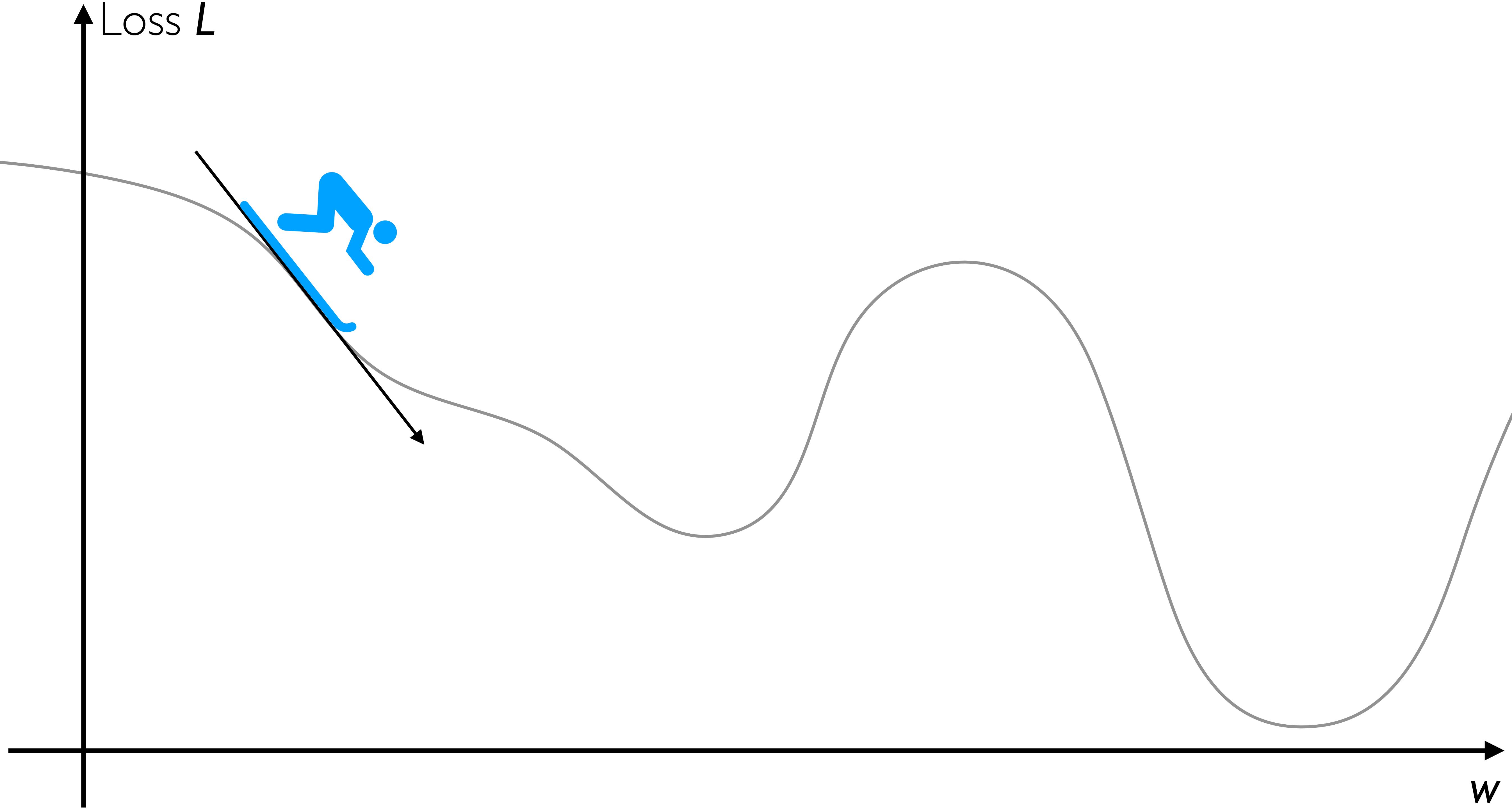
Learning a Deep Net

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



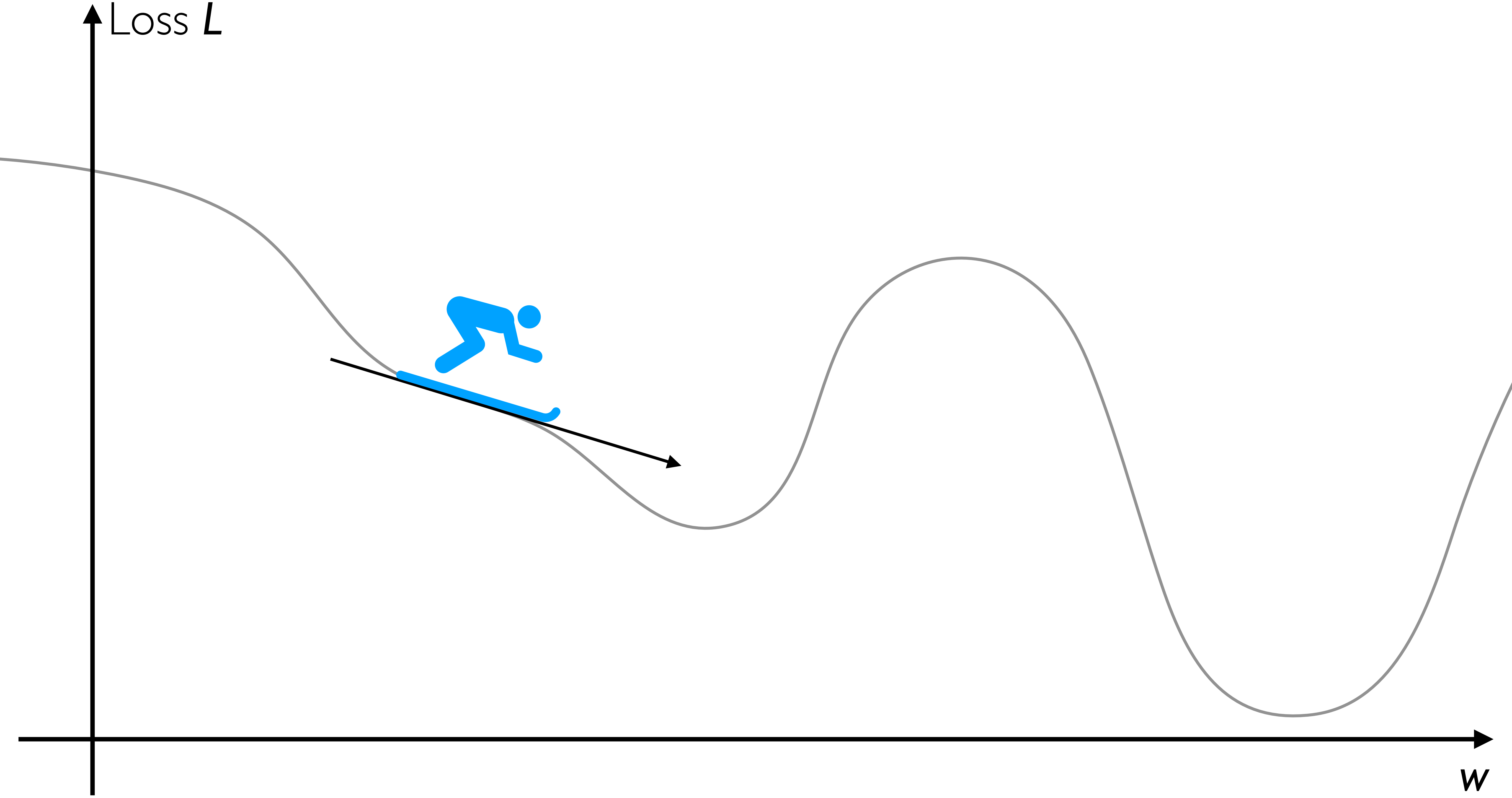
Learning a Deep Net

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



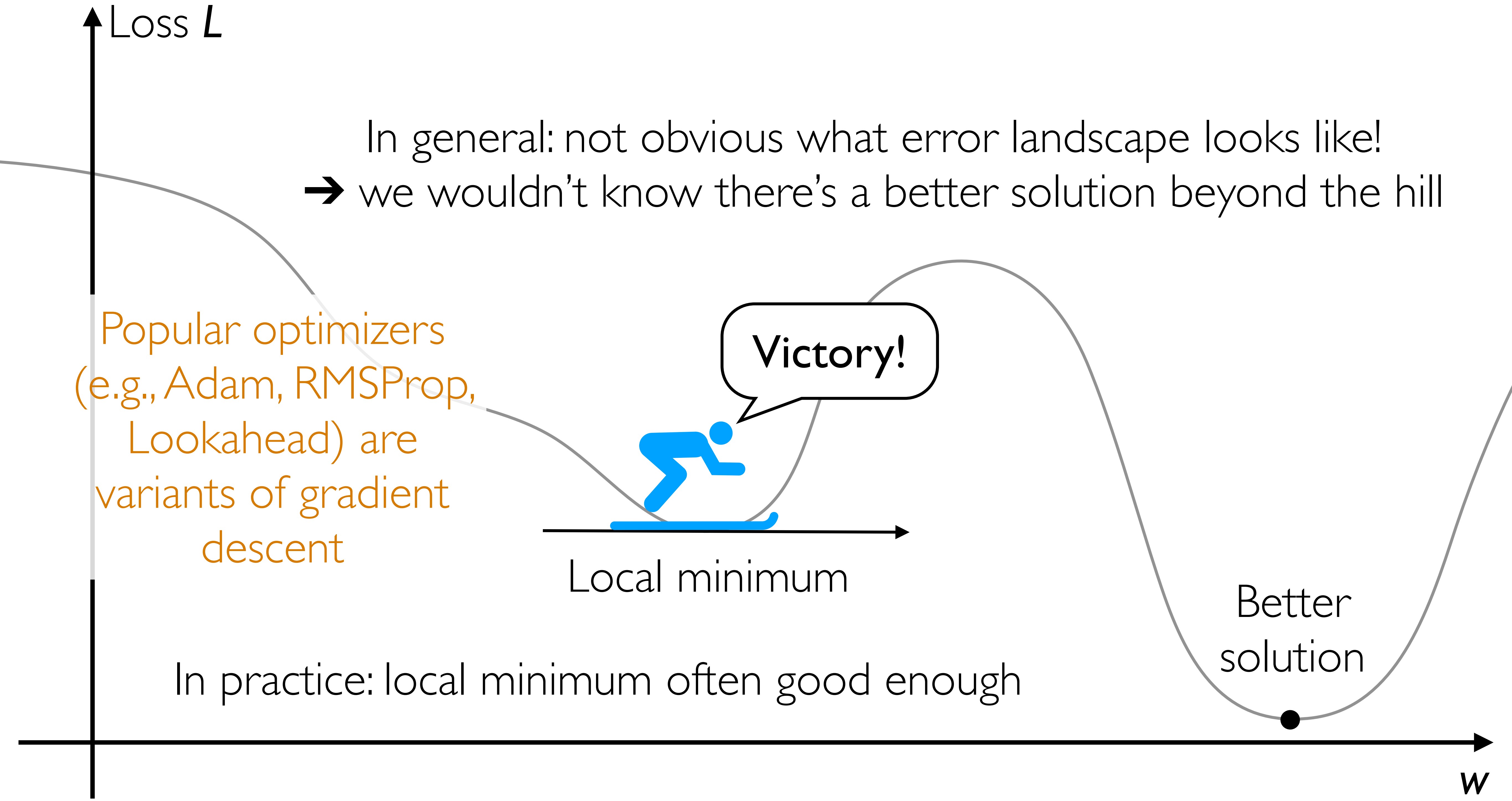
Learning a Deep Net

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



Learning a Deep Net

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

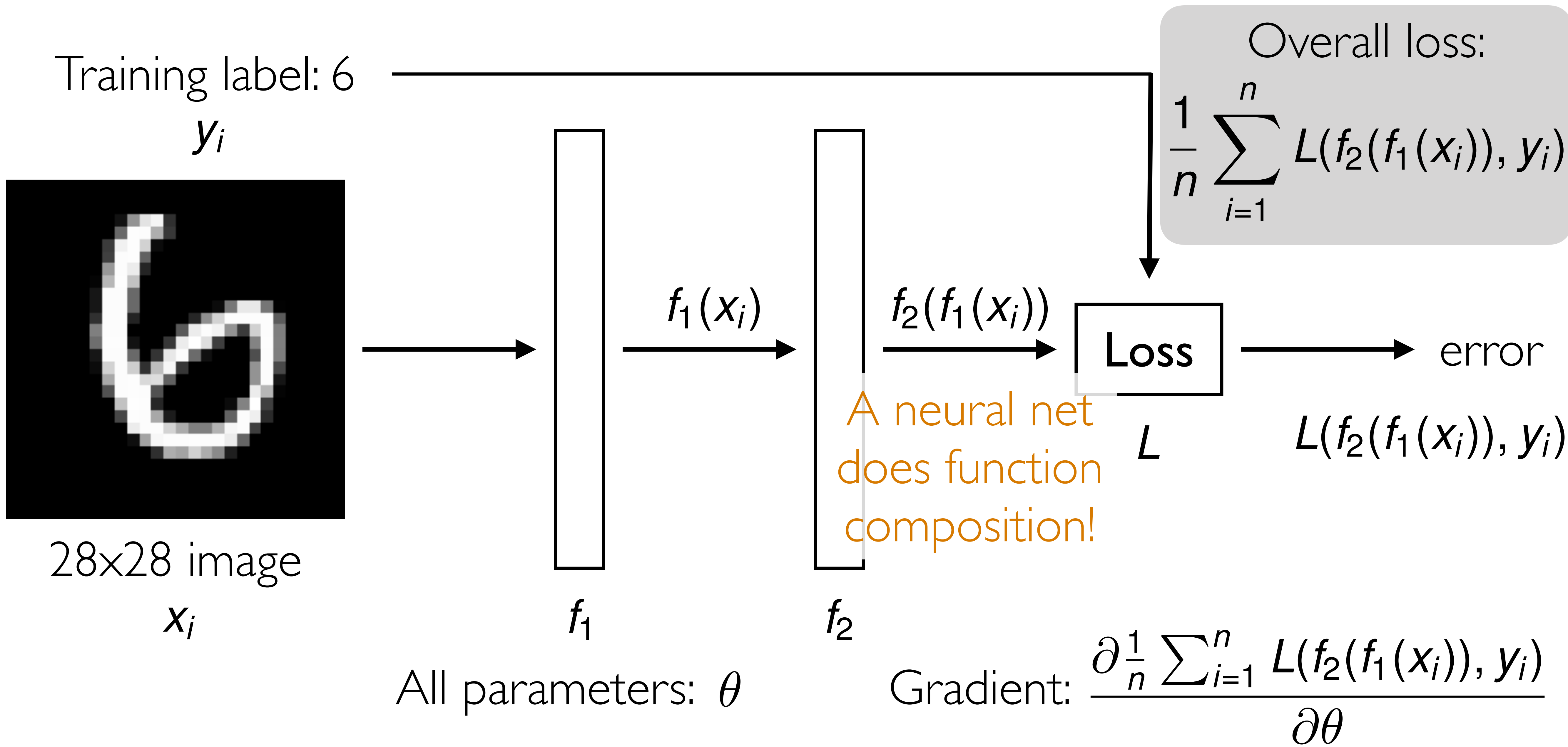


In general: not obvious what error landscape looks like!
➔ we wouldn't know there's a better solution beyond the hill

Popular optimizers (e.g., Adam, RMSProp, Lookahead) are variants of gradient descent

In practice: local minimum often good enough

Handwritten Digit Recognition



Automatic differentiation is crucial in learning deep nets!

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

Gradient Descent

Training example 1 | Training example 2 | Training example 3 | Training example 4 | Training example 5 | … | Training example n

loss 1 | loss 2 | loss 3 | loss 4 | loss 5 | … | loss n

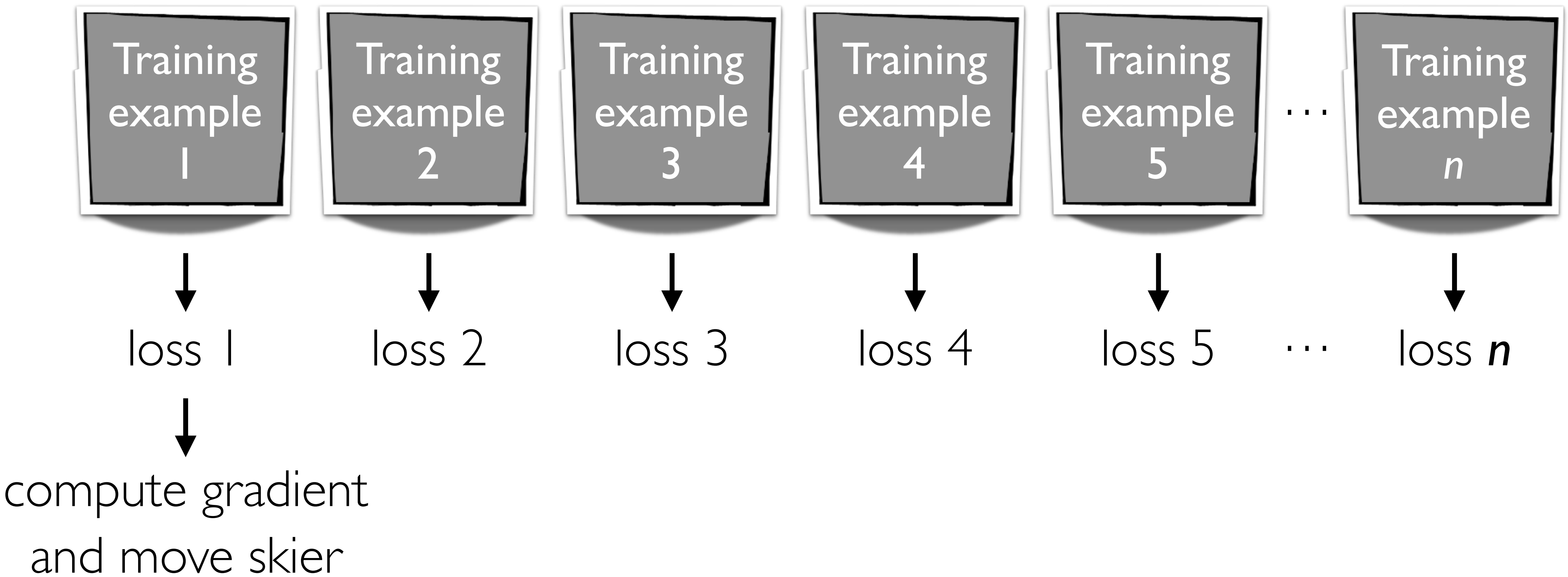
average loss

compute gradient and move skier

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

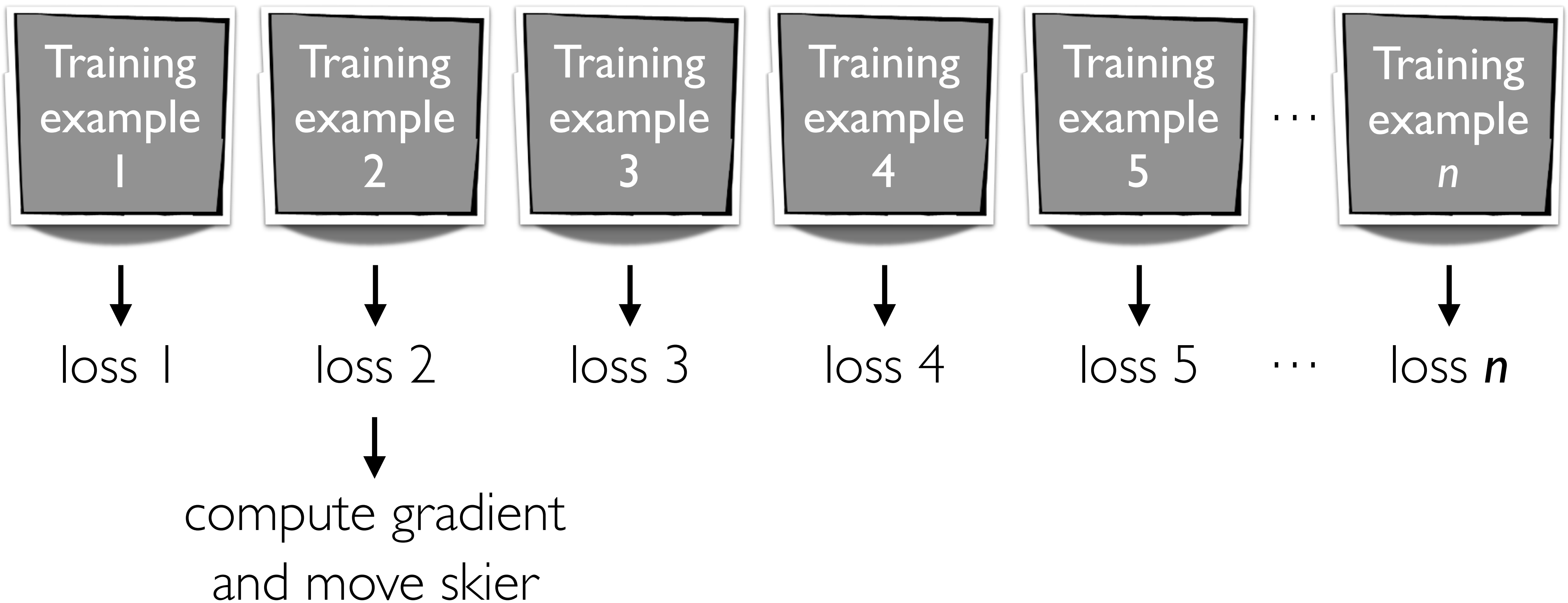
Computing gradients using all the training data seems really expensive!

Stochastic Gradient Descent (SGD)



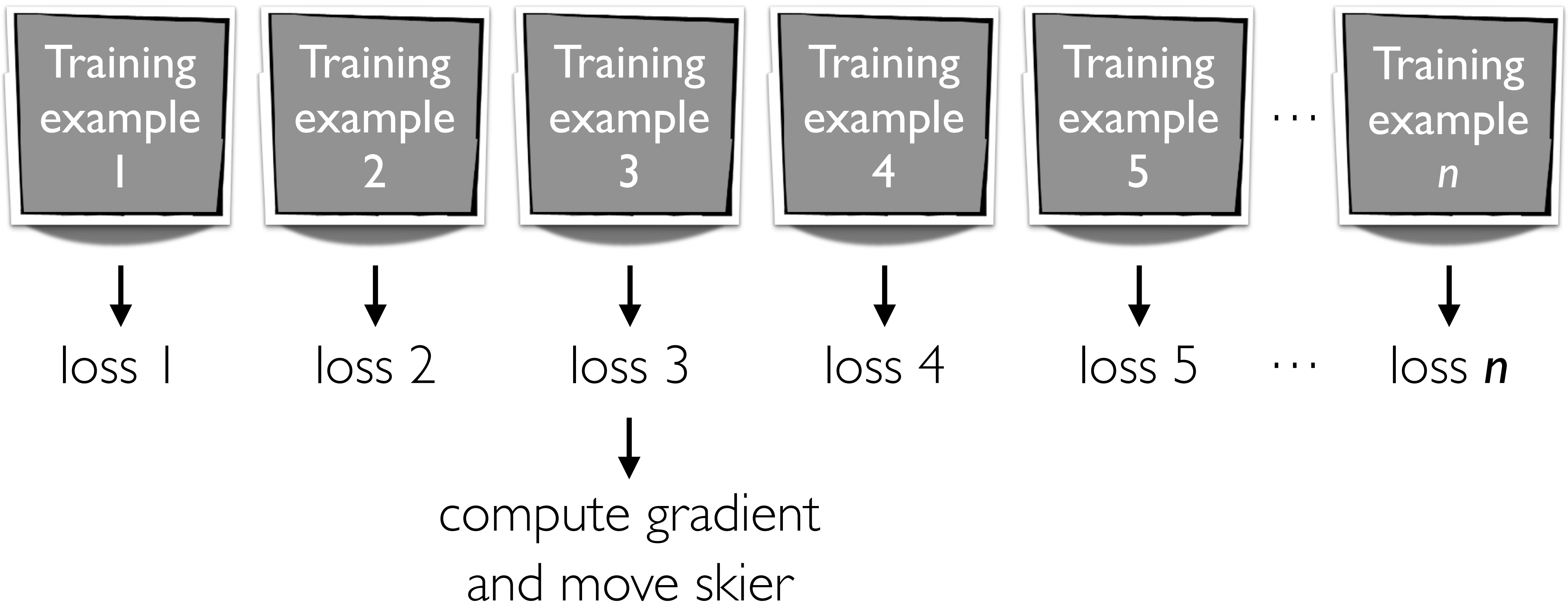
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



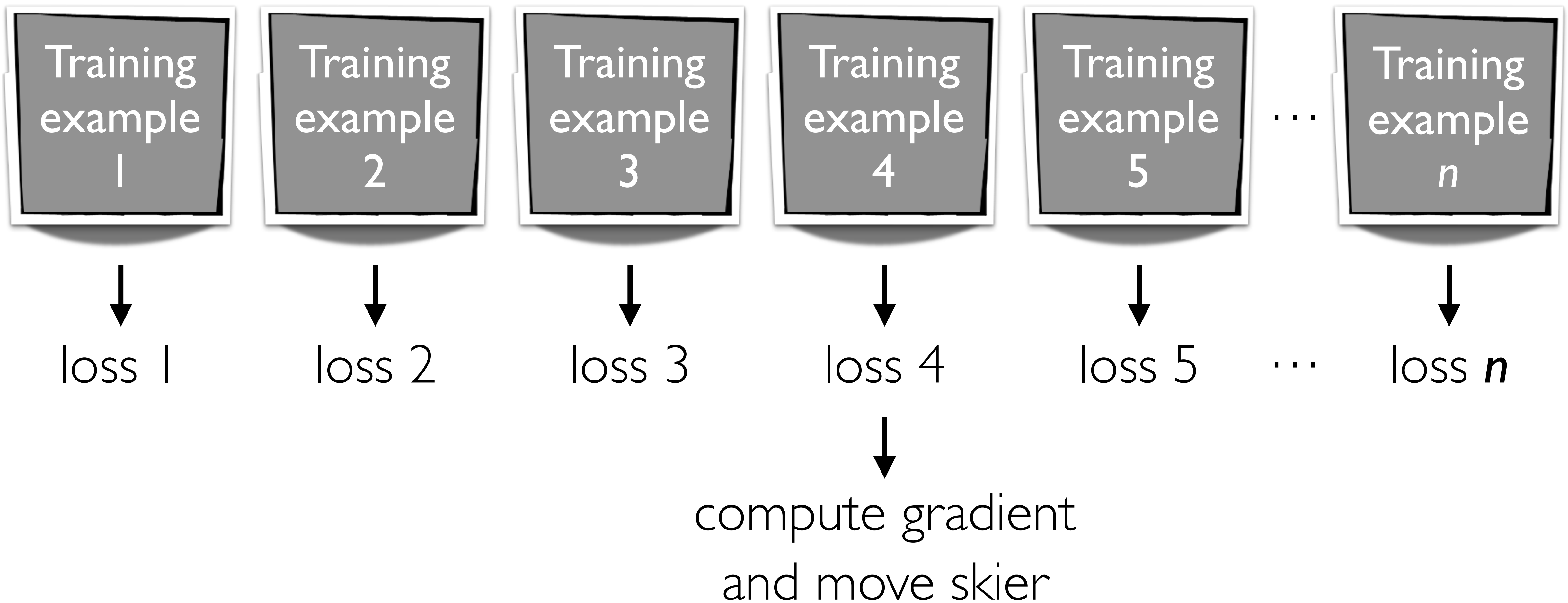
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



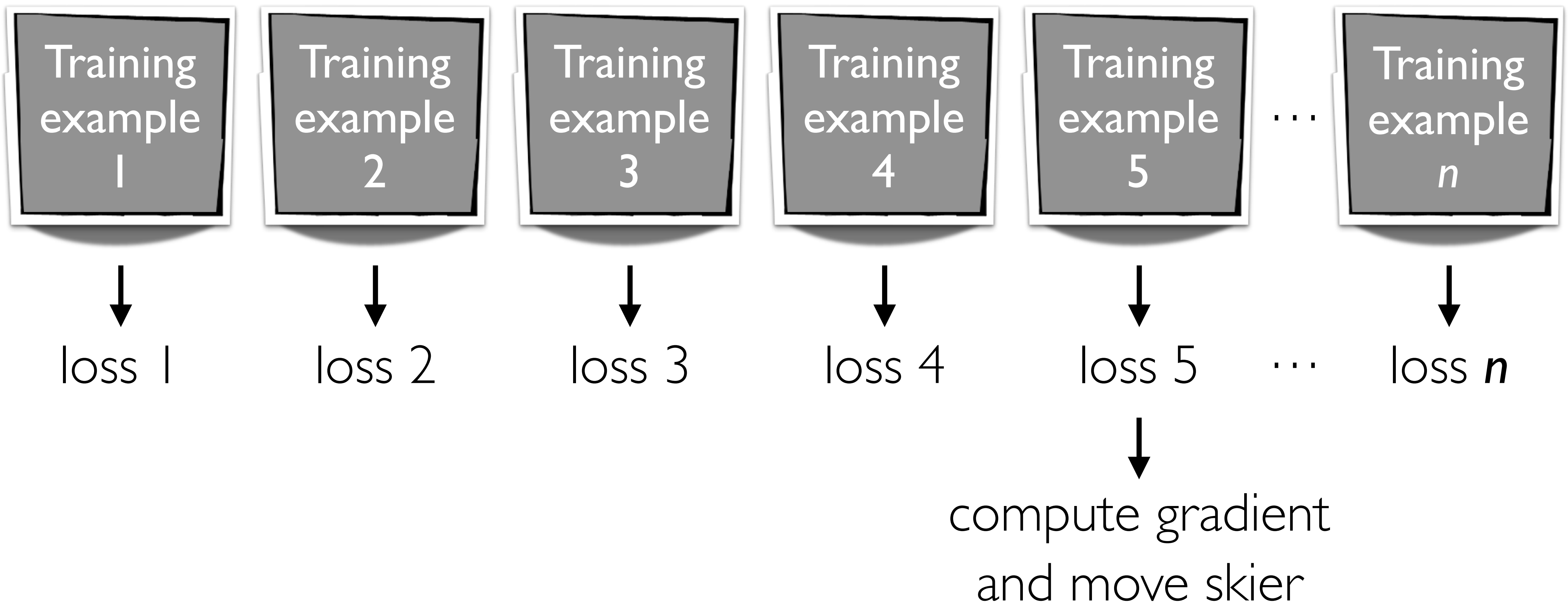
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



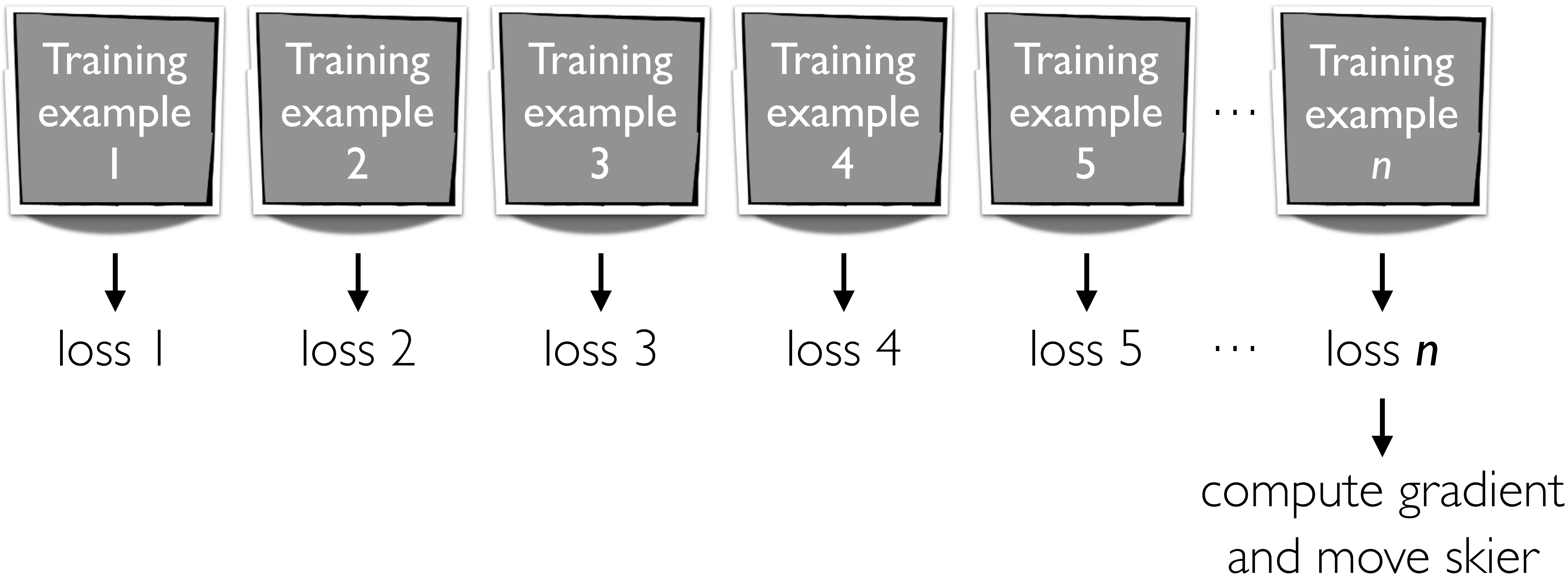
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

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)

Training example 1	Training example 2	Training example 3	Training example 4	Training example 5	⋯	Training example *n*
↓	↓	↓	↓	↓		↓
loss 1	loss 2	loss 3	loss 4	loss 5	⋯	loss ***n***

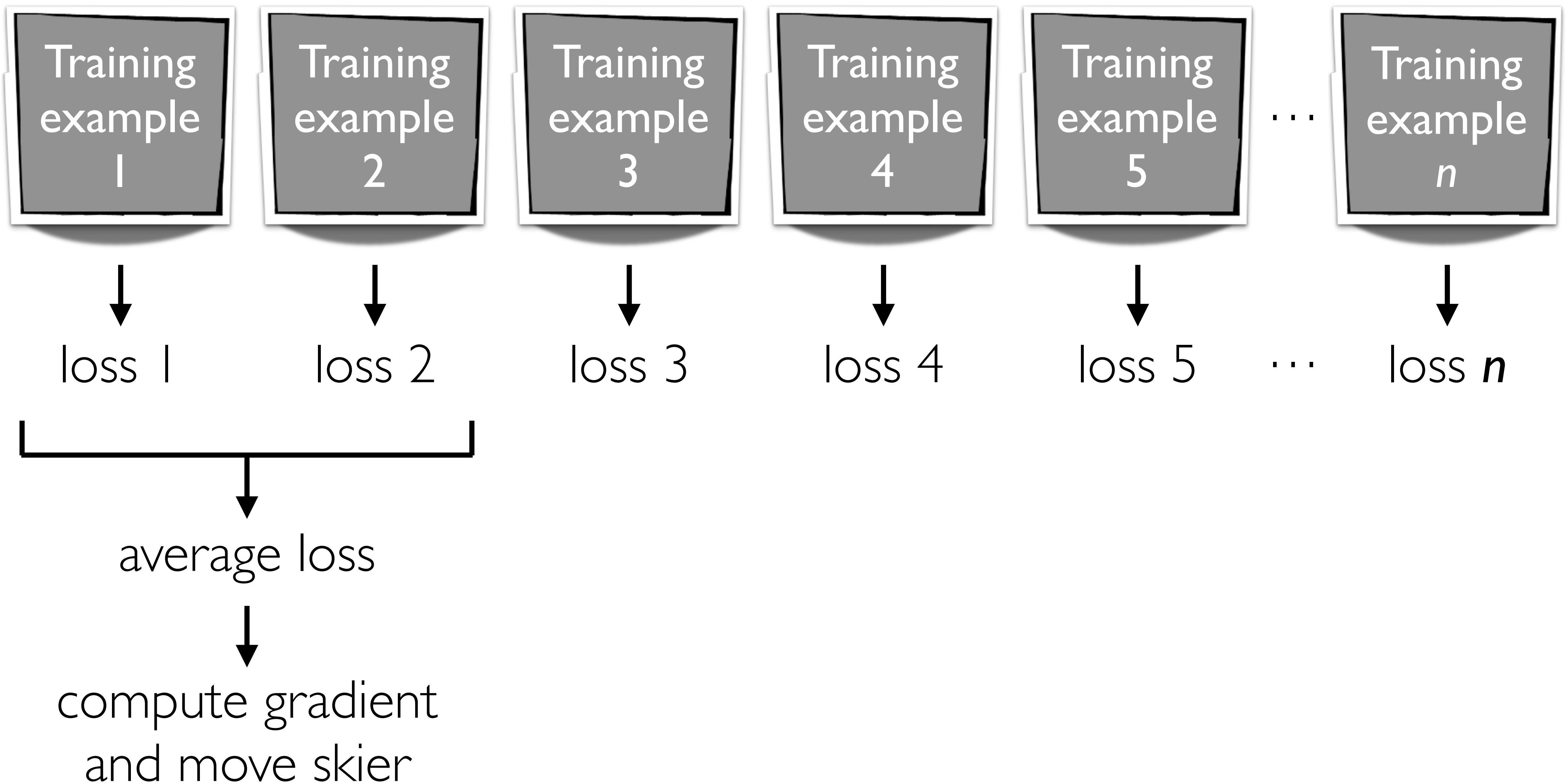
↓

compute gradient
and move skier

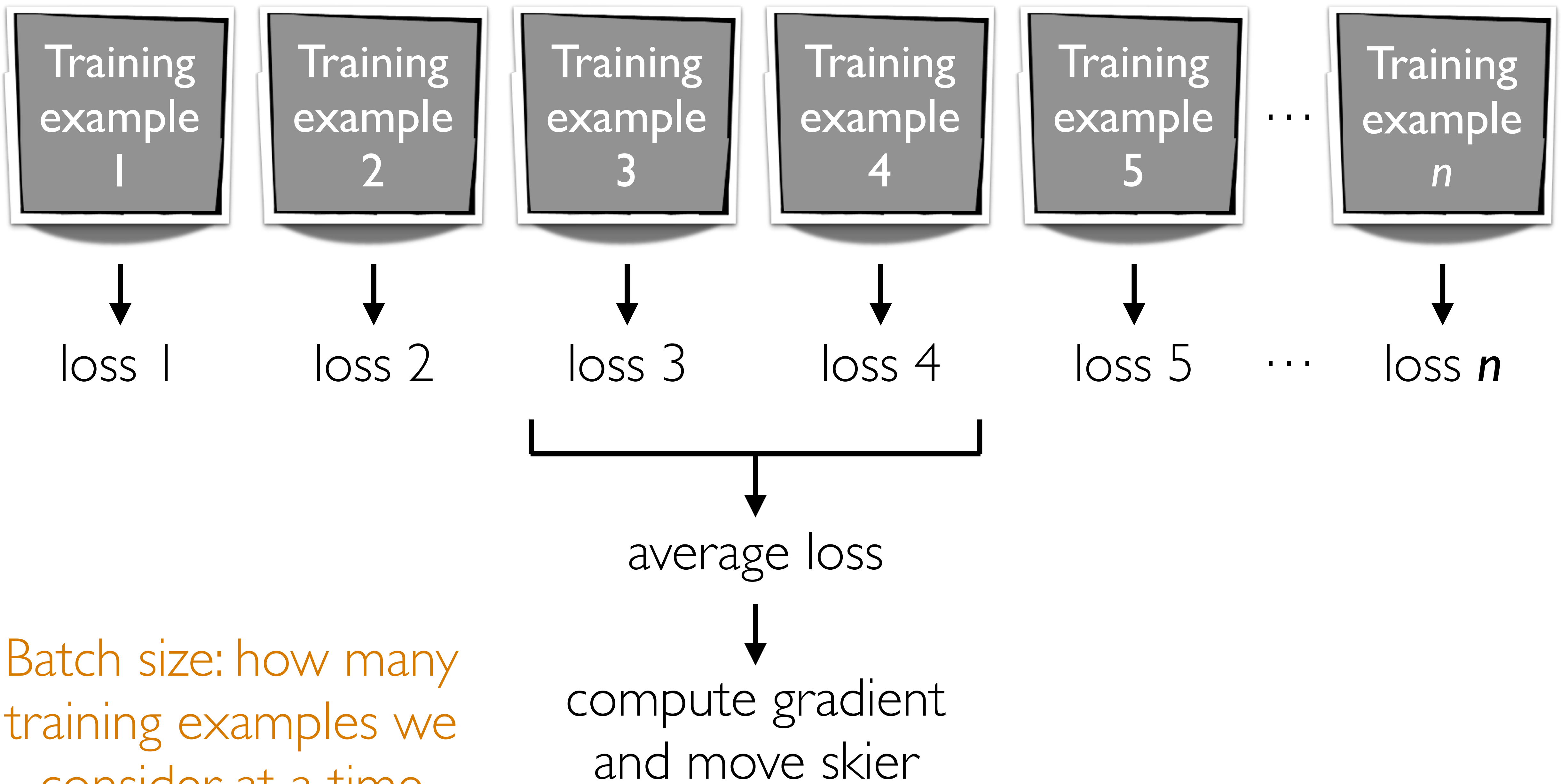
An epoch refers to 1 full pass through
all the training data

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

Minibatch Gradient Descent



Minibatch Gradient Descent



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

Best optimizer? Best learning rate? 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!!!

A Look Under the Hood

UDA_pytorch_utils.py

Dealing with Small Datasets

Data Augmentation

Generate perturbed versions of your training data to get a 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)

Fine Tuning

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

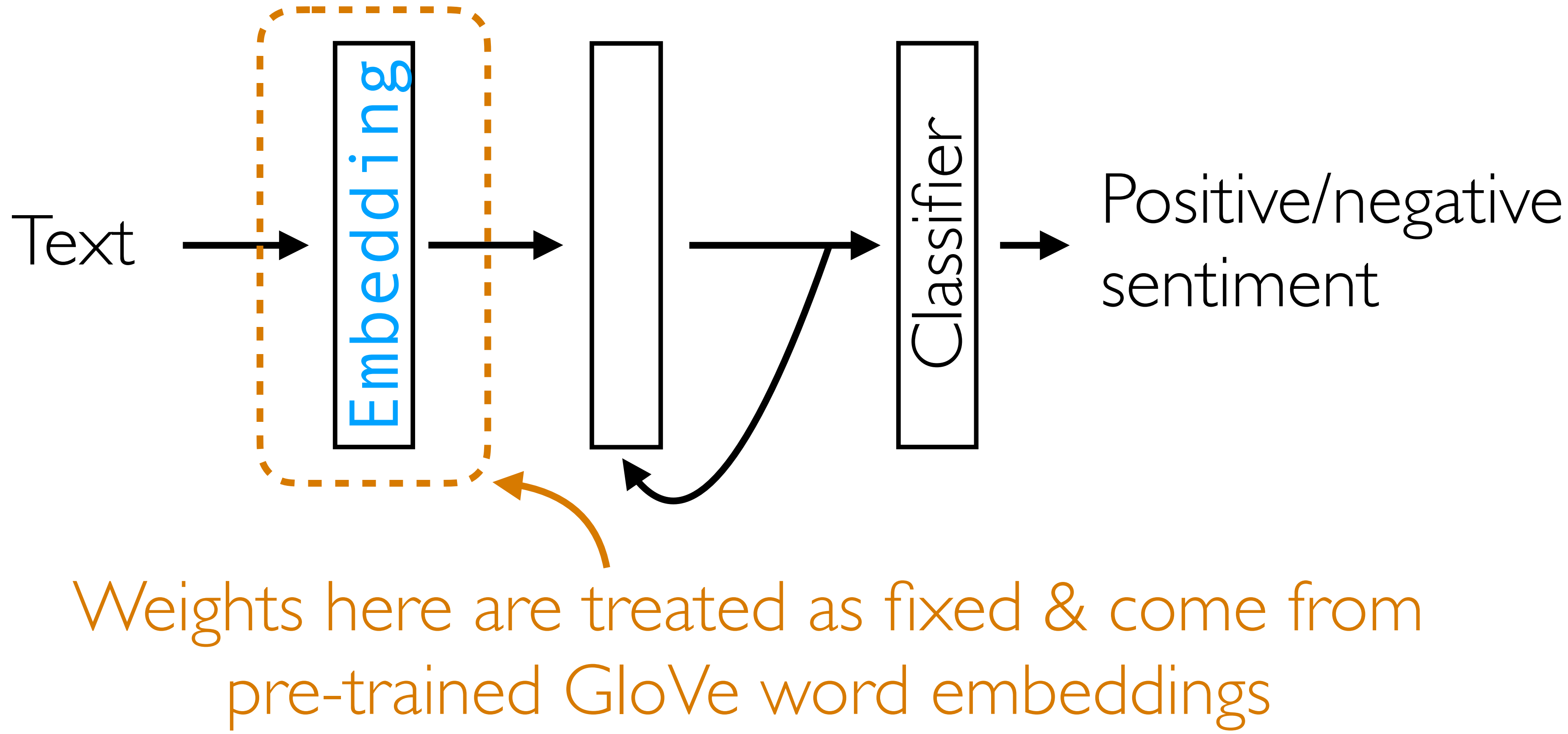
Example: classify between Tesla's and Toyota's

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

Strategy: take pre-trained convnet (such as a state-of-the-art one like ResNet, trained to classify between 1000 objects) and change final layers to do classification between Tesla's and Toyota's

Fine Tuning

Sentiment analysis RNN demo



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

IMDb review dataset is small in comparison

Word Embeddings: Even without labels, we can set up a prediction problem!

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

Word Embeddings: word2vec

Can solve tasks like the following:

Man is to King as Woman is to **???**

Word Embeddings: word2vec

Can solve tasks like the following:

Man is to King as Woman is to Queen

Word Embeddings: word2vec

Can solve tasks like the following:

Man is to King as Woman is to <u>Queen</u>

Which word doesn't belong?
blue, red, green, crimson, transparent

Word Embeddings: word2vec

Can solve tasks like the following:

Man is to King as Woman is to Queen

Which word doesn't belong?

blue, red, green, crimson, transparent

Word Embeddings: word2vec

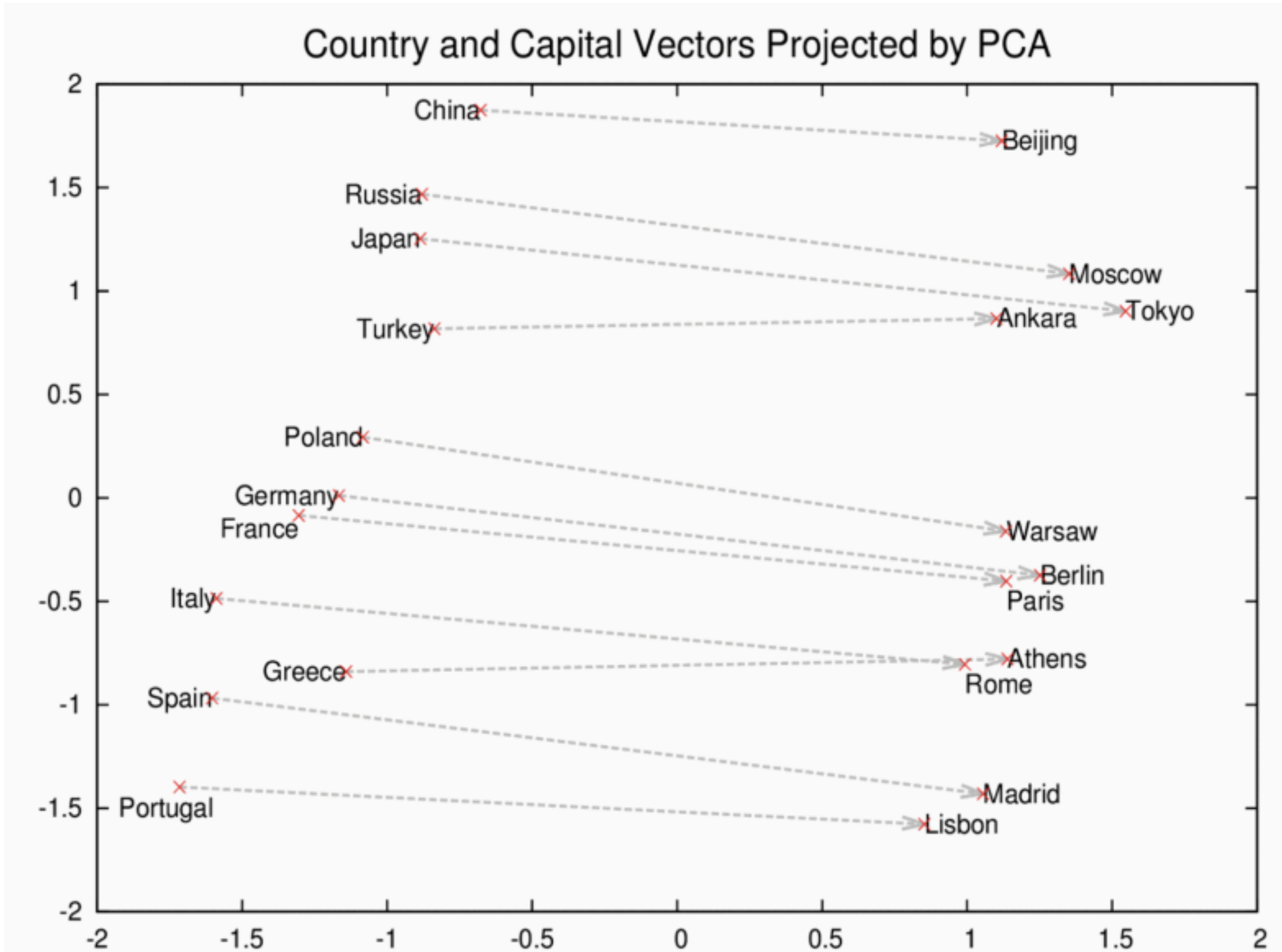


Image source: https://deeplearning4j.org/img/countries_capitals.png

Word Embeddings: word2vec

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

Training data point: epidemic

"Training labels": the, opioid, or, opioid

Word Embeddings: word2vec

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

Training data point: or

"Training labels": opioid, epidemic, opioid, crisis

Word Embeddings: word2vec

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

Predict context of each word!

These are "positive" (correct) examples of what context words are for "opioid"

Training data point: opioid

"Training labels": epidemic, or, crisis, is

Also provide "negative" examples of words that are *not* likely to be context words (by randomly sampling words elsewhere in document)

Word Embeddings: word2vec

The opioid epidemic or opioid crisis is the rapid increase in the use of prescription and non-prescription opioid drugs in the United States and Canada in the 2010s.

randomly sampled word

Predict context of each word!

Training data point: opioid

“Negative training label”: 2010s

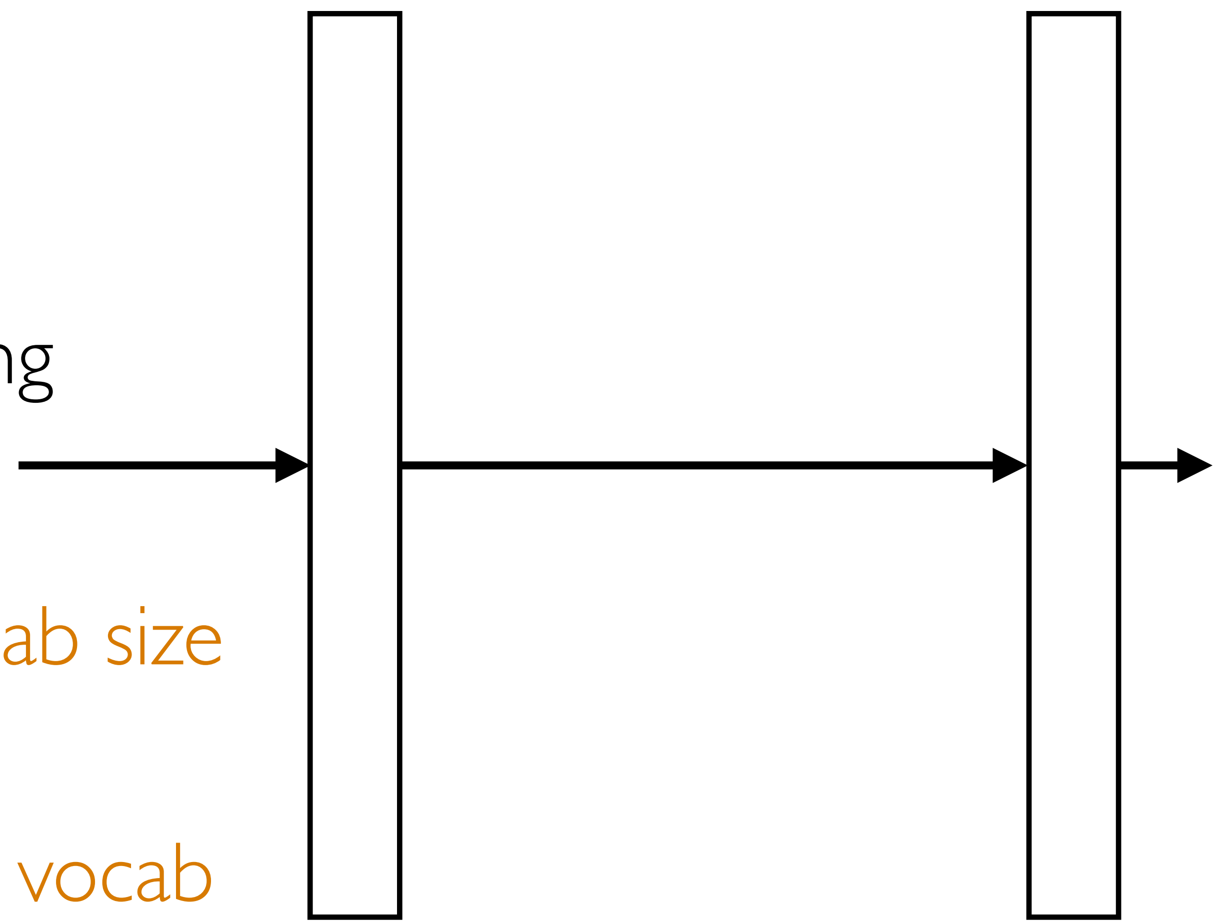
Also provide “negative” examples of words that are *not* likely to be context words (by randomly sampling words elsewhere in document)

Word2vec Neural Net

"opioid"
Use one-hot encoding
[0, 0, . . ., 1, . . ., 0]

vector length = vocab size

index of "opioid" in vocab



Want real context words (e.g., "epidemic", "crisis") to have high probability

Linear
(100 nodes)

Learned weight matrix used as word embedding!

Linear
(# nodes = vocab size),
Softmax

(Treat ***i***-th col of weight matrix as word embedding for ***i***-th word)

Word Embeddings as a Special Case of *Self-Supervised Learning*

- Key idea: hide part of the training data and try to predict hidden part using other parts of the training data
- No actual training labels required — we are defining what the training labels are just using the unlabeled training data!
- This is an *unsupervised* method that sets up a *supervised prediction* task
- Other word embeddings methods are possible

(Flashback)

What about a word that has multiple meanings?

Challenging: try to split up word into multiple words depending on meaning (requires inferring meaning from context)

This problem is called **word sense disambiguation** (WSD)

Word Embeddings as a Special Case of *Self-Supervised Learning*

- Key idea: hide part of the training data and try to predict hidden part using other parts of the training data
- No actual training labels required — we are defining what the training labels are just using the unlabeled training data!
- This is an *unsupervised* method that sets up a *supervised prediction* task
- Other word embeddings methods are possible
  - Word embedding that handles word-sense disambiguation: BERT (to figure out embedding for word, provide sentence the word is used in)

Unstructured Data Analysis

Question → Data → Finding Structure → Insights

Becoming good at data scientist requires you to think like a detective!

The dead body | *The evidence* | *Puzzle solving,* | *When? Where?*

Much like how some murder mysteries are hard to solve, many data analysis problems (unstructured or not) are hard to solve too!

Th… by …

more evidence!

analysis

catchable?

Answer original question

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

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
    - 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!**

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
- One of the best ways to learn material is to teach it!

Apply to be a TA for me next term!