Carnegie Mellon University
Heinz College

Unstructured Data Analysis

Lecture 14: Time series analysis with recurrent neural nets; some other deep learning topics; course wrap-up

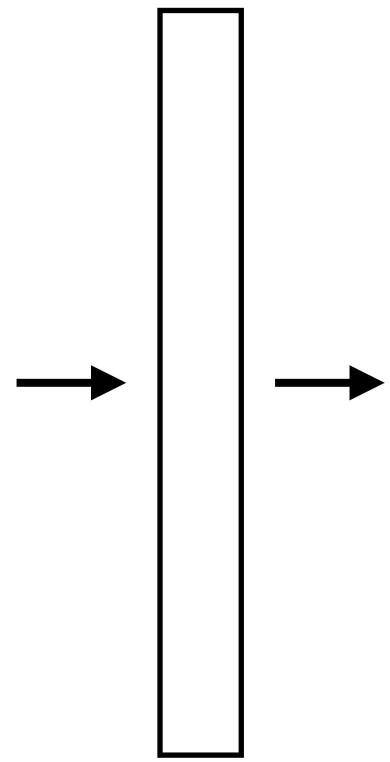
George Chen

Last Lecture!

- More on deep learning:
  - Time series analysis with recurrent neural nets
    - **Demo is supplemental & posted on course webpage**
  - I'll also talk about some other deep learning topics
    - Self-supervised learning and *word embeddings*
    - Roughly how learning a neural net works
    - How to deal with small datasets
    - Generating fake data that look real
    - AI agents that interact with environments
- I'll end with a course wrap-up

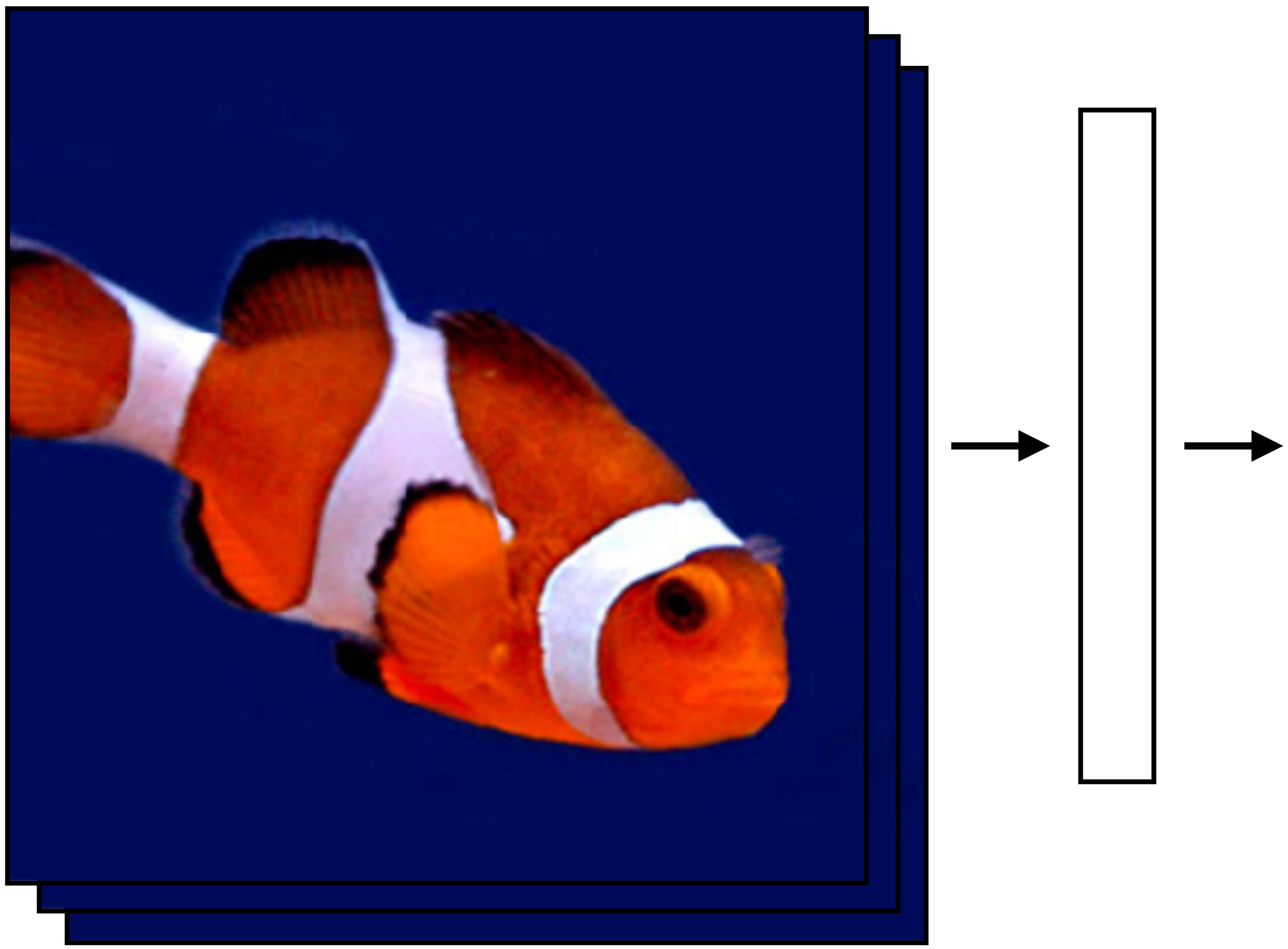
Sequence Data

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



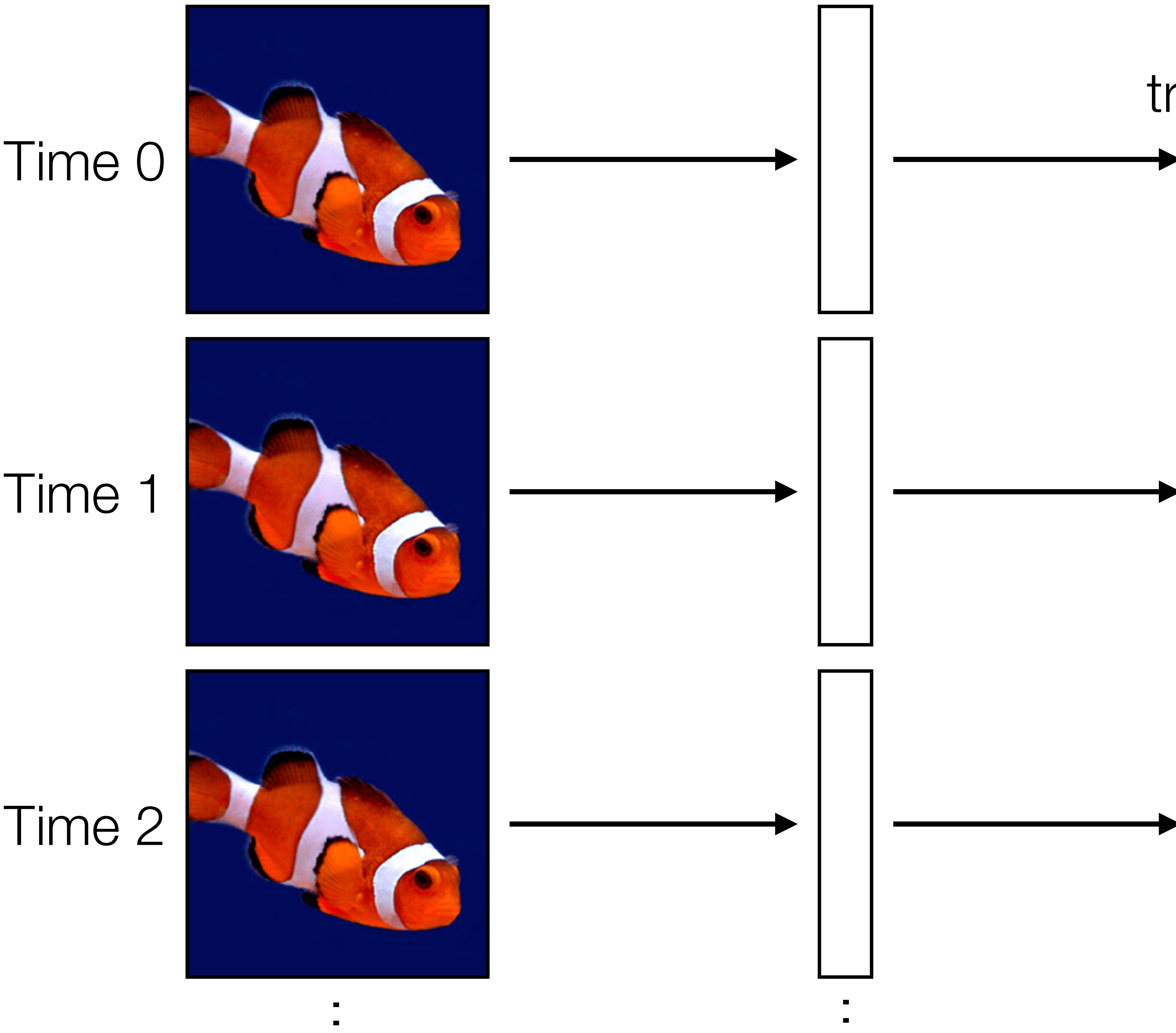
Sequence Data

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



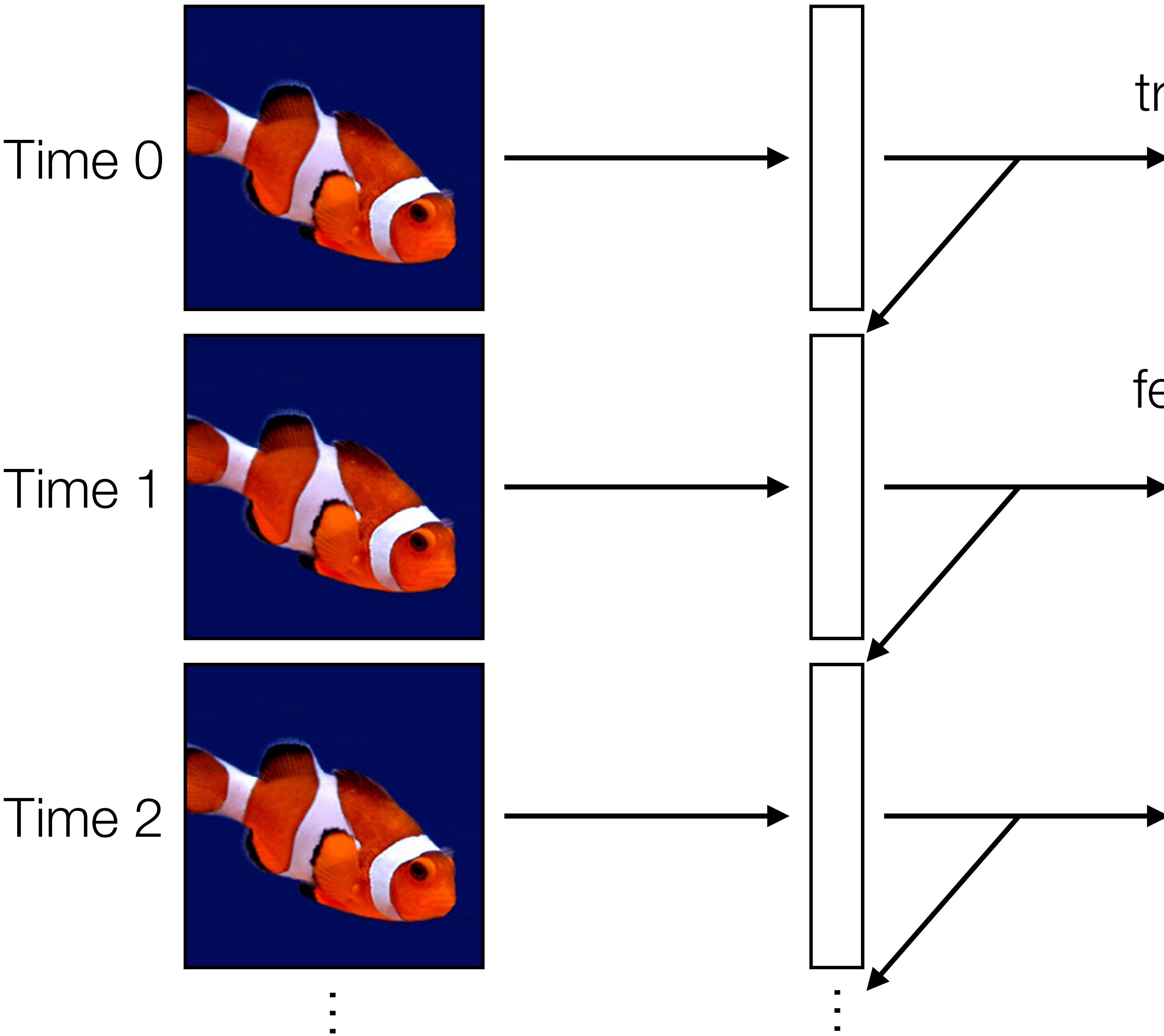
What if we had a video?

Recurrent Neural Nets



Feedforward NN's:
treat each video frame
separately

Recurrent Neural Nets

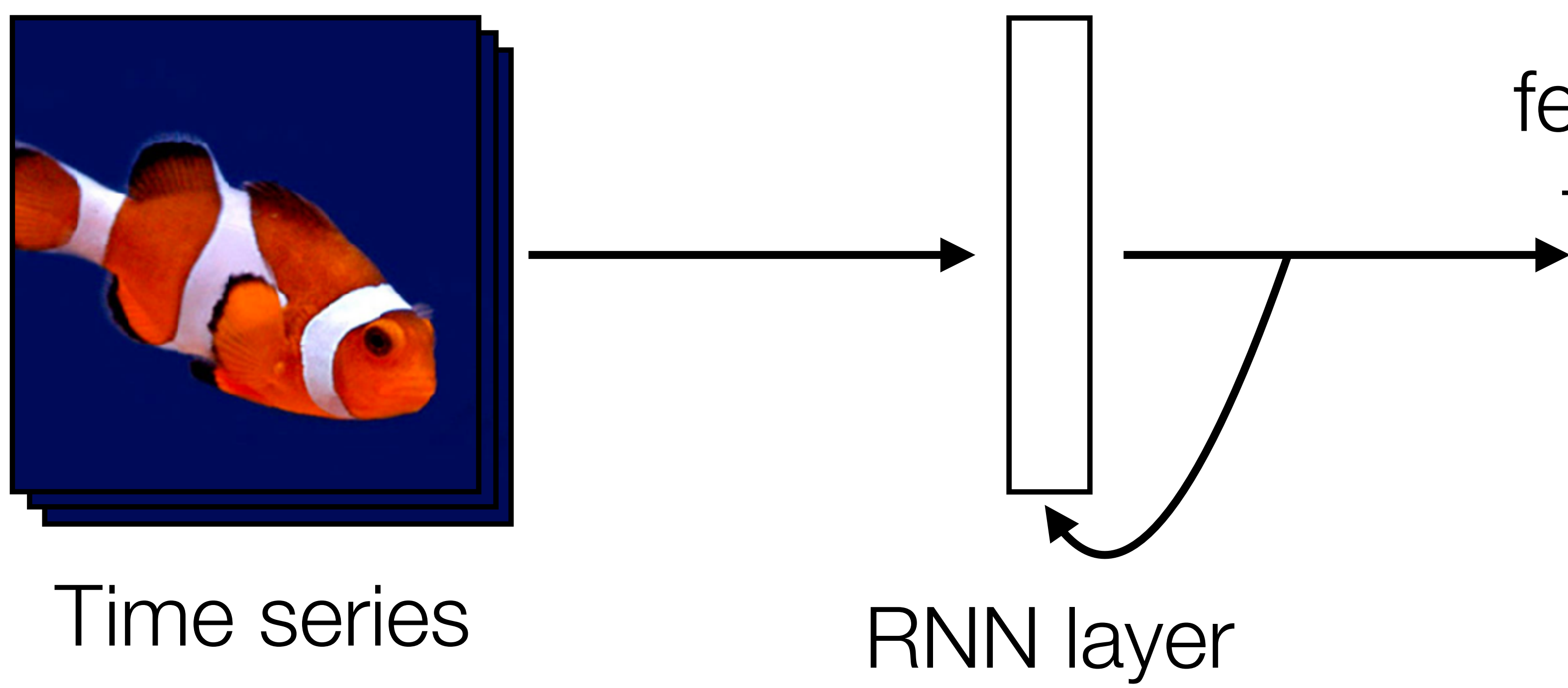


Feedforward NN's:
treat each video frame separately

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

In PyTorch, different RNN options:
`RNN` (vanilla), `LSTM`, `GRU`

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

In PyTorch, different RNN options:
`RNN` (vanilla),
`LSTM`, `GRU`

Vanilla ReLU RNN

memory that evolves over time; we want to learn how it changes

```
current_state = np.zeros(num_nodes)

for input in input_sequence:

    linear = np.dot(input, W) \
             + np.dot(current_state, U) \
             + b

    output = np.maximum(0, linear) # ReLU

    current_state = output
```

`W` is a 2D table: # rows: (length of single time step's `input`), # cols: `num_nodes`

`U` is a 2D table: `num_nodes` by `num_nodes`

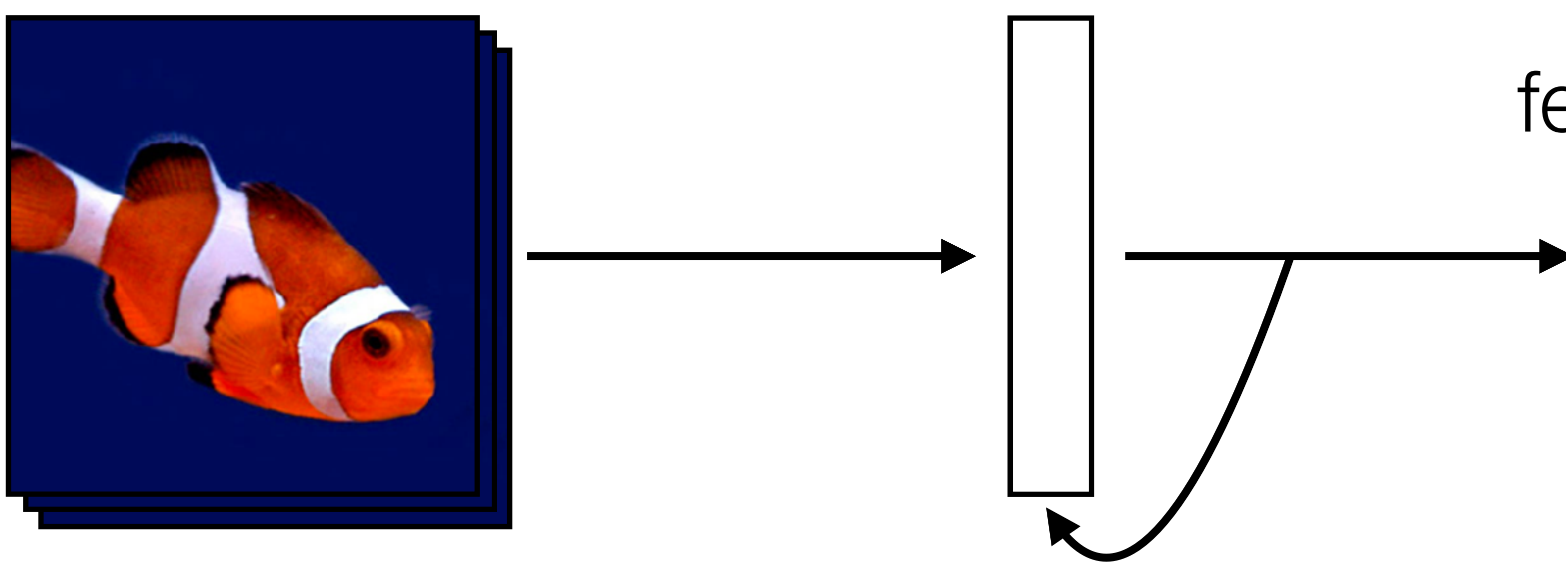
`b` is a 1D table: `num_nodes` entries

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

Key idea: **it's like a linear layer in a `for` loop that tracks how memory changes over time**

Recurrent Neural Nets

readily chains together with
other neural net layers



Time series

RNN layer

like a linear layer
that has memory

does not incorporate
image structure!!!

Feedforward NN's:
treat each video frame
separately

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

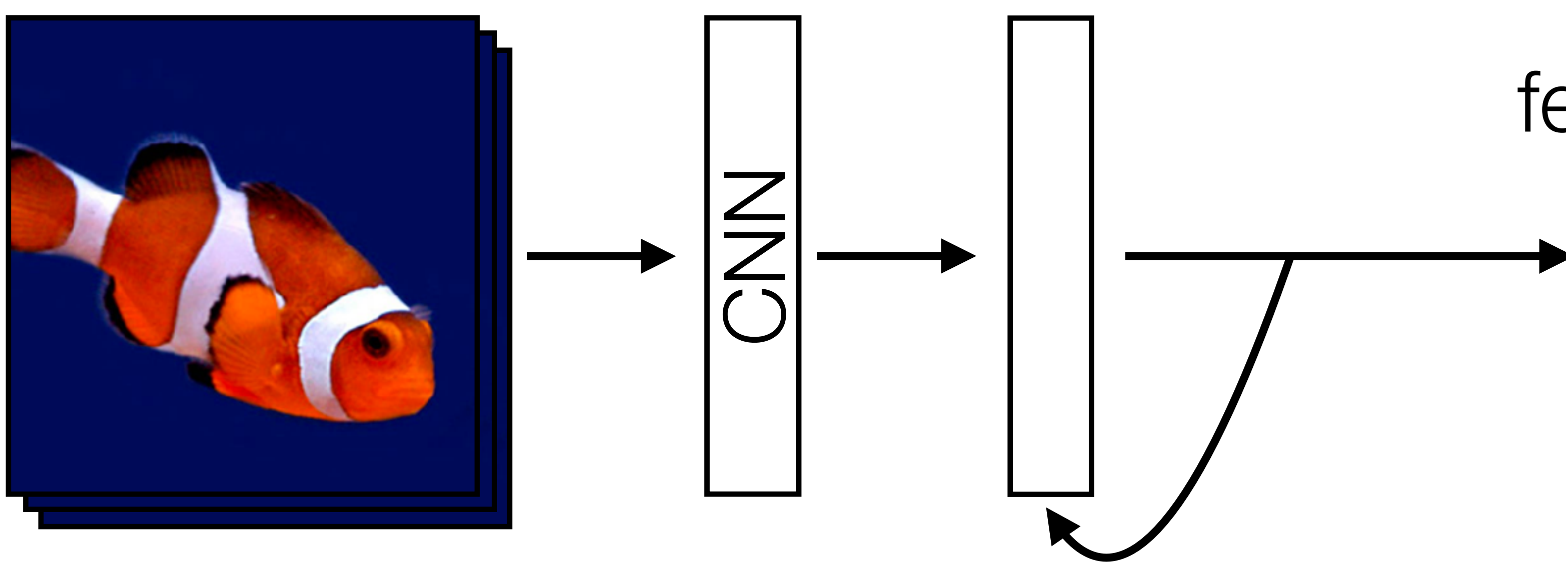
In PyTorch, different
RNN options:
`RNN` (vanilla),
`LSTM`, `GRU`

Recurrent Neural Nets

Feedforward NN's:
treat each video frame
separately

readily chains together with
other neural net layers

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



Time series

RNN layer

like a linear layer
that has memory

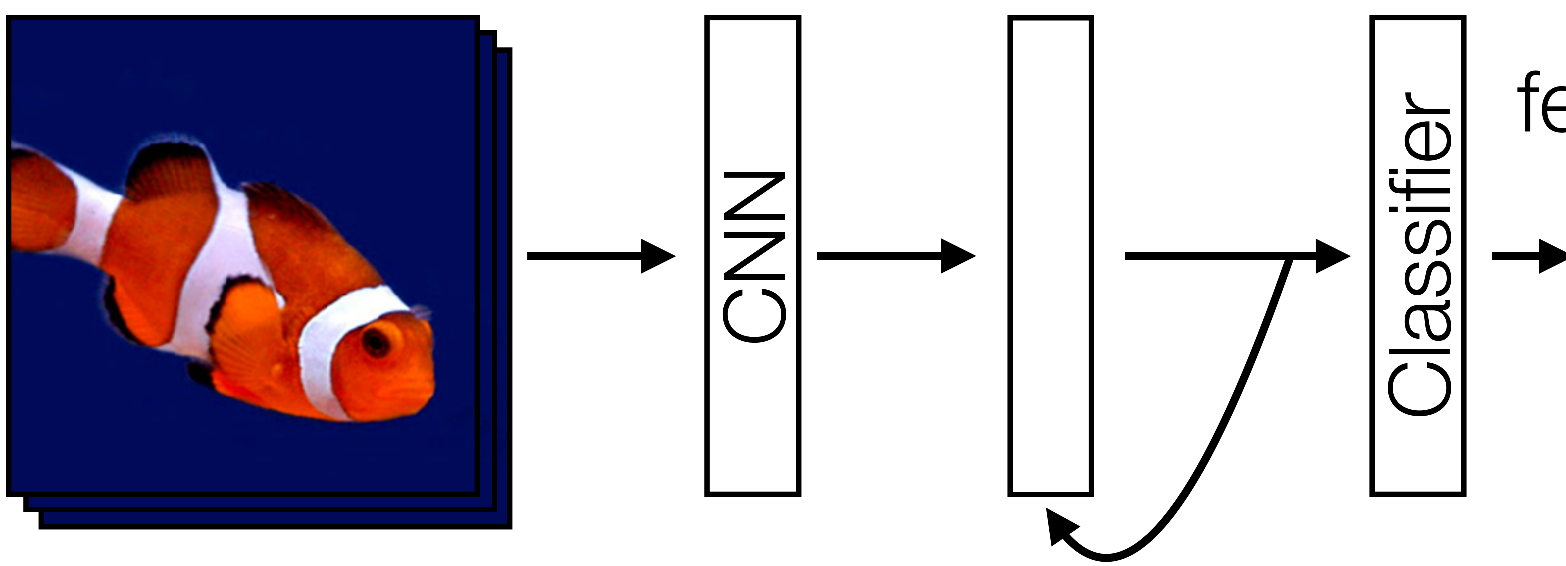
does not incorporate
image structure!!!

In PyTorch, different
RNN options:
`RNN` (vanilla),
`LSTM`, `GRU`

Recurrent Neural Nets

Feedforward NN's:
treat each video frame separately

readily chains together with other neural net layers



Time series

RNN layer

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

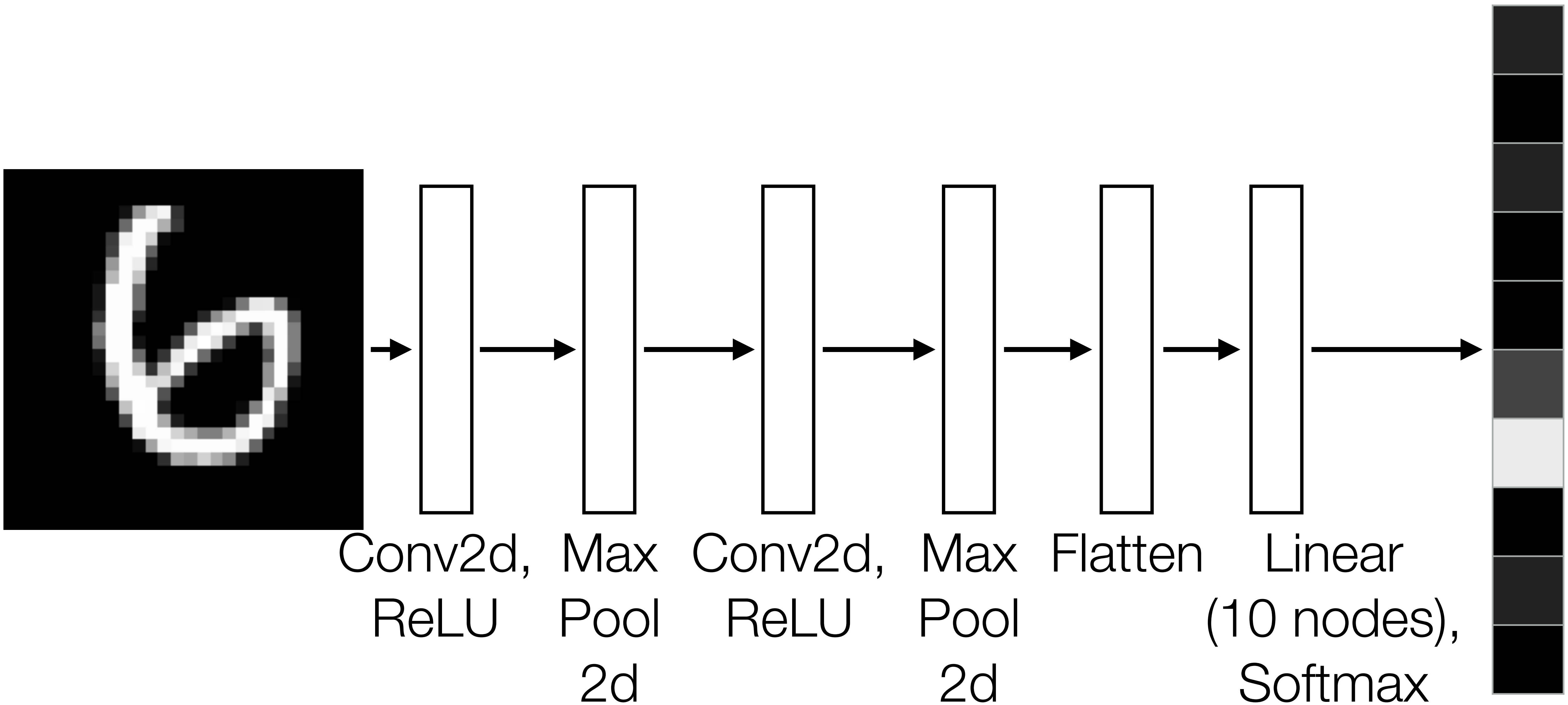
Use CNN to incorporate image structure!

like a linear layer that has memory

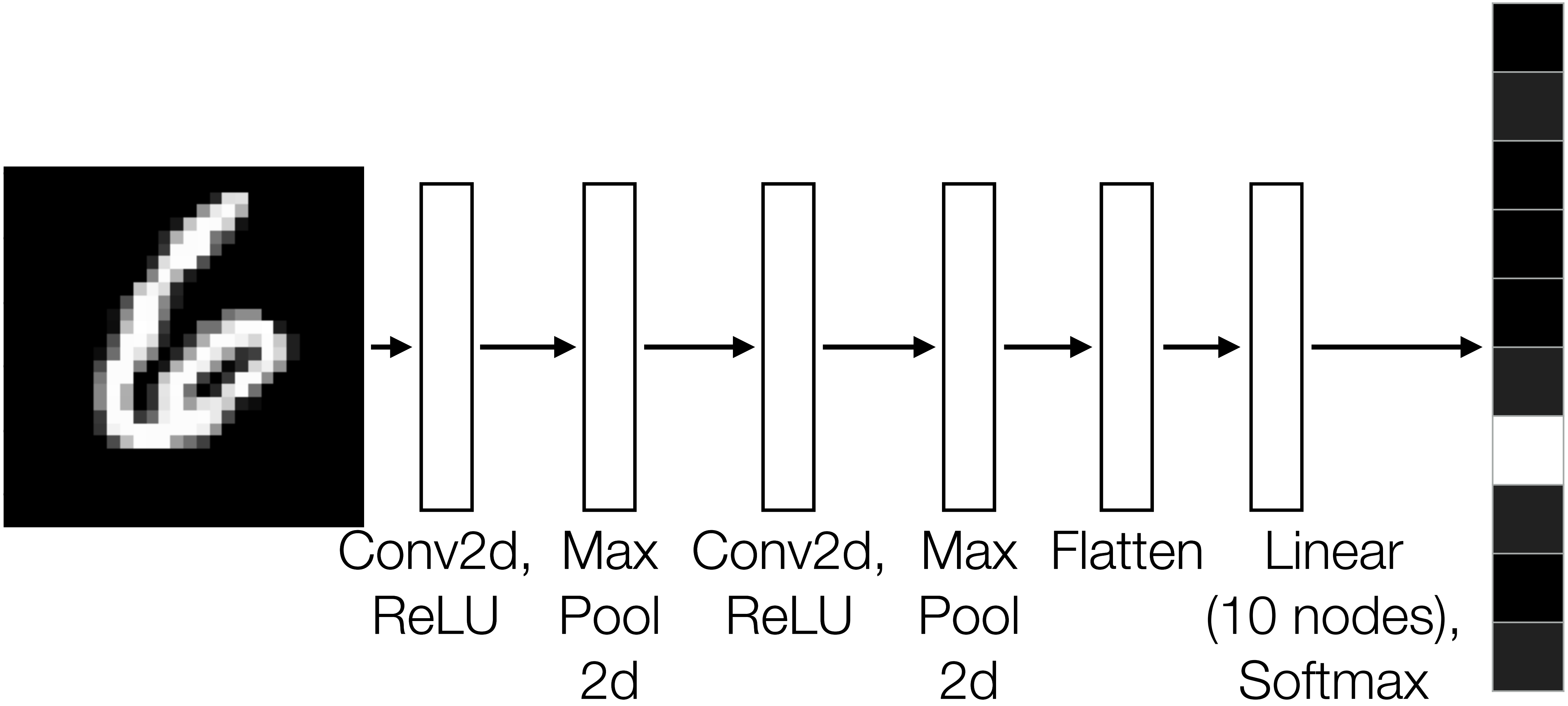
does not incorporate image structure!!!

In PyTorch, different RNN options:
`RNN` (vanilla),
`LSTM`, `GRU`

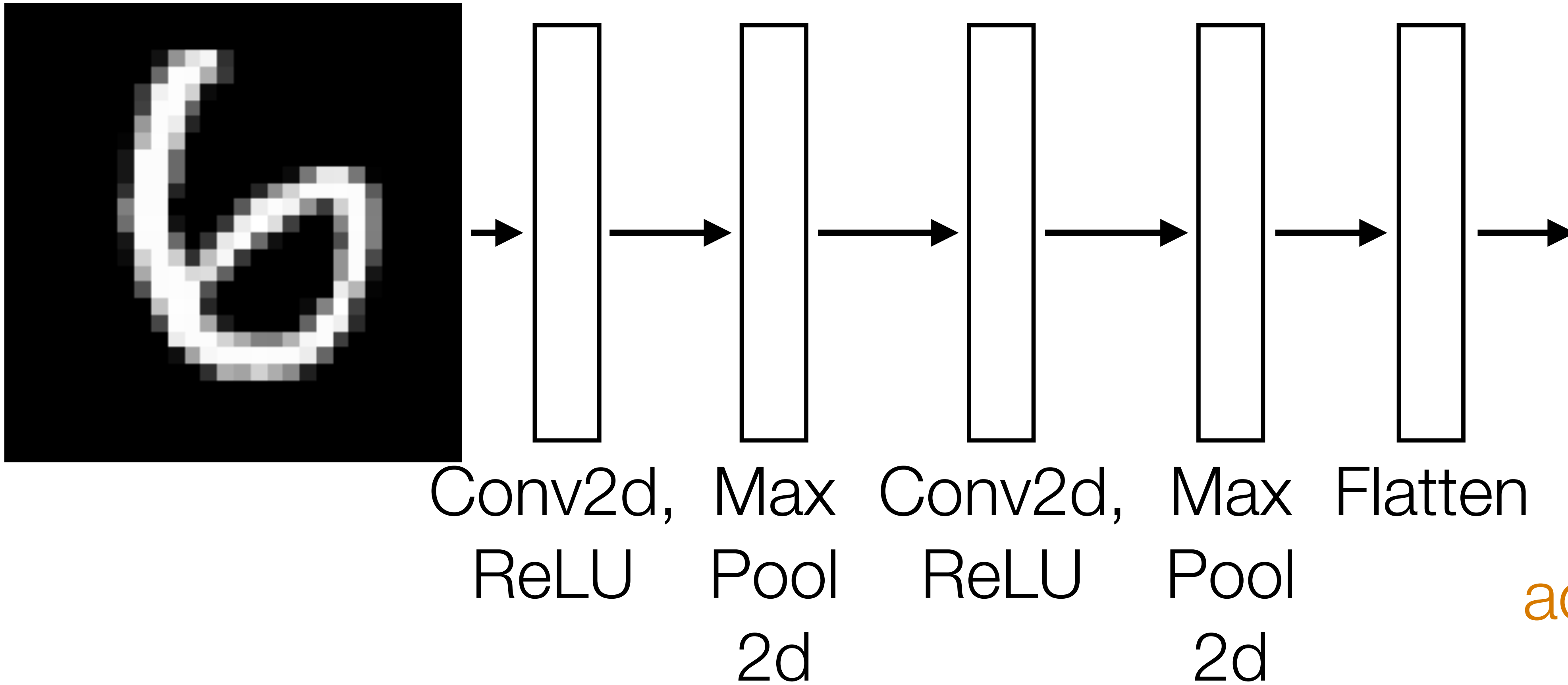
Intuition: CNNs Encode Semantic Structure for Images



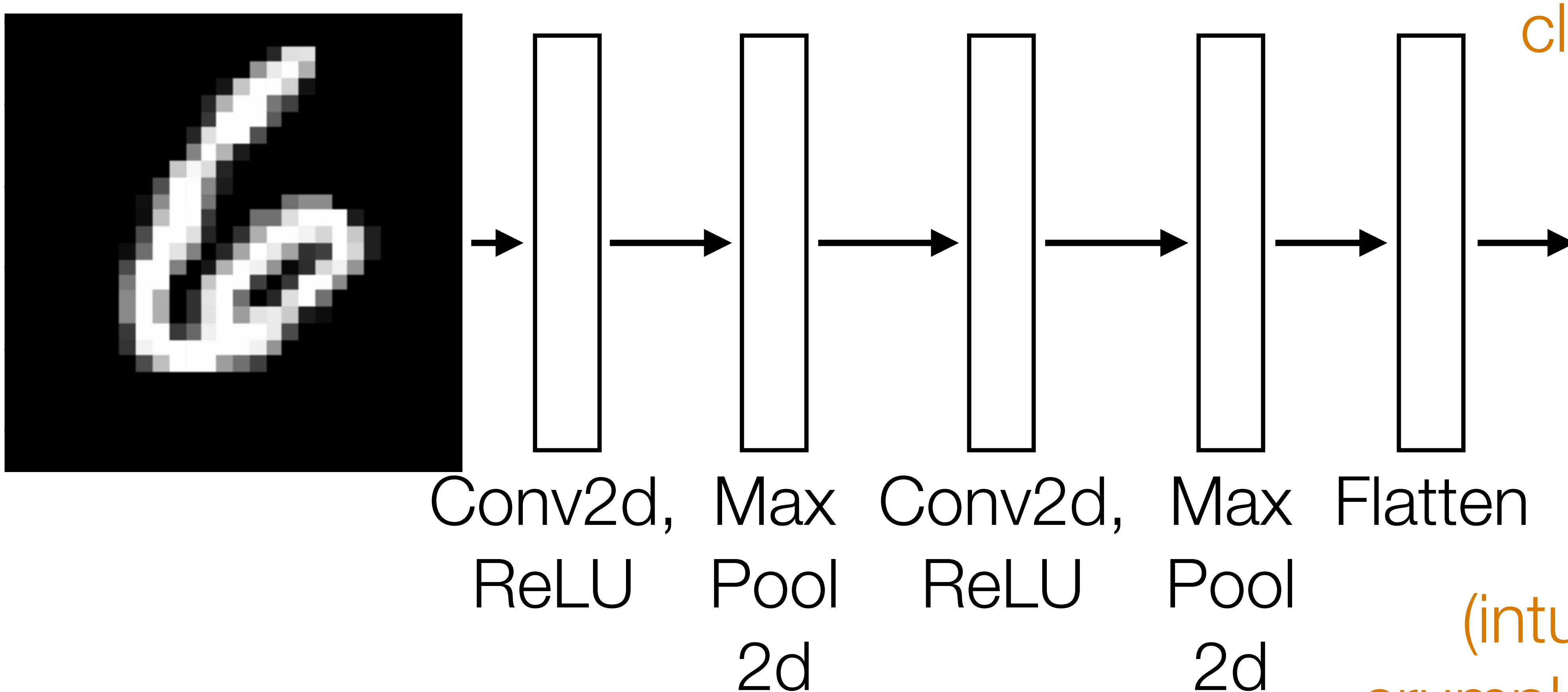
Intuition: CNNs Encode Semantic Structure for Images



final output for different input 6’s is similar



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

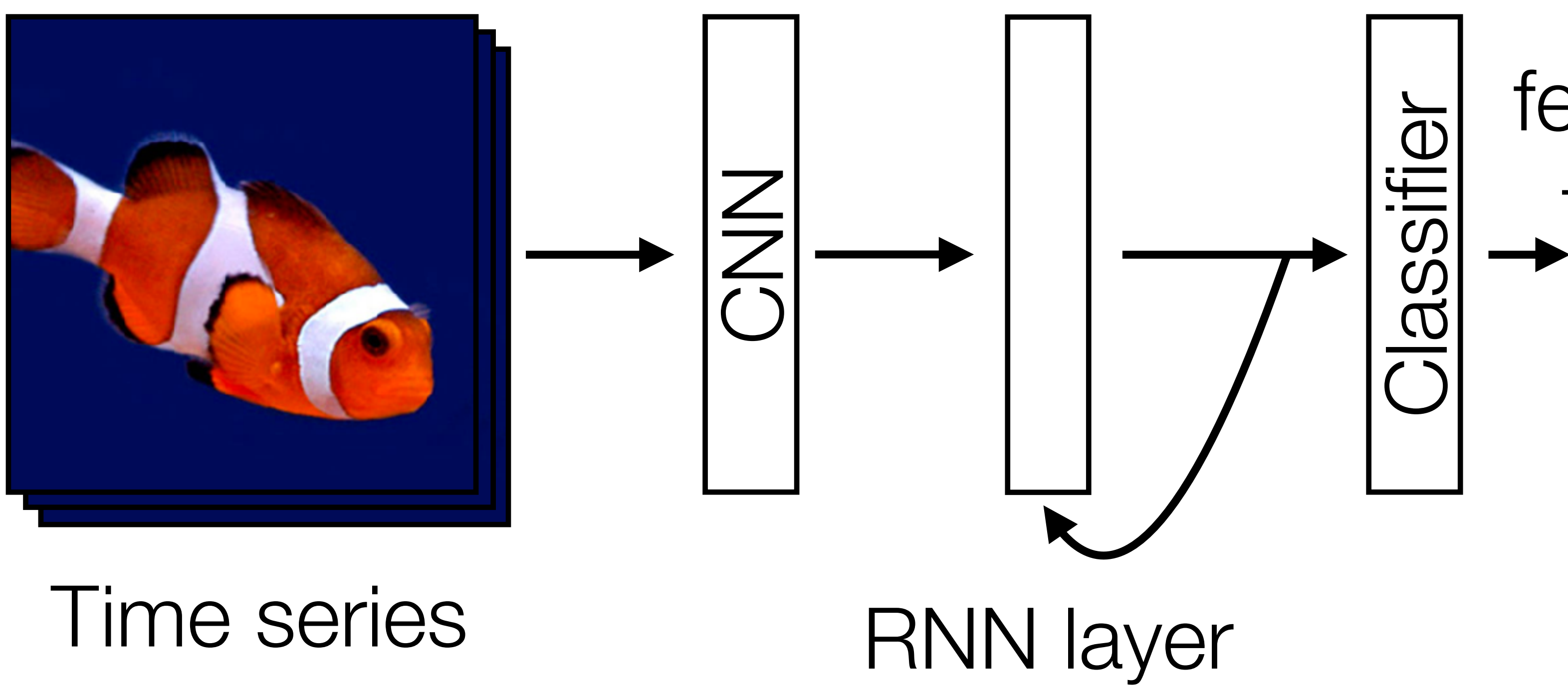


(intuition: recall the crumpled paper analogy!)

Recurrent Neural Nets

Feedforward NN's:
treat each video frame separately

readily chains together with other neural net layers



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

Use CNN to incorporate image structure!

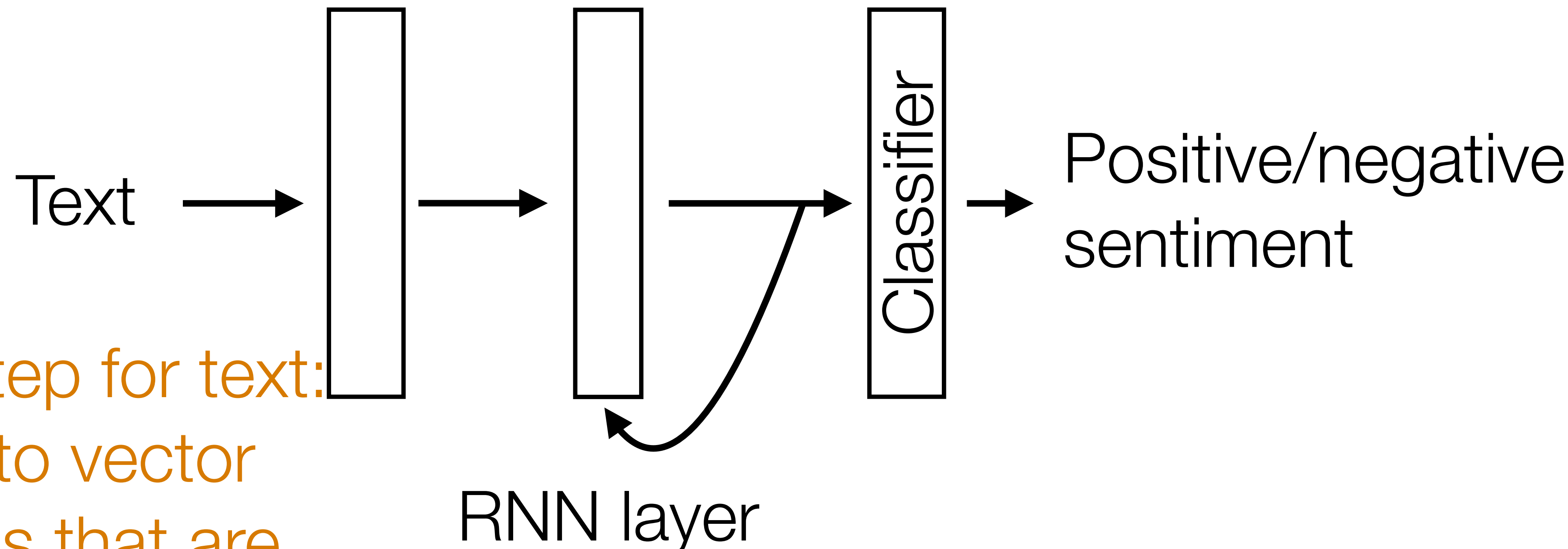
like a linear layer that has memory

does not incorporate image structure!!!

In PyTorch, different RNN options:
`RNN` (vanilla),
`LSTM`, `GRU`

Recurrent Neural Nets

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



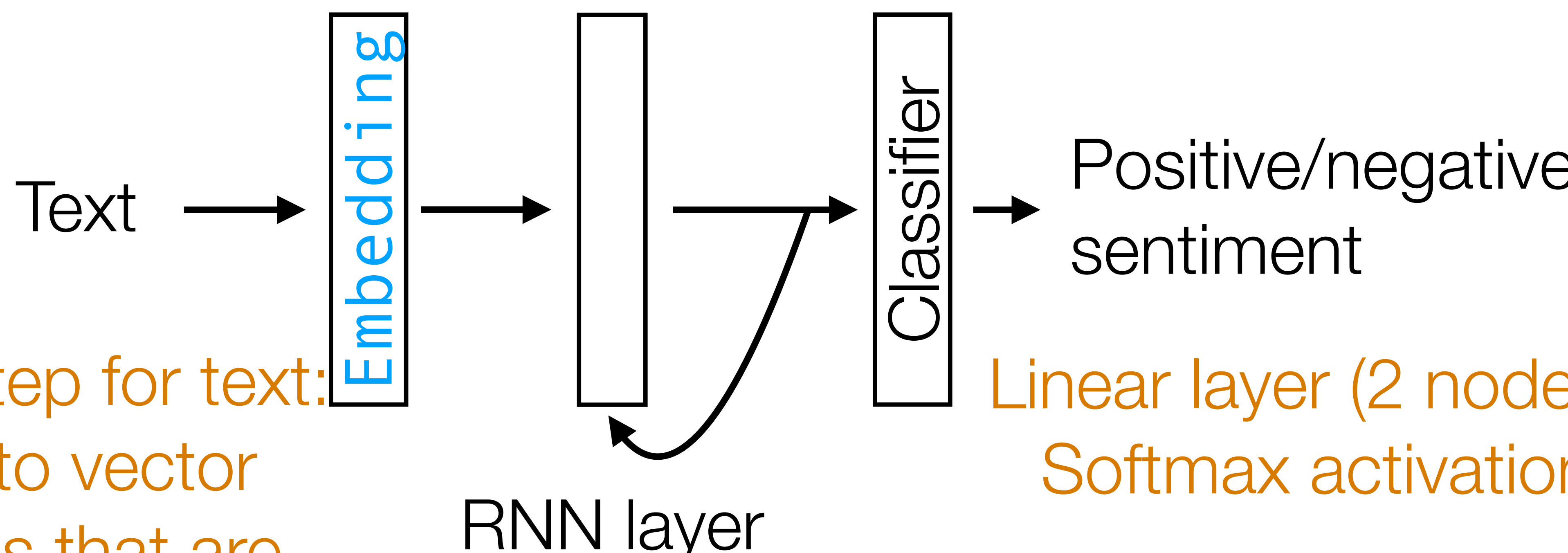
Common first step for text: turn words into vector representations that are semantically meaningful

(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)

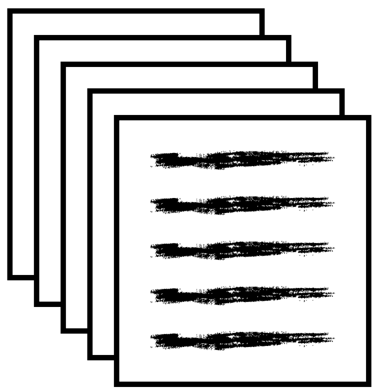


Common first step for text: turn words into vector representations that are semantically meaningful

In PyTorch, use the `Embedding` layer

Linear layer (2 nodes), Softmax activation

Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

Step 2: Encode each review as a sequence of word indices into the vocab

Ordering of words matters

Different reviews can have different lengths

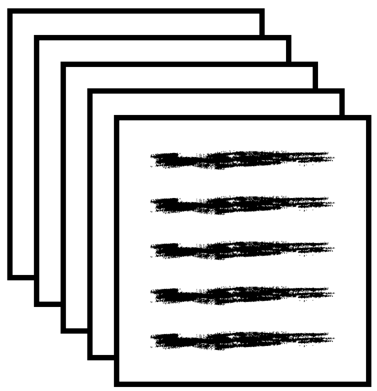
"this movie rocks" → 0 1 2

"this movie sucks" → 0 1 3

"this sucks" → 0 3

Step 3: Use word embeddings to represent each word

Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

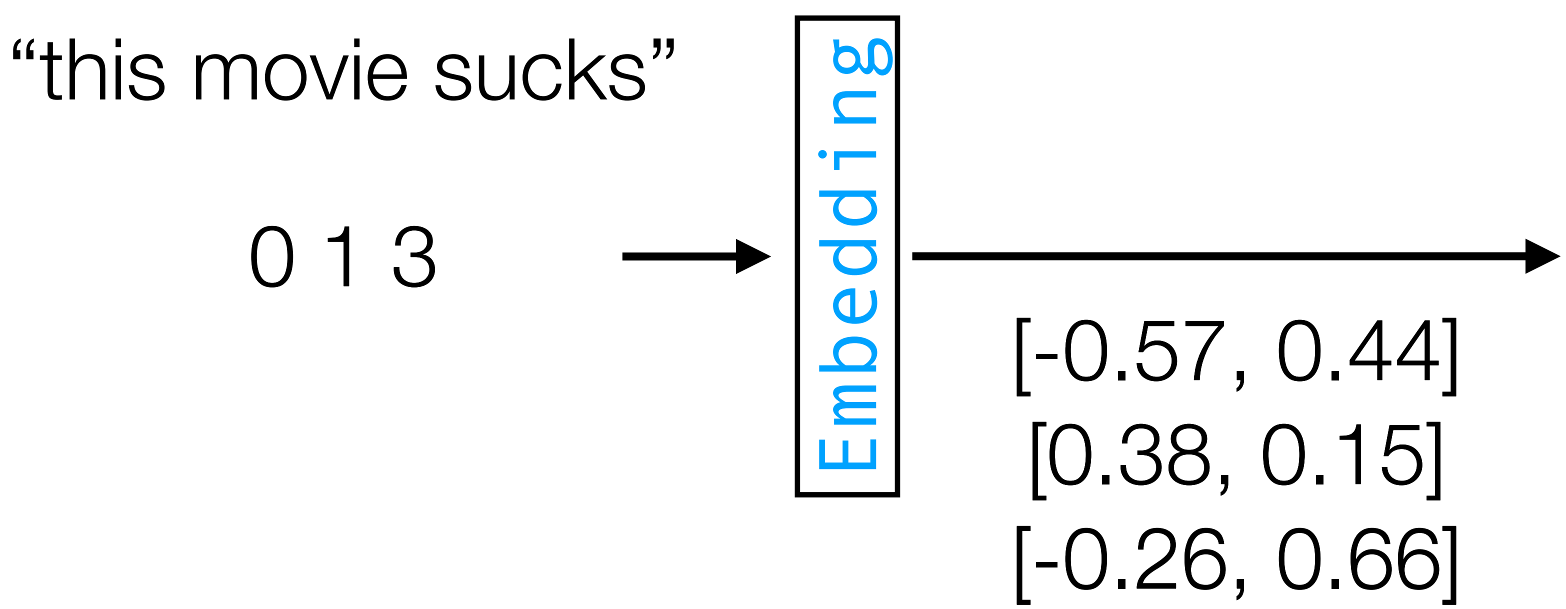
Step 2: Encode each review as a sequence of word indices into the vocab

"this movie sucks" → 0 1 3

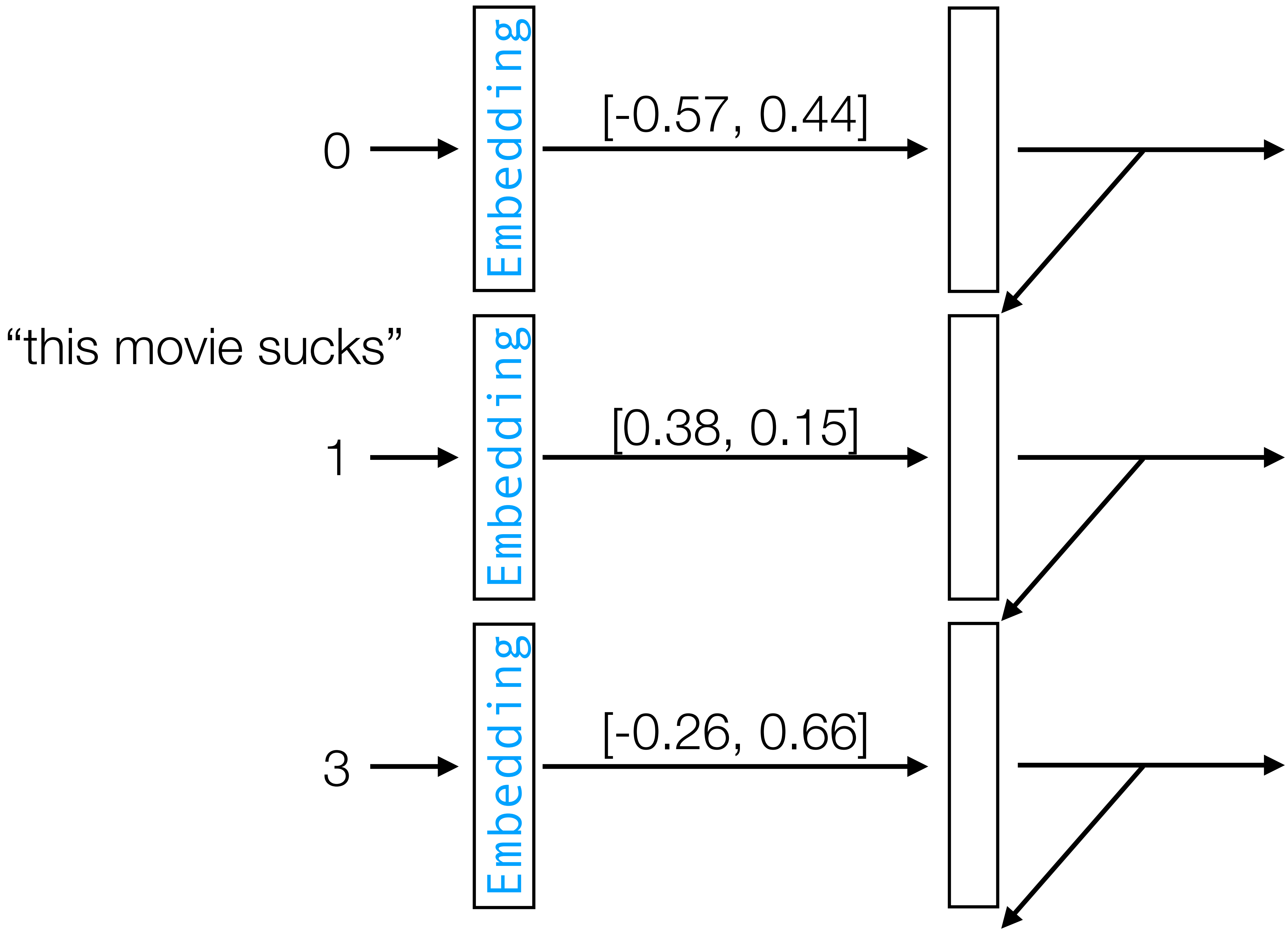
Step 3: Use word embeddings to represent each word

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]

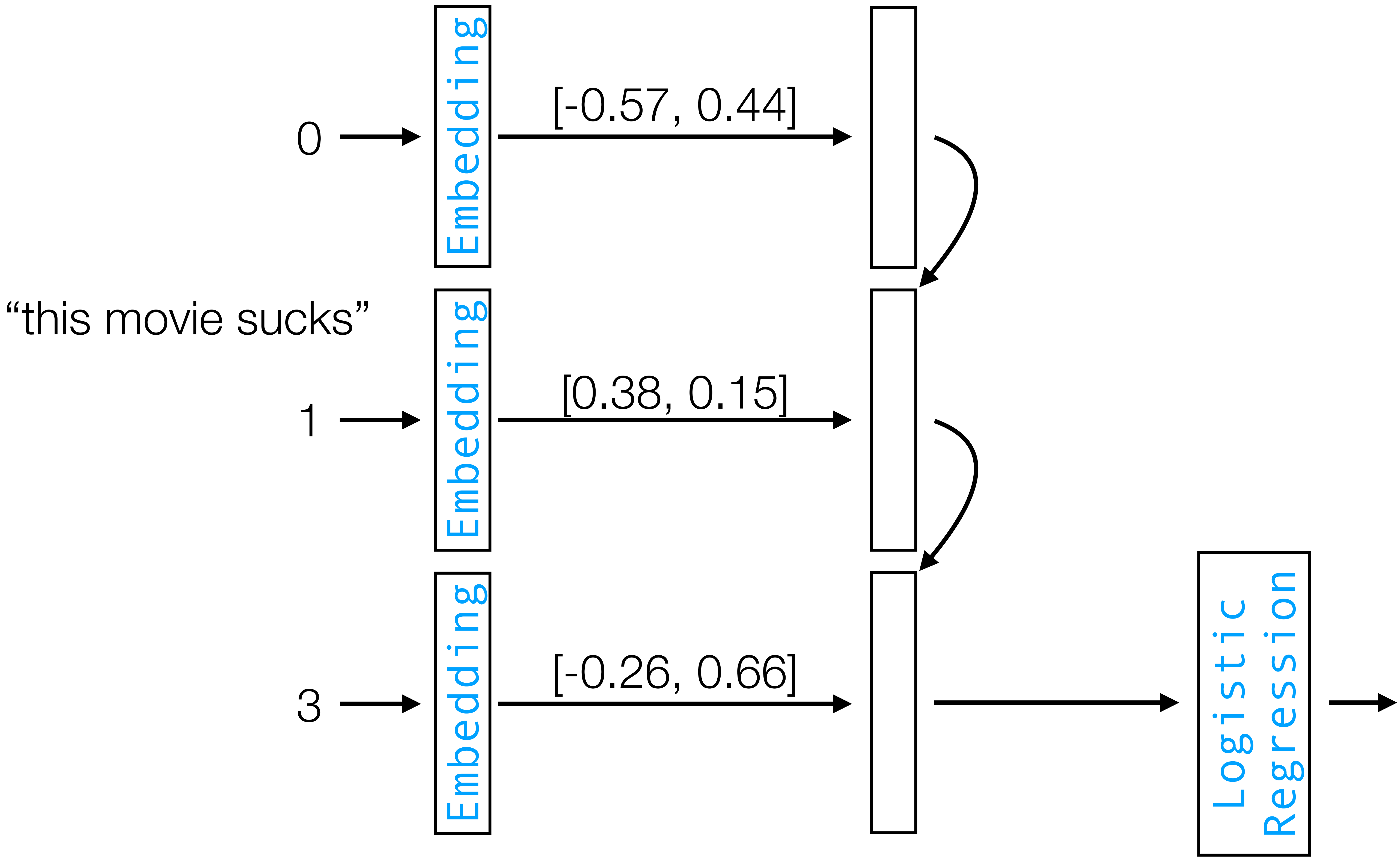
Sentiment Analysis with IMDb Reviews



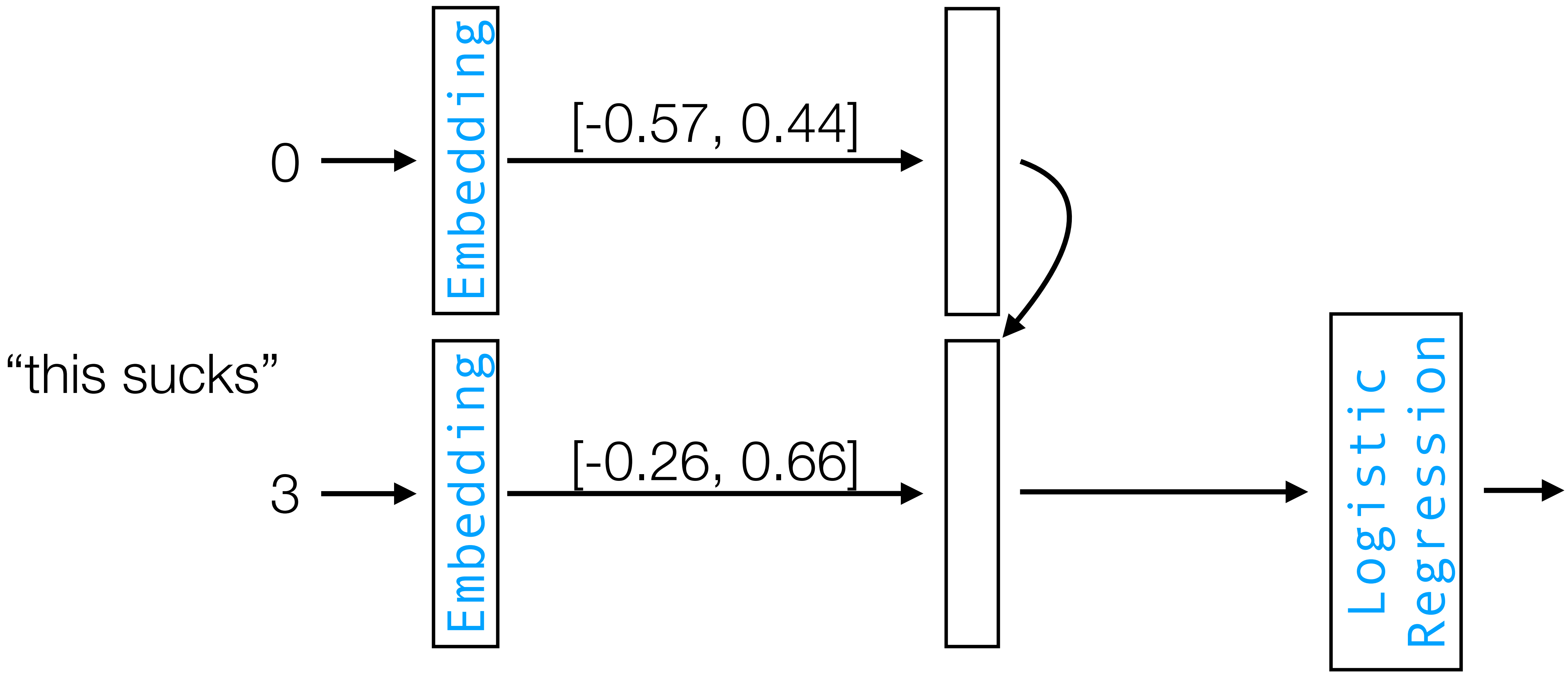
Sentiment Analysis with IMDb Reviews



Sentiment Analysis with IMDb Reviews



Sentiment Analysis with IMDb Reviews

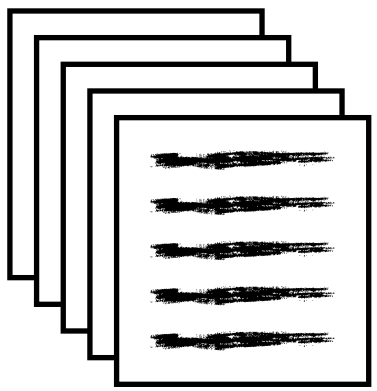


RNN's work with variable-length inputs

Note: Often in text analysis, the word embeddings are treated as fixed, so we do *not* update them during training

What if we didn’t use word embeddings?

Sentiment Analysis with IMDb Reviews



Training reviews

Step 1: Tokenize & build vocabulary

Word index	Word	2D Embedding
0	this	[-0.57, 0.44]
1	movie	[0.38, 0.15]
2	rocks	[-0.85, 0.70]
3	sucks	[-0.26, 0.66]

Step 2: Encode each review as a sequence of word indices into the vocab

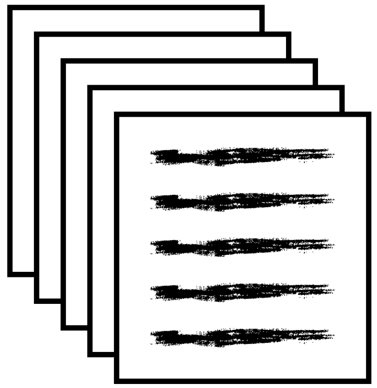
"this movie sucks" → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]

Bad Strategy: One-Hot Encoding

Step 1: Tokenize & build vocabulary



Word index	Word	One-hot encoding
0	this	[1, 0, 0, 0]
1	movie	[0, 1, 0, 0]
2	rocks	[0, 0, 1, 0]
3	sucks	[0, 0, 0, 1]

Step 2: Encode each review as a sequence of word indices into the vocab

"this movie sucks" → 0 1 3

Step 3: Use one-hot encoding to represent each word

[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 0, 1]

This strategy tends to work poorly in practice: distance between every pair of words is the same in one-hot encoding!

Recap/Important Reminder

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

Sentiment Analysis with IMDb Reviews

Supplemental demo posted on course webpage;
uses a better kind of RNN (called an LSTM)
compared to the vanilla ReLU RNN
(LSTM's remember things for longer periods of time)

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 does not have long-range dependencies that require long-term memory, CNNs can do well already!
    - If you need long-term memory, use RNNs

Some Other Deep Learning Topics

Self-Supervised Learning

Self-Supervised Learning

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

Example: word embeddings like word2vec, GloVe

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 label”: the, opioid, or, opioid

Self-Supervised Learning

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

Example: word embeddings like word2vec, GloVe

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 label”: opioid, epidemic, opioid, crisis

Self-Supervised Learning

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

Example: word embeddings like word2vec, GloVe

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!

There are "positive" examples of what context words are for "opioid"

Training data point: opioid

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

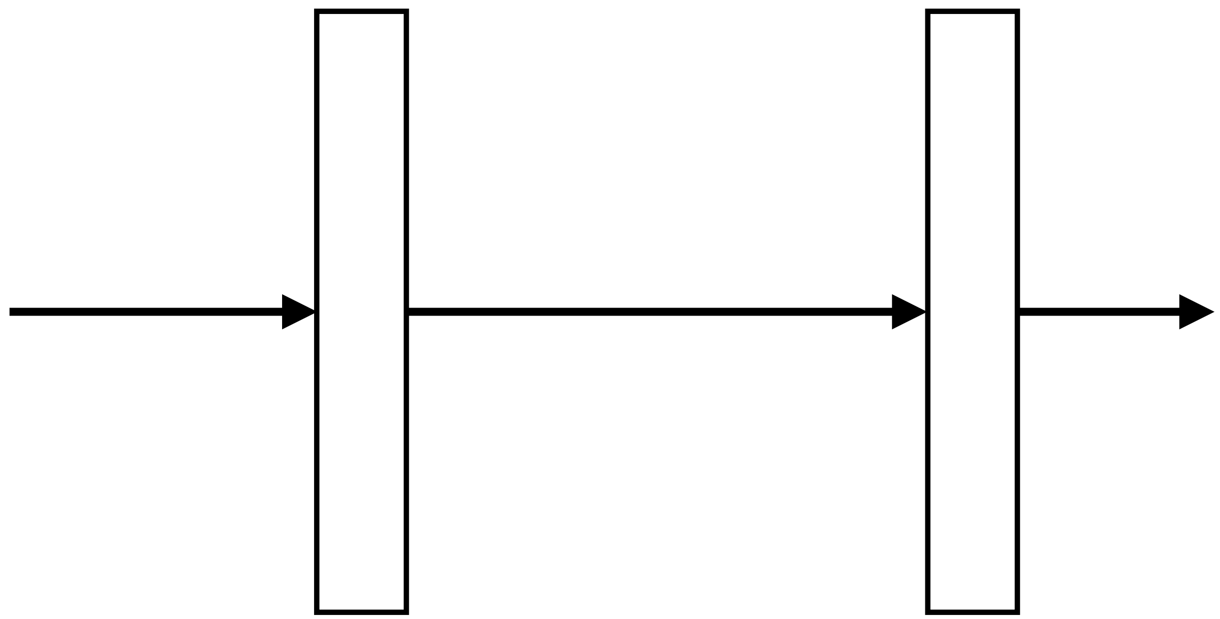
Also provide "negative" examples of words that are *not* likely to be context words (e.g., randomly sample words elsewhere in document)

Self-Supervised Learning

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

Example: word embeddings like word2vec, GloVe

Input word (categorical "one hot" encoding)



Vector saying the probabilities of different words being context words

This actually relates to PMI!

Linear layer,
nodes equal to "embedding dim"

Linear layer,
nodes equal to vocab size, softmax activation

Weight matrix: (vocab size) by (embedding dim)

Dictionary word *i* has "word embedding" given by row *i* of weight matrix

Self-Supervised Learning

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

- Key idea: predict part of the training data from other parts of the training data

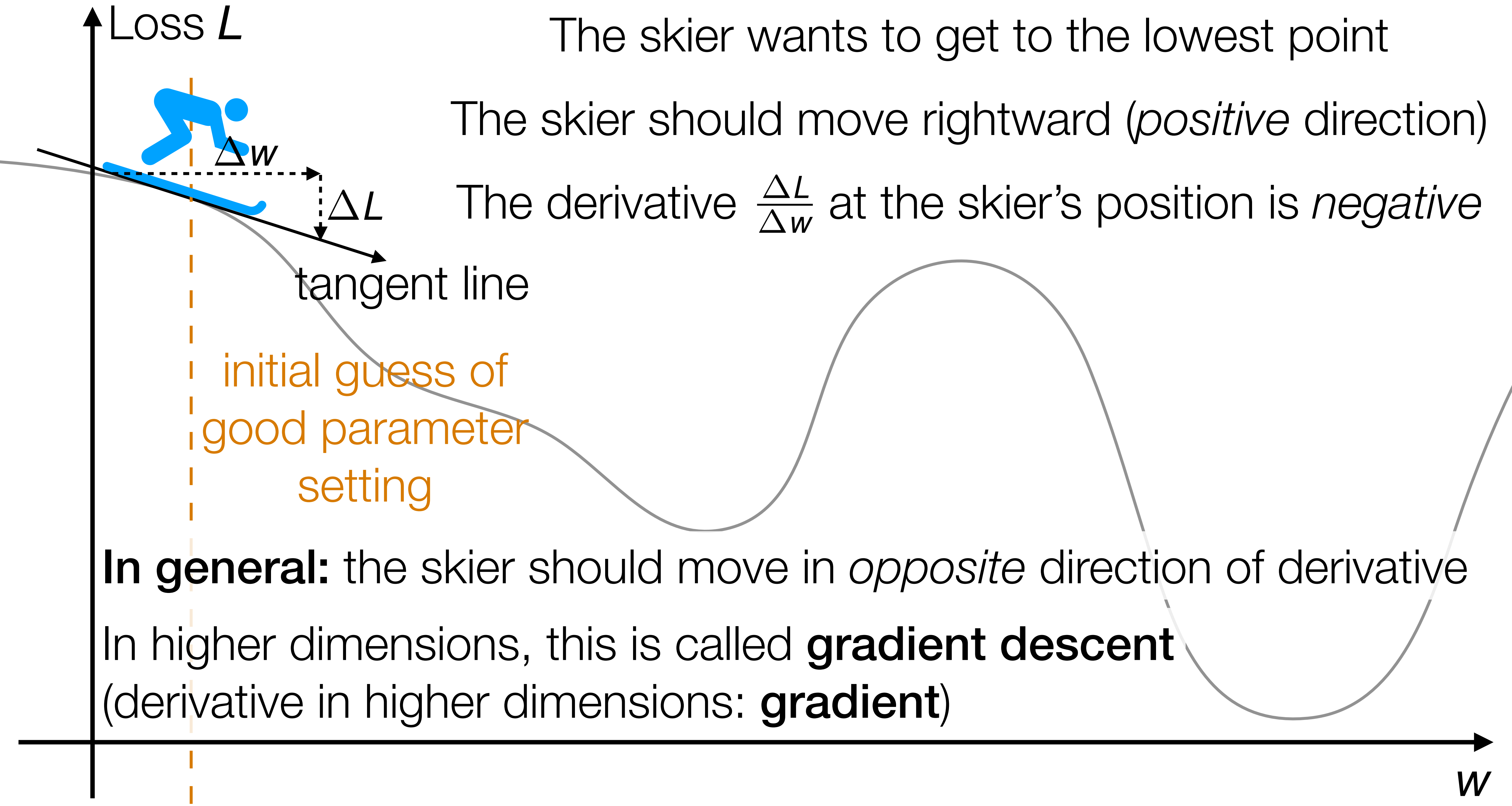
- No actual training labels required — we are defining what the training labels are just using the unlabeled training data

- This is an *unsupervised* method that sets up a *supervised prediction* task

Learning a Deep Net

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)

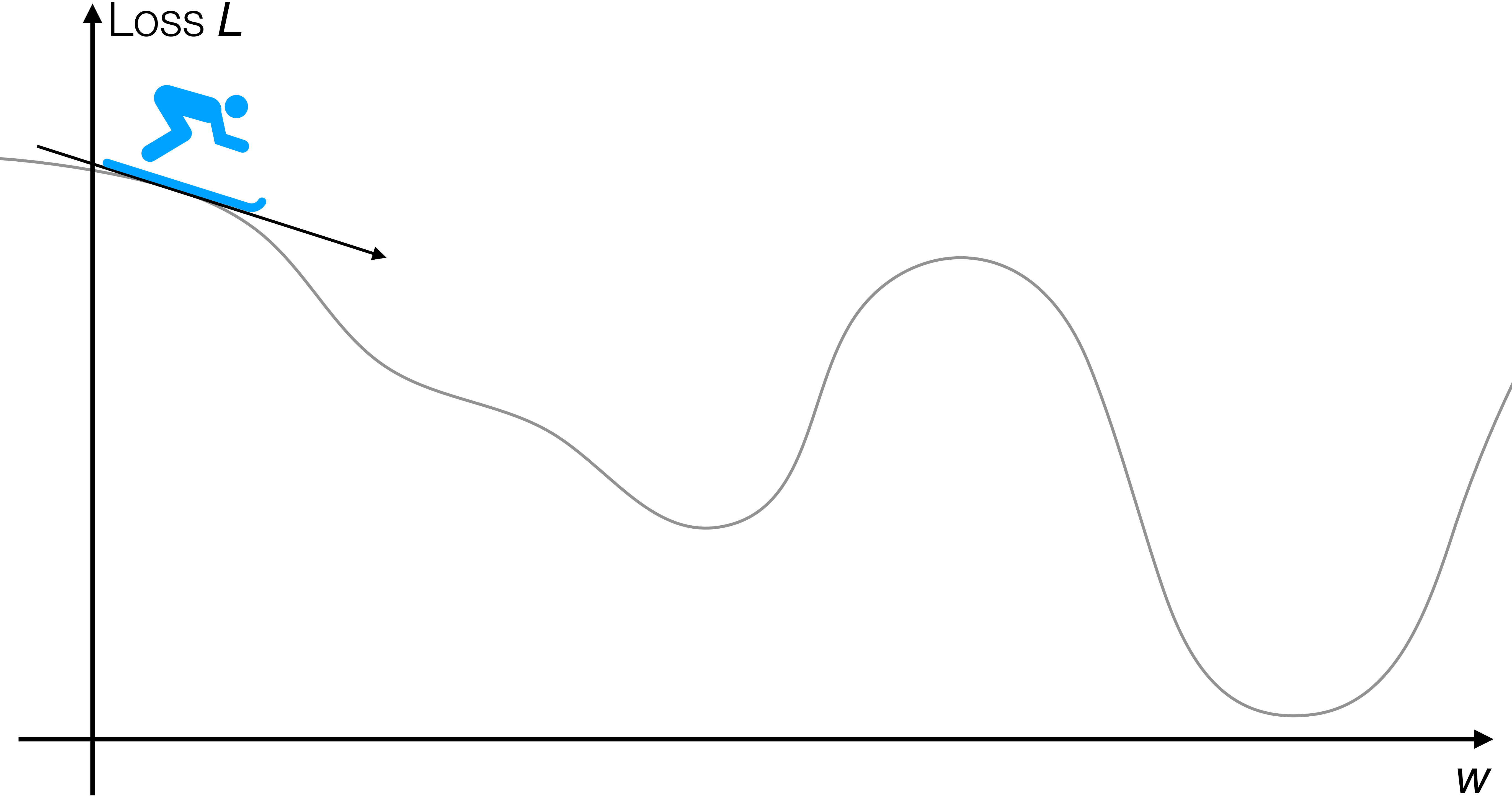
The derivative $\frac{\Delta L}{\Delta w}$ at the skier's position is *negative*

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

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

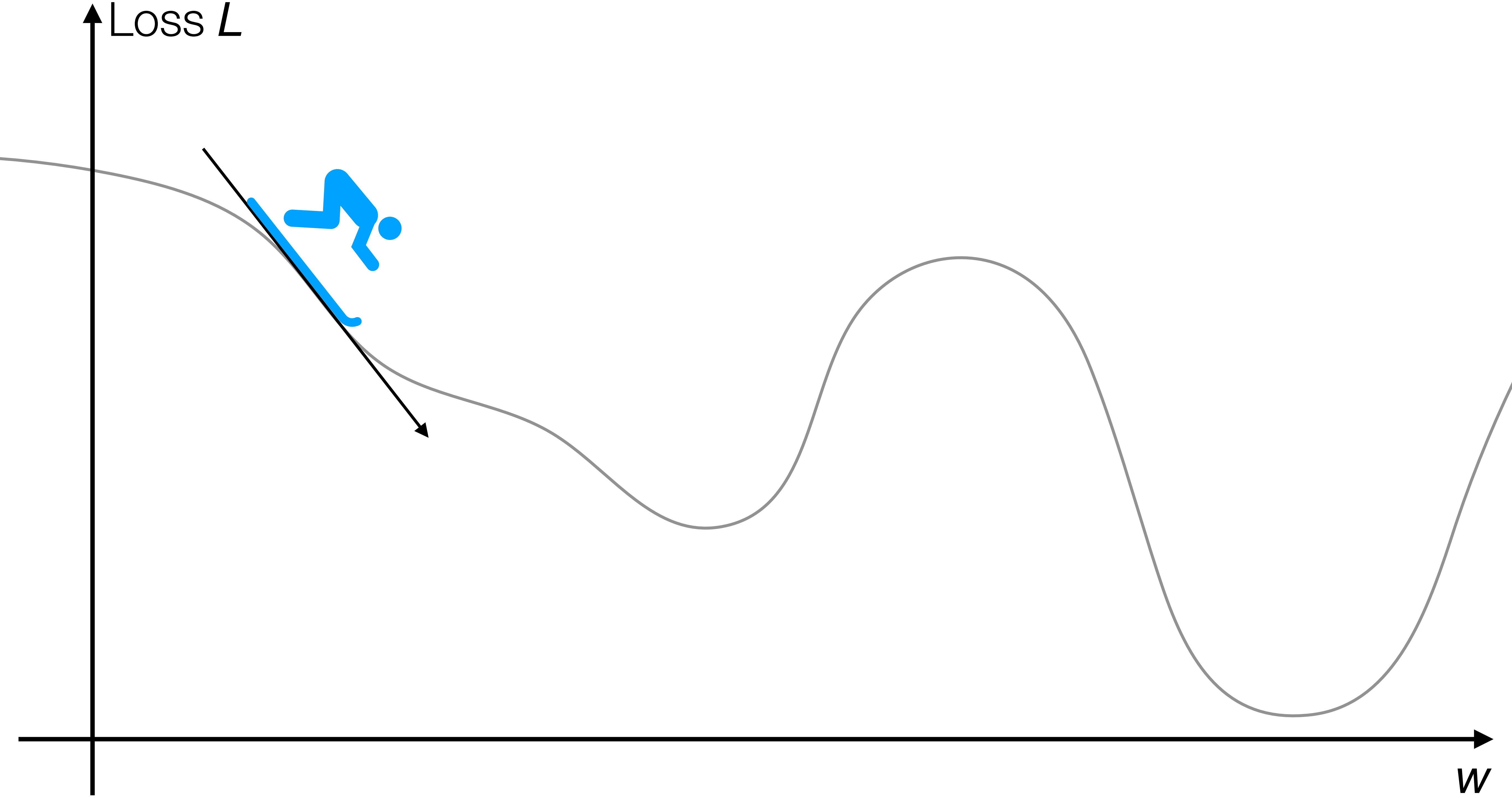
Learning a Deep Net

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



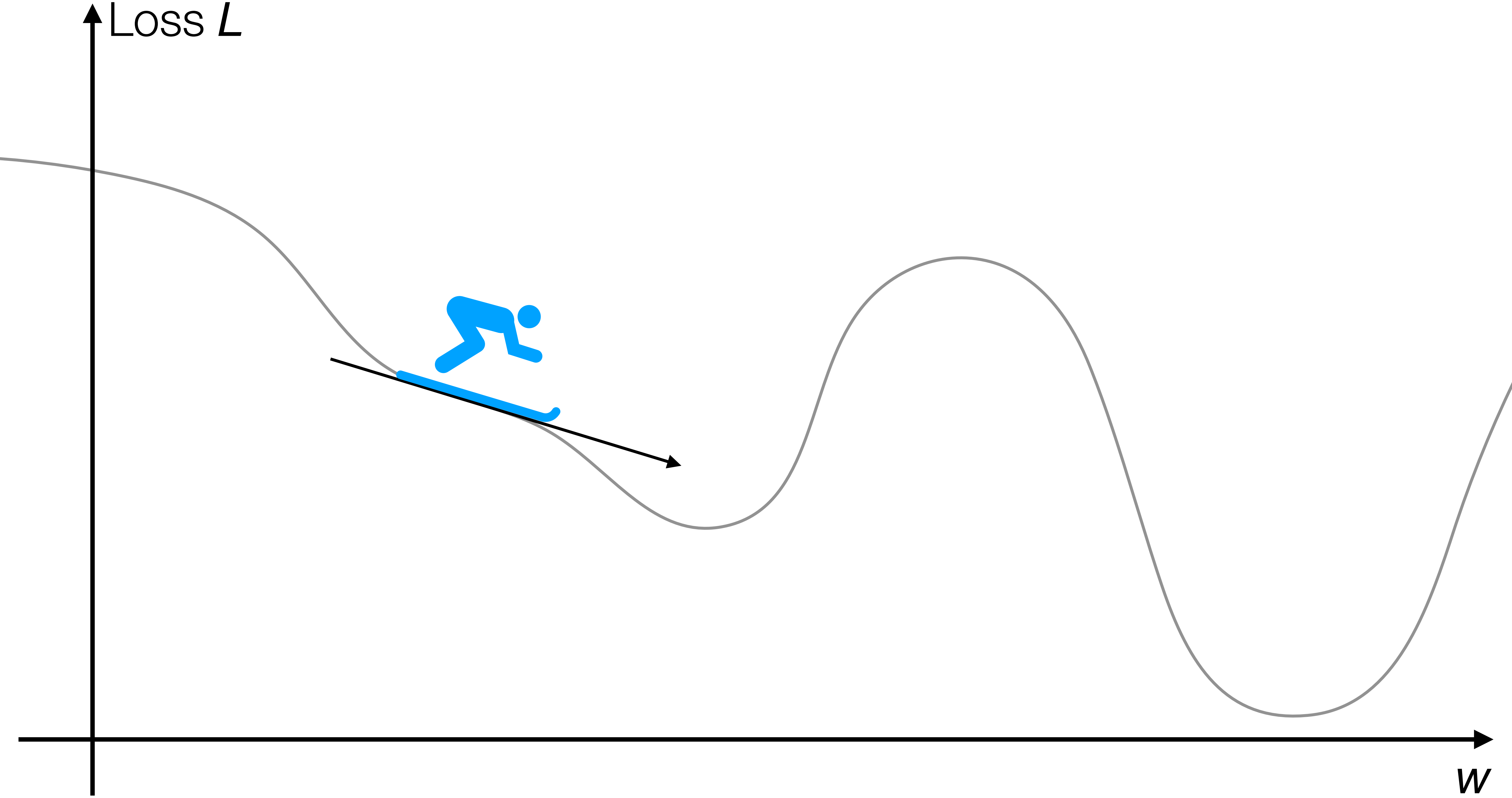
Learning a Deep Net

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



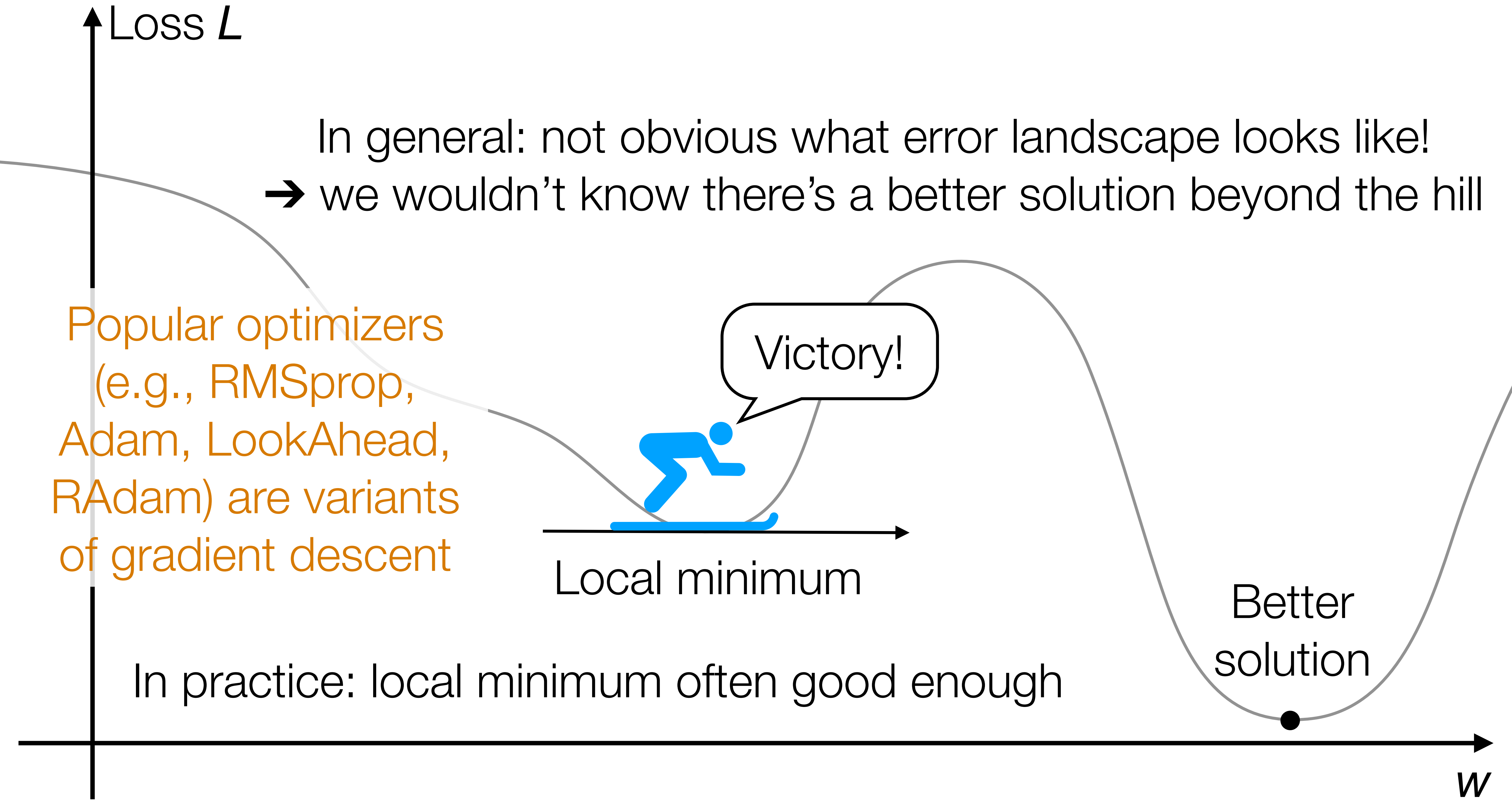
Learning a Deep Net

Suppose the neural network has a single real number parameter w



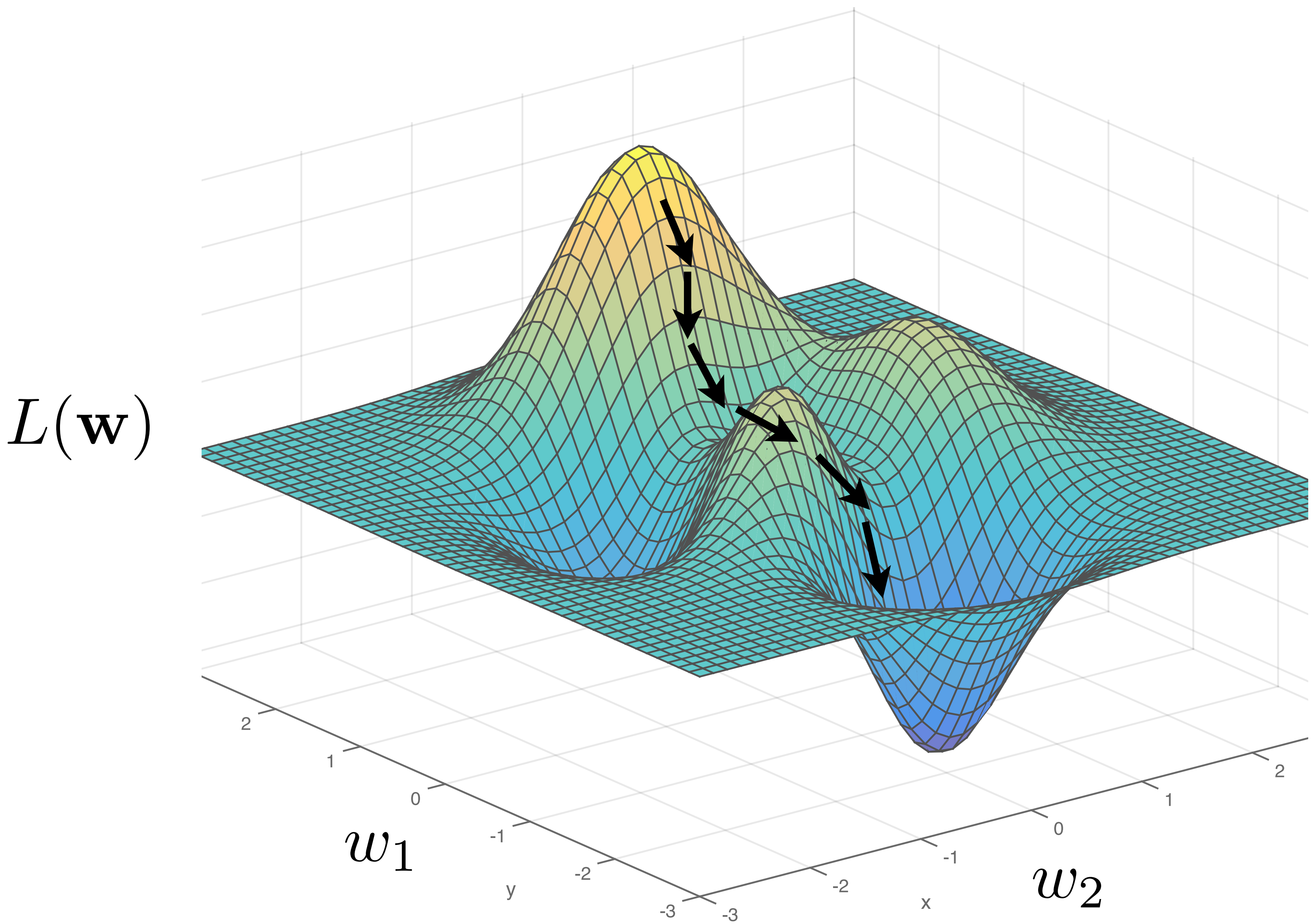
Learning a Deep Net

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



Learning a Deep Net

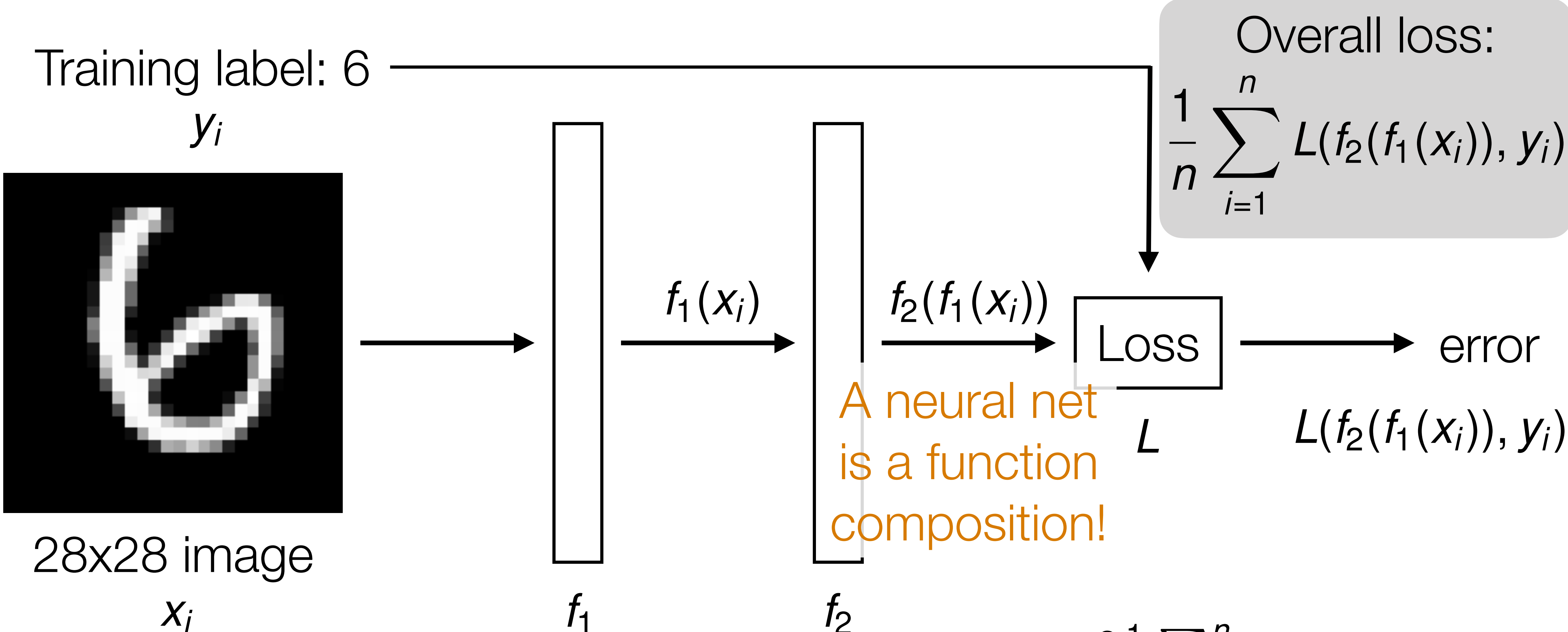
2D example



Slide by Phillip Isola

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

Handwritten Digit Recognition



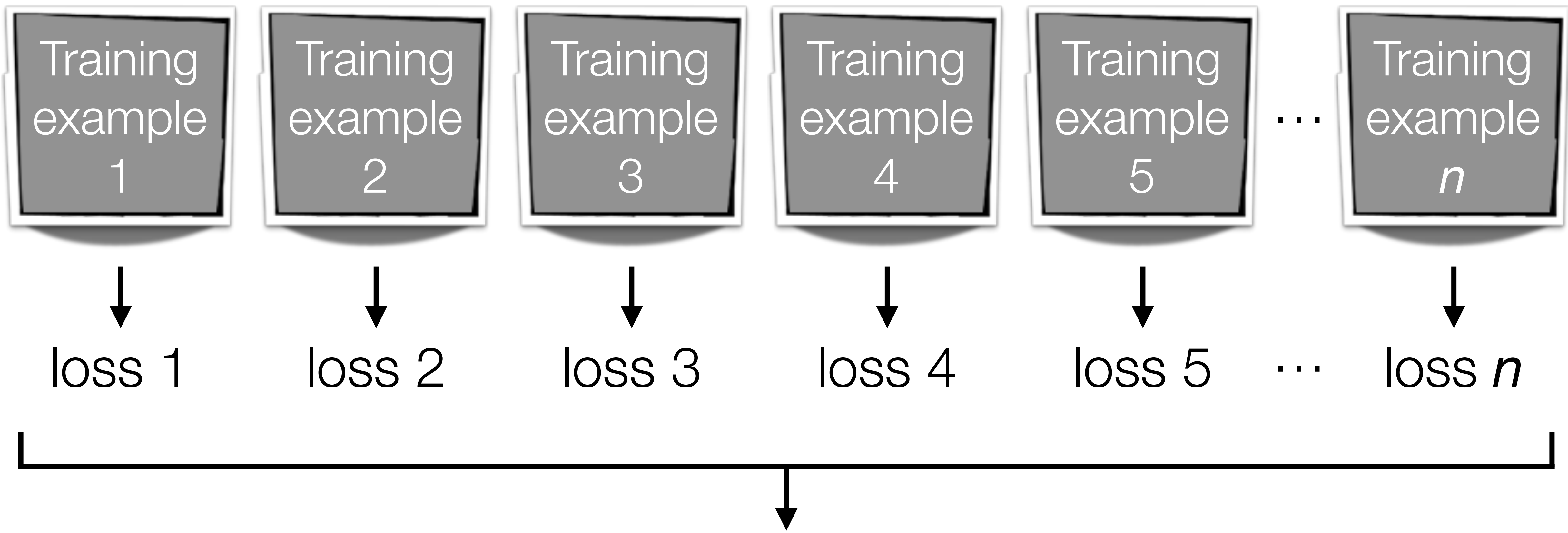
All parameters: θ

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



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

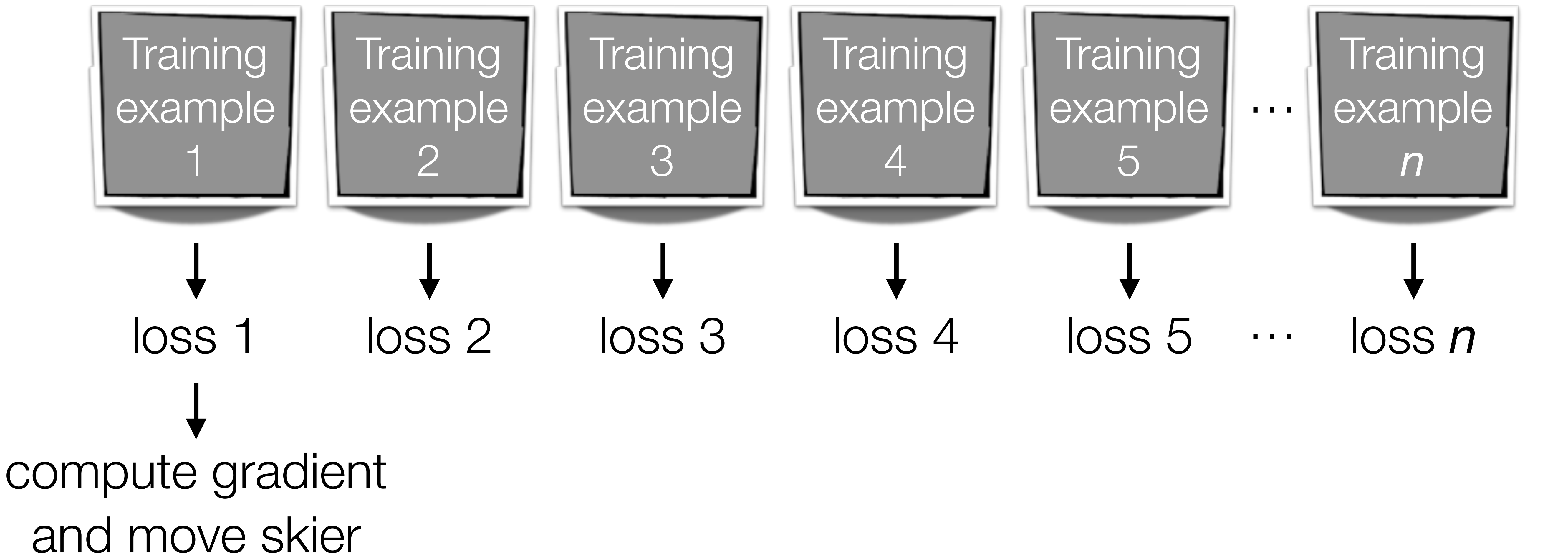
average loss

↓

compute gradient and move skier

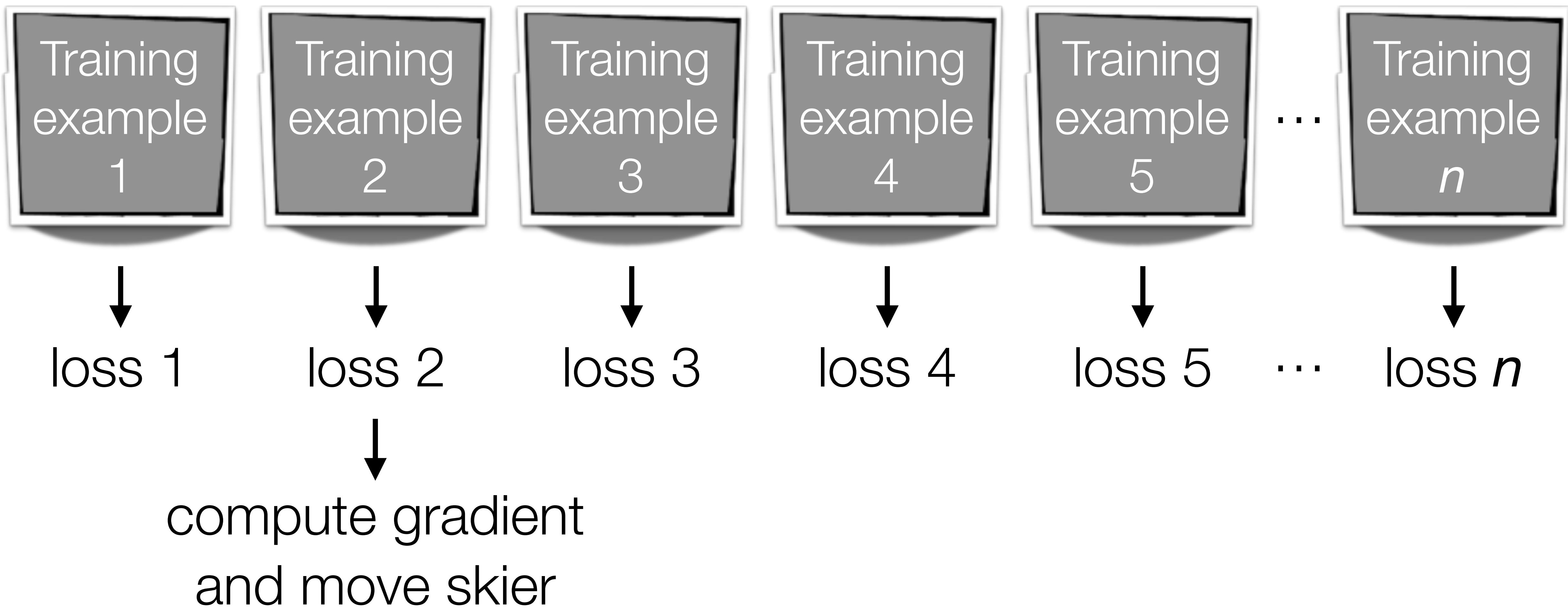
Computing gradients using all the training data seems really expensive!

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)

Training example 1 | Training example 2 | Training example 3 | Training example 4 | Training example 5 | ... | Training example *n*

↓

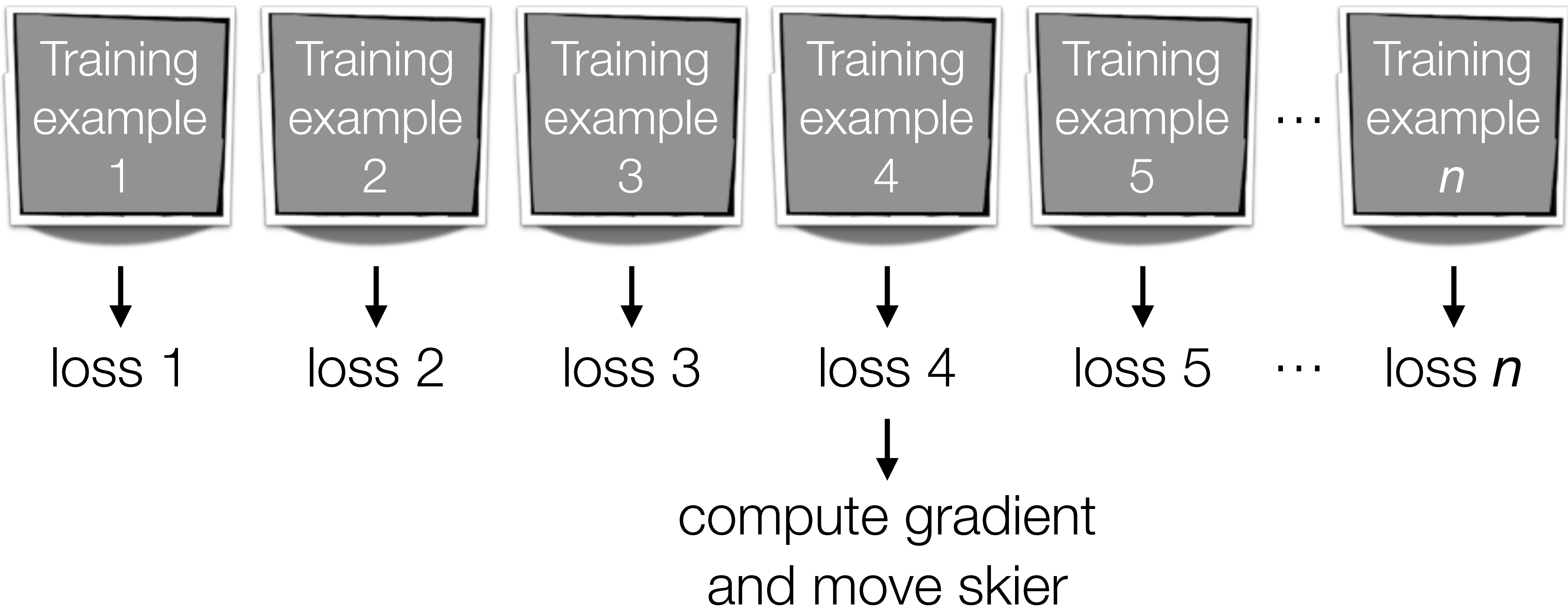
loss 1 | loss 2 | loss 3 | loss 4 | loss 5 | ... | loss *n*

loss 3 ↓

compute gradient
and move skier

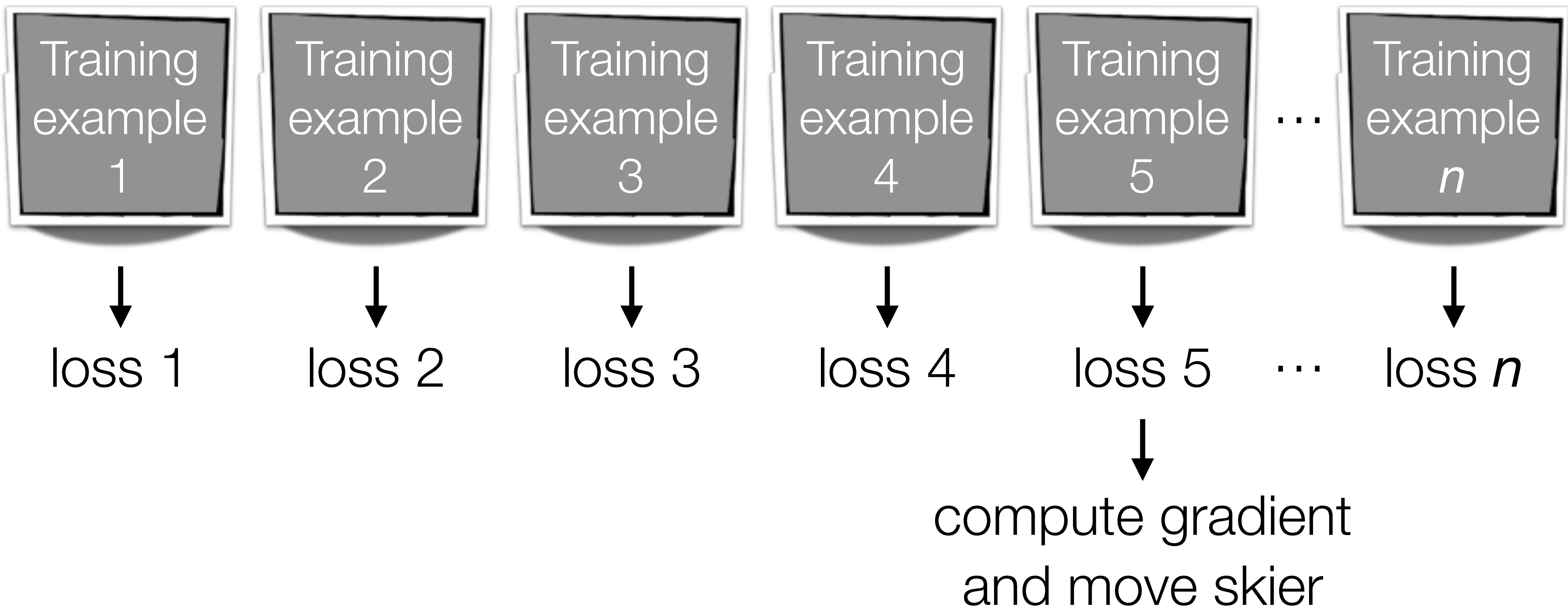
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



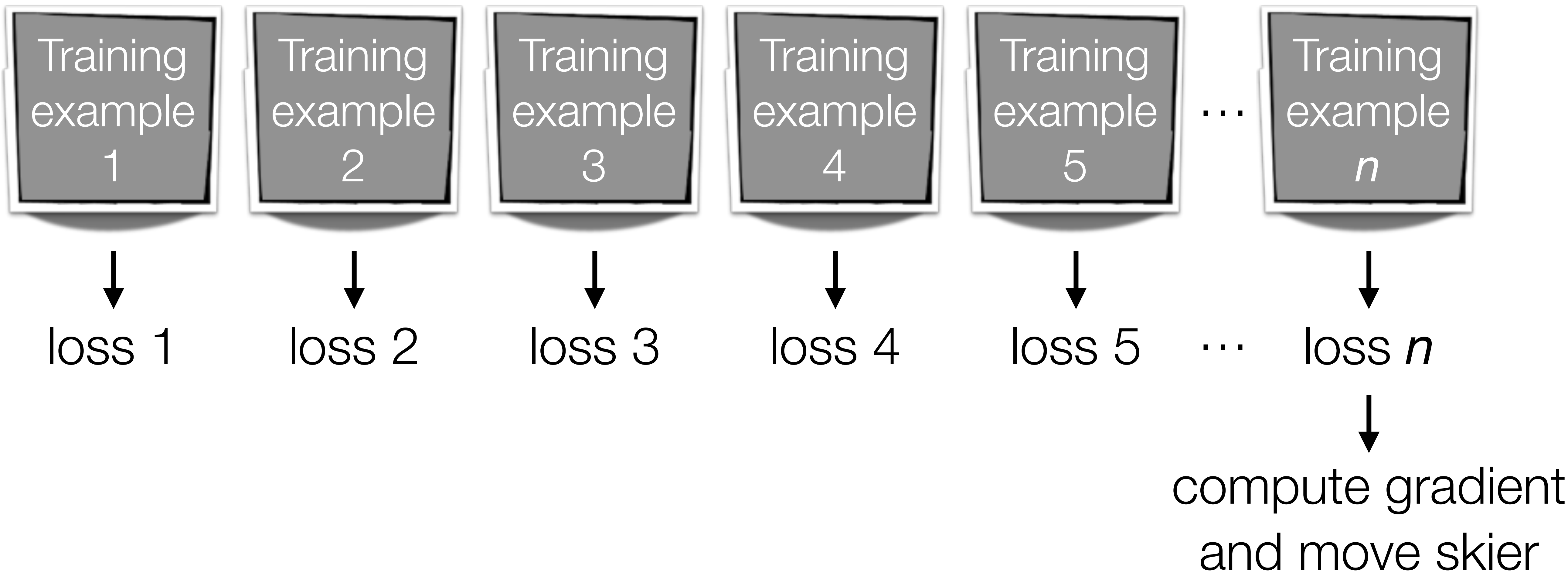
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

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)

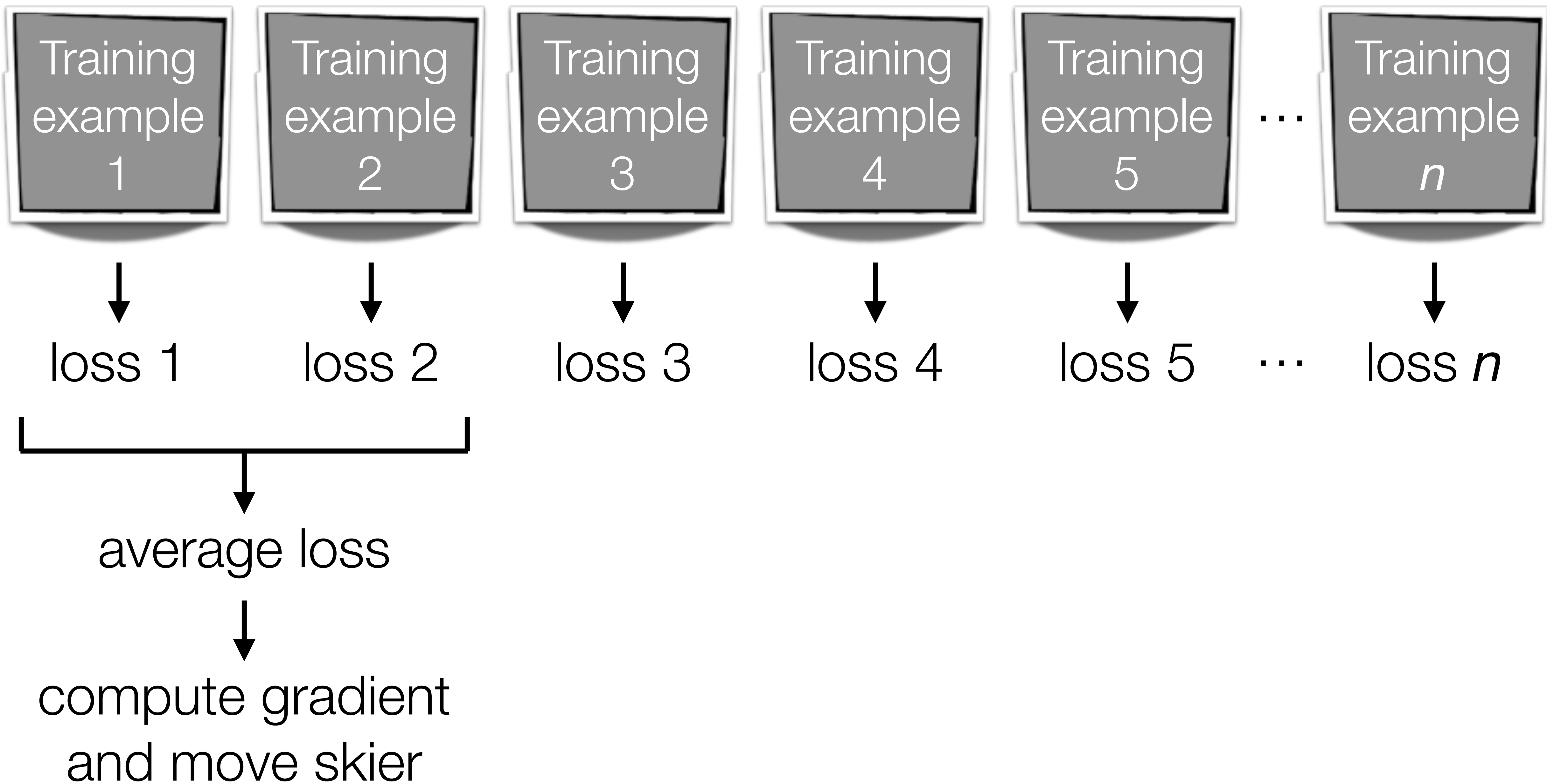
Stochastic Gradient Descent (SGD)

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						

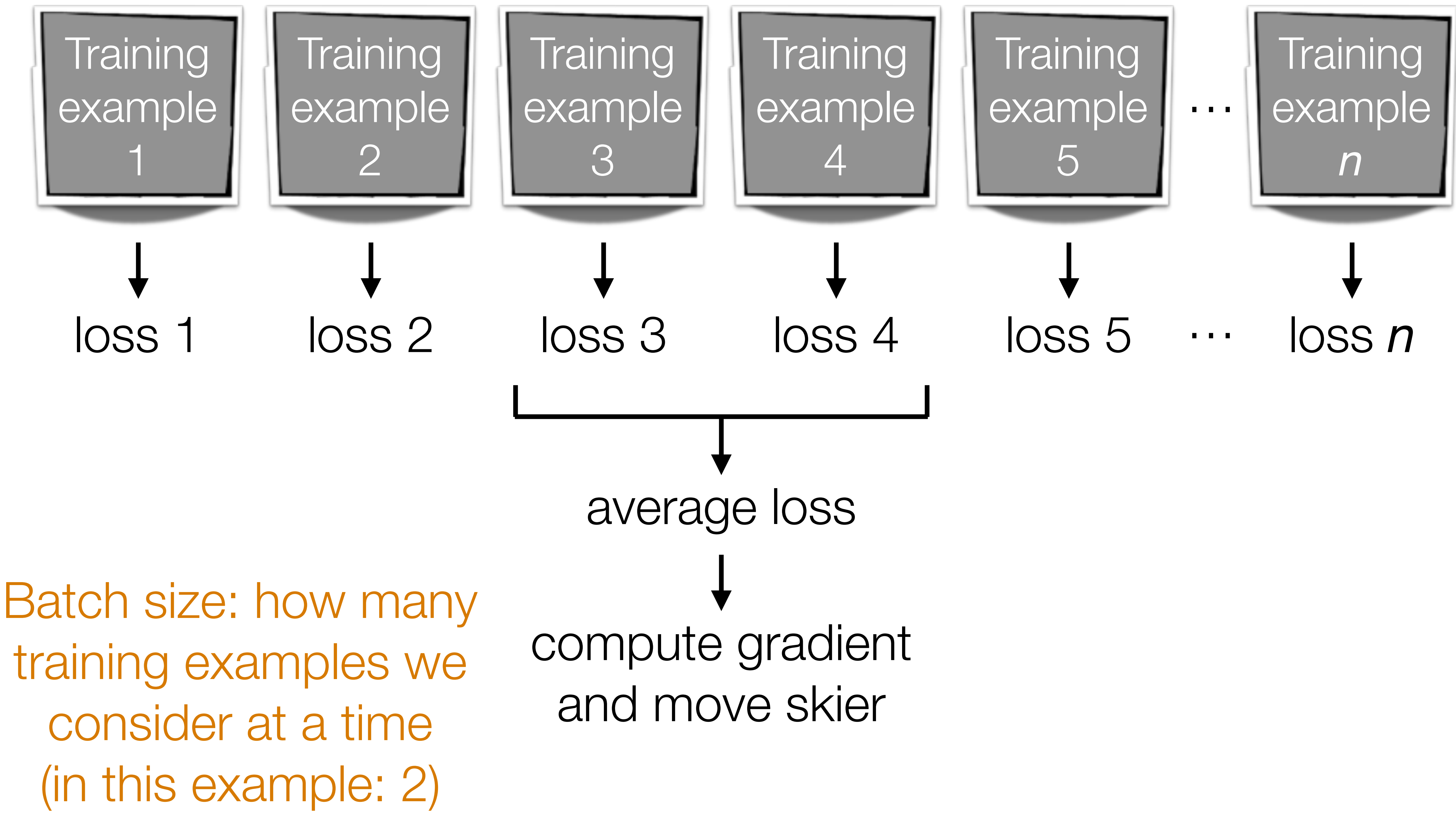
An epoch refers to 1 full pass through all the training data

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



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!!!

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

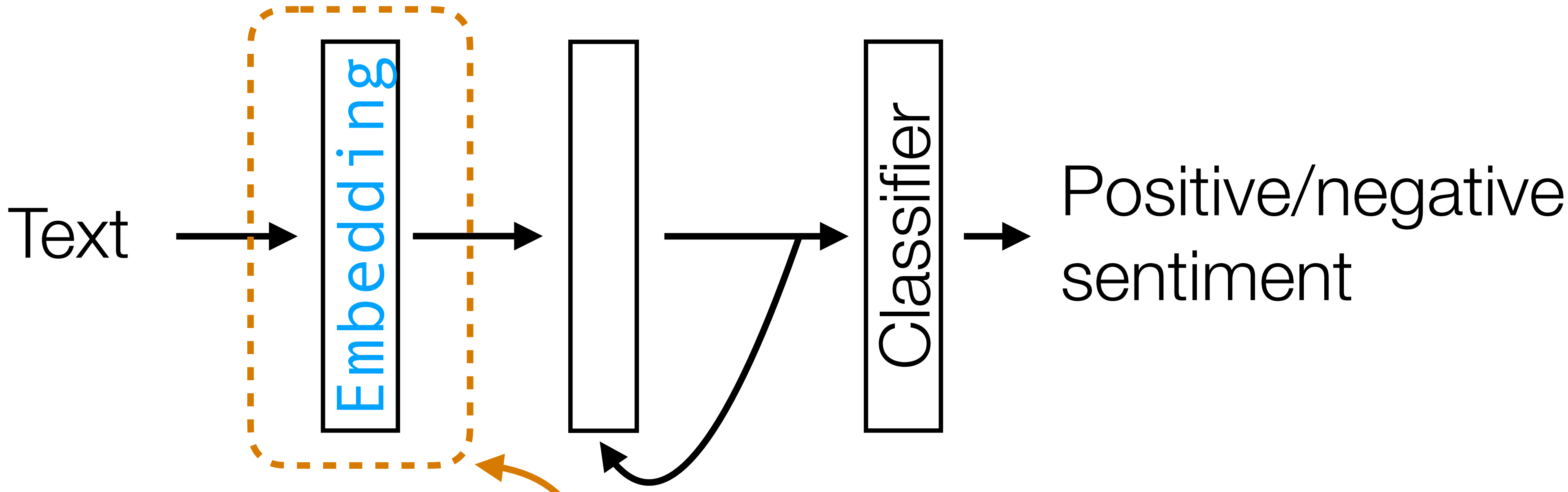
Example: classify between Tesla's and Toyota's

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

Strategy: take pre-trained convnet (such as the state-of-the-art ResNet) for ImageNet classification and change final layers to do classification between Tesla's and Toyota's instead of classifying 1000 objects

Fine Tuning

Sentiment analysis RNN demo



Weights here are treated as fixed & come from pre-trained GloVe word embeddings

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

IMDb review dataset is small in comparison

Data Augmentation

Another way of dealing with small datasets: 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

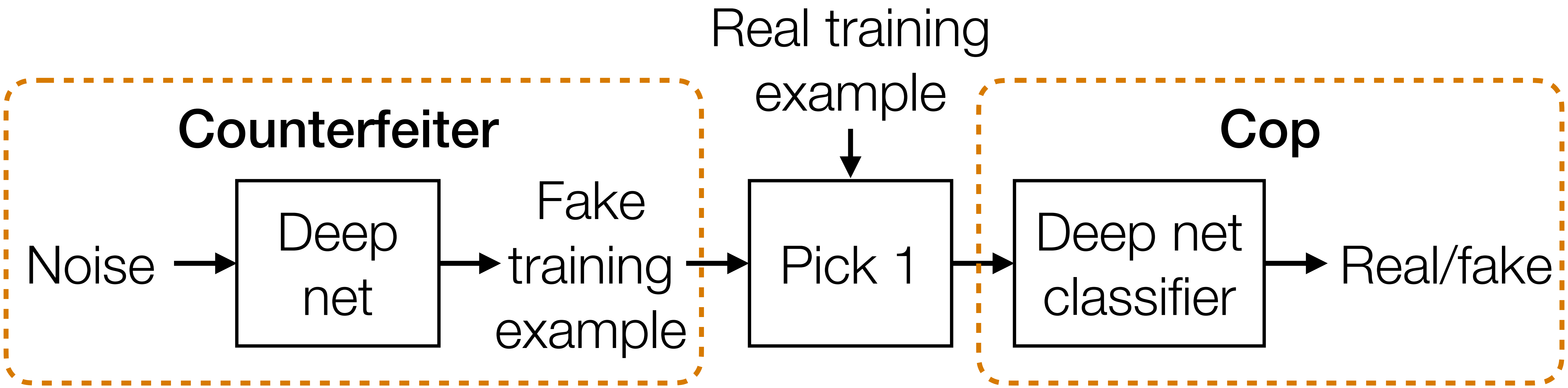
Allowable perturbations depend on data
(e.g., for handwritten digits, rotating by 180 degrees would be bad: confuse 6's and 9's)

Generating Fake Data That Look Real

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

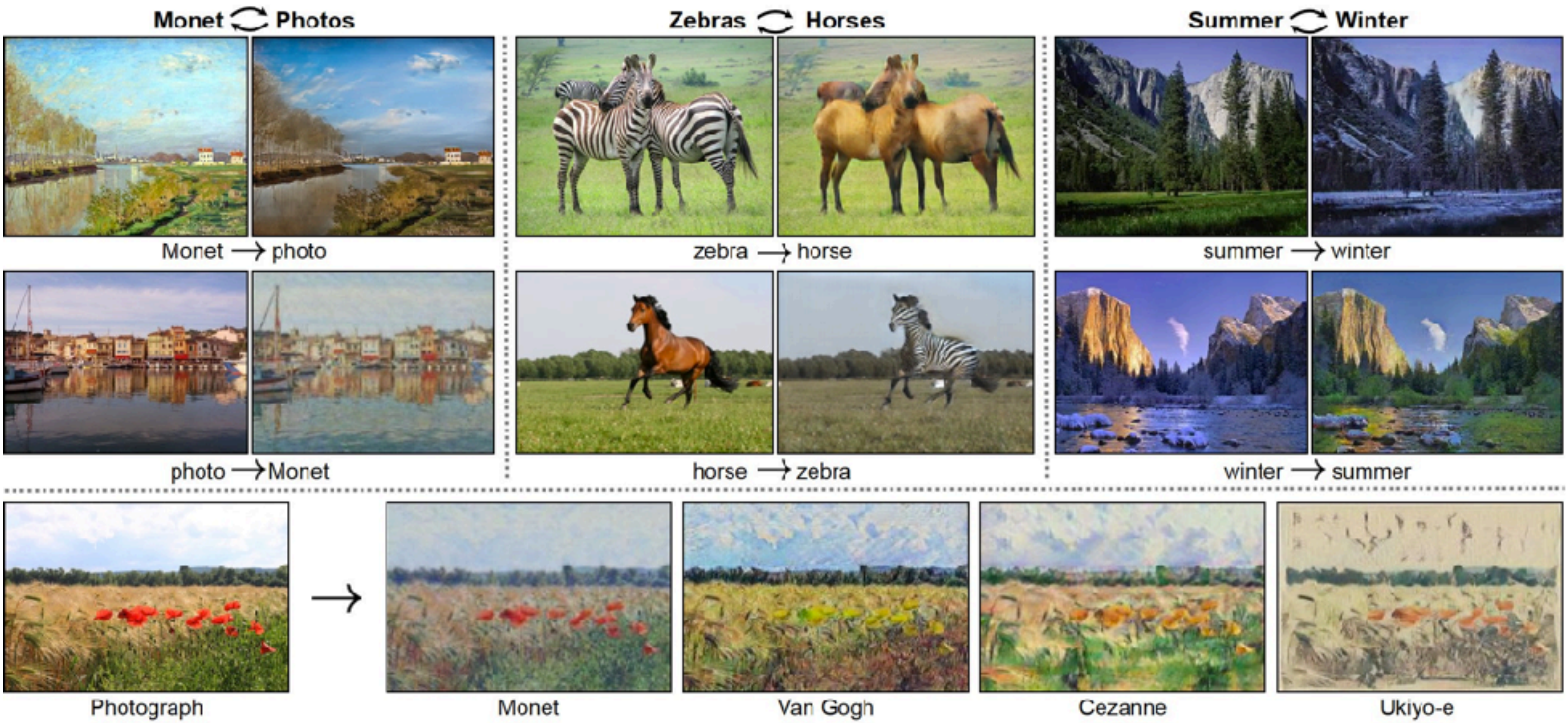


Image-to-image translation results from UC Berkeley using GANs
(Isola et al 2017, Zhu et al 2017)

AI News Anchor

China's Xinhua agency unveils AI news presenter

By Chris Baraniuk
Technology reporter

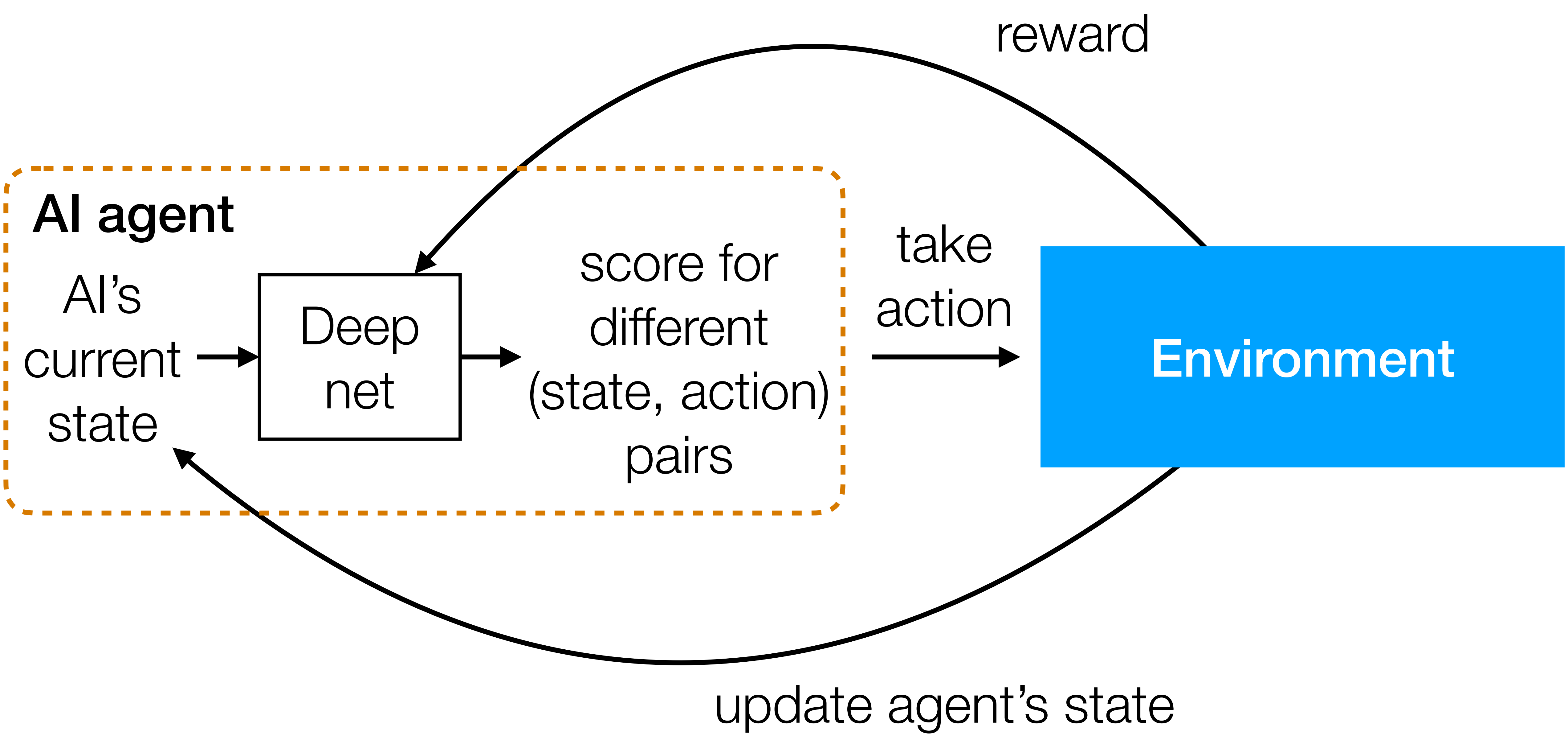
8 November 2018

Share

Source: https://www.bbc.com/news/technology-46136504

Deep Reinforcement Learning

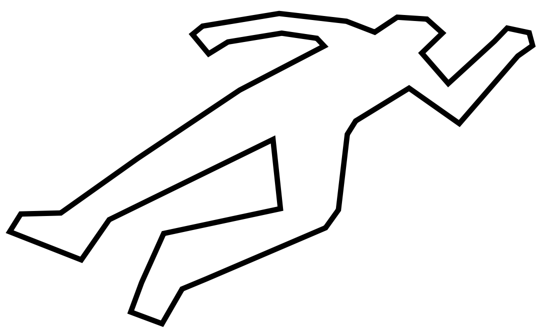
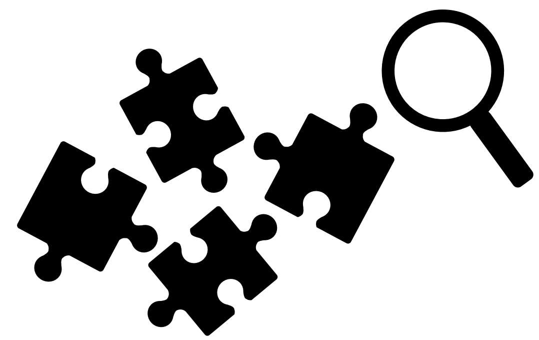
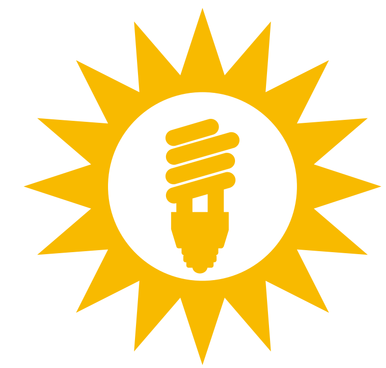
The machinery behind AlphaGo and similar systems



The Future of Deep Learning

- Deep learning currently is still very limited in what it can do
  - Learns simple computer programs (functions) comprised of a series of basic operations — need to be able to compute derivatives of these basic operations
- Adversarial examples at test time remain a problem
- Pretty much all the best ideas that lead to amazing prediction results incorporate problem-specific structure
  - For example, think about how CNNs and RNNs incorporate structure of images/time series
  - How do we get away with using less expert knowledge?
- How do we do lifelong learning?
- How do we reason about causality?

Unstructured Data Analysis

Question	Data	Finding Structure	Insights
			
The dead body	*The evidence*	*Puzzle solving, careful analysis*	*When? Where? Why? How? Perpetrator catchable?*
This is provided by a practitioner	Some times you have to collect more evidence!	Exploratory data analysis	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 year!