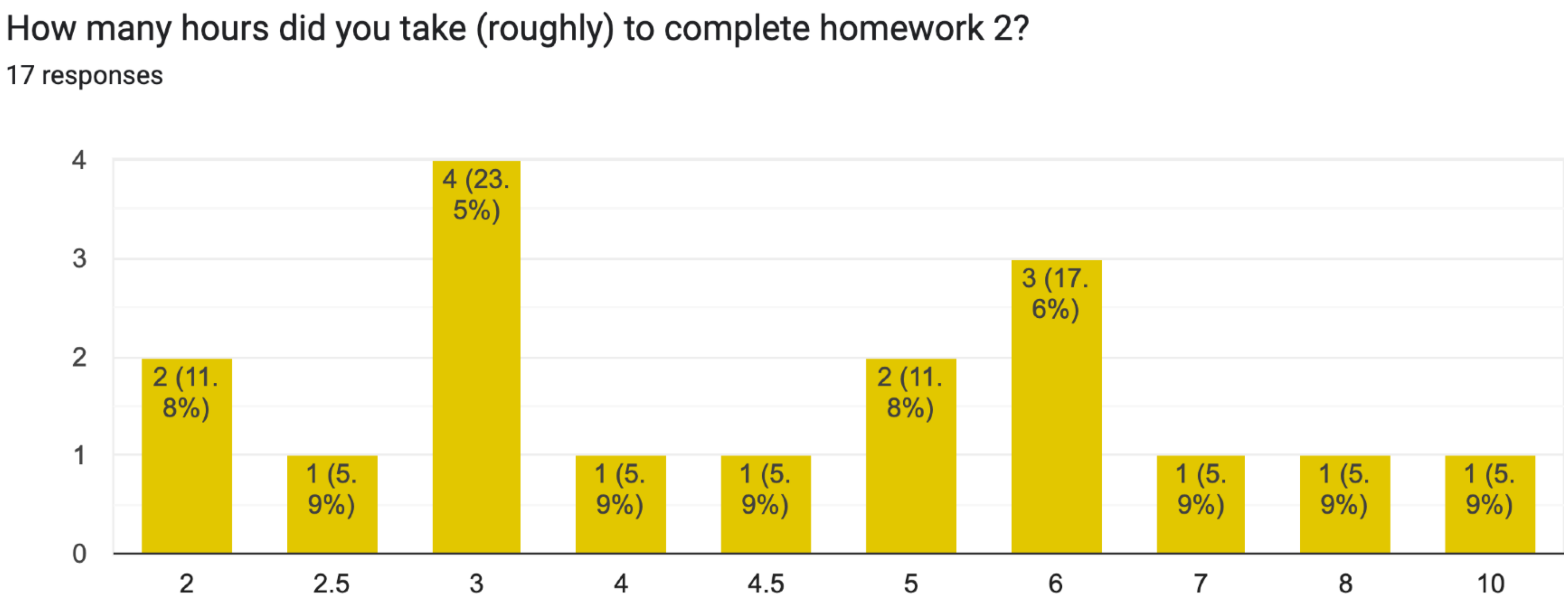
Carnegie Mellon University
Heinz College

94-775 Unstructured Data Analytics

Last lecture: More on transformers; a few more deep learning concepts; course wrap-up

Slides by George H. Chen

HW2 Questionnaire (1/2)



Overall: looks good!
(HW is designed to take at most 15 hrs)

HW2 Questionnaire (2/2)

- Some students said that interpreting clusters or topics can be hard
  - Interpreting clusters or topics can indeed be challenging!
  - Even with newer topic models developed (such as BERTopic), interpretation can still be challenging depending on the dataset
- Some students said that t-SNE plots are confusing to interpret
  - Yes, this is indeed the case…
  - If you have some ground truth annotation that can be used to help color the data points, it might be easier seeing what's going on…
- Some students said that it's not clear when specific steps should be done in preprocessing (e.g., when should PCA be applied before clustering? when should it not be applied? etc)
  - This is indeed not straightforward
  - **Perhaps the best starting point: see if someone else has come up with a preprocessing pipeline for a similar sort of data as yours**
    - If not, try multiple options and see how conclusions change!

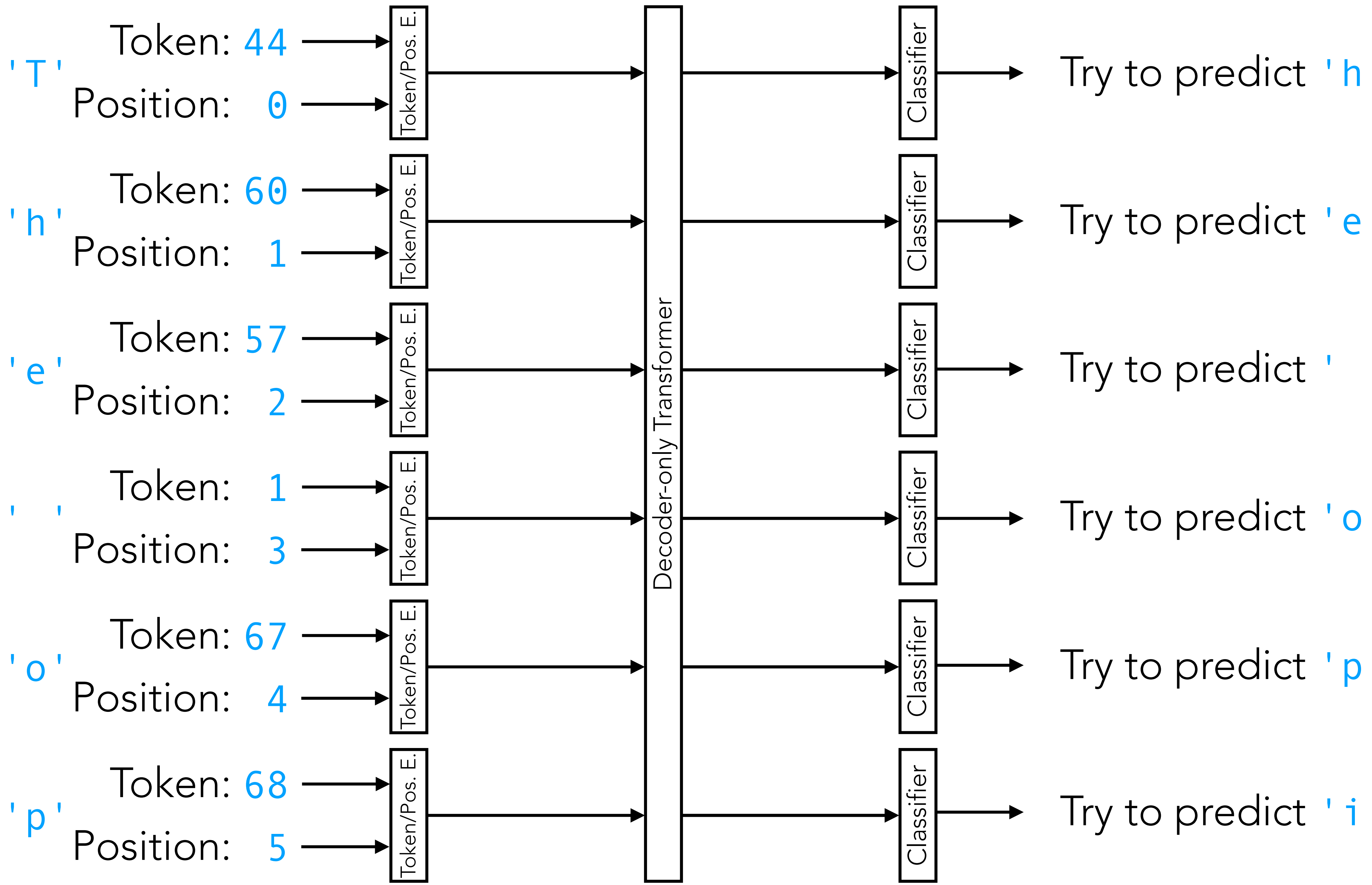
Outline

- A bit more on transformers
    - What is BERT?
    - How can BERT or similar models help us solve a prediction task?

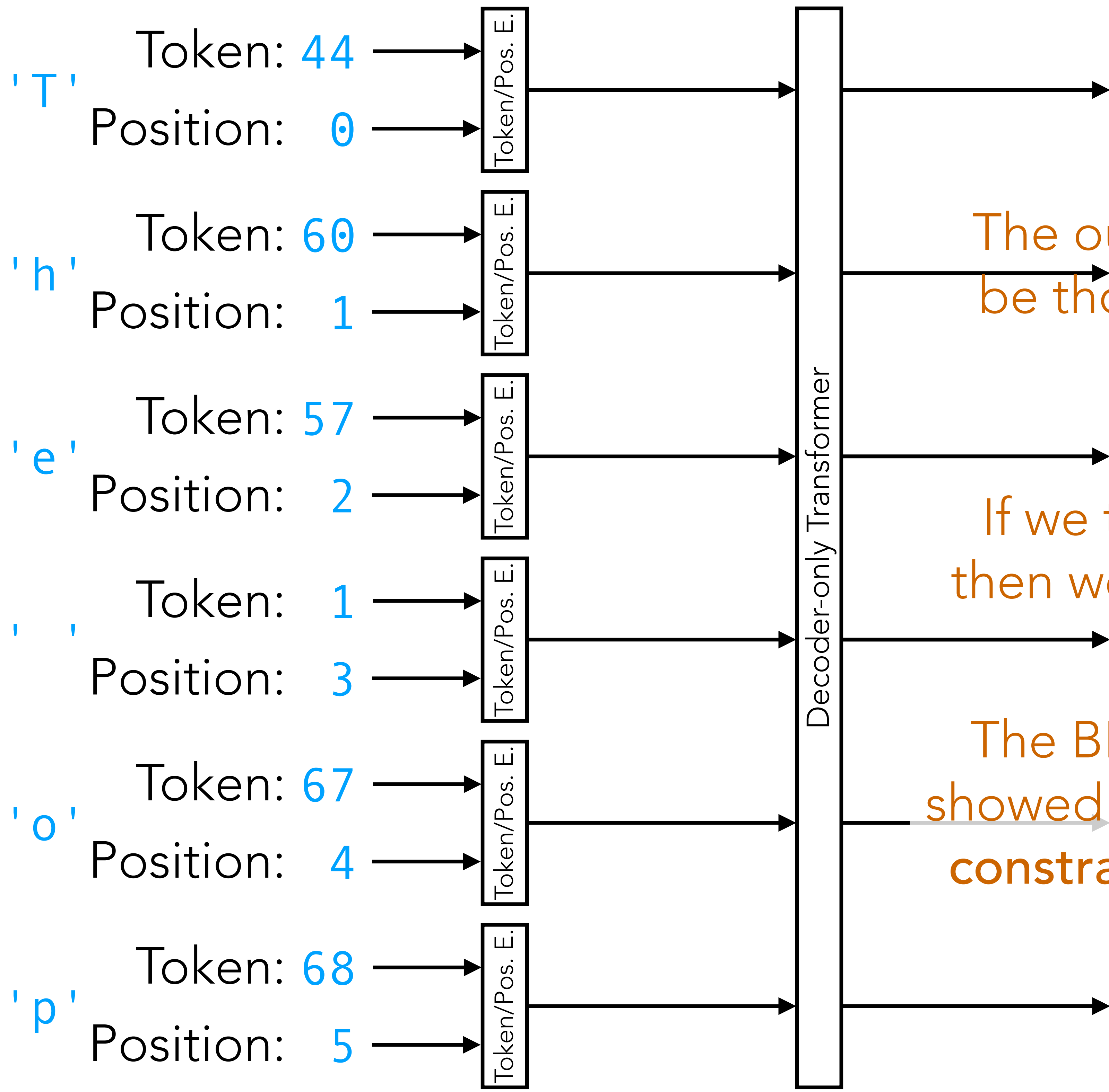
      We'll specifically look at sentiment analysis with IMDb reviews
- How do we train deep nets on small datasets?
- How do we interpret what a deep net has learned?
- How does training a deep net work?
- Course wrap-up

A bit more on transformers

(Flashback) Generative Pre-trained Transformer (GPT)



(Flashback) Generative Pre-trained Transformer (GPT)

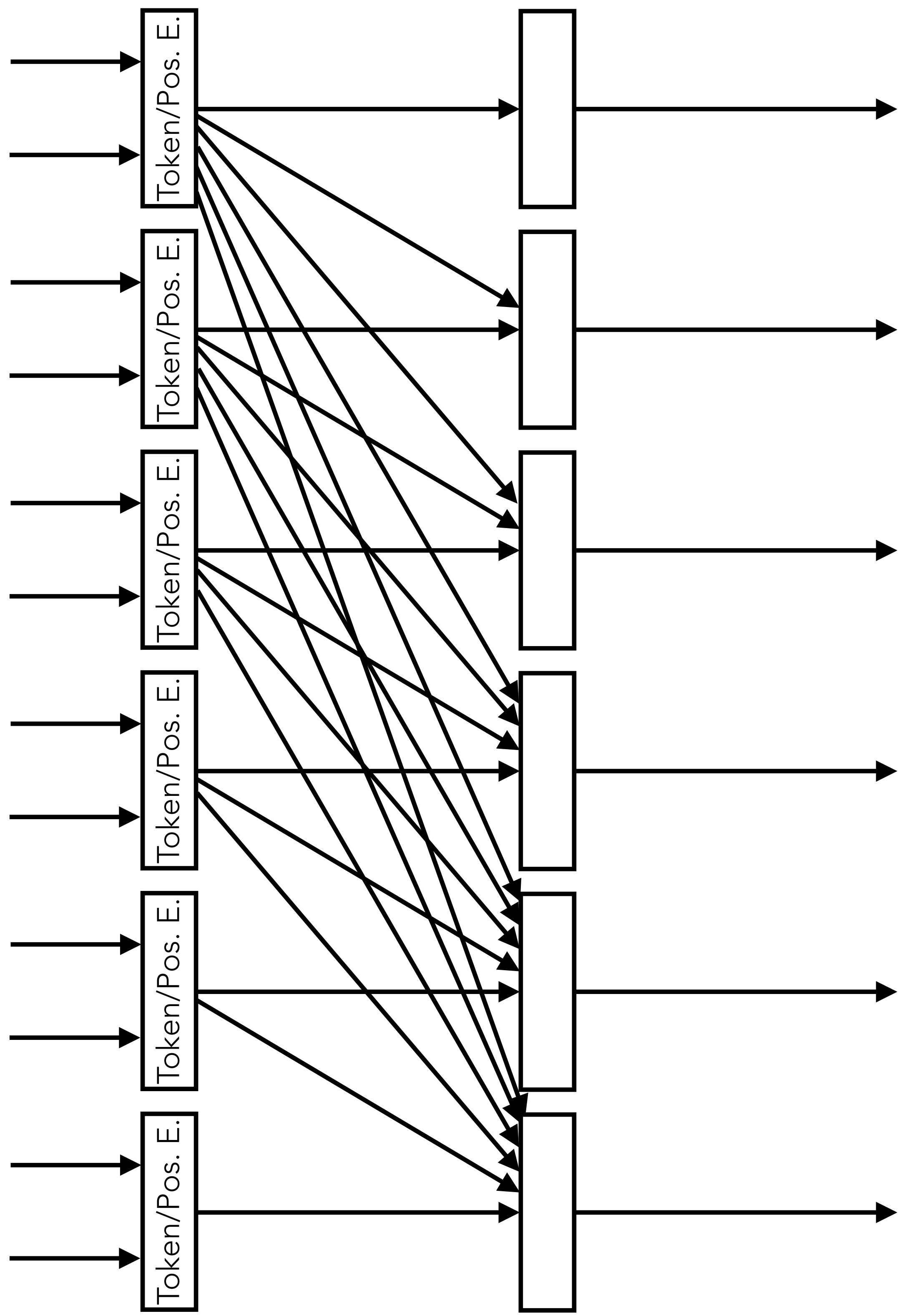


The output before the classifier could be thought of as a token embedding that accounts for context

If we tokenized instead using words, then we would have word embeddings

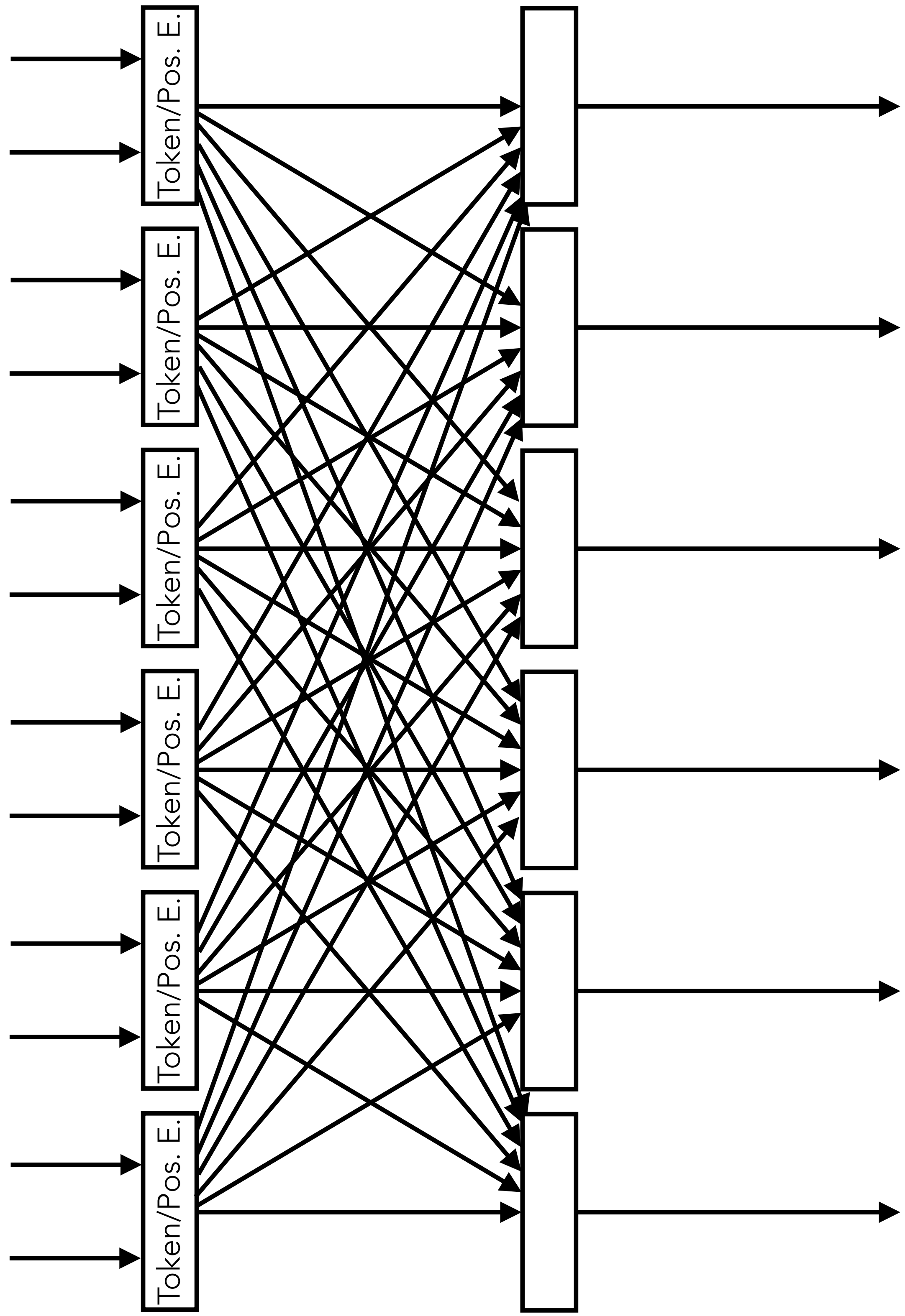
The BERT model is basically what we showed today except **without the causal constraint** & **with a different tokenizer**

(As we saw previously, BERT's tokenization uses words and sometimes subwords)



causal dependence

BERT (2018)



no causal dependence

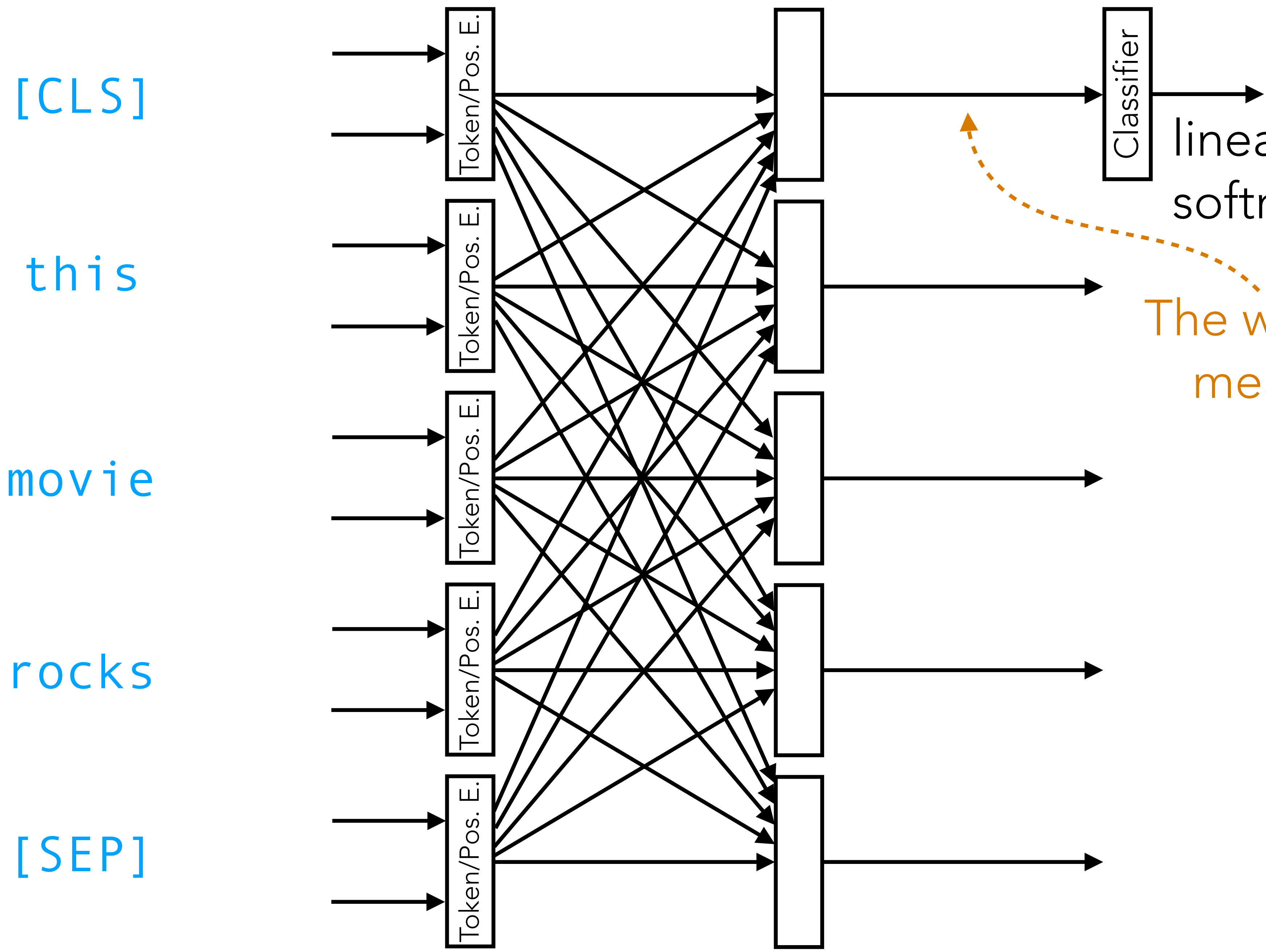
The prediction at any time step depends on the input at all time steps

This lack of causal dependence is also sometimes referred to as "bidirectional"

A transformer layer like this without a causal constraint is sometimes called an "encoder-only" transformer layer

BERT is short for Bidirectional Encoder Representations from Transformers

BERT (2018)



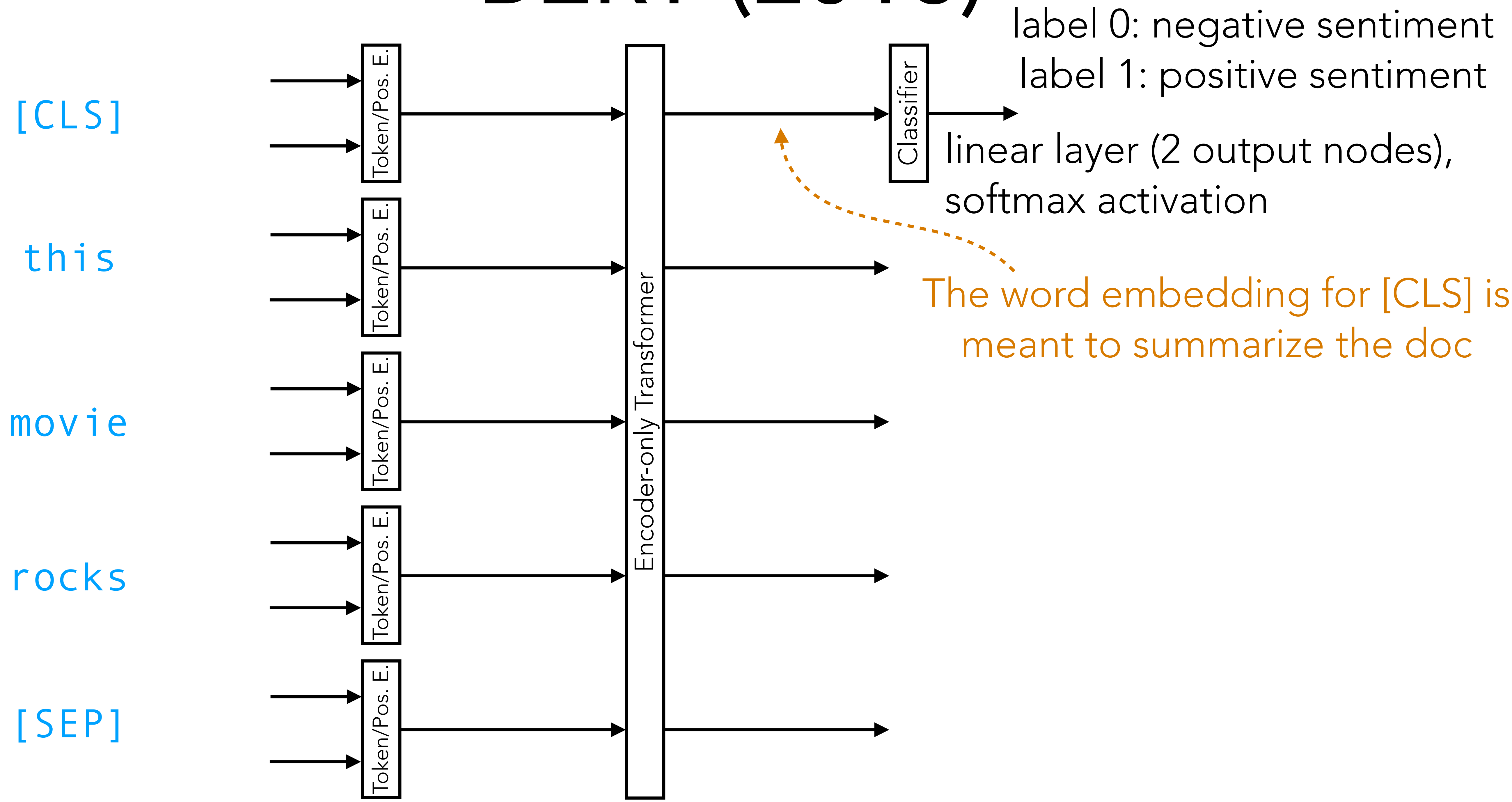
label 0: negative sentiment
label 1: positive sentiment

linear layer (2 output nodes), softmax activation

The word embedding for [CLS] is meant to summarize the doc

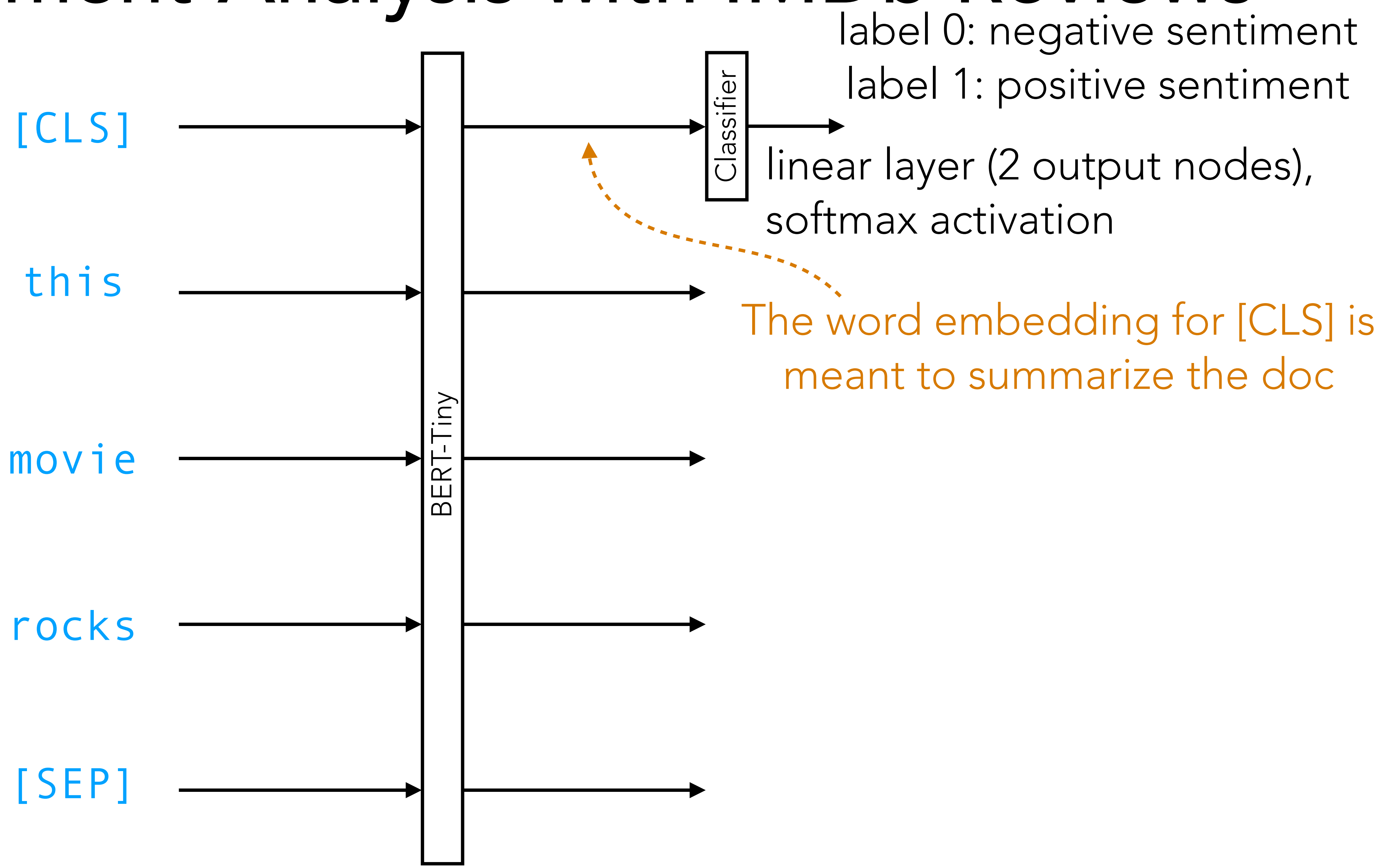
BERT actually adds an initial "[CLS]" token and an ending "[SEP]" token

BERT (2018)



BERT actually adds an initial "[CLS]" token and an ending "[SEP]" token

Sentiment Analysis with IMDb Reviews



We're about to look at a demo where we use a tiny version of BERT called BERT-Tiny (so that things run faster!)

Sentiment Analysis Demo Cheatsheet

Important: we do not build a vocabulary from scratch since we just use BERT-Tiny's vocabulary!

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 <u>proper training</u> review into token IDs using BERT-Tiny's `encode` method

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

↓

```
['[CLS]', 'master', 'ci', '##eas', '##te", 'alain', 'res',
 '##nais', 'likes', 'to', 'work', 'with', 'those', ...]
```

↓

```
[101, 3040, 25022, 26737, 2618, 15654, 24501, 28020, 7777,
 2000, 2147, 2007, 2216, ...]
```

Important: we do not build a vocabulary from scratch since we just use BERT-Tiny's vocabulary!

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 <u>proper training</u> review into token IDs using BERT-Tiny's `encode` method

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

```
['[CLS]', 'master', 'ci', '##eas', '##te", 'alain', 'res',
 '##nais', 'likes', 'to', 'work', 'with', 'those', ...]
```

```
[101, 3040, 25022, 26737, 2618, 15654, 24501, 28020, 7777,
 2000, 2147, 2007, 2216, ...]
```

`proper_train_dataset_encoded`
`val_dataset_encoded`

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

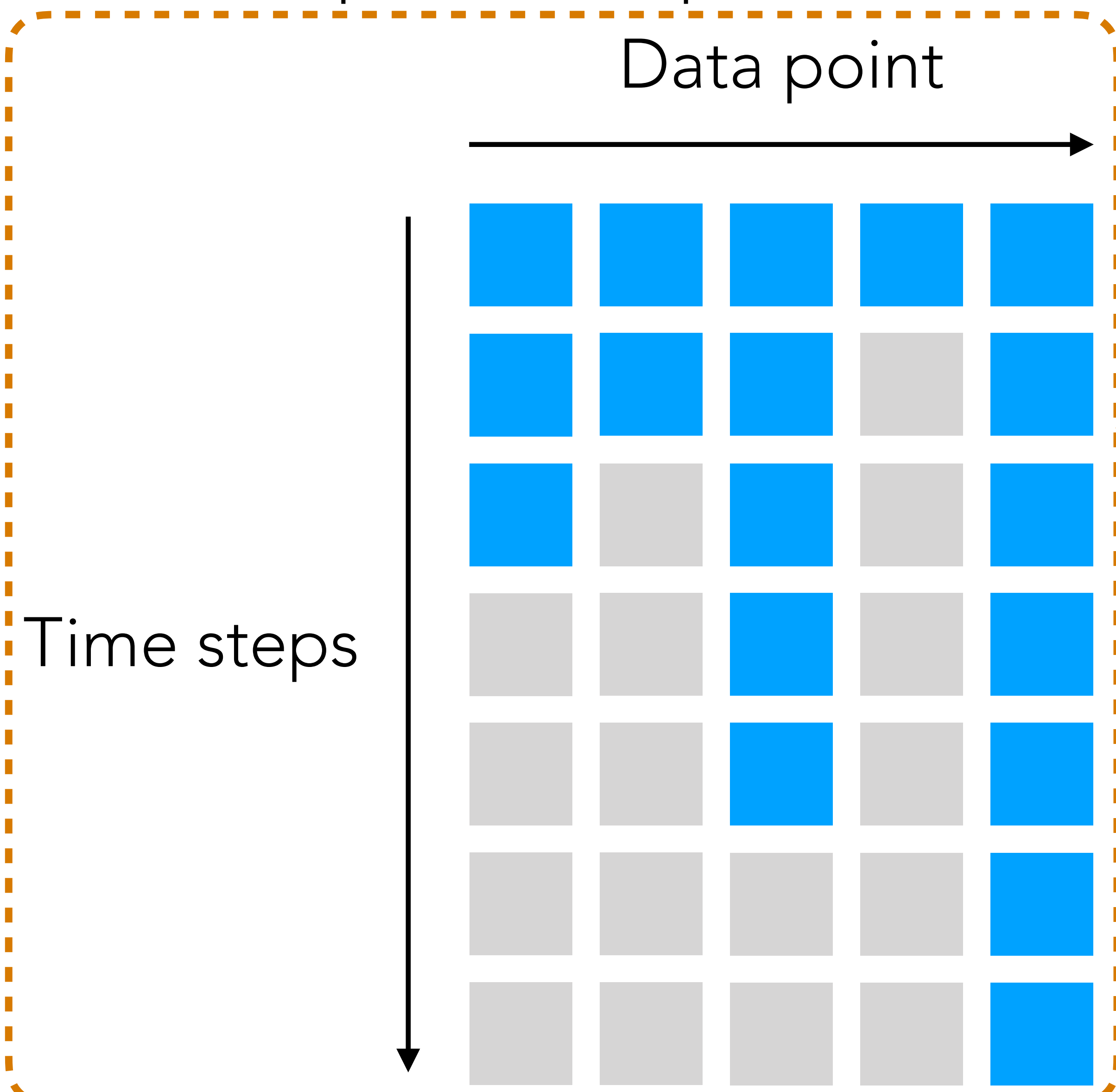
`proper_train_dataset_encoded`
`val_dataset_encoded`

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

4. 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

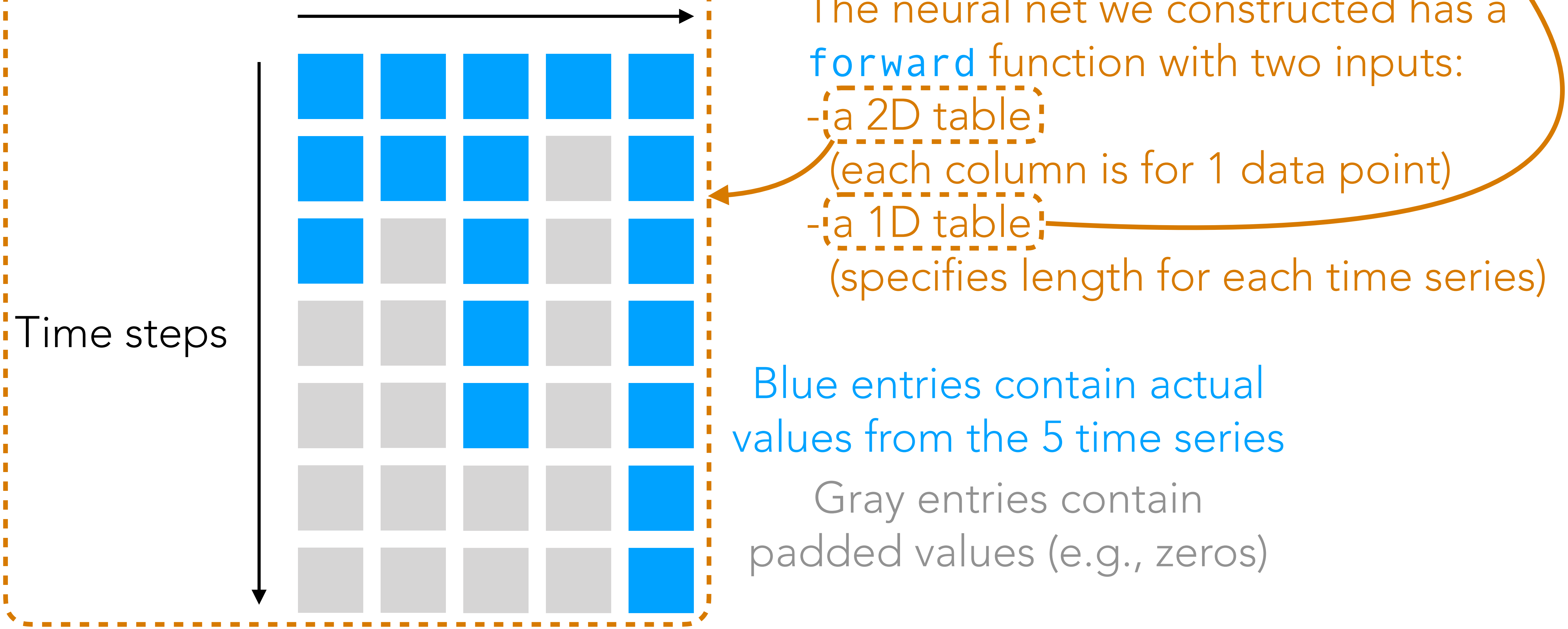
Example: 5 data points (each one is a time series) of lengths 3, 2, 5, 1, 7



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)



5. Train the neural net for some user-specified max number of epochs

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

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

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

Demo: Sentiment Analysis with IMDb Reviews

Demo

Fine-Tuning

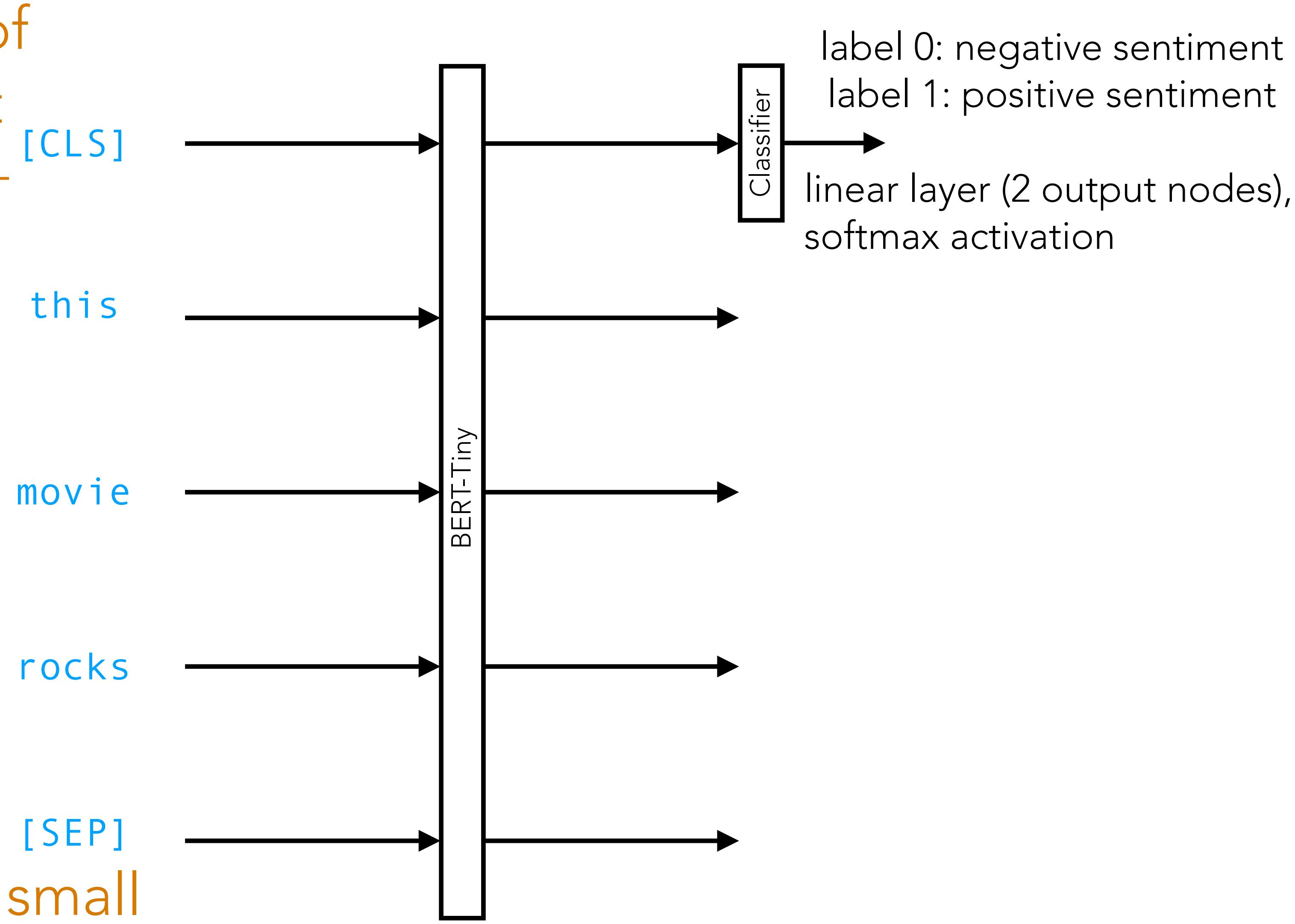
Load in an already trained model, possibly change the last few layers, and modify it for our purposes

We loaded in a pre-trained BERT-Tiny model, which is a compressed version of BERT-Large, trained on a large dataset including BooksCorpus (800M words) + English Wikipedia (2500M words)

We then fine-tuned BERT-Tiny for our sentiment analysis neural net

Note that we fine-tuned on a relatively small dataset (only 25000 training reviews, which is much smaller than BooksCorpus/English Wikipedia)

Sentiment analysis demo



Handling Small Datasets

- Fine-tuning has an extremely important application: it allows us to use an existing model trained on a *massive* dataset to help us with a new prediction task where we might only have a *small* dataset

We just talked about this for the sentiment analysis demo (previous slide)

ChatGPT/GPT 4.0:

GPT pre-trained on massive dataset (exact size undisclosed…)

Fine-tune on human-annotated training dataset (of Q&A pairs and scores of how good the system's automatically generated Q&A pairs are), known to be much smaller than what the model is pre-trained on

Handling Small Datasets

- Fine-tuning has an extremely important application: it allows us to use an existing model trained on a *massive* dataset to help us with a new prediction task where we might only have a *small* dataset
- Another extremely important strategy: **data augmentation (randomly perturb training data to get more training data)**

Training image

Training label: cat

Mirrored

Still a cat!

Rotated & translated

Still a cat!

We just turned 1 training example in 3 training examples!

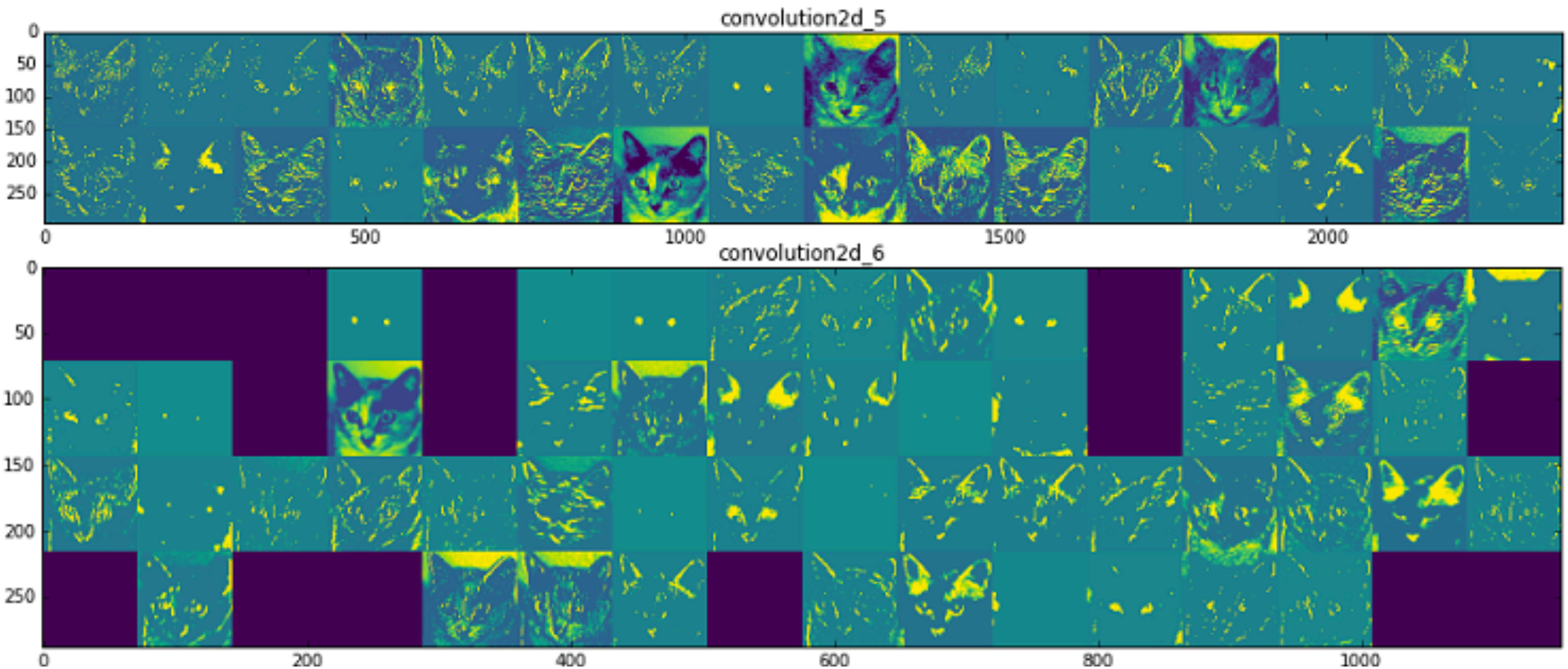
State-of-the-art vision systems are all trained with data augmentation!

Allowable perturbations depend on data
(e.g., for handwritten digits, rotating a 6 or 9 by 180 degrees would be bad)

Interpreting/explaining deep nets

Visualizing What a CNN Learned

Plot filter outputs at different layers



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

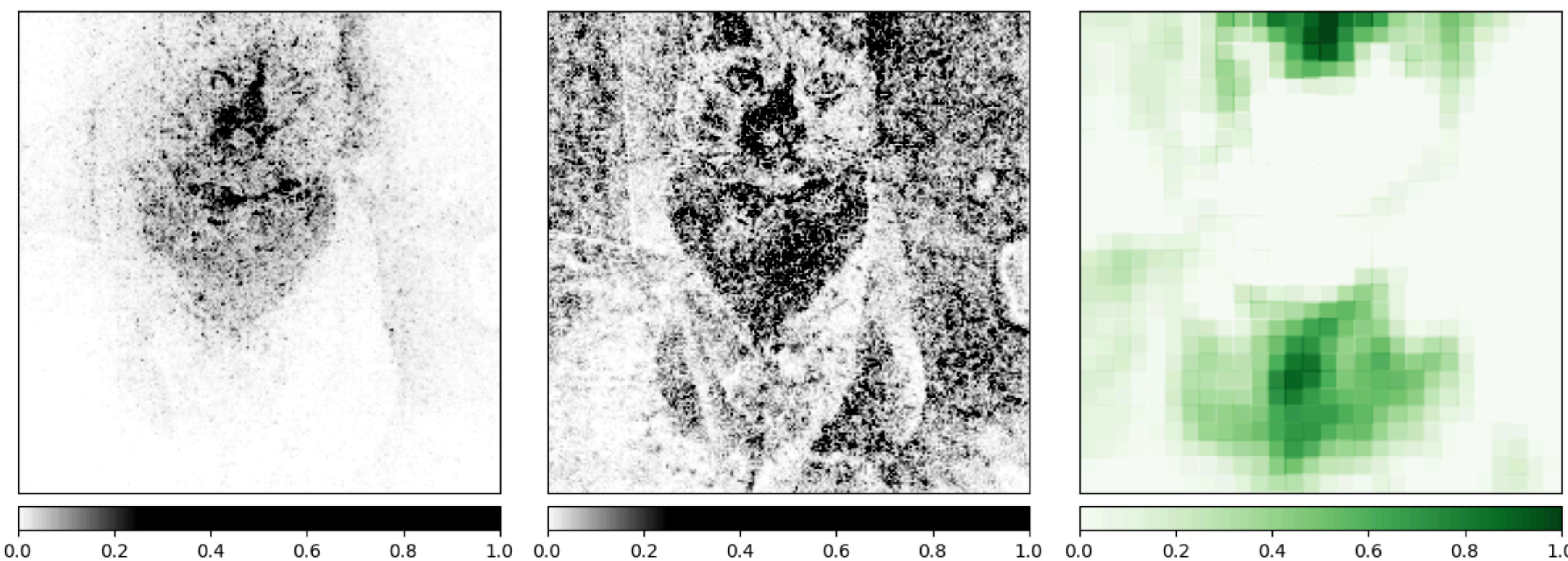
Interpretability/Explainability: Current State of Affairs

- There are lots of "explanation" approaches that can be used after training a deep net to try to understand what has been learned
  - Many of these are implemented in the Python package Captum developed by Meta/Facebook: https://captum.ai/

ResNet-18 (a CNN) predicts my cat to be an "Egyptian cat"

Crop image
(many CNNs need the input image to be a specific size)

What pixels are important for prediction?



These are the answers from 3 different explanation models (they give different answers!)

Warning: there's a **lot** of debate as to how much we should actually trust these explanations, as they can often be misleading

Interpretability/Explainability: Current State of Affairs

There are neural net architectures that *by design are interpretable* (e.g., prototypical part networks, neural topic models, neural decision tree models…)

- No separate explanation approach needed since model directly provides explanation
- My opinion: if you really care about interpretability/explainability, then you're better off using this sort of model

Unfortunately, deep nets with state-of-the-art prediction accuracy tend to be difficult to interpret

It's important to distinguish between tasks where interpretability is important vs ones where it's not as important

How to train a deep net

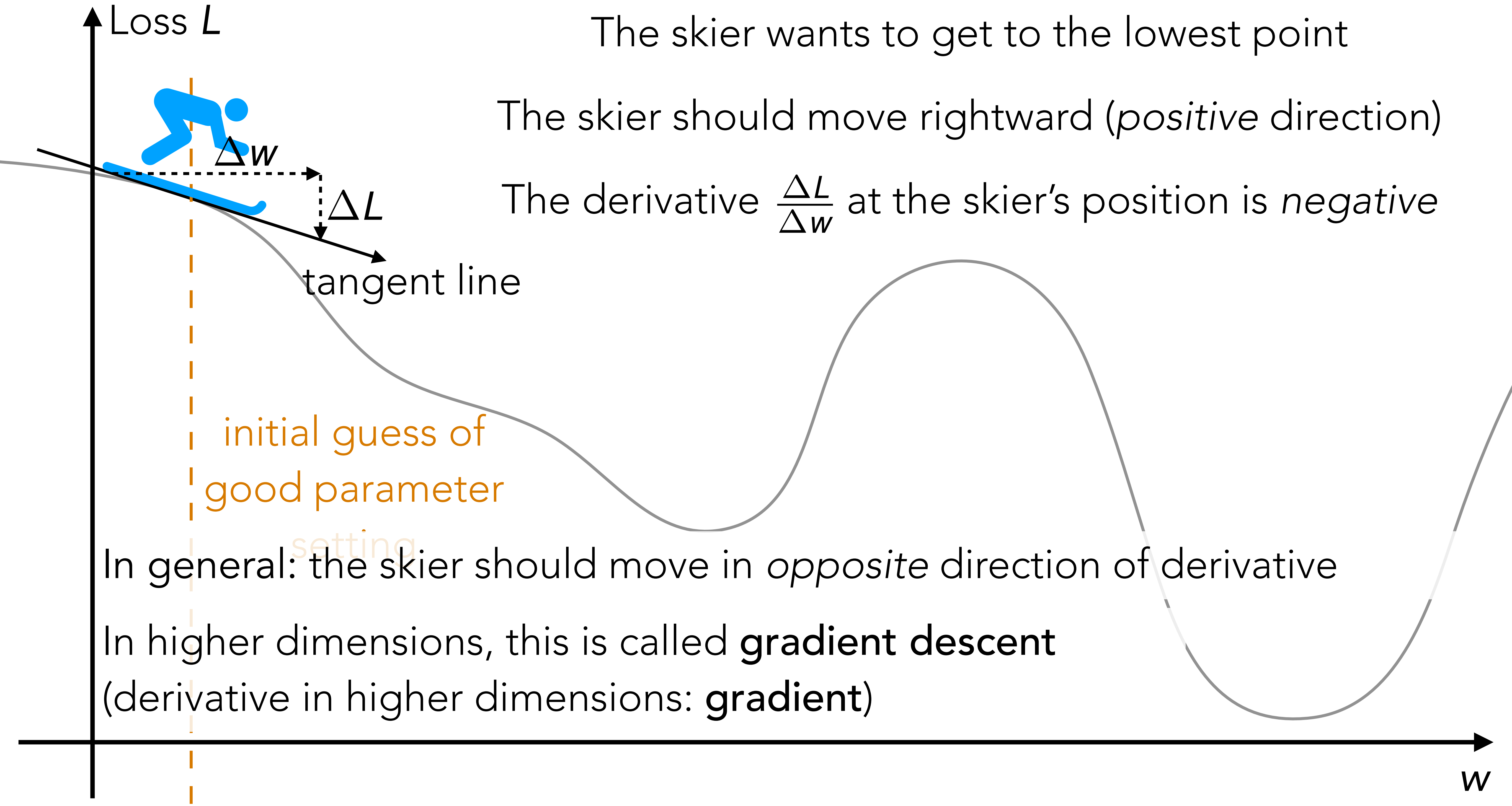
Training a Deep Net

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*

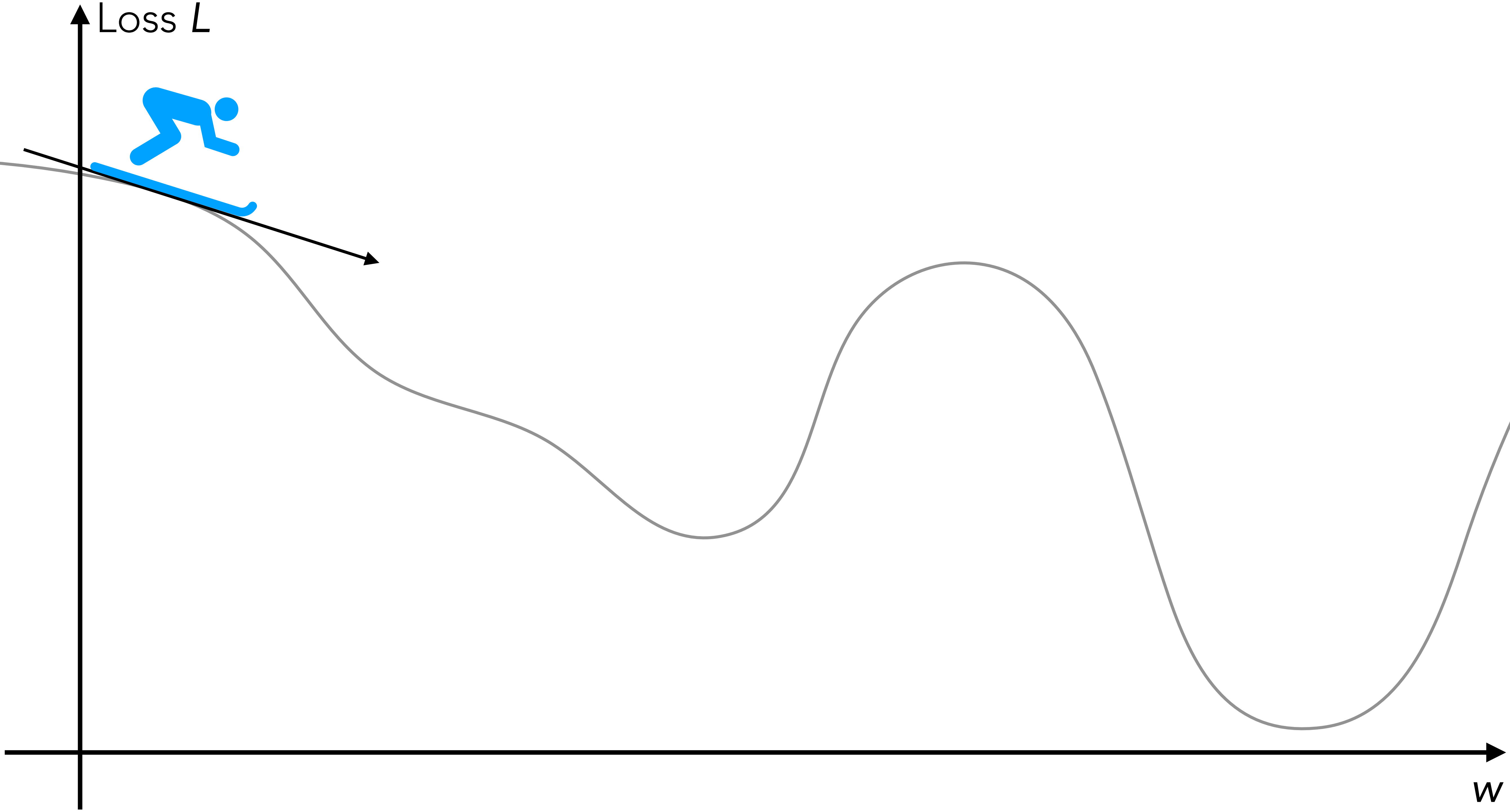


In general: the skier should move in *opposite* direction of derivative

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

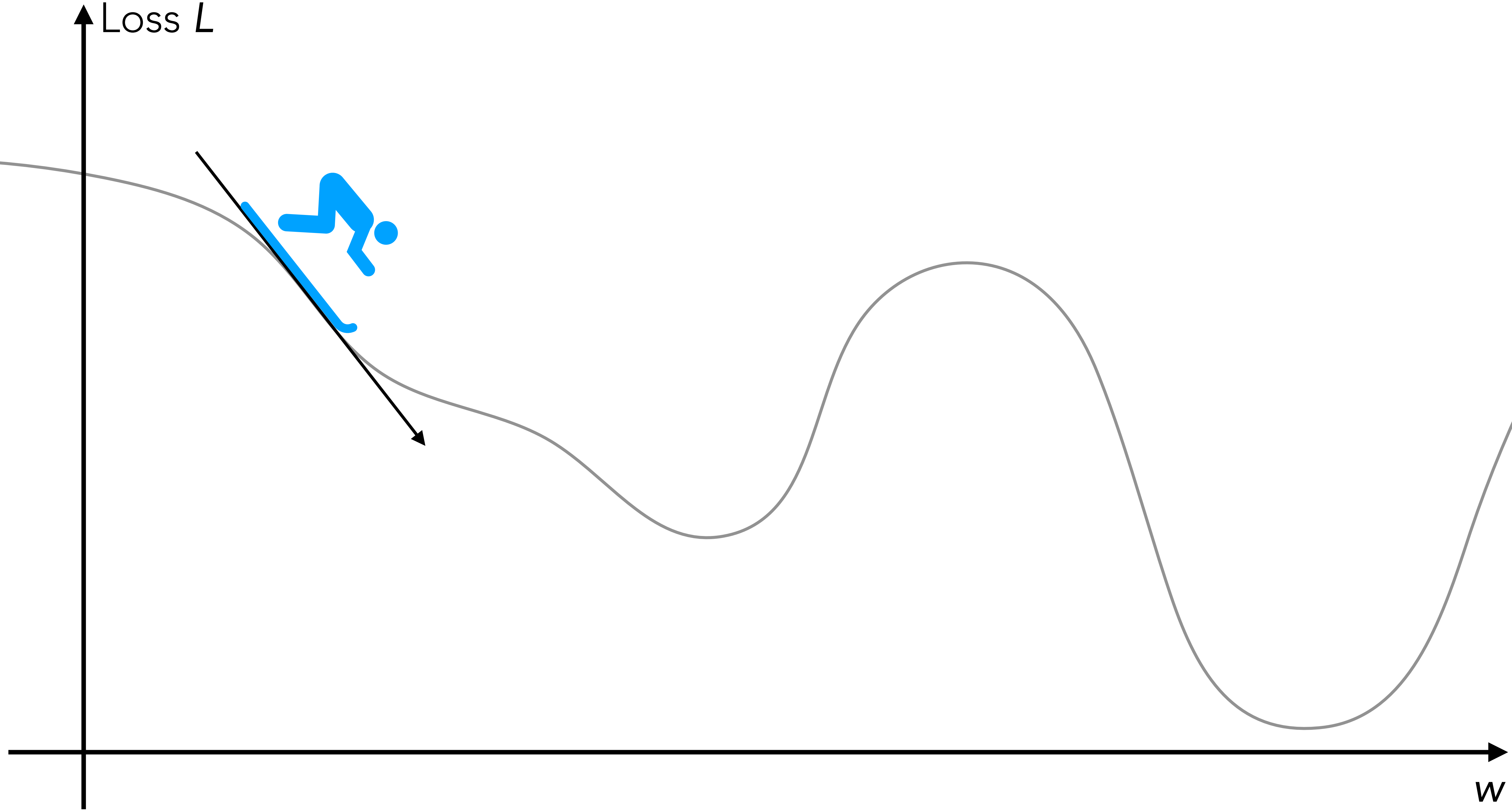
Training a Deep Net

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



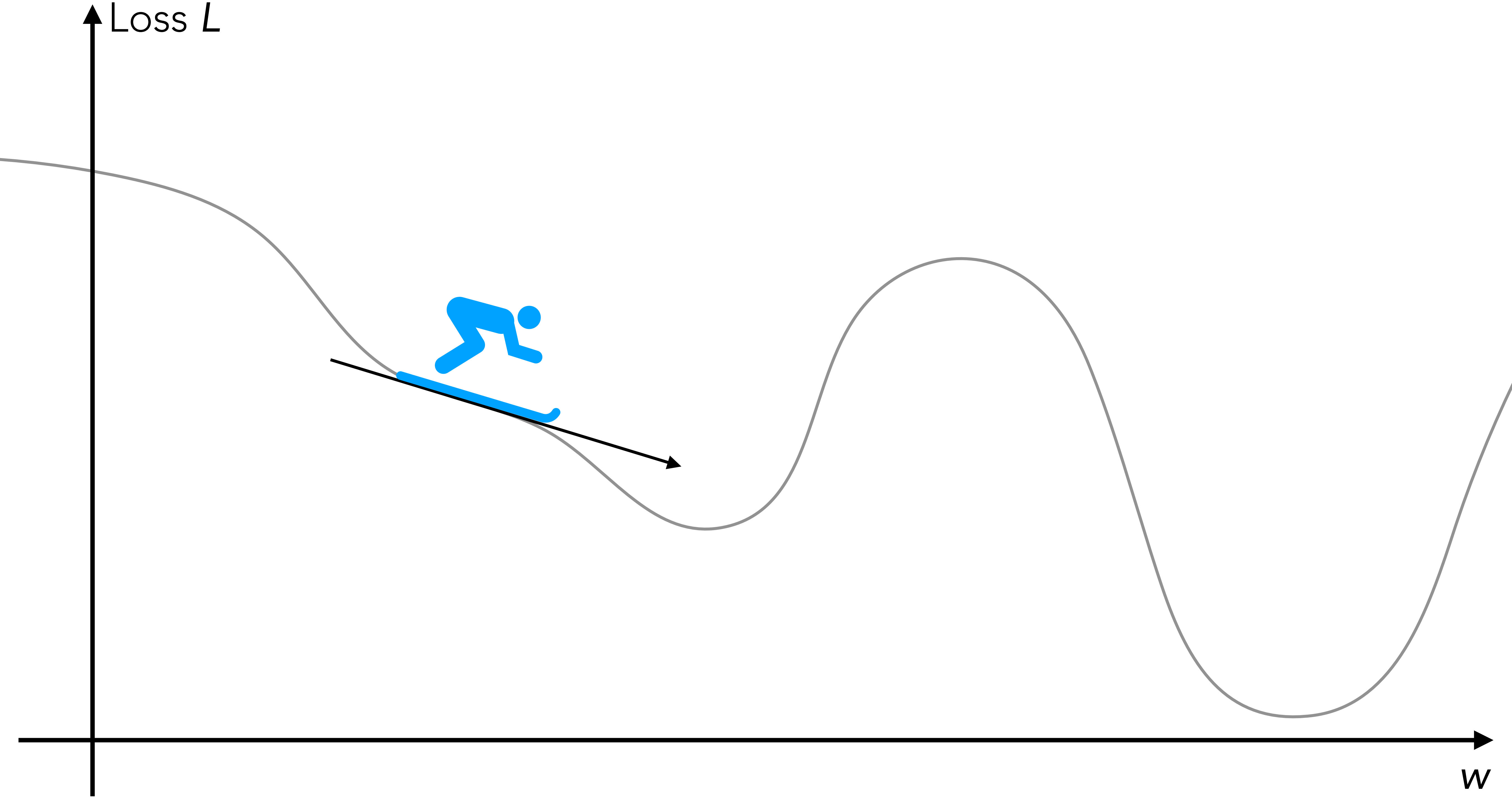
Training a Deep Net

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



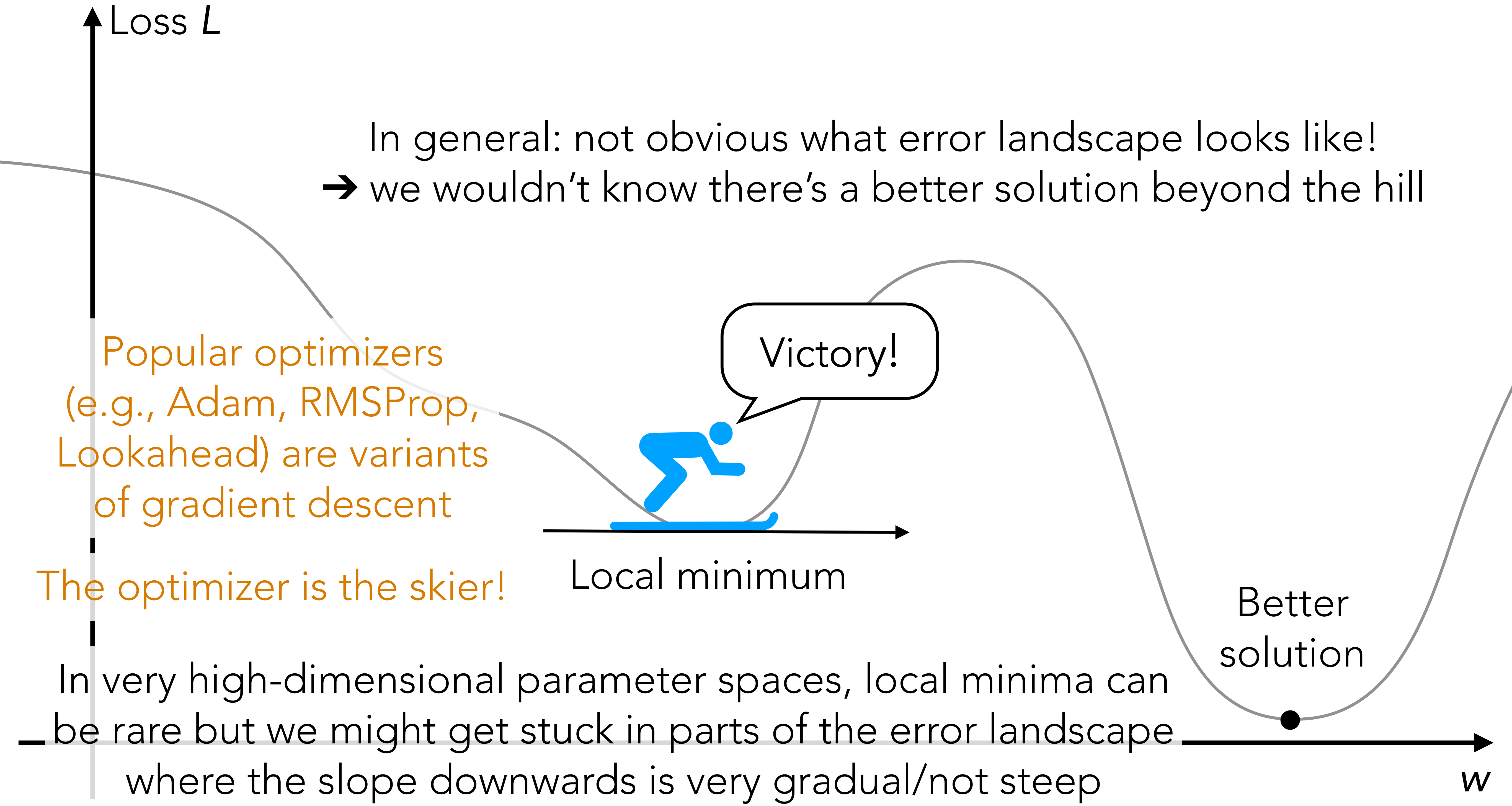
Training a Deep Net

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



Training a Deep Net

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



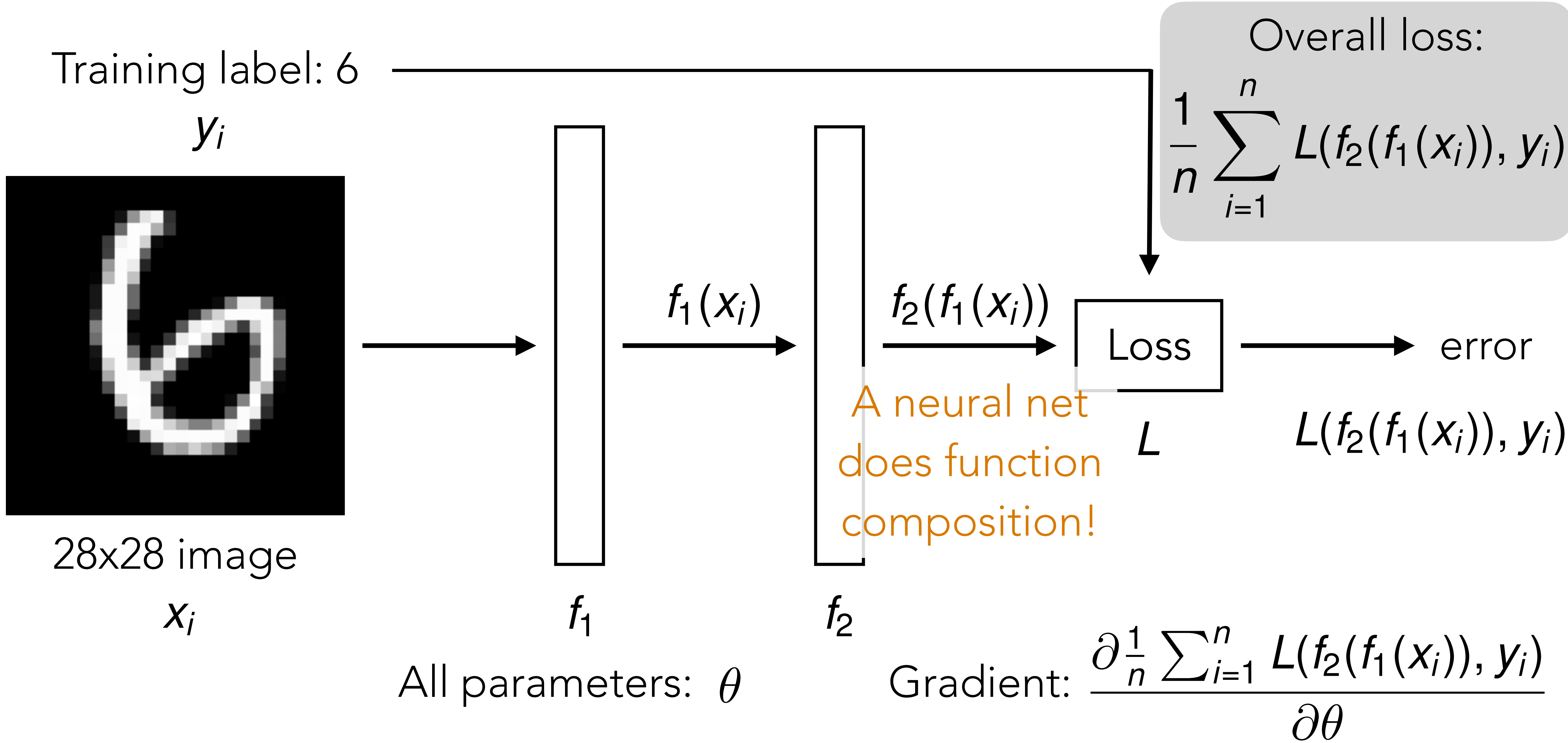
In general: not obvious what error landscape looks like!
➔ we wouldn't know there's a better solution beyond the hill

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

The optimizer is the skier!

In very high-dimensional parameter spaces, local minima can be rare but we might get stuck in parts of the error landscape where the slope downwards is very gradual/not steep

Handwritten Digit Recognition



Automatic differentiation is crucial in learning deep nets!

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

Gradient Descent

Training example 1 → loss 1

Training example 2 → loss 2

Training example 3 → loss 3

Training example 4 → loss 4

Training example 5 → loss 5

…

Training example n → loss n

↓

average loss

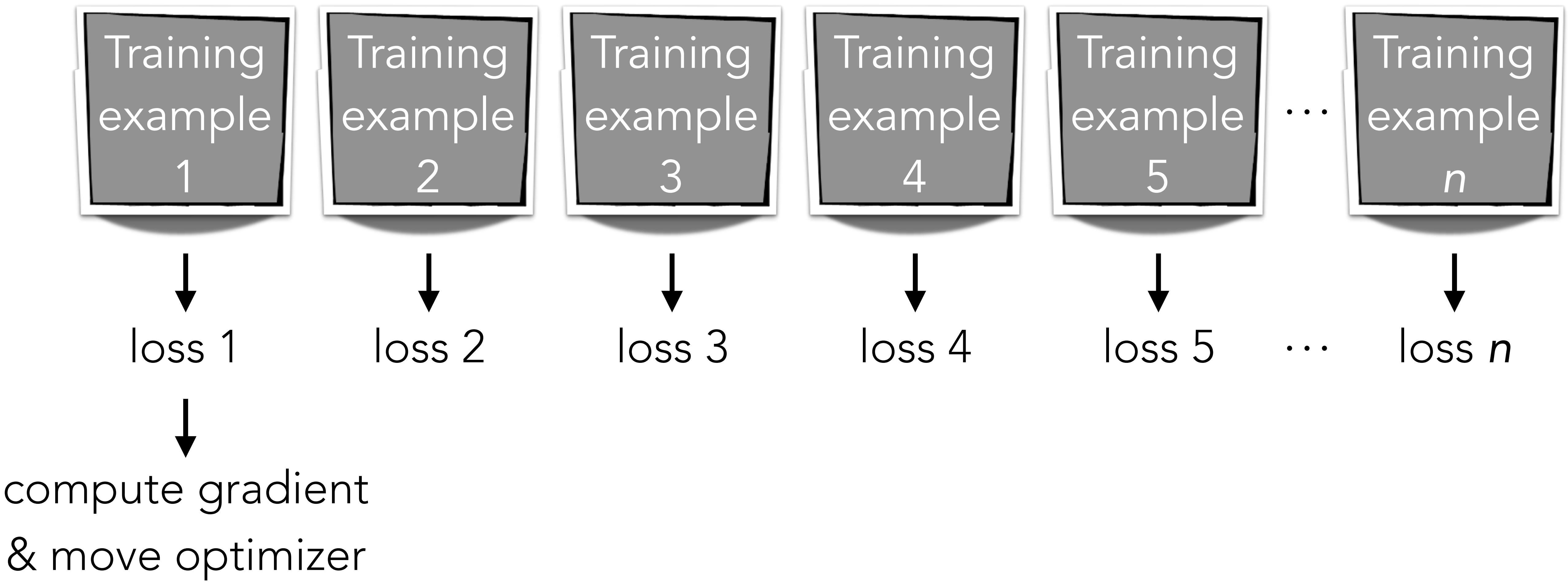
↓

compute gradient
& move optimizer

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

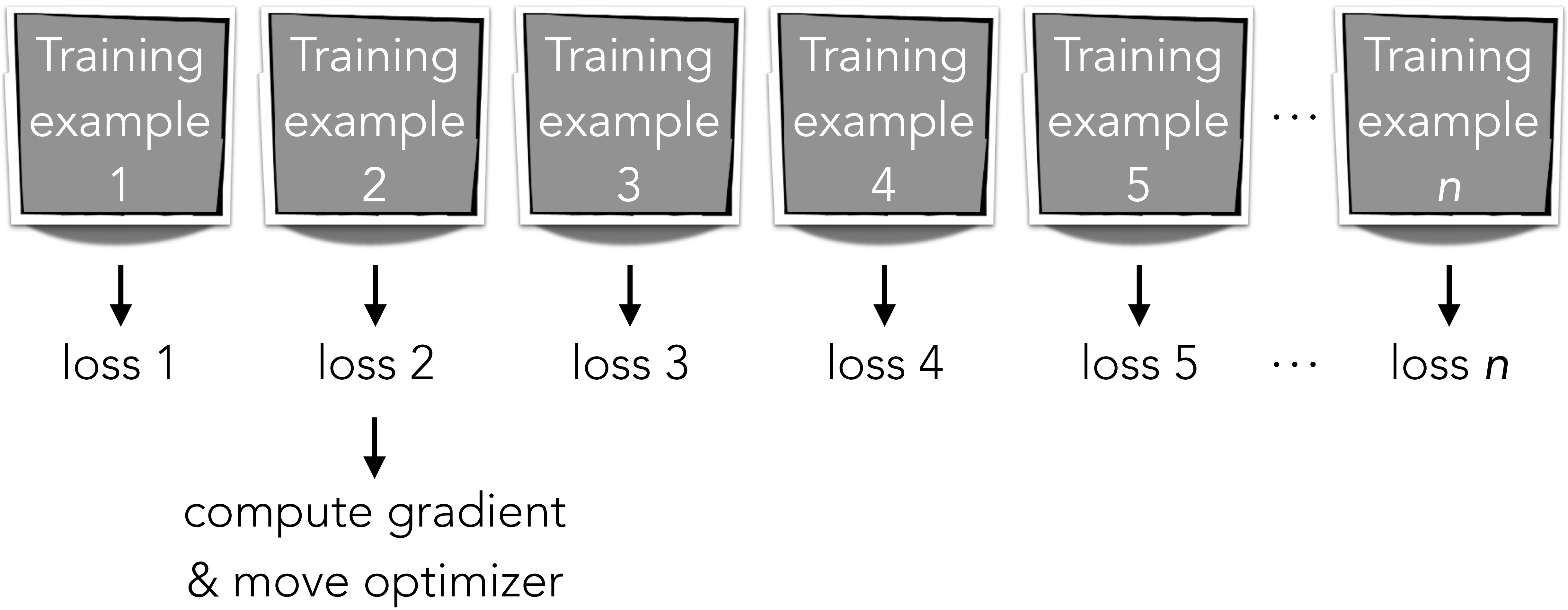
Computing gradients using all the training data seems really expensive!

Stochastic Gradient Descent (SGD)



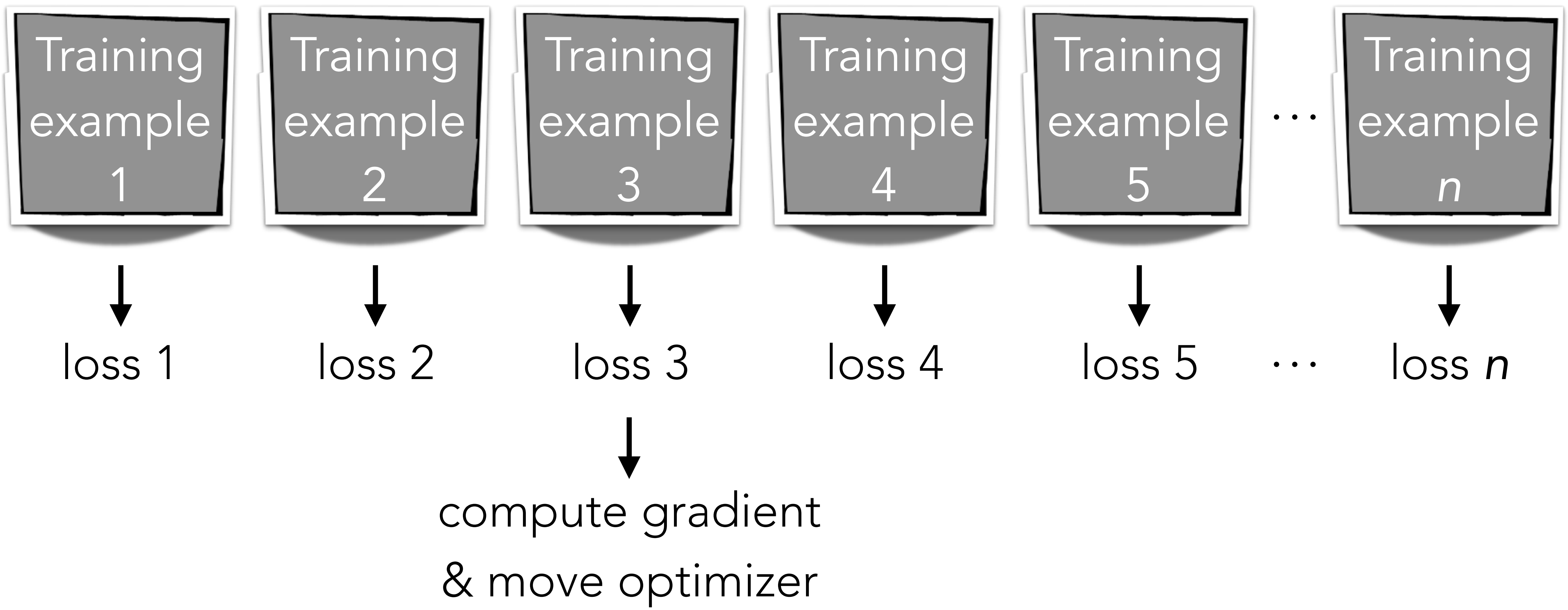
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



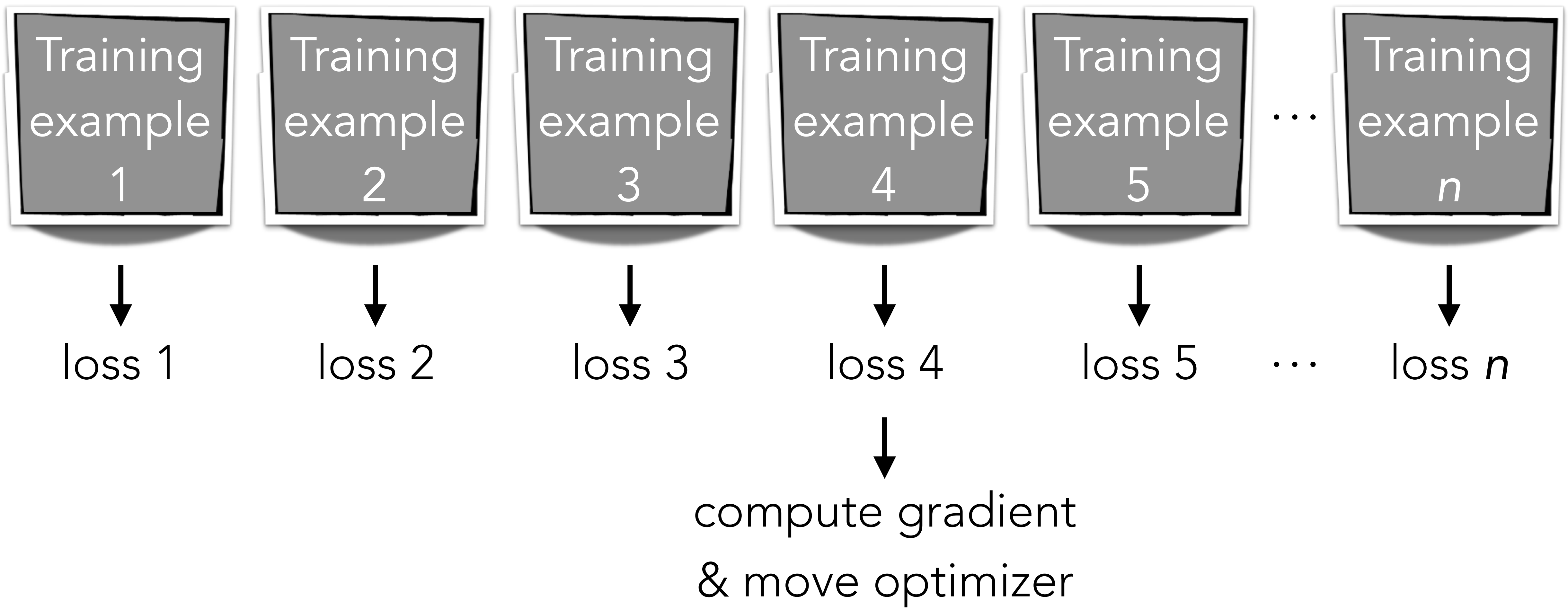
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



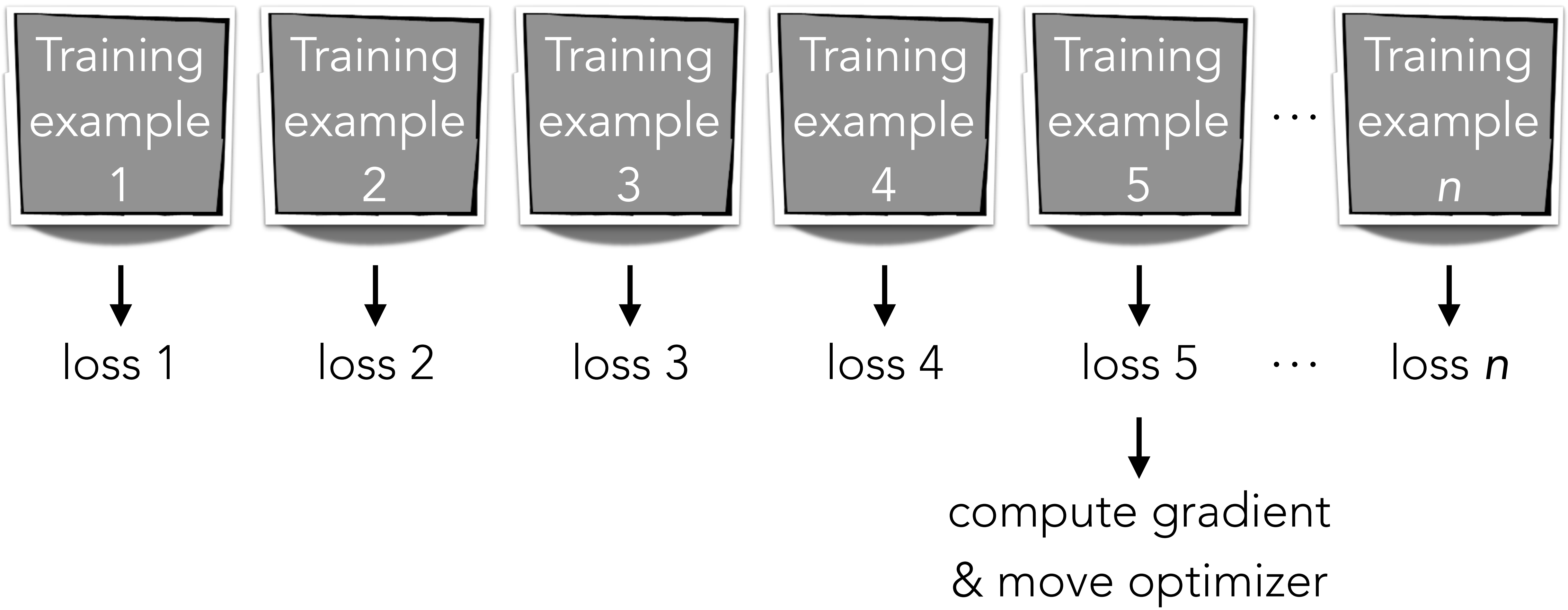
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

Stochastic Gradient Descent (SGD)



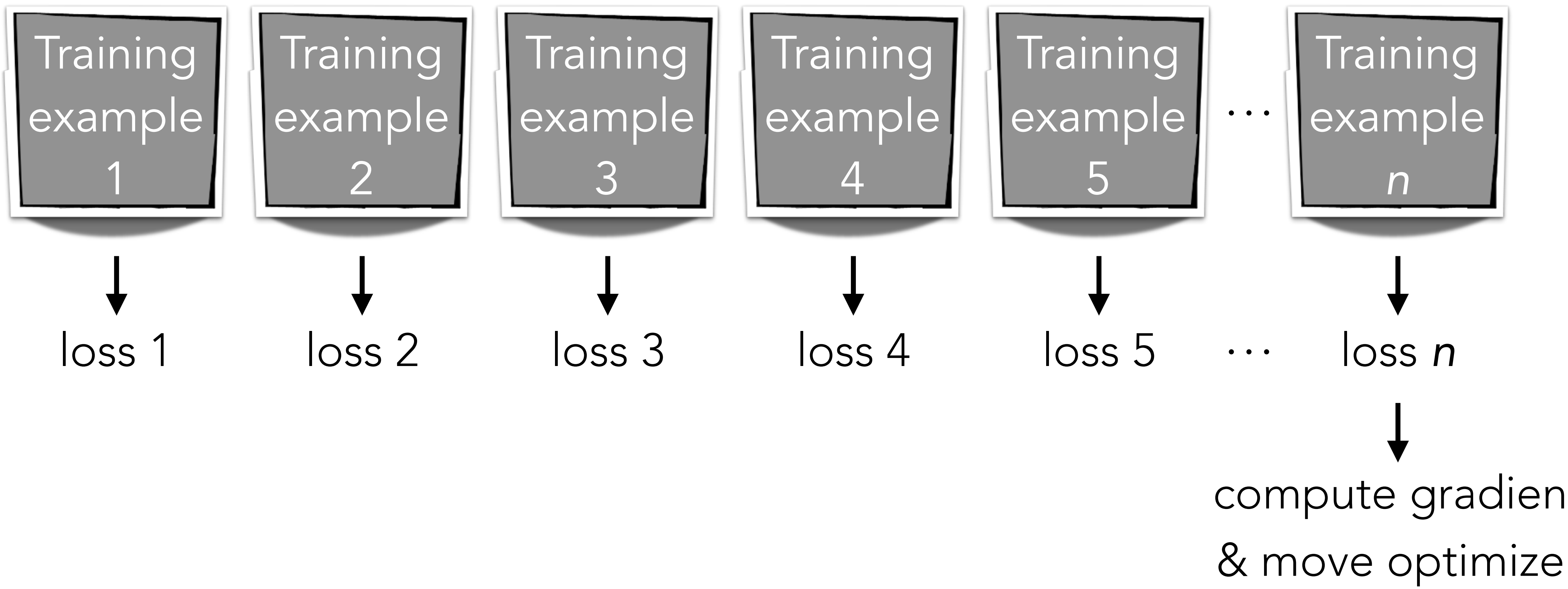
SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the "full" gradient)

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)

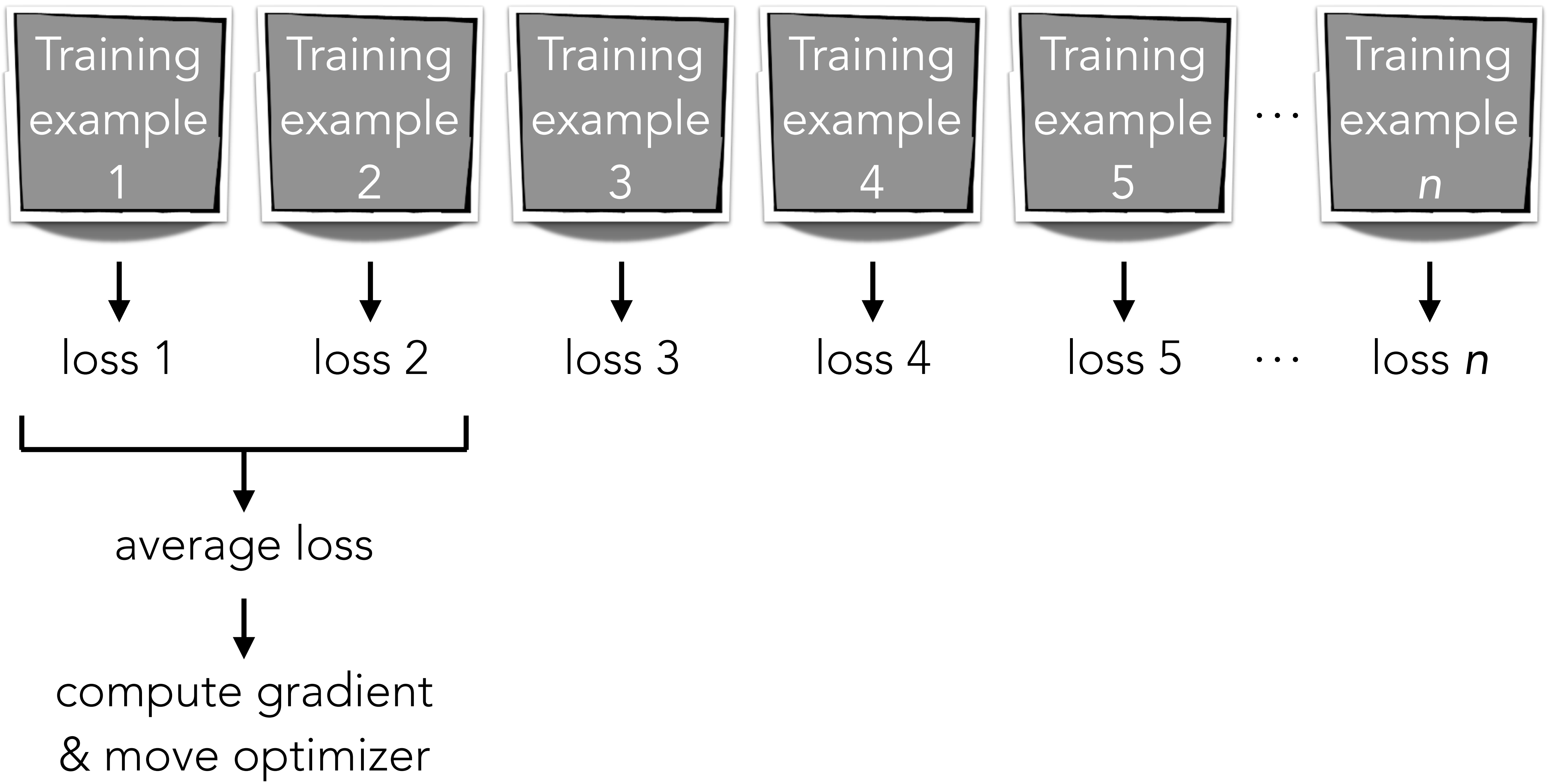
Stochastic Gradient Descent (SGD)

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 & move optimizer						

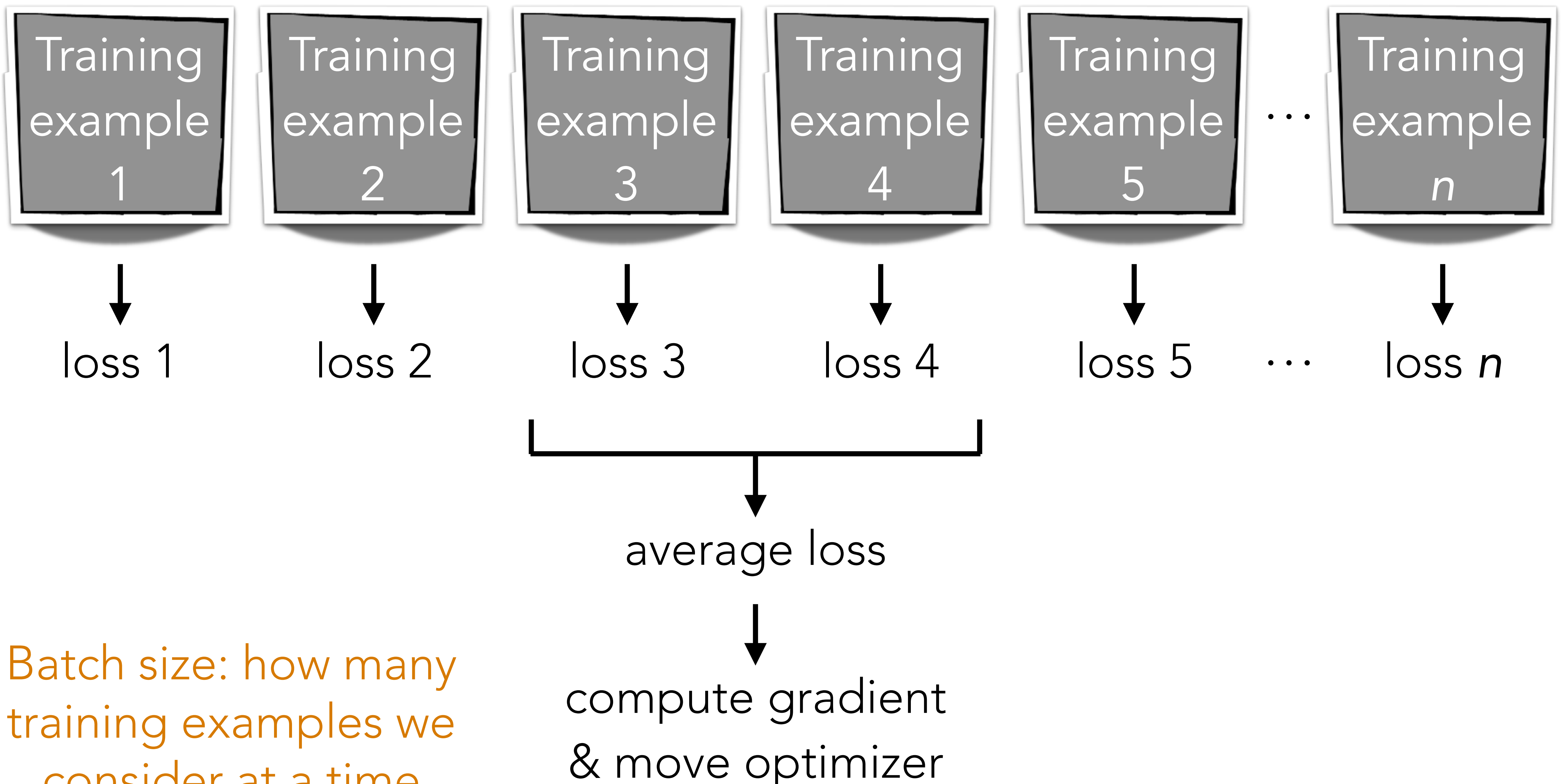
An epoch refers to 1 full pass through all the training data

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!!!

Unstructured Data Analytics (UDA)

Much like how many murder mysteries go unsolved,
many data analysis (unstructured or not) problems can be extremely difficult

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

Question

The dead body

This is provided by a domain expert

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

Not detailed in lecture but addressed by your final project, which must address a public policy problem

Sometimes: we aim to solve a prediction problem

UDA involves *lots* of data → write computer programs to assist analysis

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/hyperparameters
  - 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!
- There are *lots* of open policy questions regarding AI safety

Just earlier this month (Apr 2), Google released its AI safety plan