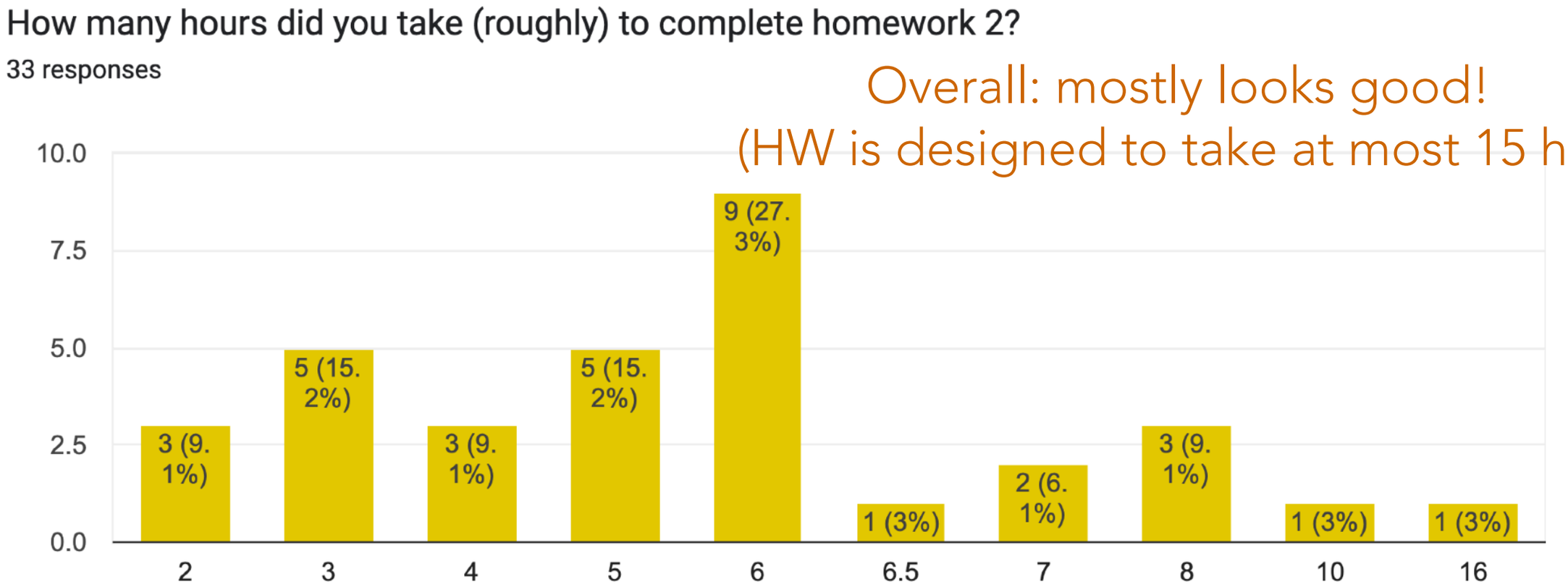
Carnegie Mellon University
Heinz College

94-775 Unstructured Data Analytics for Policy

Last lecture: Generative pre-trained transformers (GPTs); a few more deep learning concepts; course wrap-up

Slides by George H. Chen

HW2 Questionnaire (1/2)



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

Major free response recurring themes:

- Choosing settings/tuning hyperparameters for dimensionality reduction methods (and interpreting the resulting plots) is not easy
- Choosing which clustering approach to use (including settings for HDBSCAN) is not easy

Yes, setting hyperparameters/design choices is a recurring theme of 94-775 and it's often *not* easy and can be very dataset-dependent 🙁

HW2 Questionnaire (2/2)

My thoughts on choosing the number of clusters (for clustering) or number of topics (for topic models):

- I already showed that clustering/topic modeling score functions can sometimes be misleading
- I emphasized actually trying to interpret clusters or topics (e.g., visualizing top words via word clouds or just printing them)
    - The nice thing is that now you can have an LLM help you interpret top word lists (across different hyperparameters) to help you make sense of what trends there are as you vary hyperparameters
        - It's still a good idea to fact check the trends that an LLM tells you that it's finding to see if you agree or not
- Your TA Shurui will cover (among other things) how to get BERT word embeddings & how to use a neural topic model (BERTopic) in this Friday's recitation
    - BERTopic often gives more interpretable topics than LDA *but it has many hyperparameters (it uses UMAP & HDBSCAN by default)*

Final project presentations next Mon
8:30am are in **HBH 1202**
(not the usual classroom)

(Flashback)

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!

Text generation as a prediction problem

Just like the word2vec prediction problem:
we set up a prediction task even though there aren't any human-annotated ground truth labels

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.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters
(for simplicity; other ways to tokenize are possible)

Given `['T']`, predict next token `'h'`

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.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters (for simplicity; other ways to tokenize are possible)

Given `['T']`, predict next token `'h'`

Given `['T', 'h']`, predict next token `'e'`

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.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters (for simplicity; other ways to tokenize are possible)

Given `['T']`, predict next token `'h'`

Given `['T', 'h']`, predict next token `'e'`

Given `['T', 'h', 'e']`, predict next token `' '`

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.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters
(for simplicity; other ways to tokenize are possible)

Given `['T']`, predict next token `'h'`

Given `['T', 'h']`, predict next token `'e'`

Given `['T', 'h', 'e']`, predict next token `' '`

Given `['T', 'h', 'e', ' ']`, predict next token `'o'`

> 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.

Let's treat this string as a single data point (a time series of tokens)

For tokenization, let's split by individual characters (for simplicity; other ways to tokenize are possible)

Given `['T']`, predict next token `'h'`

Given `['T', 'h']`, predict next token `'e'`

Given `['T', 'h', 'e']`, predict next token `' '`

Given `['T', 'h', 'e', ' ']`, predict next token `'o'`

…

If the string has $L + 1$ tokens total, then there are L such prediction tasks

Vocabulary

First, let's agree on a vocabulary to use
(e.g., pick the unique ones seen in the dataset)

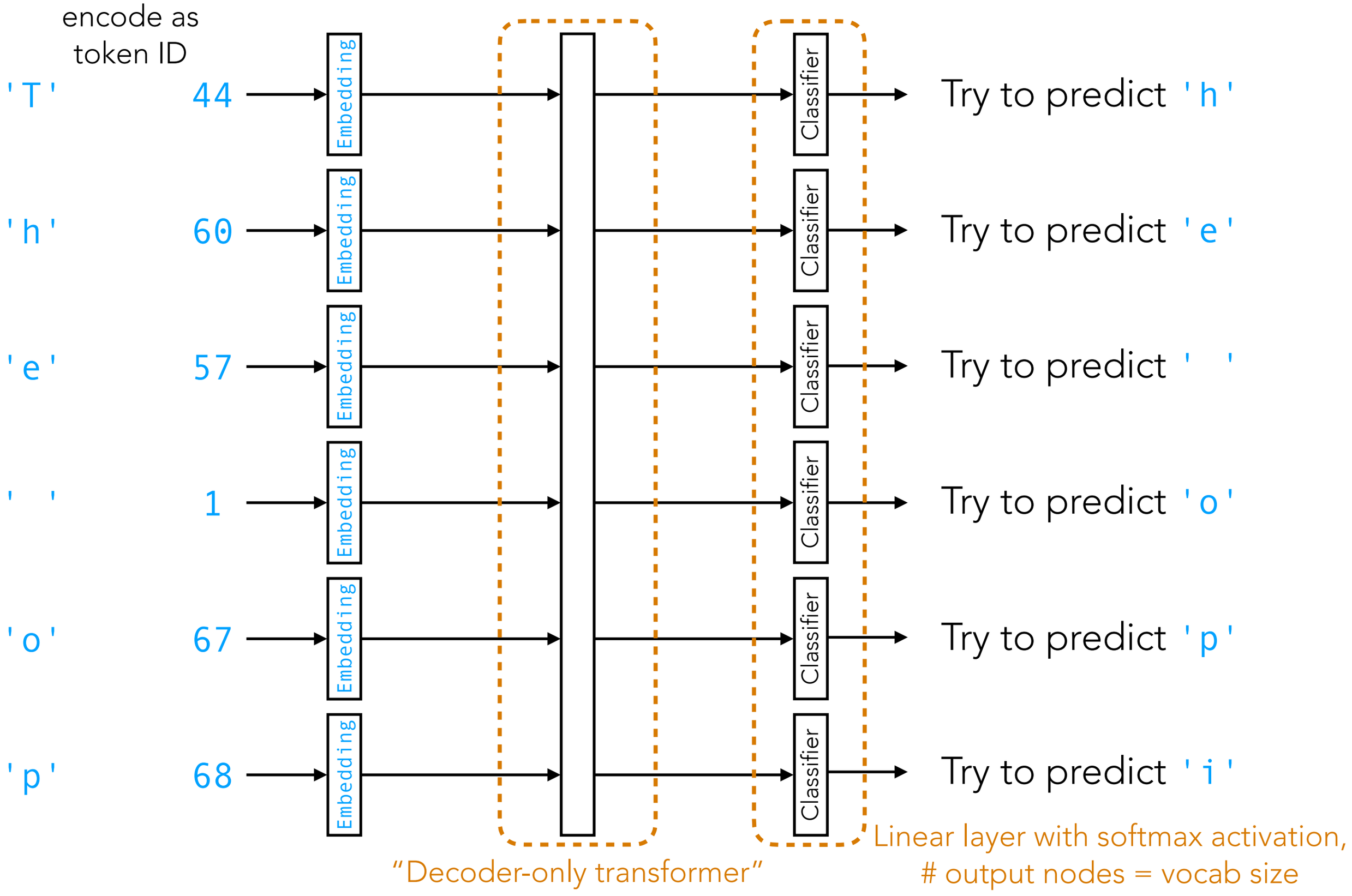
Token ID	Token
0	"\n"
1	" "
2	"\" "
3	"$"
⋮	⋮

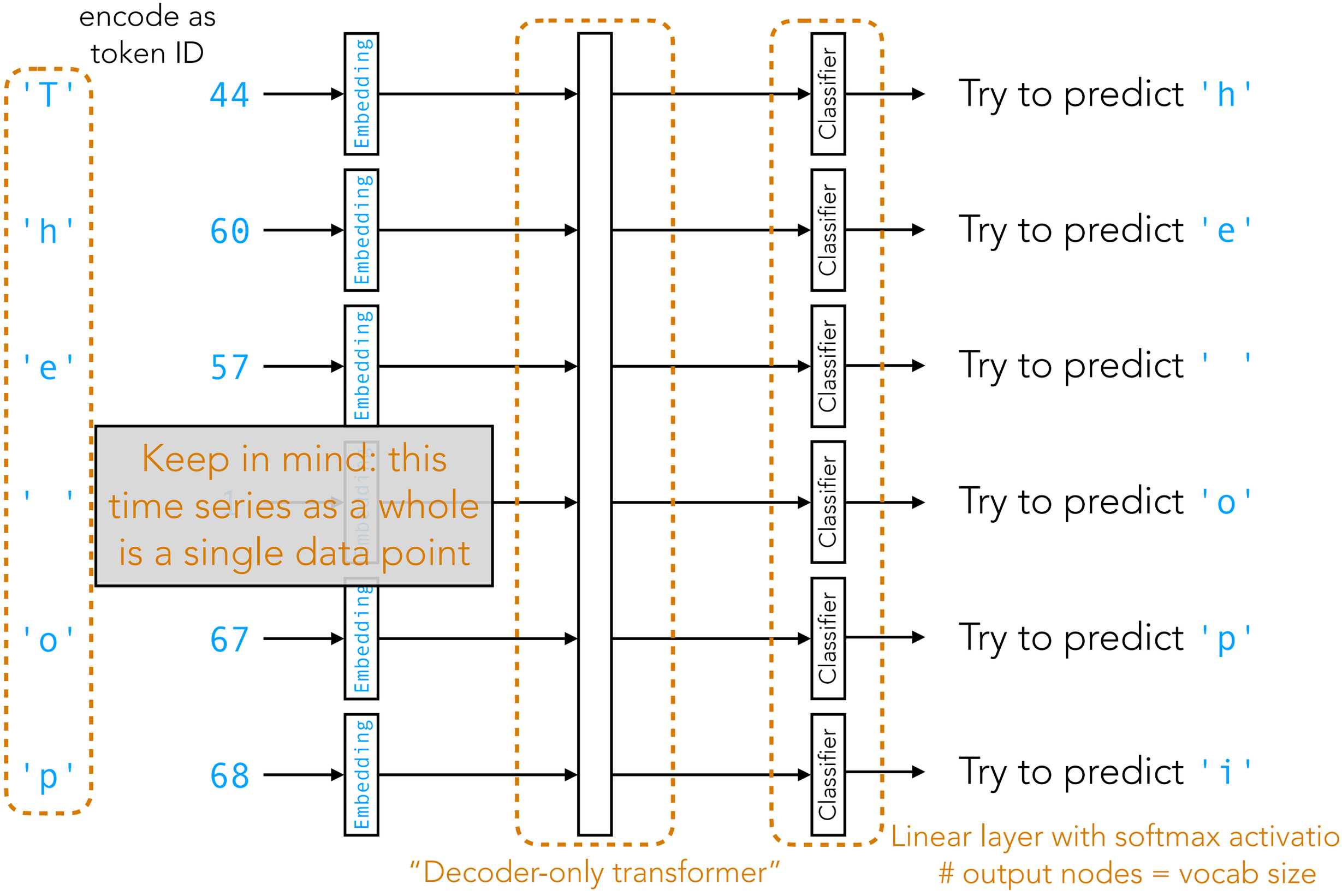
The demo has a vocabulary size of 86
(same as the text generation using n-grams lecture demo)

```
['T', 'h', 'e', ' ', 'o', 'p', 'i']
```

length = $L + 1$

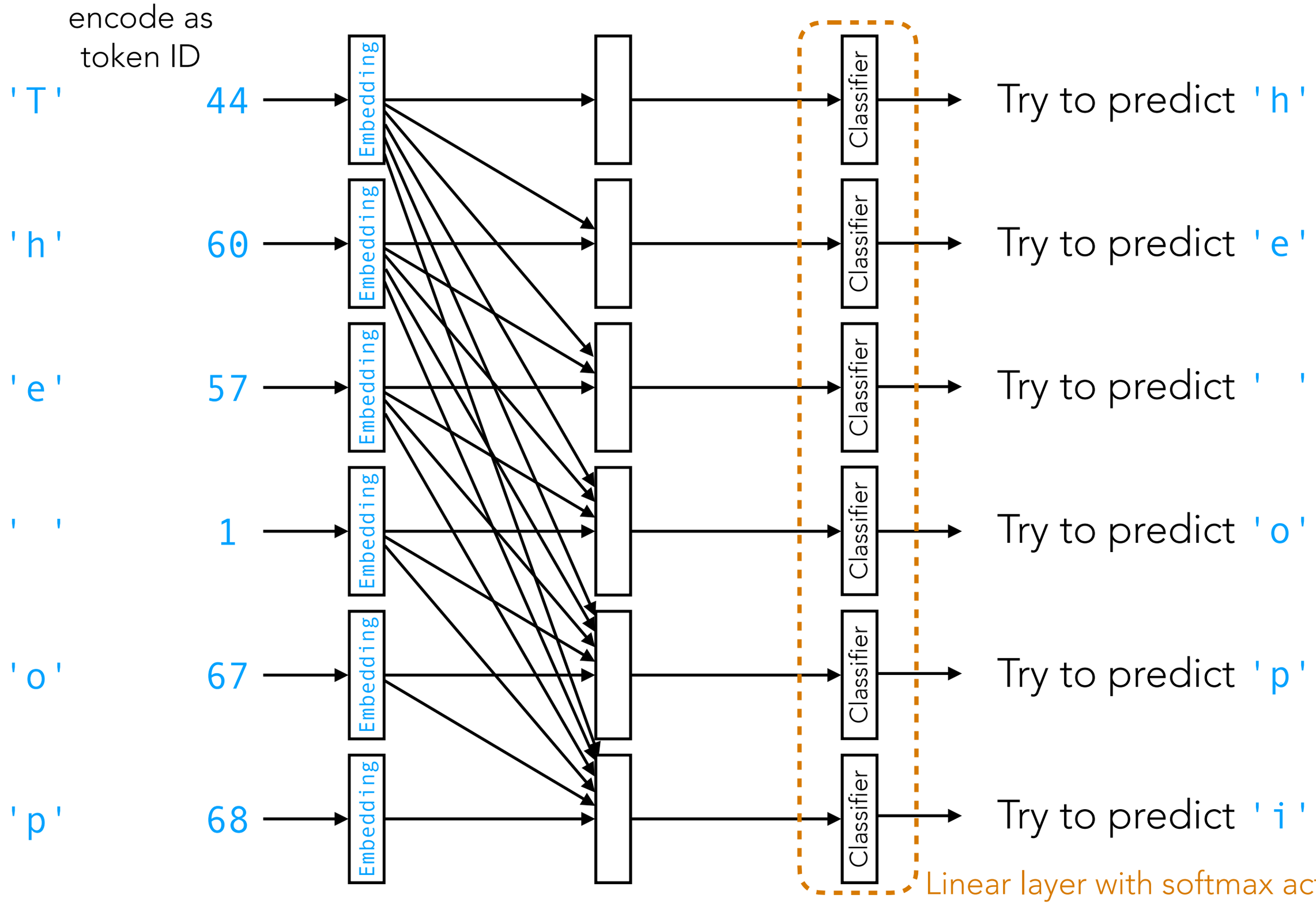
$L = 6$ in this example


encode as token ID
'T' 44
'h' 60
'e' 57
' ' 1
'o' 67
'p' 68
Embedding
Classifier
Try to predict 'h'
Try to predict 'e'
Try to predict ' '
Try to predict 'o'
Try to predict 'p'
Try to predict 'i'
"Decoder-only transformer"
Linear layer with softmax activation,
output nodes = vocab size

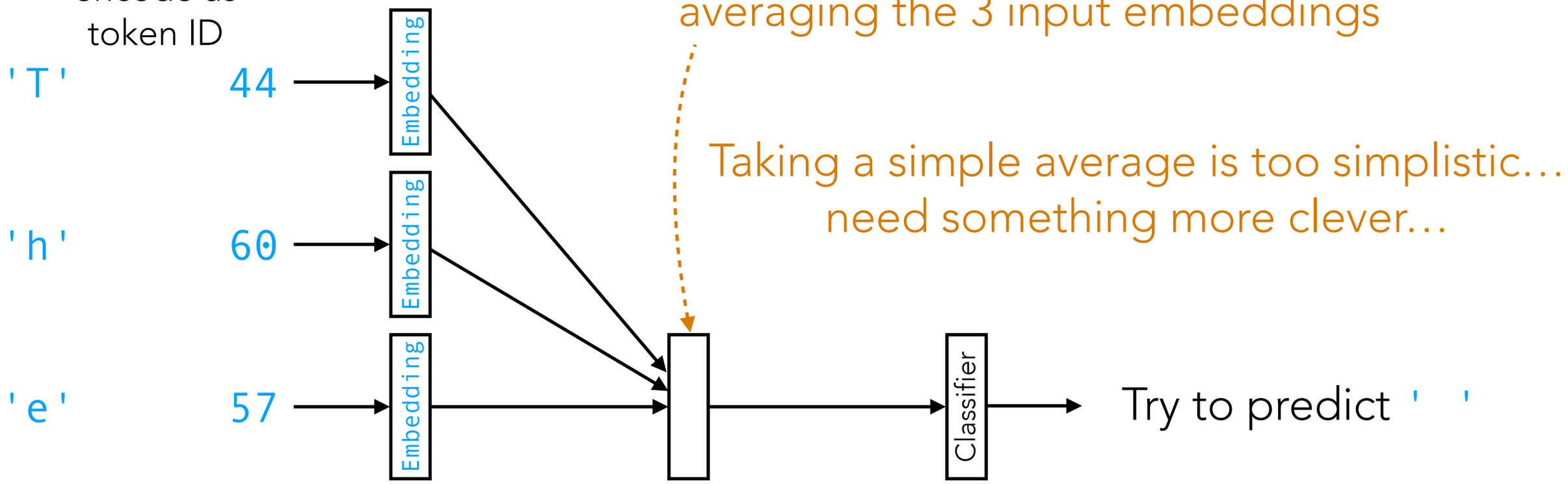

encode as token ID
'T'
'h'
'e'
' '
'o'
'p'
44
60
57
1
67
68
Embedding
Keep in mind: this time series as a whole is a single data point
Classifier
Try to predict 'h'
Try to predict 'e'
Try to predict ' '
Try to predict 'o'
Try to predict 'p'
Try to predict 'i'
"Decoder-only transformer"
Linear layer with softmax activation,
output nodes = vocab size

What is a decoder-only transformer?

This sort of dependence is "causal": any time step can only depend on its current input and all past inputs (and **not** on future time steps' inputs)



Let's focus on time step 2's prediction task for the moment…



Bad idea: have this box correspond to averaging the 3 input embeddings

Taking a simple average is too simplistic… need something more clever…

How should we combine information from the input embeddings?

Another issue: the input embeddings by themselves do not contain information about *when* the time steps happened

→ **Let's address this issue first**

encode as
token ID

'T' 44

Position: 0

'h' 60

Position: 1

'e' 57

Position: 2

Embedding

Embedding

Embedding

Classifier

Try to predict ' '

Each embedding: 1D table with D entries

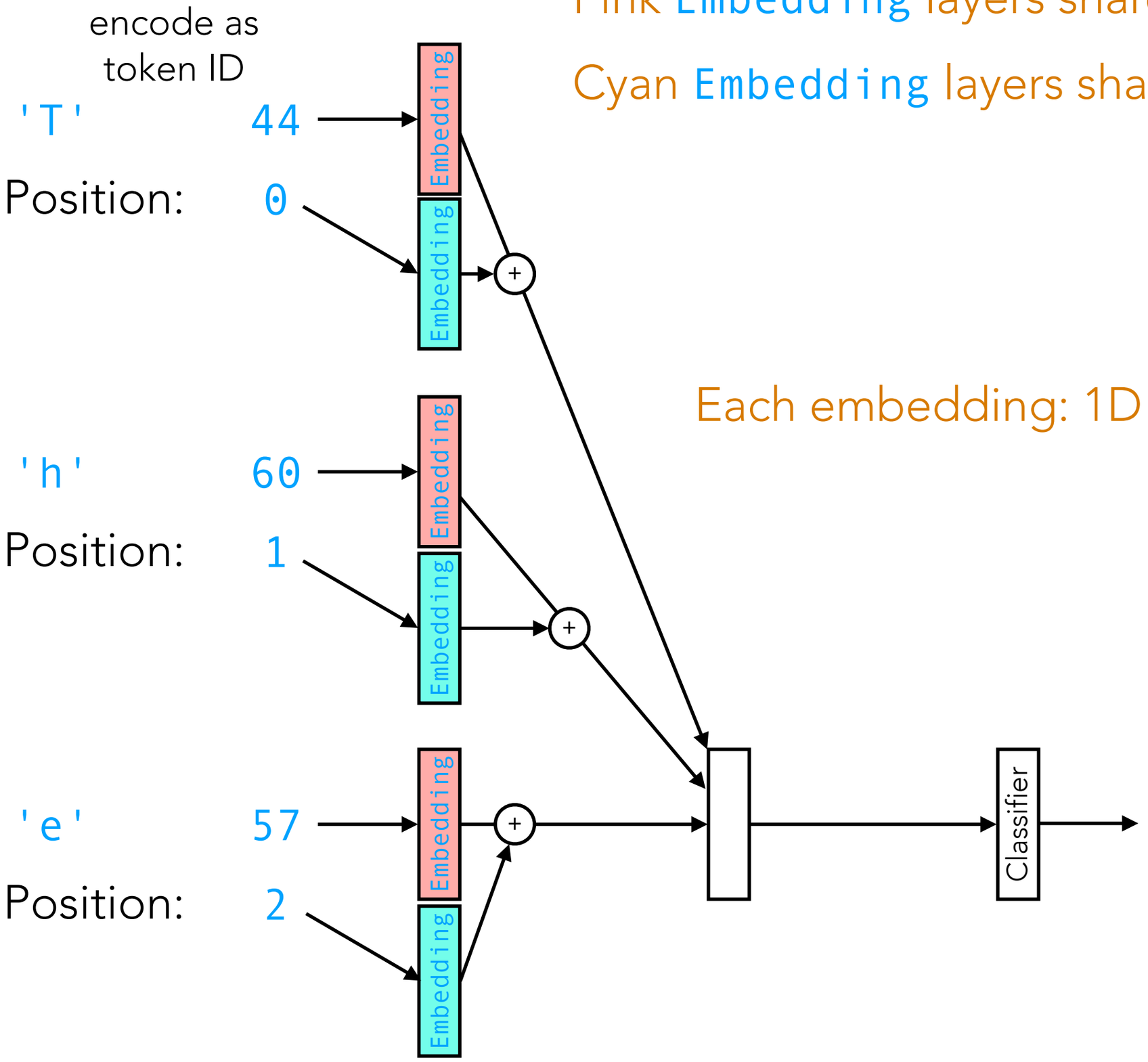
a hyperparameter
(512 in the demo)

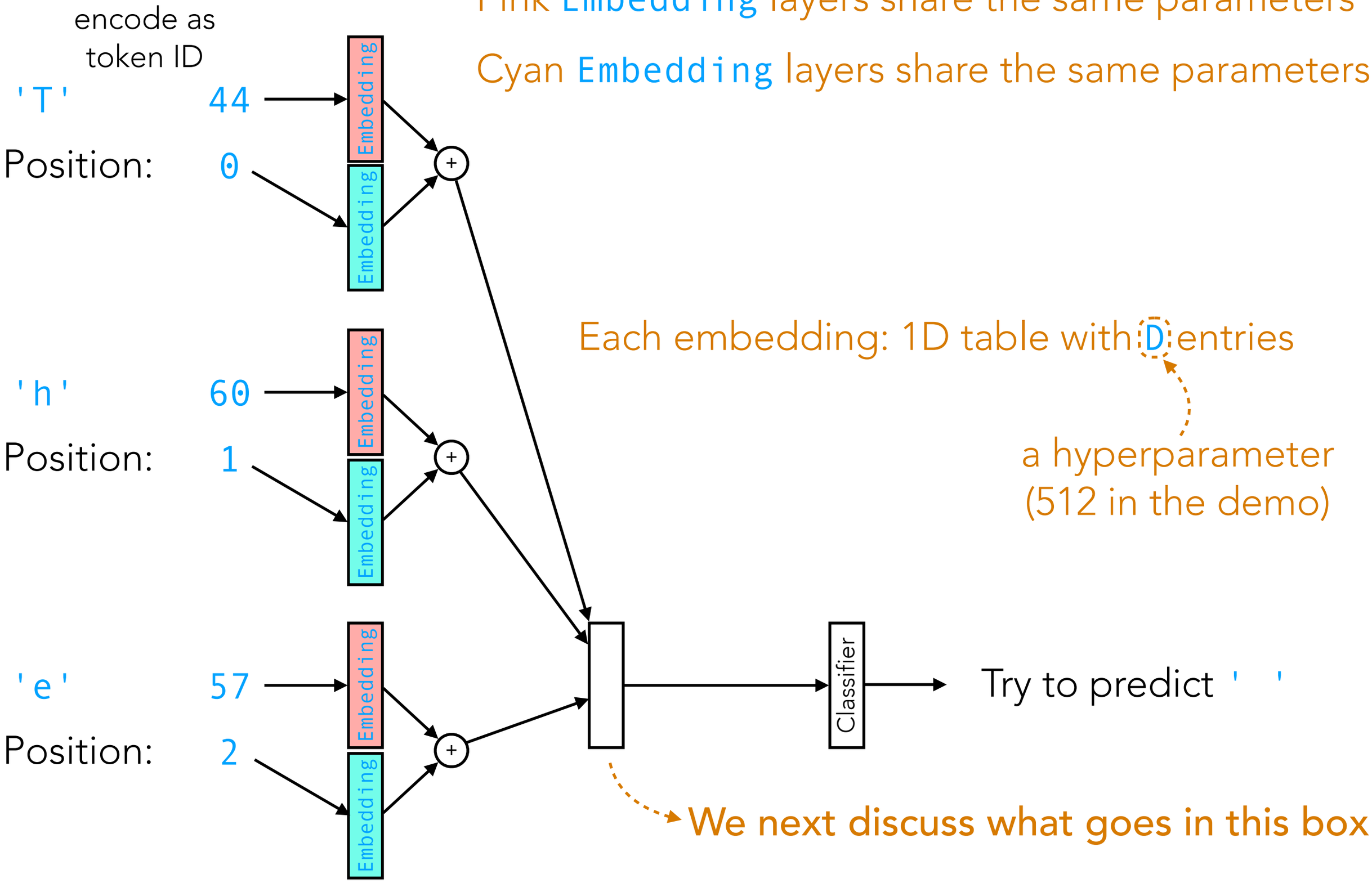
Pink `Embedding` layers share the same parameters

Cyan `Embedding` layers share the same parameters

Each embedding: 1D table with `D` entries

a hyperparameter
(512 in the demo)

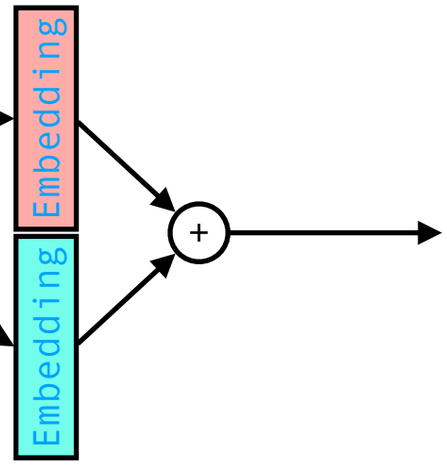



encode as
token ID
'T'
44
Position:
0
'h'
60
Position:
1
'e'
57
Position:
2
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
+
+
+
Classifier
Try to predict ' '
Pink Embedding layers share the same parameters
Cyan Embedding layers share the same parameters
Each embedding: 1D table with D entries
a hyperparameter
(512 in the demo)
We next discuss what goes in this box

encode as
token ID

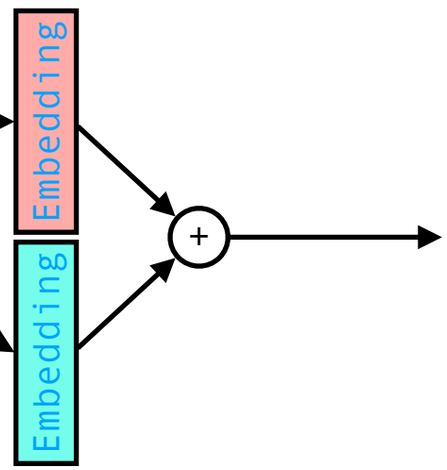
'T' 44

Position: 0



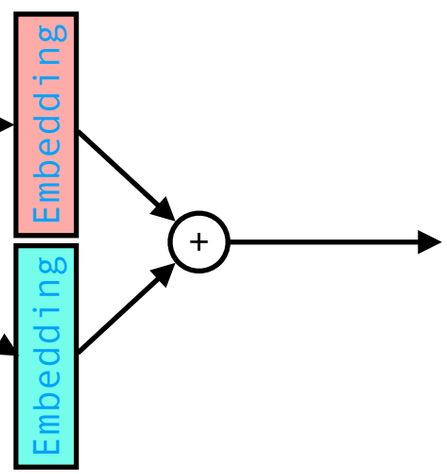
'h' 60

Position: 1



'e' 57

Position: 2



encode as token ID

'T' 44

Position: 0

Embedding

Embedding

+

Linear → key → key[0] "Here's how to find me"

Linear → query → query[0] "Here's what I'm looking for"

Linear → value → value[0] "Here's my information"

'h' 60

Position: 1

Embedding

Embedding

+

Linear → key → key[1]

Linear → query → query[1]

Linear → value → value[1]

'e' 57

Position: 2

Embedding

Embedding

+

Linear → key → key[2]

Linear → query → query[2]

Linear → value → value[2]

keys/queries/values are 1D tables with the same # entries: H

a hyperparameter (256 in the demo)

Remember: at this point, we are only computing the output for time step 2

How much should time step 0's information contribute (to the output for time step 2)?

Idea: make the contribution amount dependent on:

```
w[0] = np.dot(query[2], key[0])
```

How much should time step 1's information contribute?

```
w[1] = np.dot(query[2], key[1])
```

How much should time step 2's information contribute?

```
w[2] = np.dot(query[2], key[2])
```

Let's normalize the weights so they are probabilities:

```
w_norm = softmax(w)
```

Output at time step 2:

```
w_norm[0]*value[0] + w_norm[1]*value[1] \
+ w_norm[2]*value[2]
```

Remember: at this point, we are only computing the output for time step 2

How much should time step 0's information contribute (to the output for time step 2)?

Idea: make the contribution amount dependent on:

```
w[0] = np.dot(query[2], key[0])
```

How much should time step 1's information contribute?

```
w[1] = np.dot(query[2], key[1])
```

How much should time step 2's information contribute?

```
w[2] = np.dot(query[2], key[2])
```

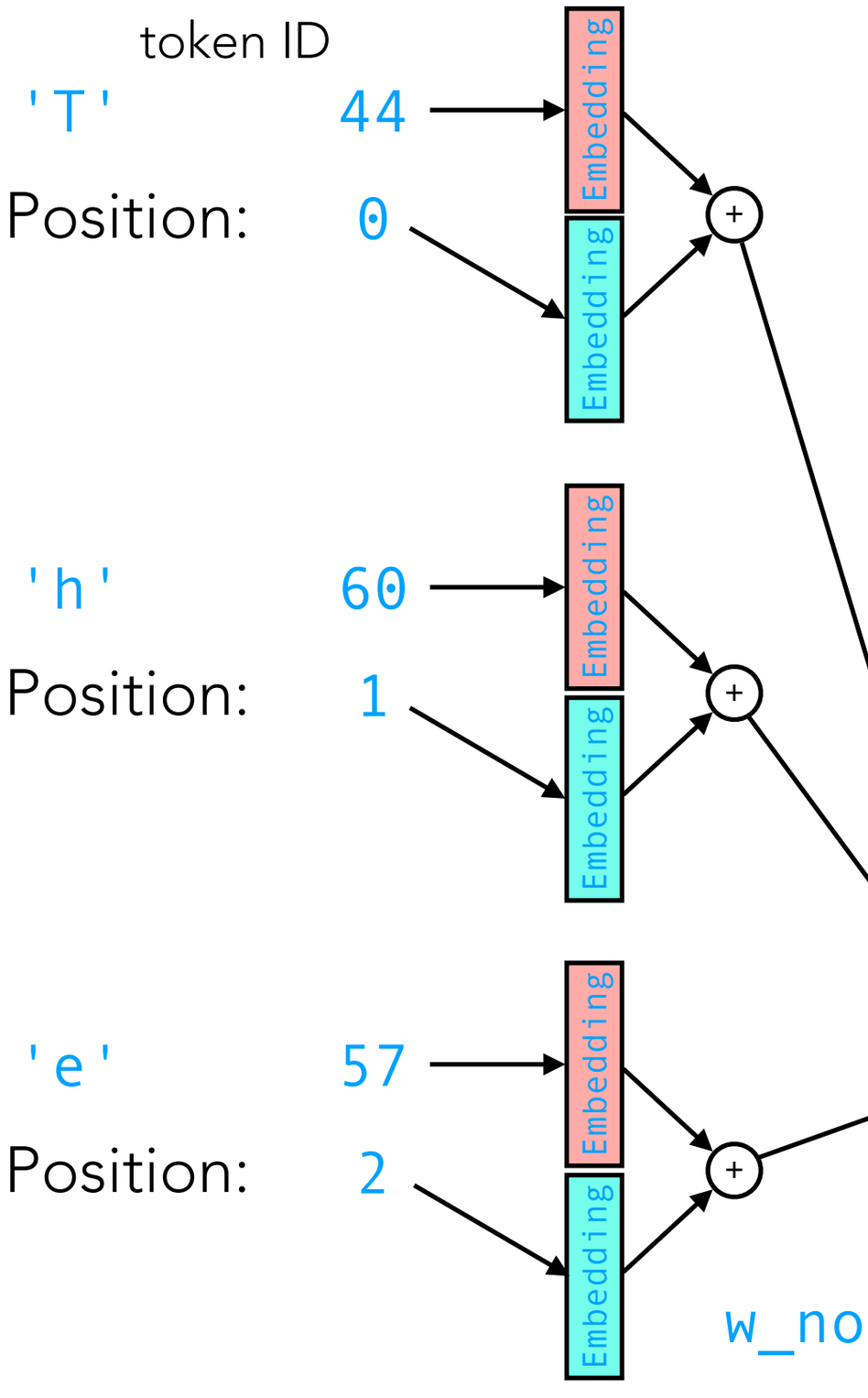
Let's normalize the weights so they are probabilities:

```
w_norm = softmax(w / np.sqrt(H))
```

In practice: include this division (helps with training)

Output at time step 2:

```
w_norm[0]*value[0] + w_norm[1]*value[1] \
    + w_norm[2]*value[2]
```



The hope: keys, queries, and values that get learned help with prediction

But what if we get unlucky and the keys, queries, and values found aren't great (or only focus on a single concept)?

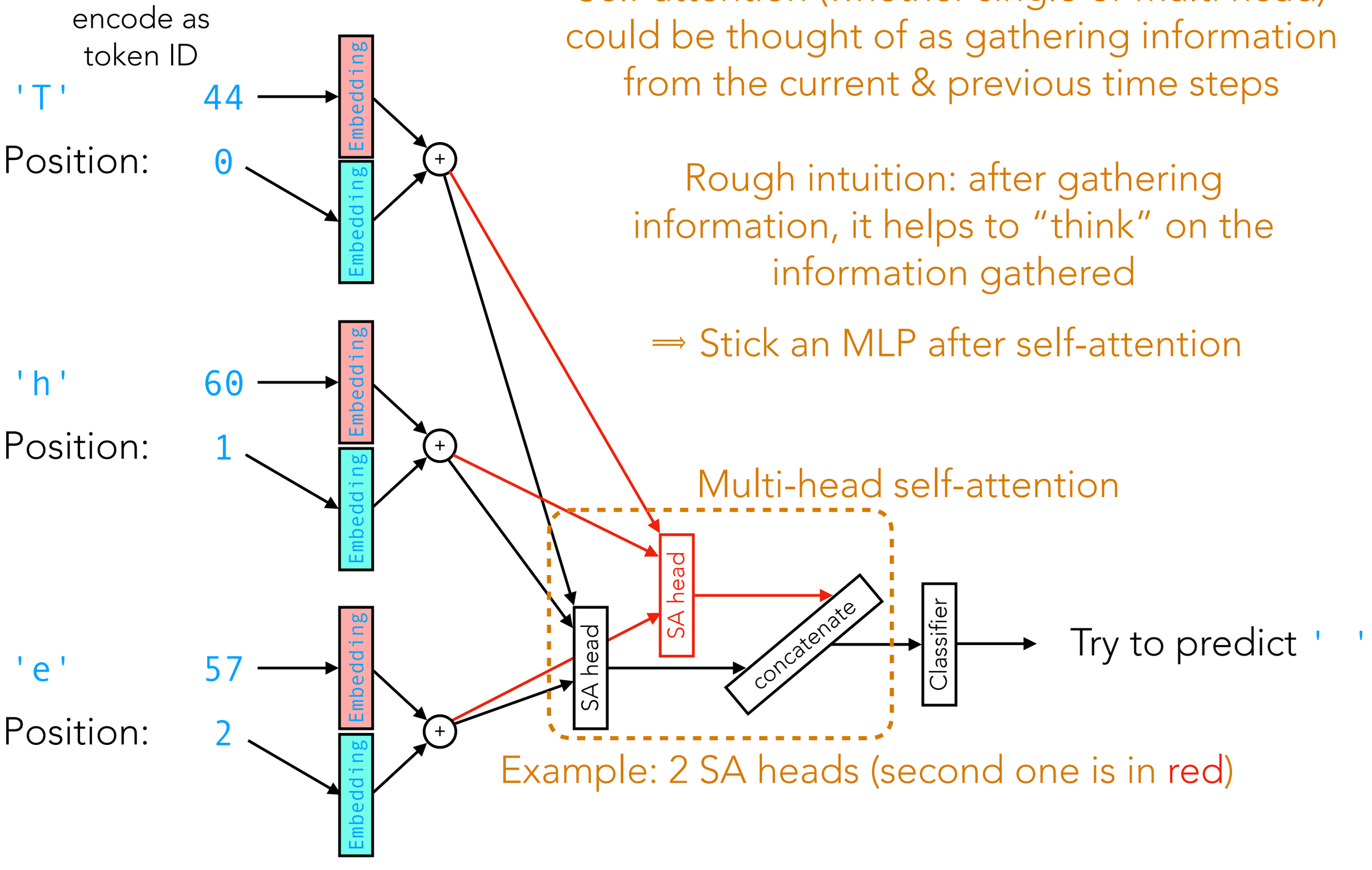
Analogy: imagine if we used a Conv2d layer but only used 1 filter and hoped that the 1 filter captures everything

The fix: use many self-attention heads

Self-attention (whether single or multi head) could be thought of as gathering information from the current & previous time steps

Rough intuition: after gathering information, it helps to "think" on the information gathered

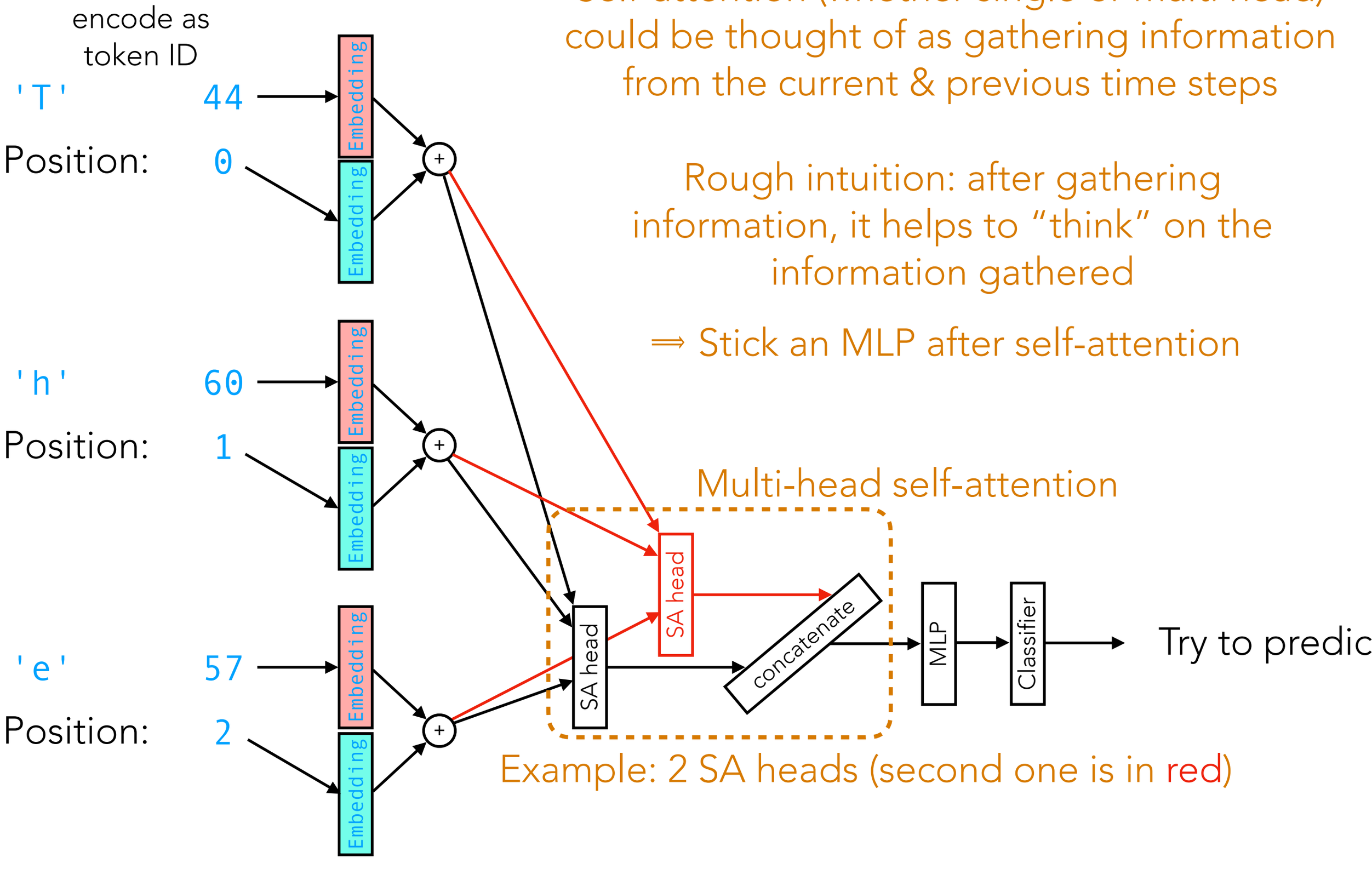
⟹ Stick an MLP after self-attention



Self-attention (whether single or multi head) could be thought of as gathering information from the current & previous time steps

Rough intuition: after gathering information, it helps to "think" on the information gathered

⟹ Stick an MLP after self-attention

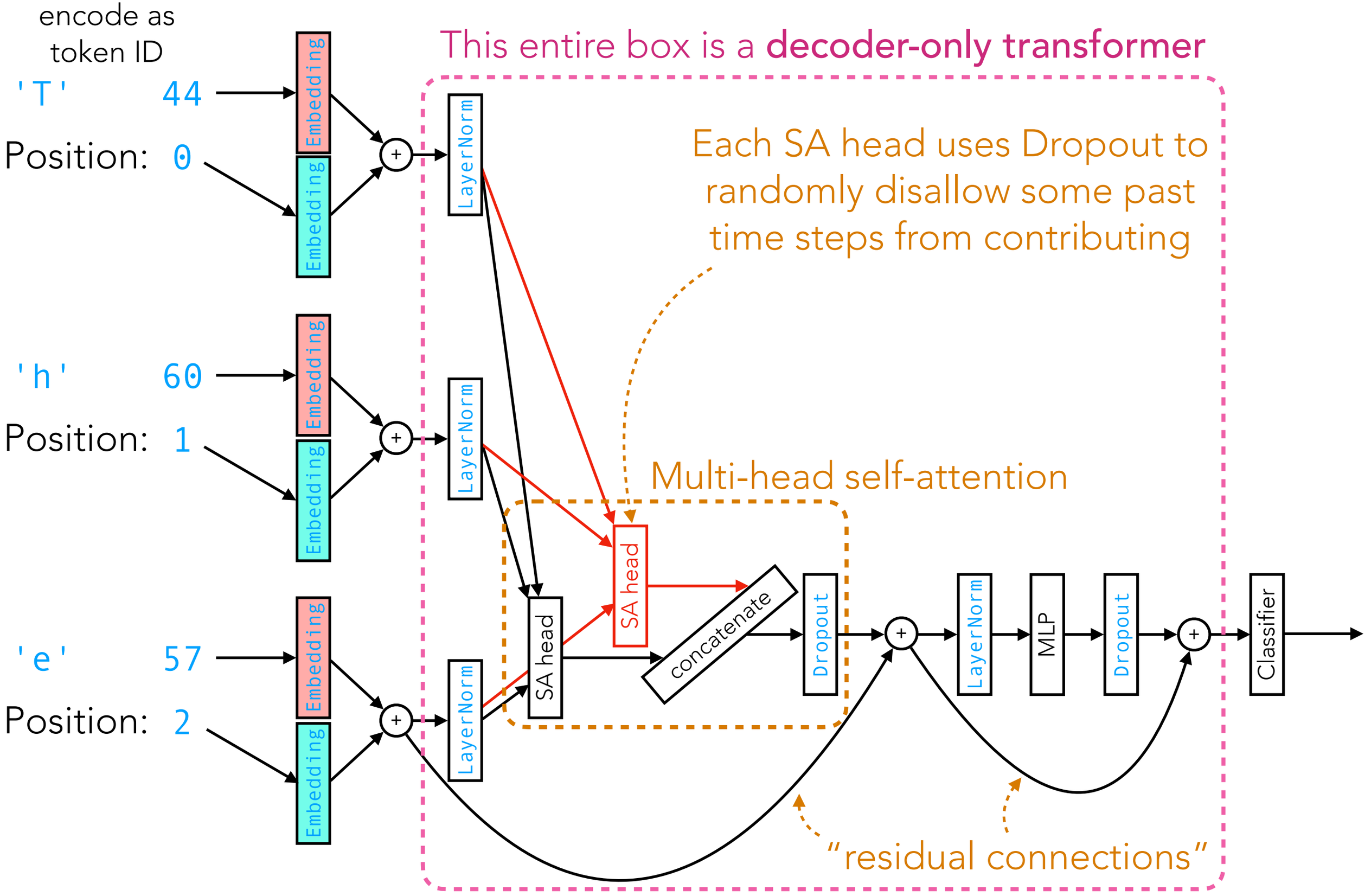


There are a few implementation details

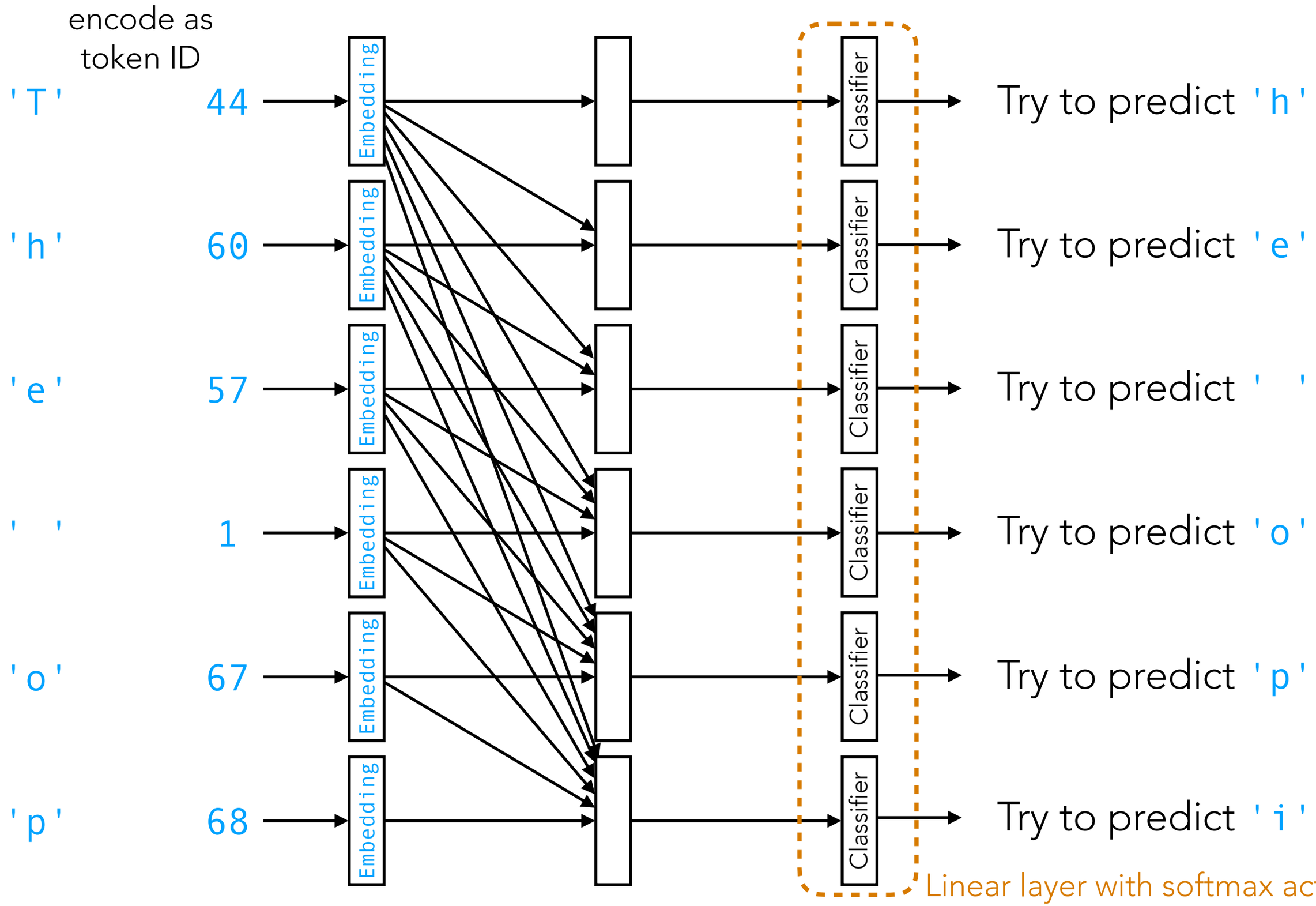
Basically, it turns out that when neural nets get very deep,
training can be more difficult without some now-standard tricks
(these tricks work with *many* neural net architectures, not just GPTs)

- LayerNorm
- Residual connections
- Dropout

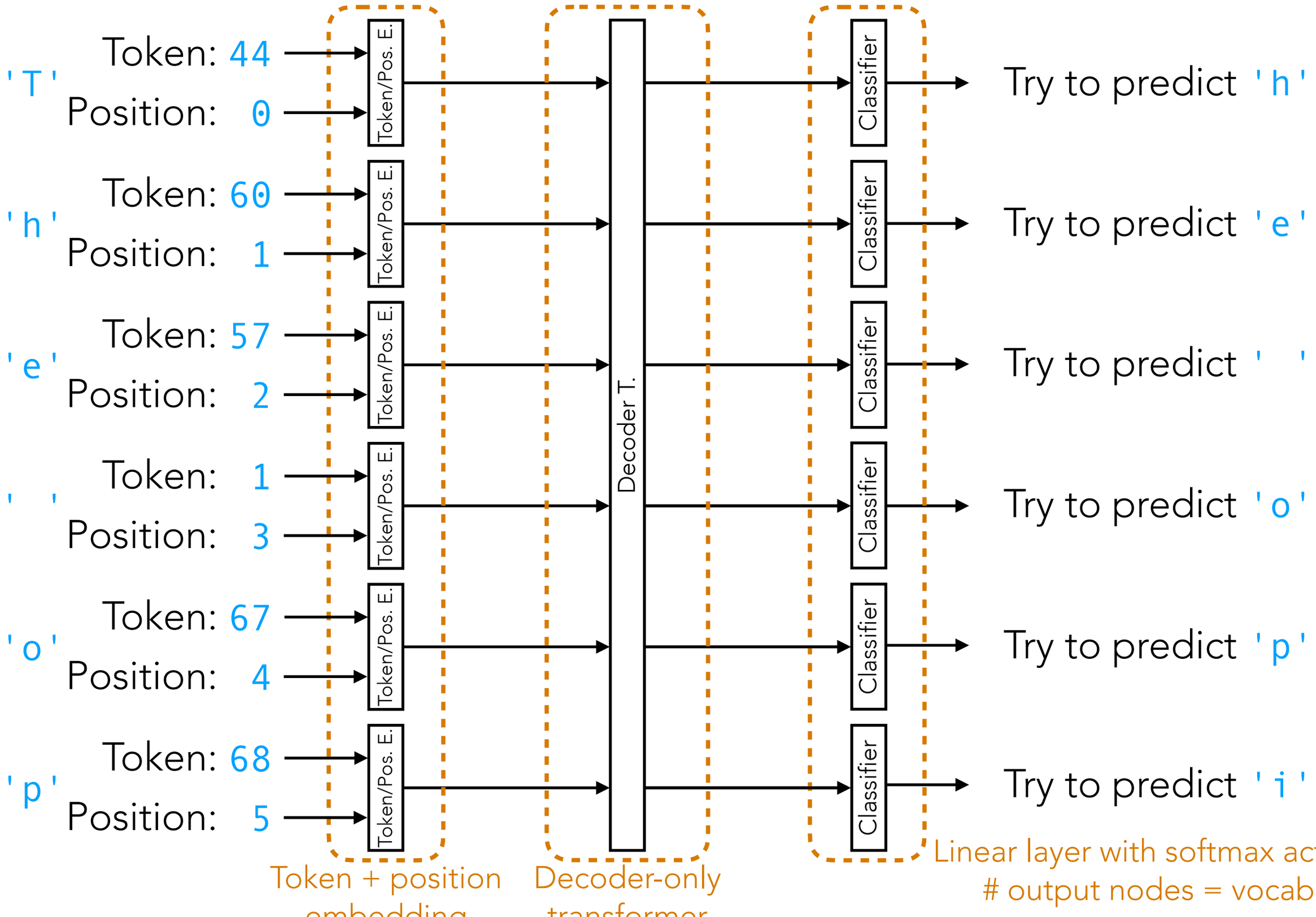
Also, there are some standard strategies for initializing GPT training


encode as token ID
'T' 44
Position: 0
'h' 60
Position: 1
'e' 57
Position: 2
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
This entire box is a decoder-only transformer
Each SA head uses Dropout to randomly disallow some past time steps from contributing
LayerNorm
LayerNorm
LayerNorm
Multi-head self-attention
SA head
SA head
concatenate
Dropout
LayerNorm
MLP
Dropout
Classifier
"residual connections"

This sort of dependence is "causal": any time step can only depend on its current input and all past inputs

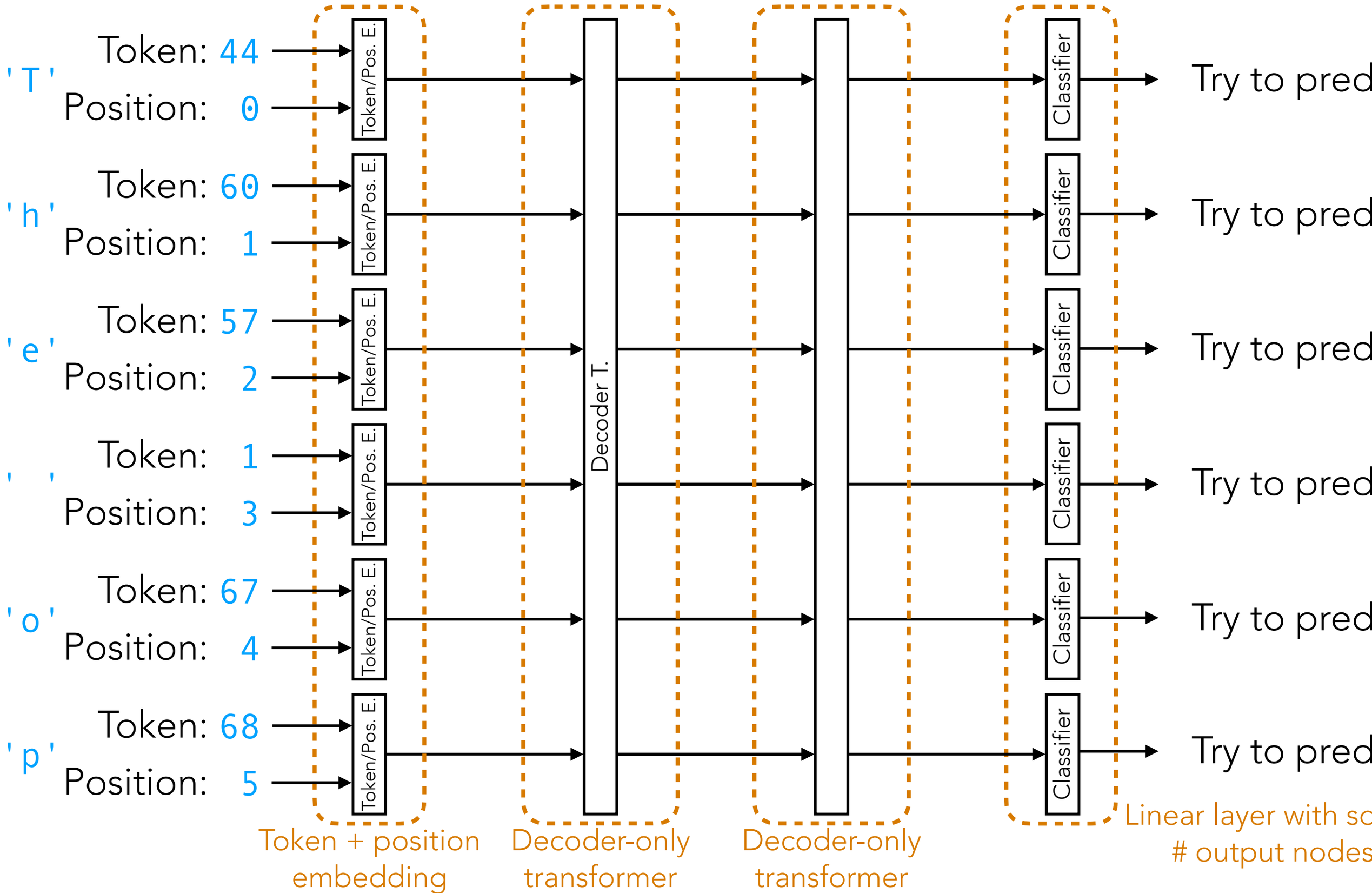


Generative Pre-trained Transformer (GPT)

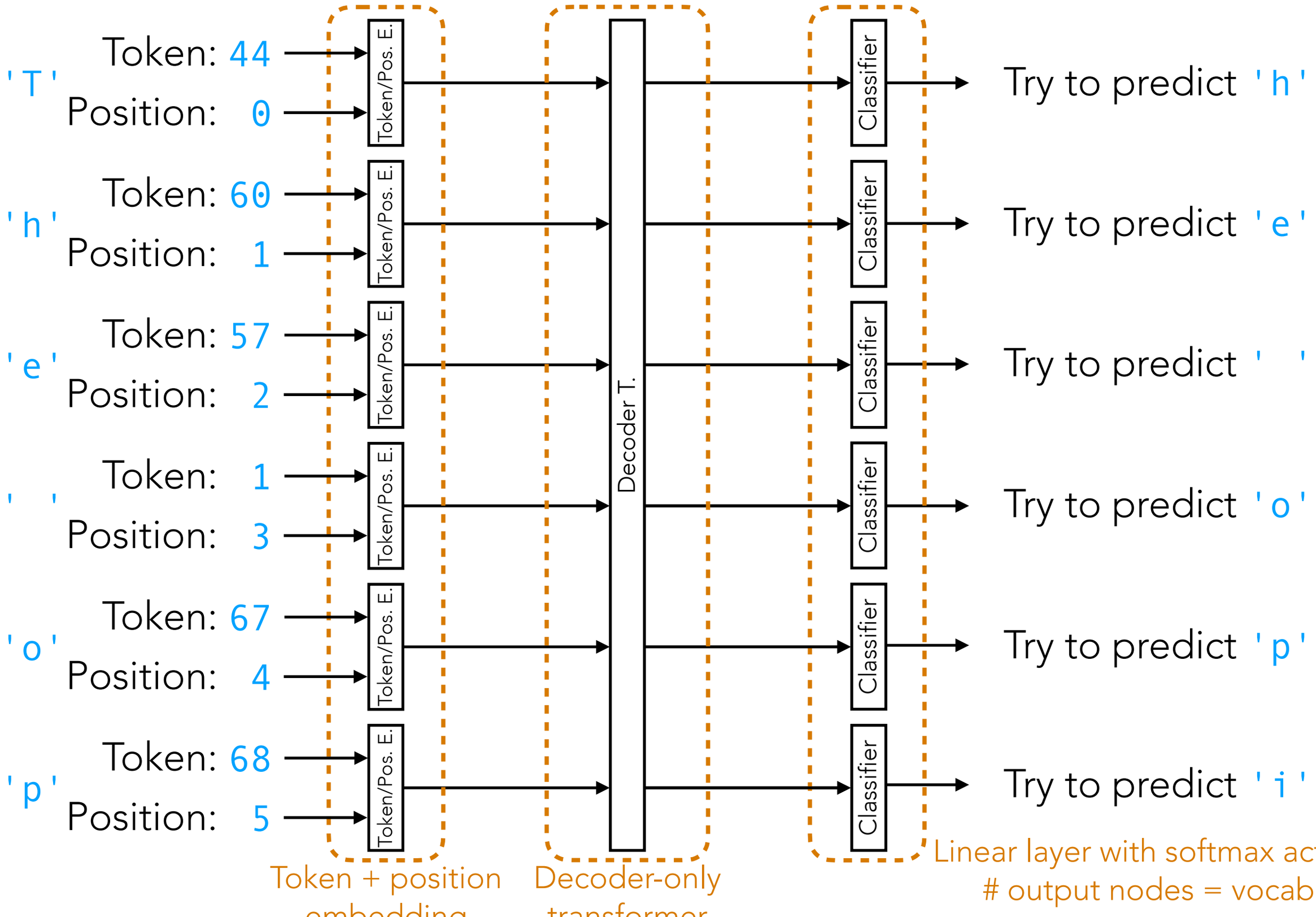


It is possible to stack transformer layers!

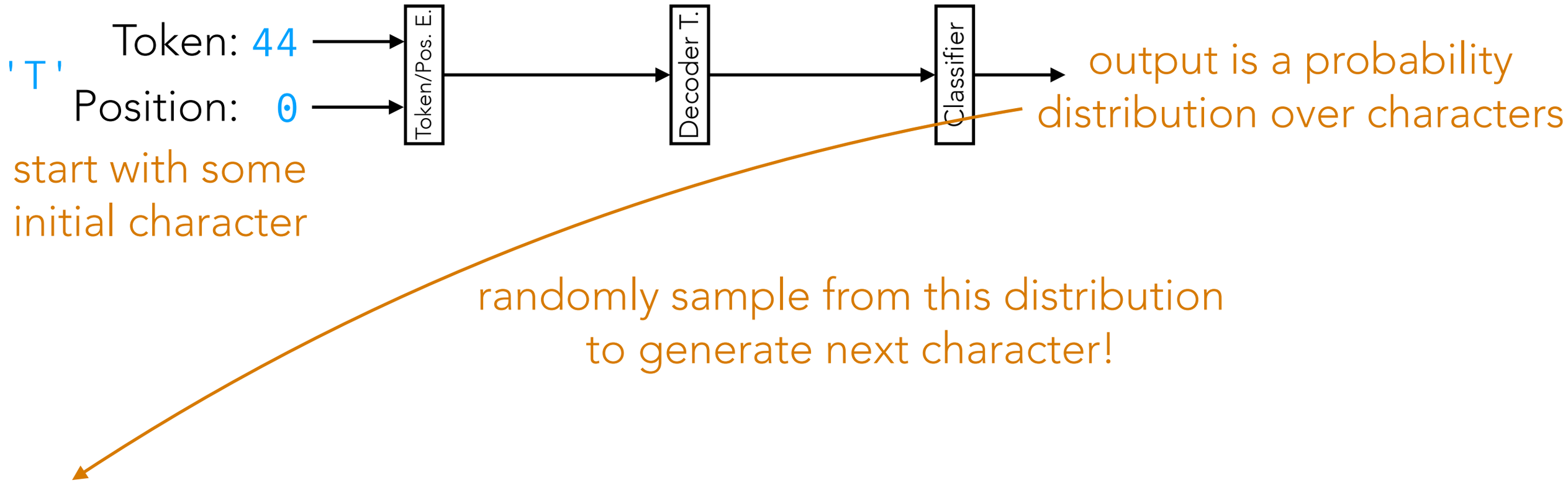
Generative Pre-trained Transformer (GPT)



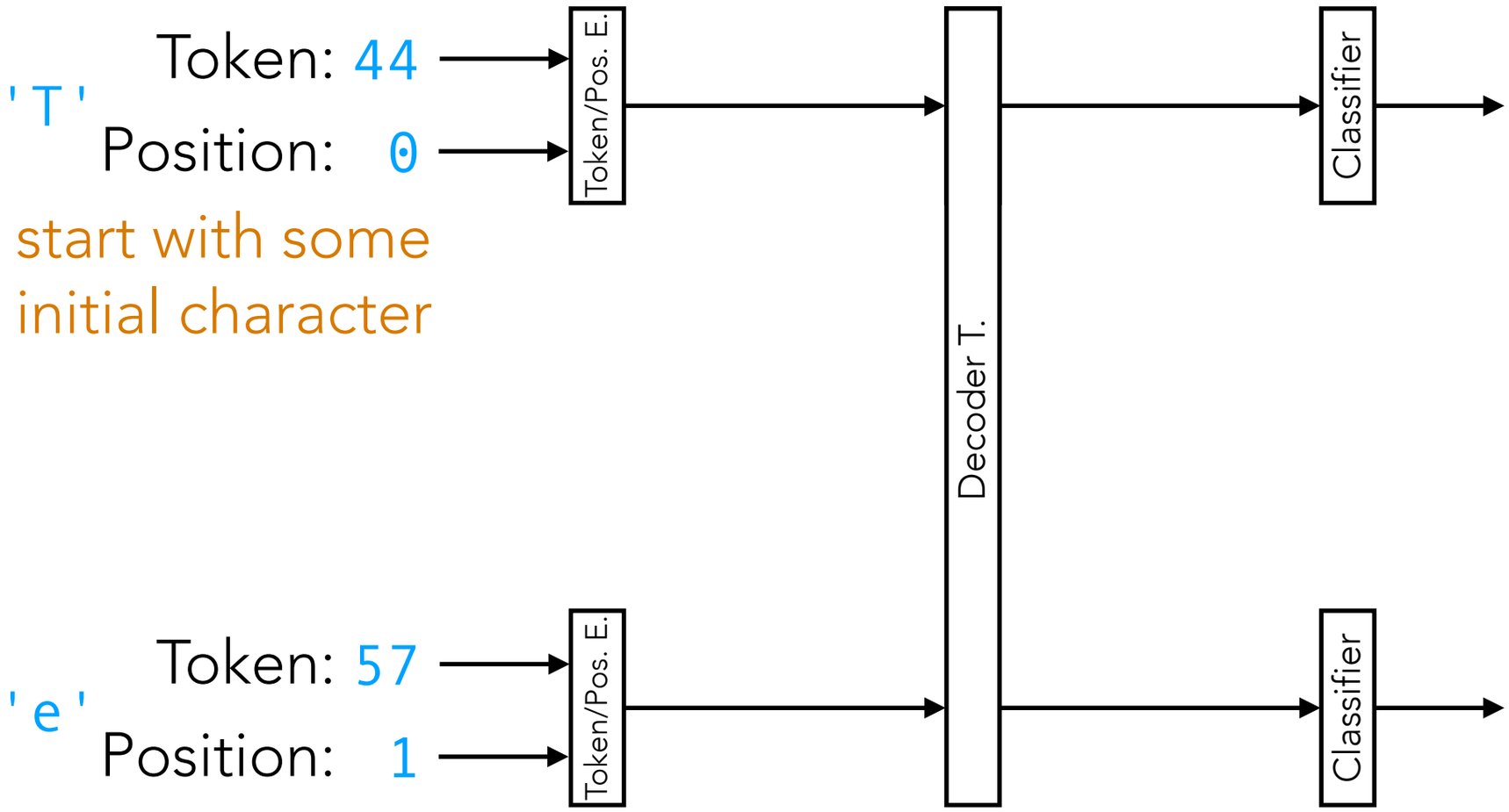
Generative Pre-trained Transformer (GPT)



How to Generate Text After Model Training



How to Generate Text After Model Training



start with some initial character

output is a probability distribution over characters

randomly sample from this distribution to generate next character!

's'

Keep generating text in this manner!

How to Get GPTs to Answer Prompts

A system like ChatGPT is trained in two phases

- First, it is "pre-trained" on a massive chunk of the internet using the prediction task we described already (this prediction task does *not* require any human annotations)

After this pre-training step, the model can randomly generate text but doesn't know how to answer prompts yet (the model is "unaligned" with human goals at this point)

- Next, we "fine-tune" the model by giving it labeled training data showing questions & answers, and over time, we improve the model by letting humans scoring responses of the model

This step is sometimes referred to as "post-training" and is often achieved via "reinforcement learning with human feedback" (RLHF)

We focused on this first step today

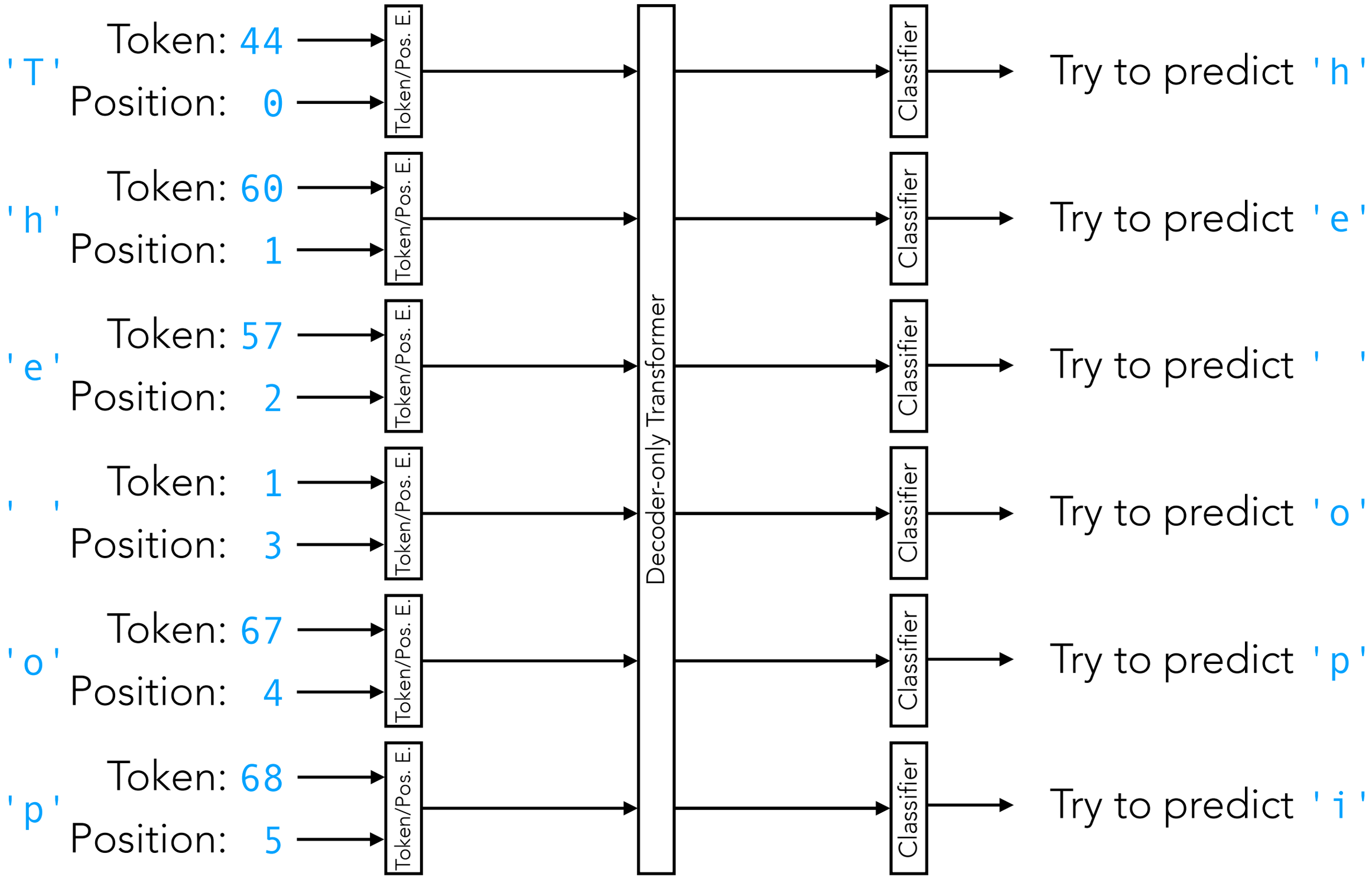
I'll briefly discuss fine-tuning (but not reinforcement learning)

GPT Pre-Training

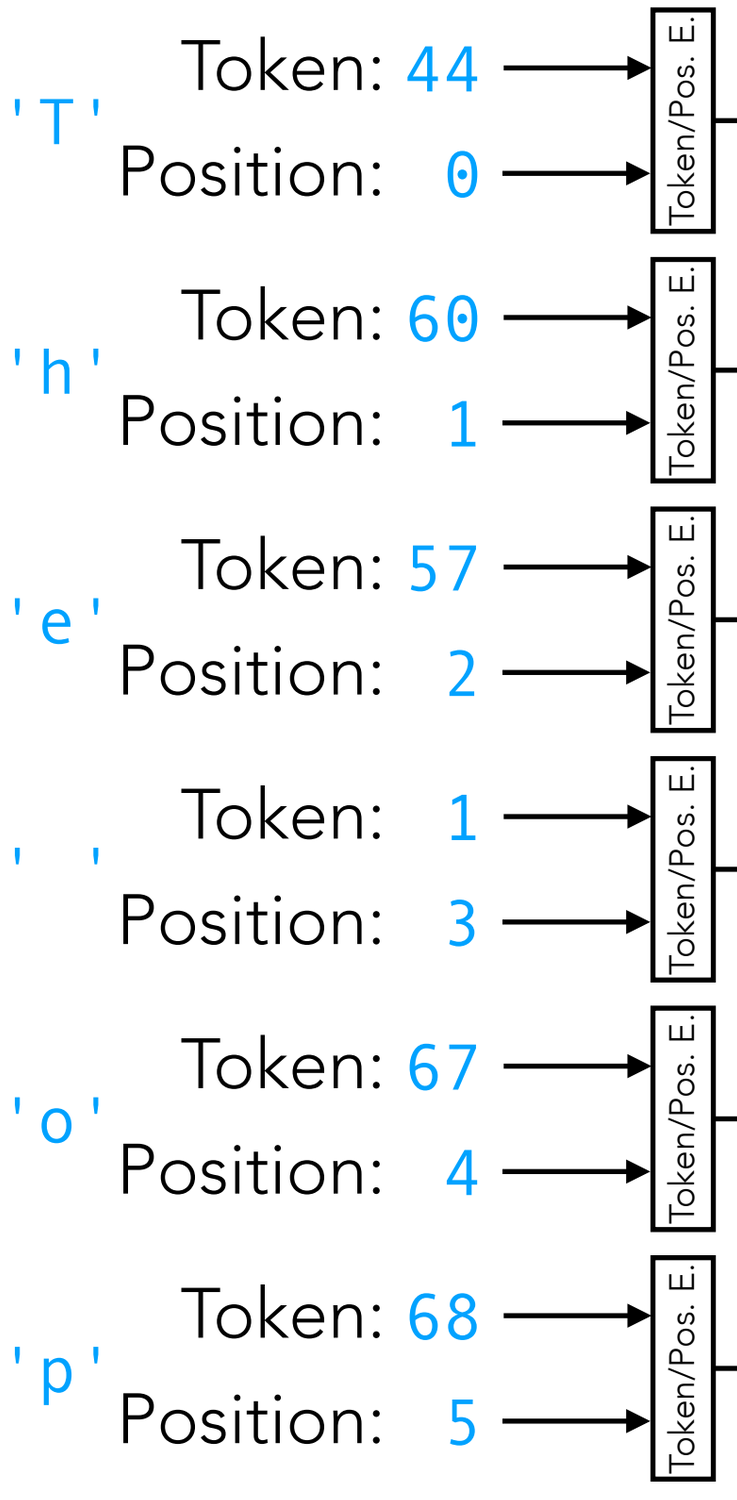
Demo will be in recitation

A bit more on transformers

(Flashback) Generative Pre-trained Transformer (GPT)



(Flashback) Generative Pre-trained Transformer (GPT)

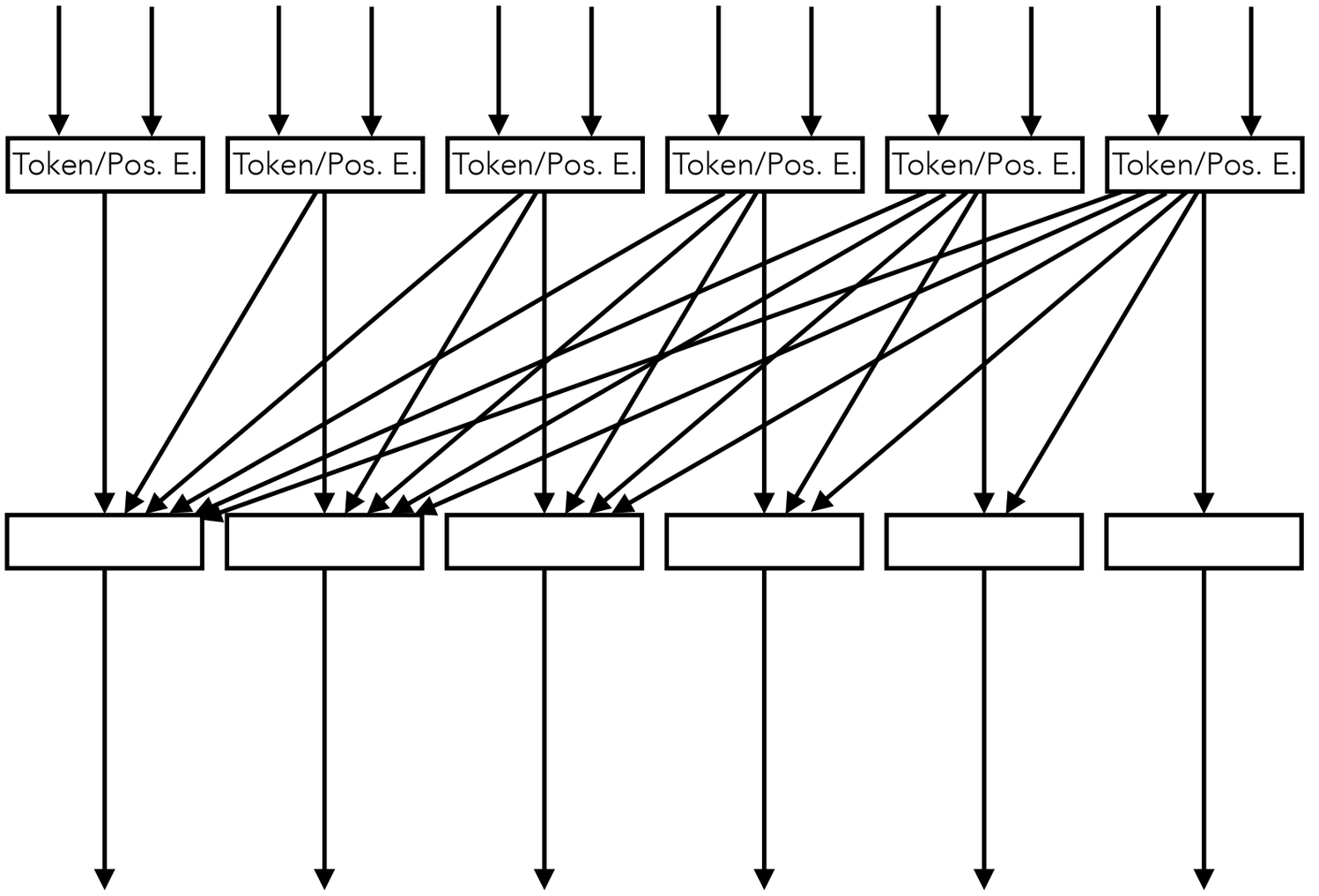


The output before the classifier could be thought of as a token embedding that accounts for context

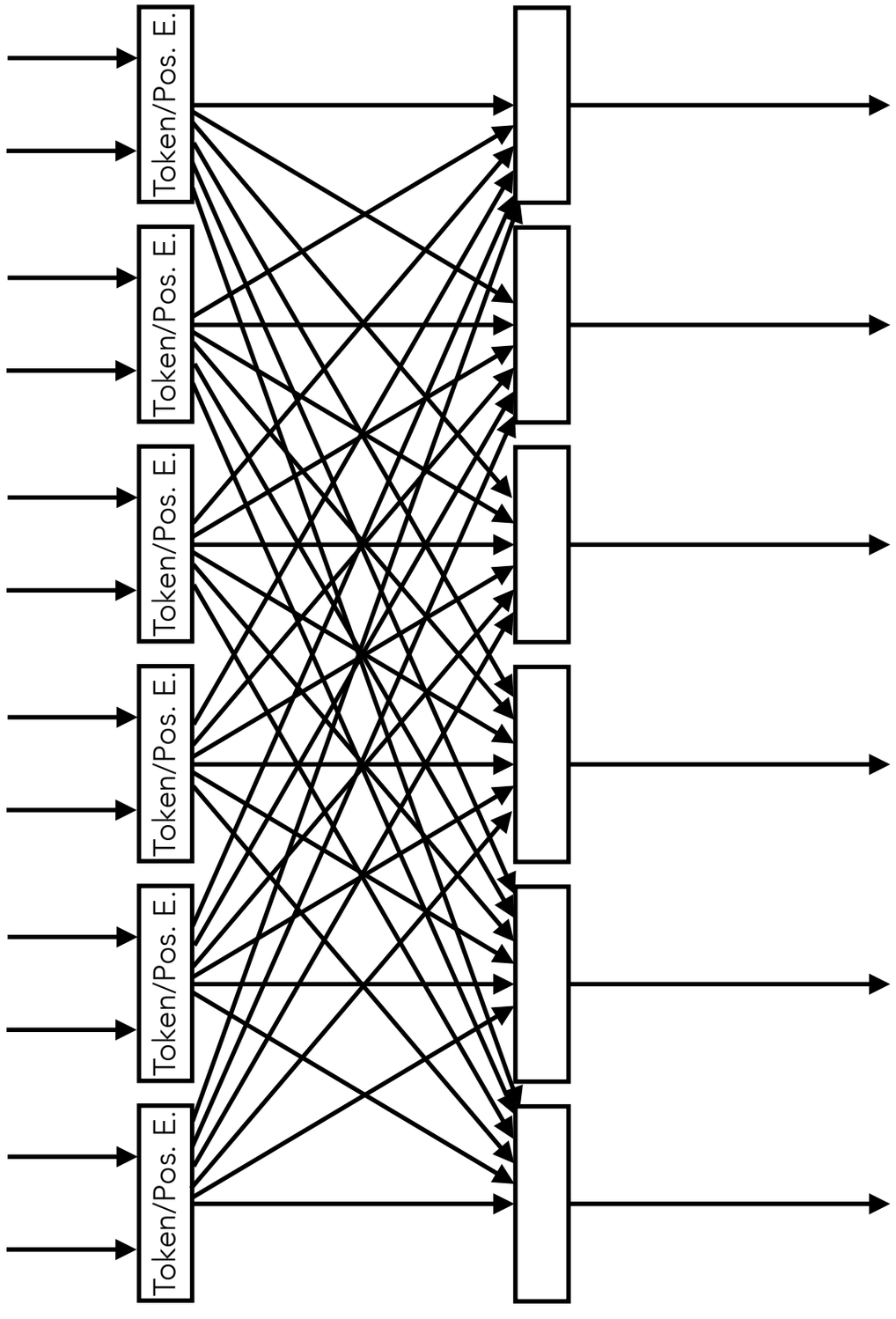
If we tokenized instead using words, then we would have word embeddings

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)


Token/Pos. E.
Token/Pos. E.
Token/Pos. E.
Token/Pos. E.
Token/Pos. E.
Token/Pos. E.
causal dependence

BERT (2018)



no causal dependence

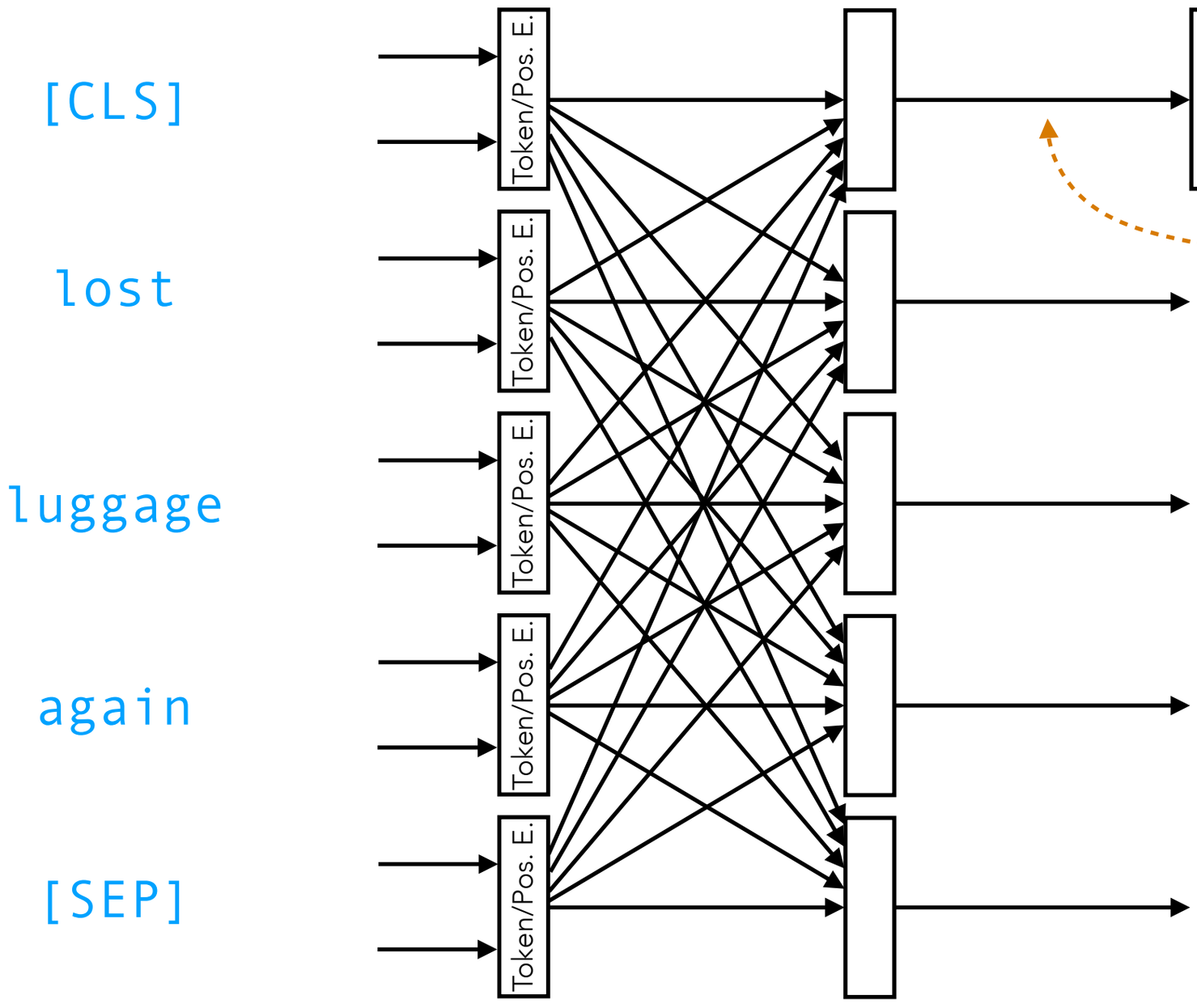
The prediction at any time step depends on the input at all time steps

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)



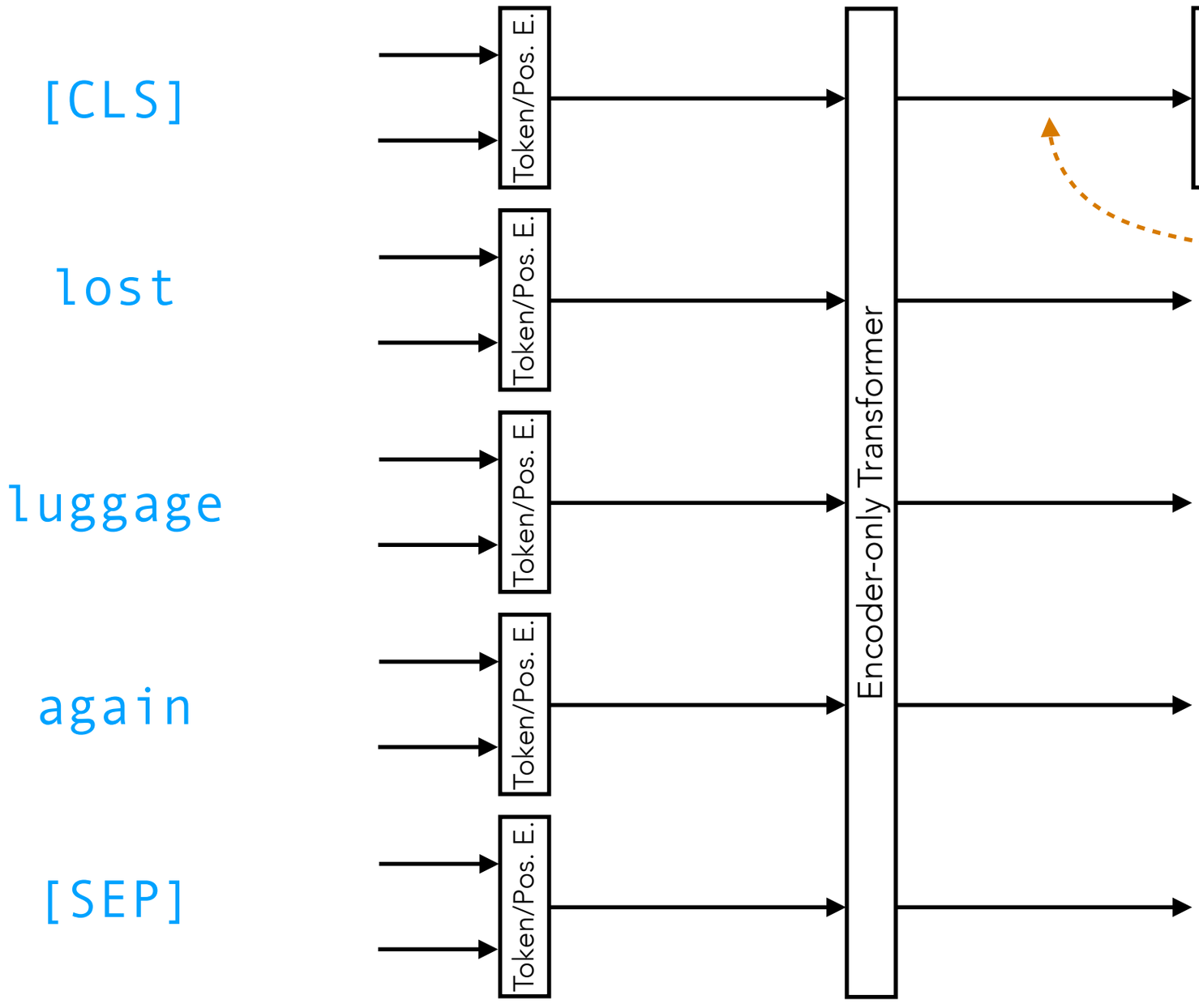
label 0: negative sentiment
label 1: positive sentiment

linear layer (2 output nodes), softmax activation

The word embedding for [CLS] is meant to summarize the doc in a manner useful for classification

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

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 in a manner useful for classification

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

Handling Small Datasets

- Fine-tuning: load in an already trained model, possibly change the last few layers, and continue training
  - Enables using an existing model trained on a *massive* dataset to help us with a new prediction task where we might only have a *small* dataset

ChatGPT:

GPT architecture 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: load in an already trained model, possibly change the last few layers, and continue training
  - Enables using 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!

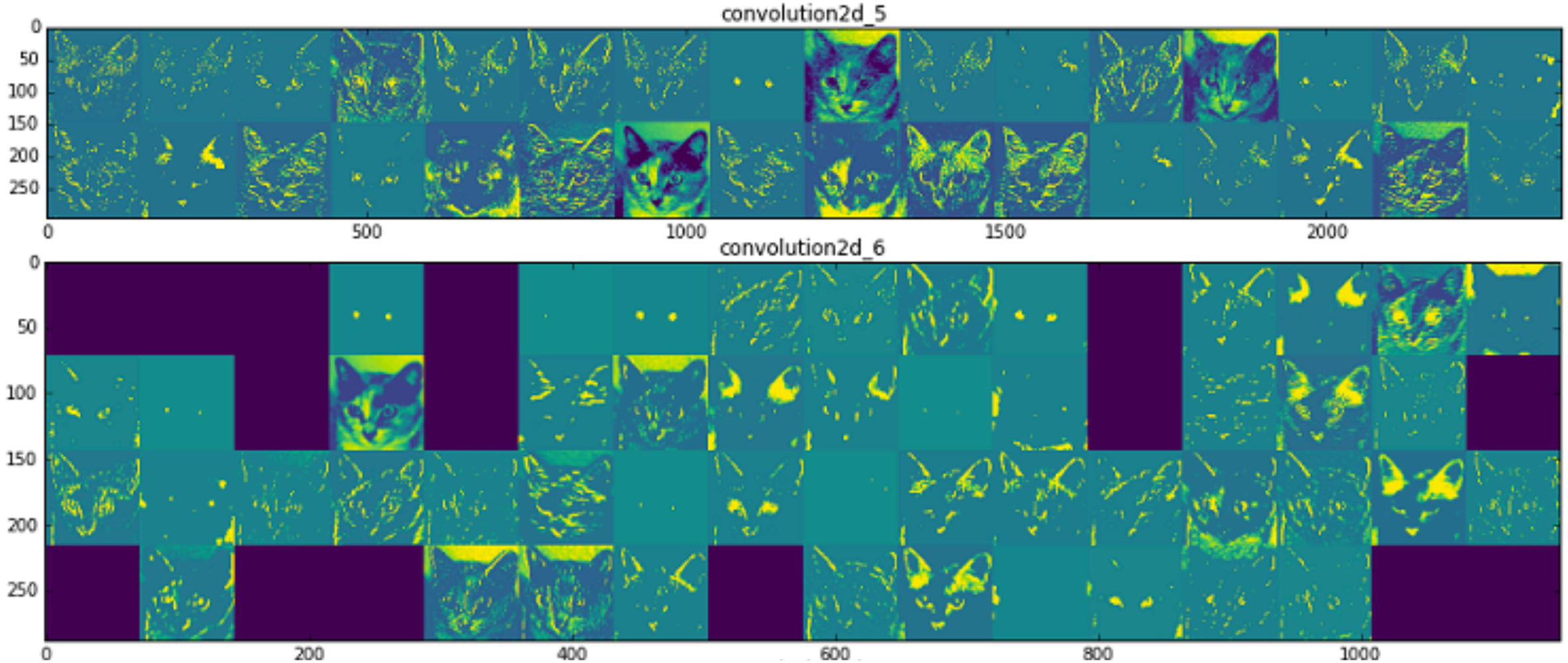
State-of-the-art vision systems are all trained with data augmentation!

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



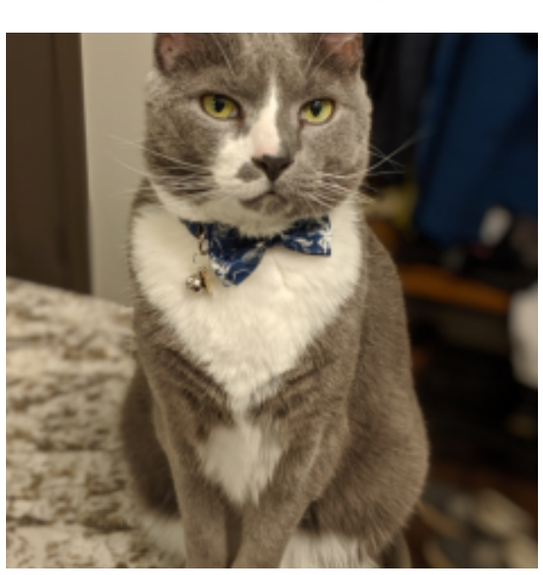
Images: From Francois Chollet's book "Deep Learning with Python"

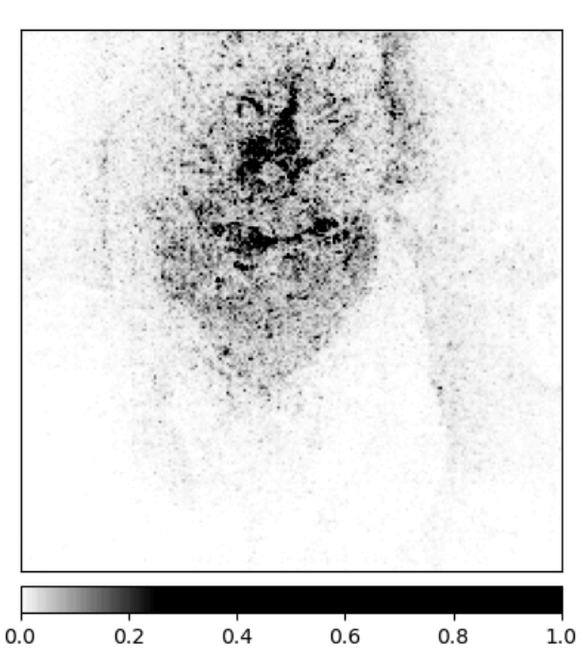
Interpretability/Explainability: Current State of Affairs

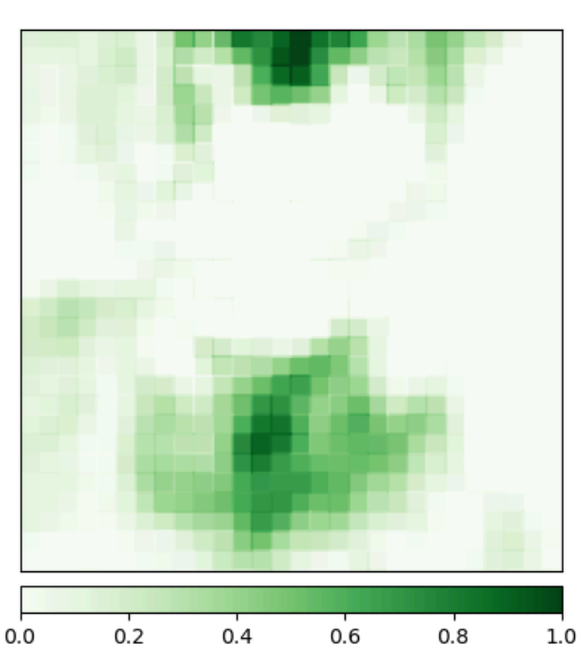
- There are lots of "explanation" approaches that can be used after training a deep net to try to understand what has been learned
  - Many of these are implemented in the Python package Captum developed by Meta/Facebook: https://captum.ai/

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

What pixels are important for prediction?







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

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

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