

94-775 Unstructured Data Analytics

Lecture 13: Text generation with generative
pretrained transformers (GPTs)

Slides by George H. Chen

Administrivia (Various Reminders)

- Final project presentations are this Friday (recitation slot)
- Final project reports are due **Monday April 28, 11:59pm** & consist of:
 - Jupyter notebook (edited down to be clean, concise)
 - Slide deck for your final project presentation
- The final slide deck could be modified compared to what is presented the Friday beforehand (e.g., to incorporate feedback)
- I'm setting a hard limit of 12 minutes for your final project presentation
 - If you go over 12 minutes, you will not receive full credit for the "presentation" aspect of your final project score
- I have office hours right after class today that goes until 7:30pm

(Flashback) Final Project Rubric

- **Policy question (15%):** what public policy question are you addressing? Please be clear and concise.
- **Data analysis (30%):** clearly state what part of your data are unstructured (some but not all of the data you are analyzing must be unstructured), and carefully justify every step of your analysis with supporting visualizations/intermediate outputs as needed
- **Code (30%):** your code should actually run!
- **Conclusions (15%):** come up with insights that are based on your quantitative data analysis and that address your original policy question
- **Presentation (10%):** how polished is your final project presentation? — this is based on the live presentation your group makes (changes made to the slides after the presentation don't affect this score)

(Flashback) Final Project Rubric

- **Code (30%):** your code should actually run!

We recommend that you send us 2 notebooks, 1 with preprocessing (that saves a small, preprocessed version of the dataset) and another notebook that runs your analyses (using the preprocessed version of the data)

We will only run the notebook that does the analysis after loading in the preprocessed data (i.e., we won't recompute the preprocessed data)

If your dataset is overall small to begin with and you'd prefer just sending in a single notebook with the dataset, that's fine too

We don't have a strict definition of how small your preprocessed dataset is

Your code should ideally take us trivially less than 1 hour to run

Please try to make your notebooks clear & concise
(in the real world, when you ask people to look at your code notebook, you should've cleaned it up so that it doesn't show exhaustively everything that you tried... and it should also be well-documented)


(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!

(Flashback) Word Embeddings: word2vec (2013)

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: opioid

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

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



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

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)

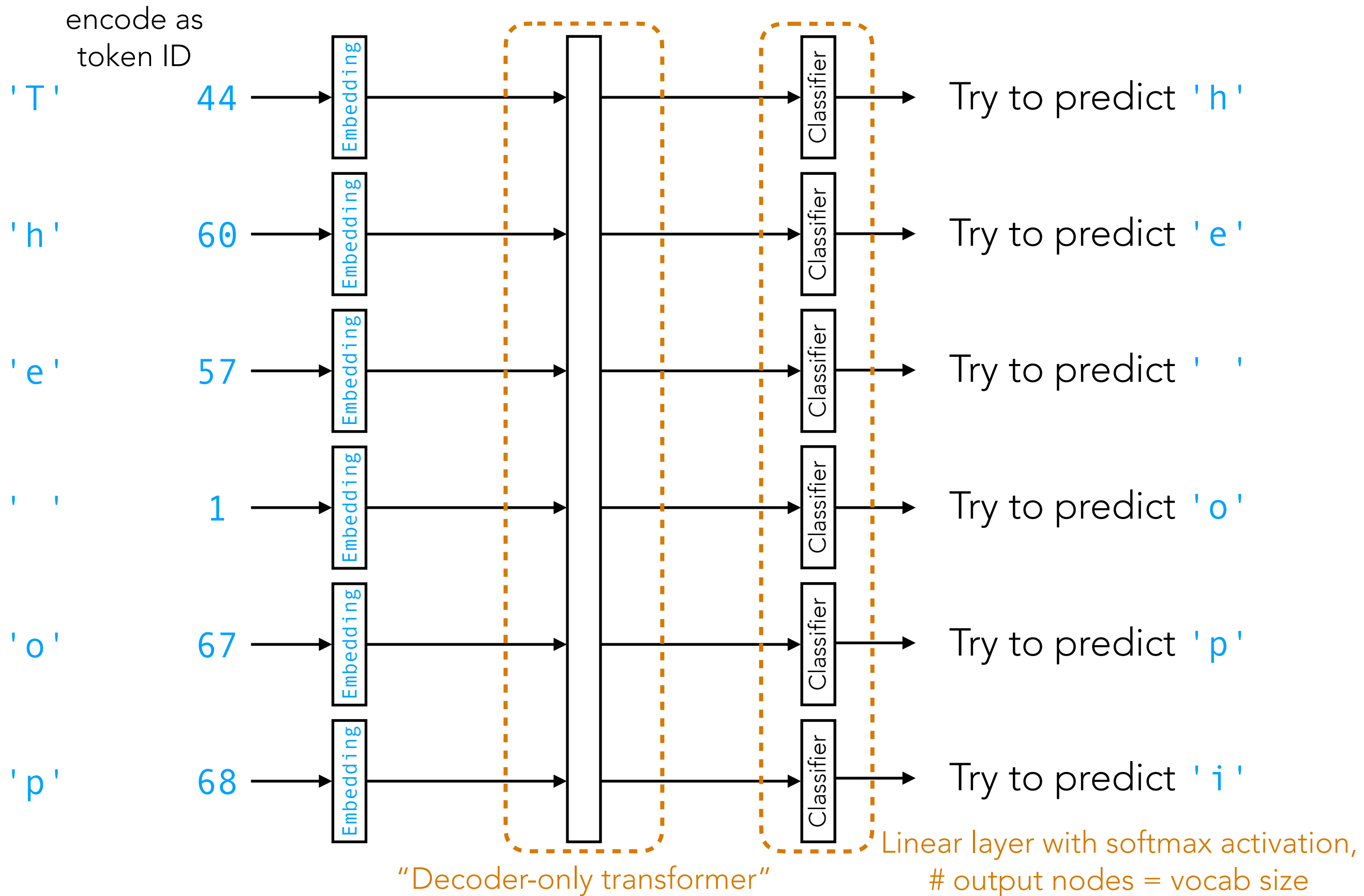
Token ID	Token
0	"\n"
1	" "
2	"\""
3	"\$"
⋮	⋮

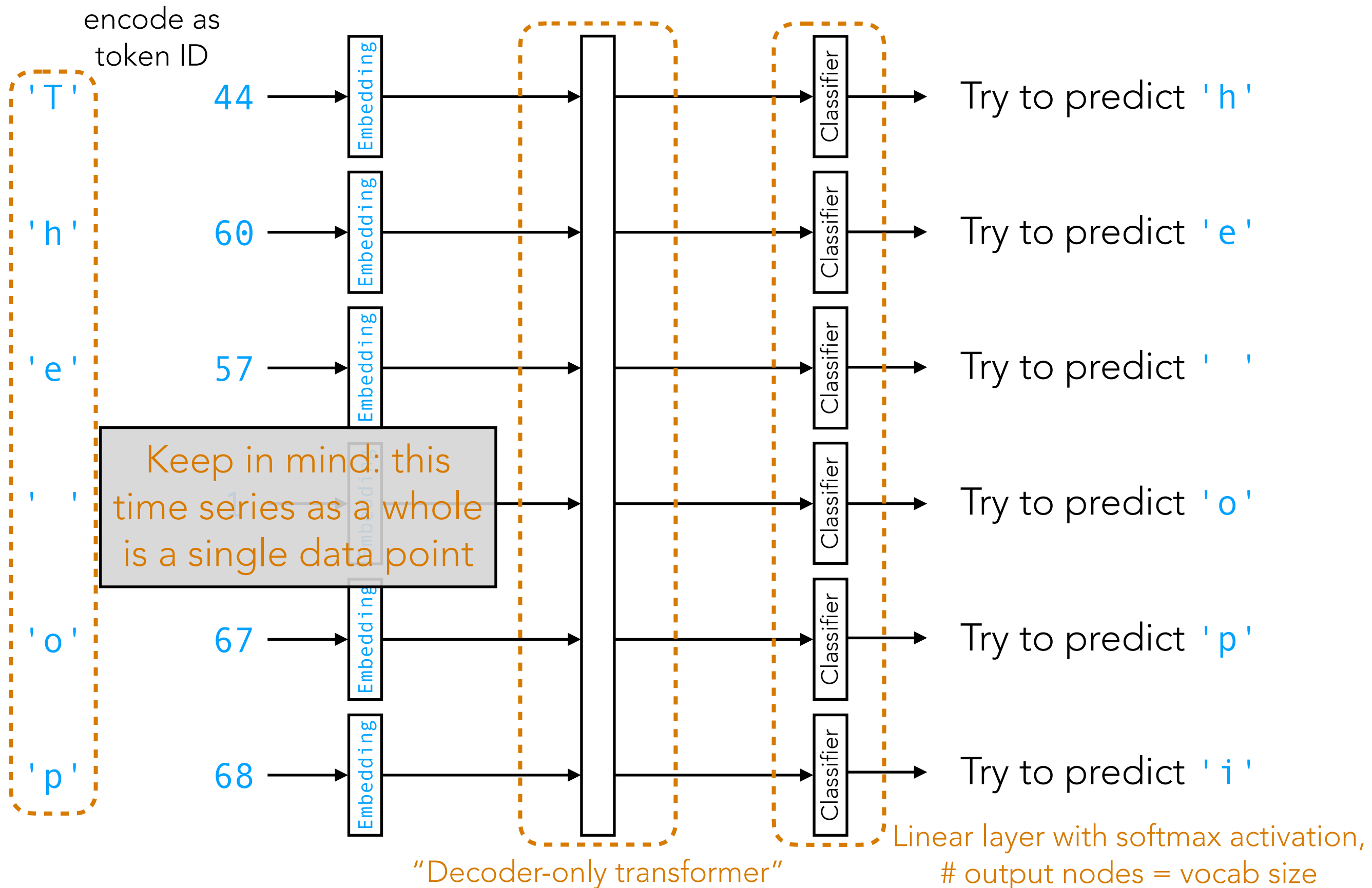
The demo has a vocabulary size of 86

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

length = $L + 1$

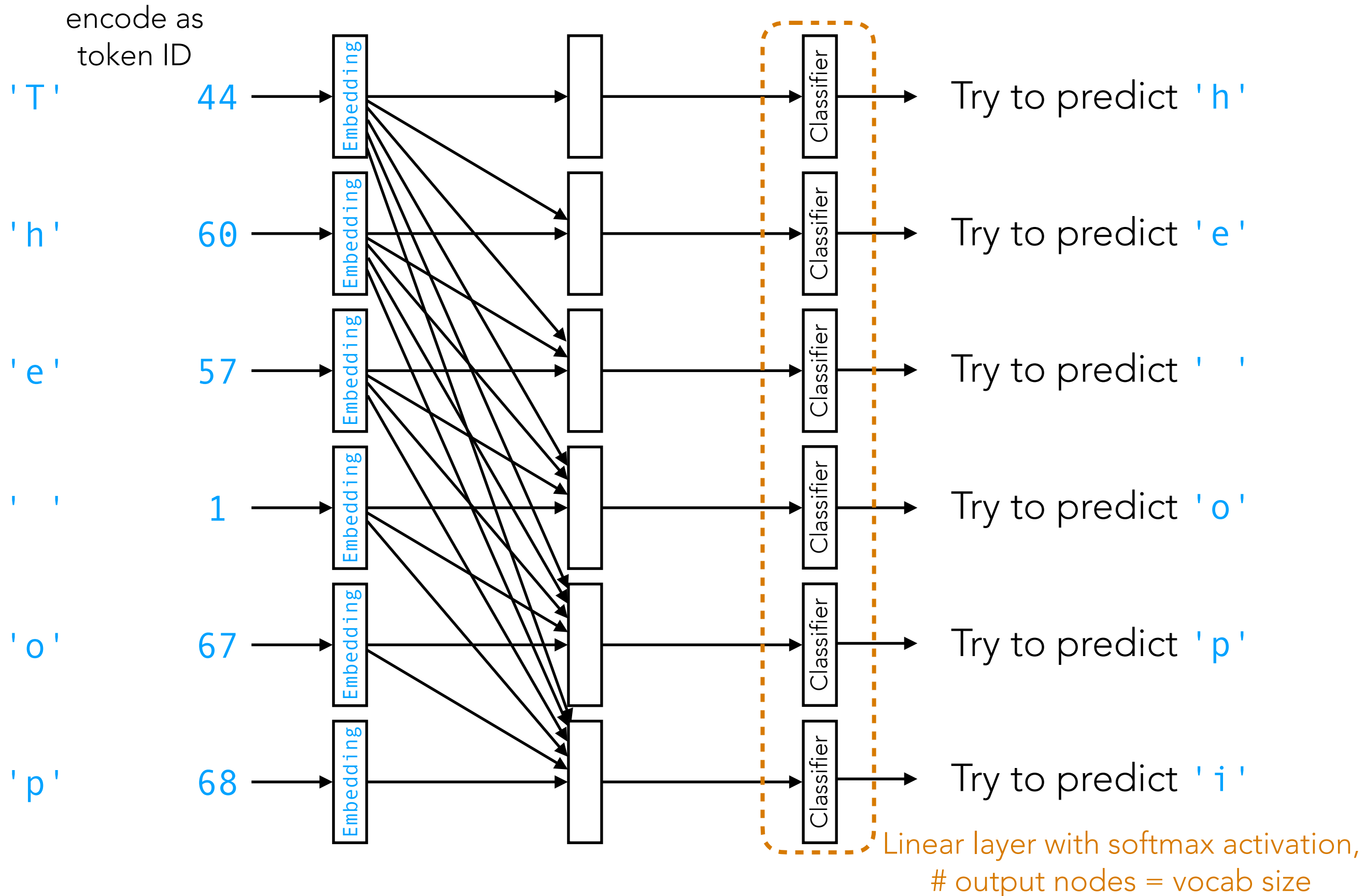
$L = 6$ in this example



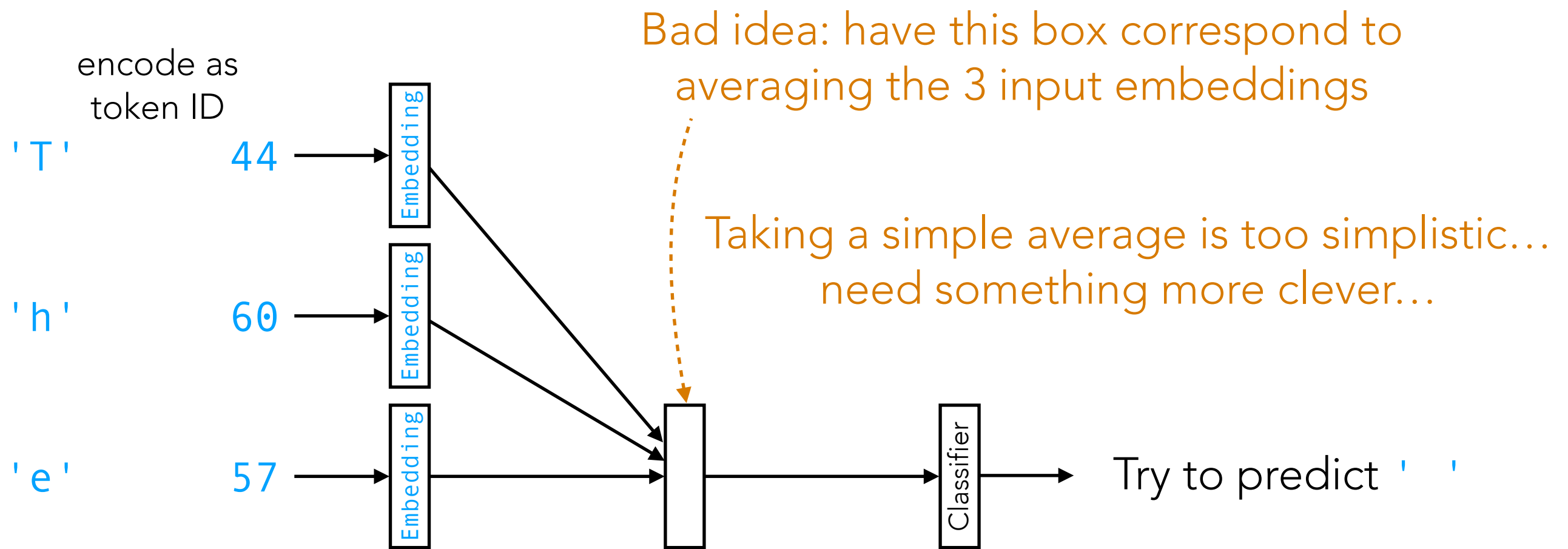


What is a decoder-only transformer?

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)



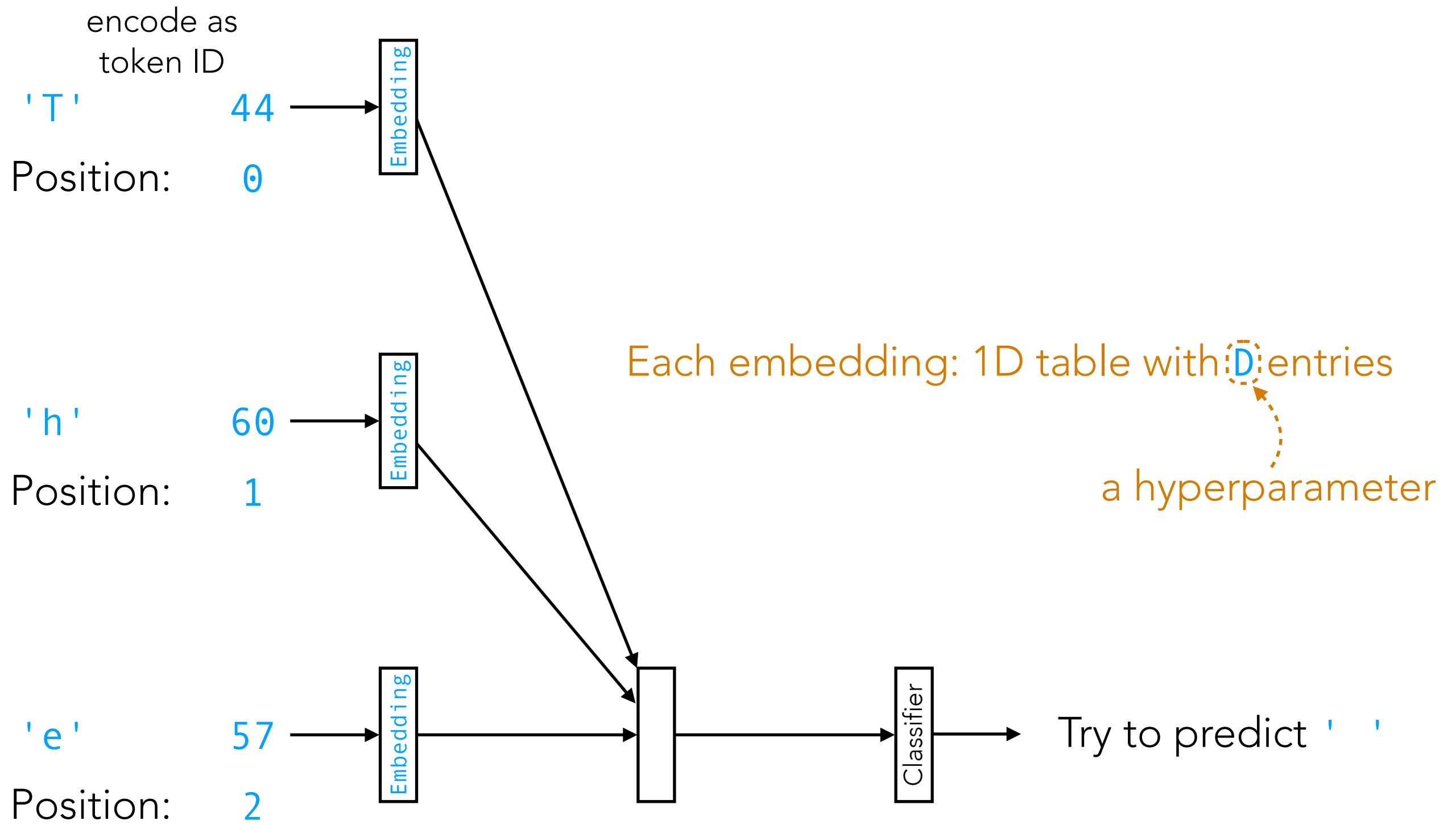
Let's focus on time step 2's prediction task for the moment...

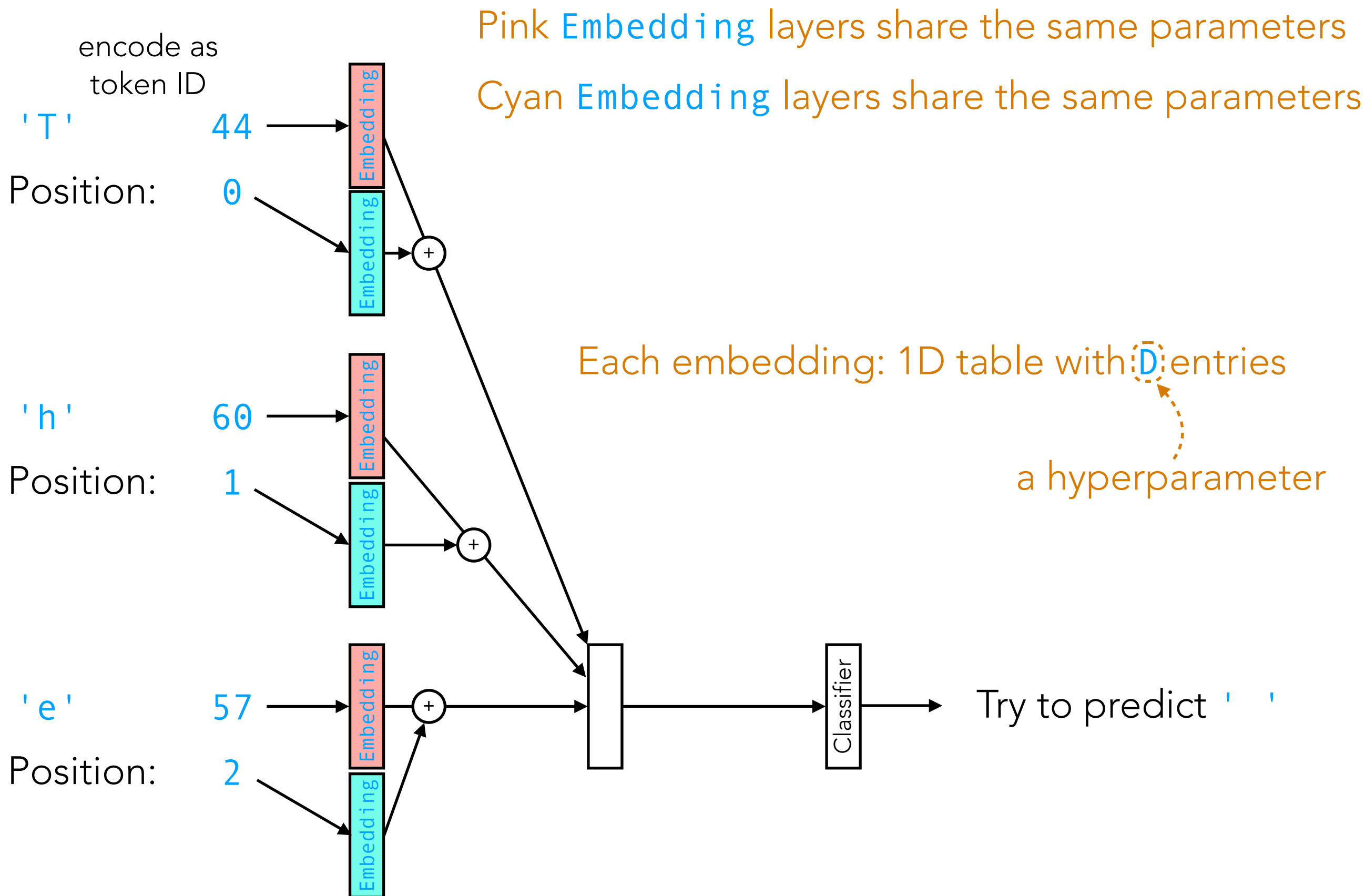


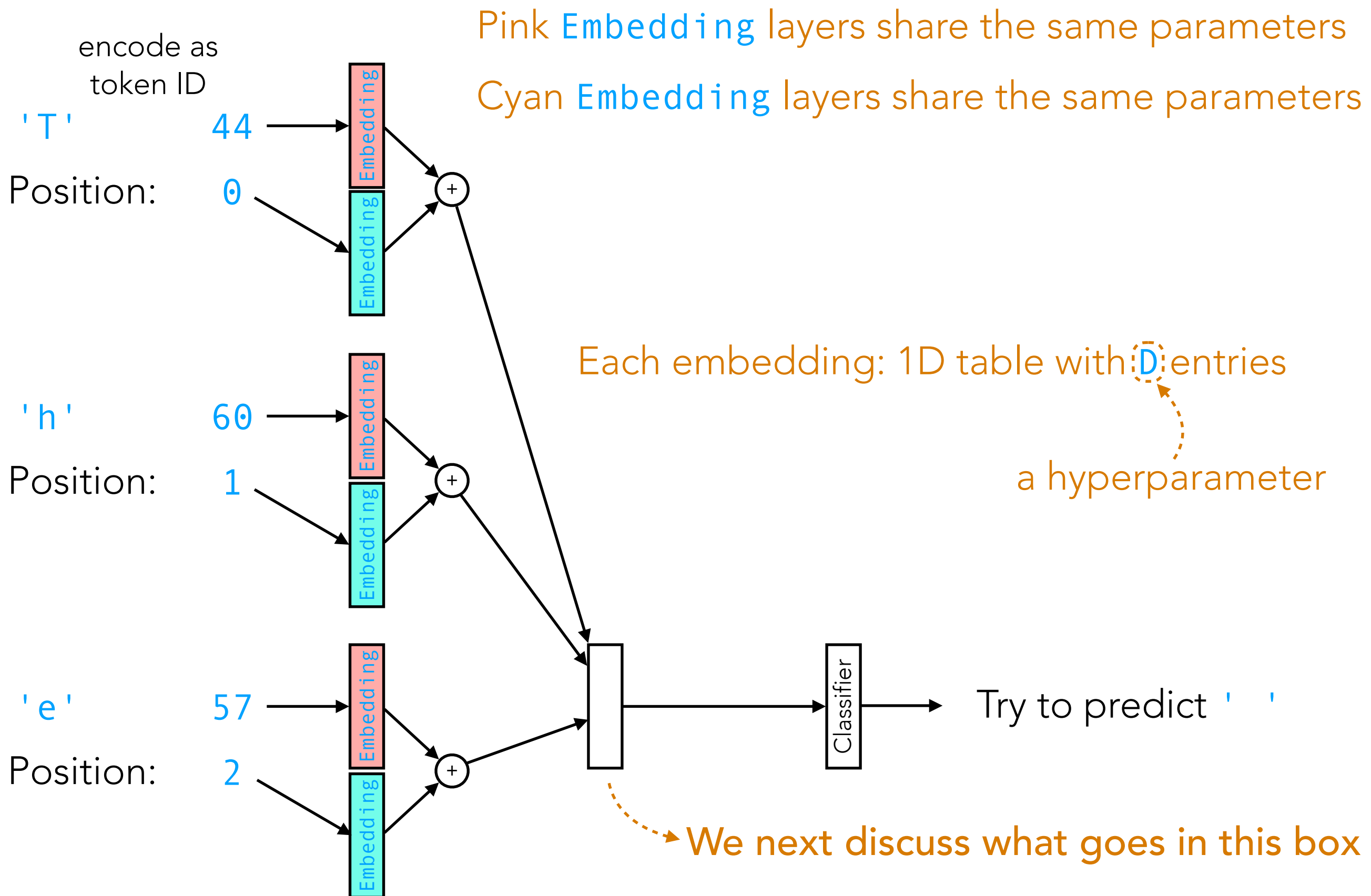
How should we combine information from the input embeddings?

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

Let's address this issue first







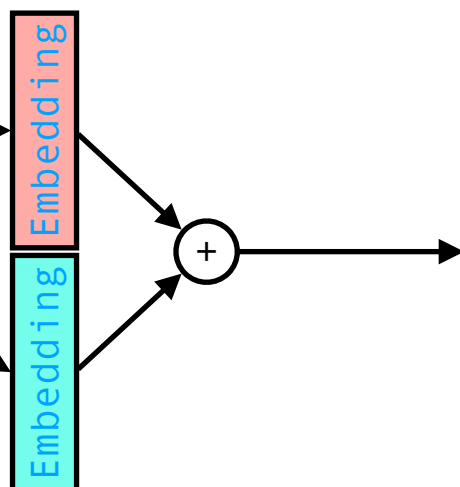
encode as
token ID

'T'

Position:

44

0

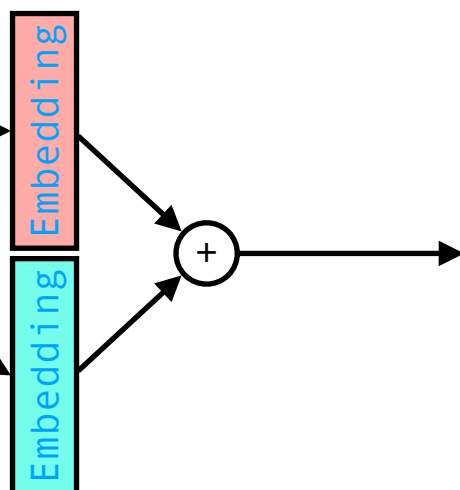


'h'

Position:

60

1

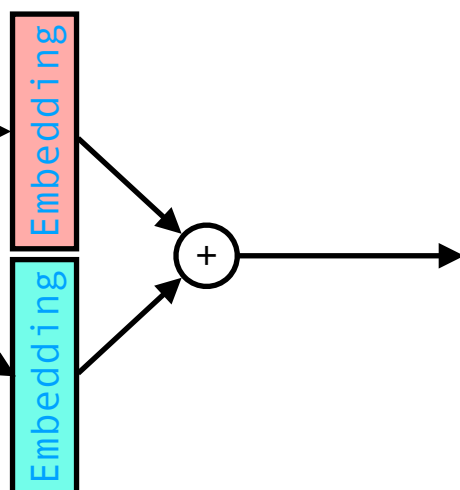


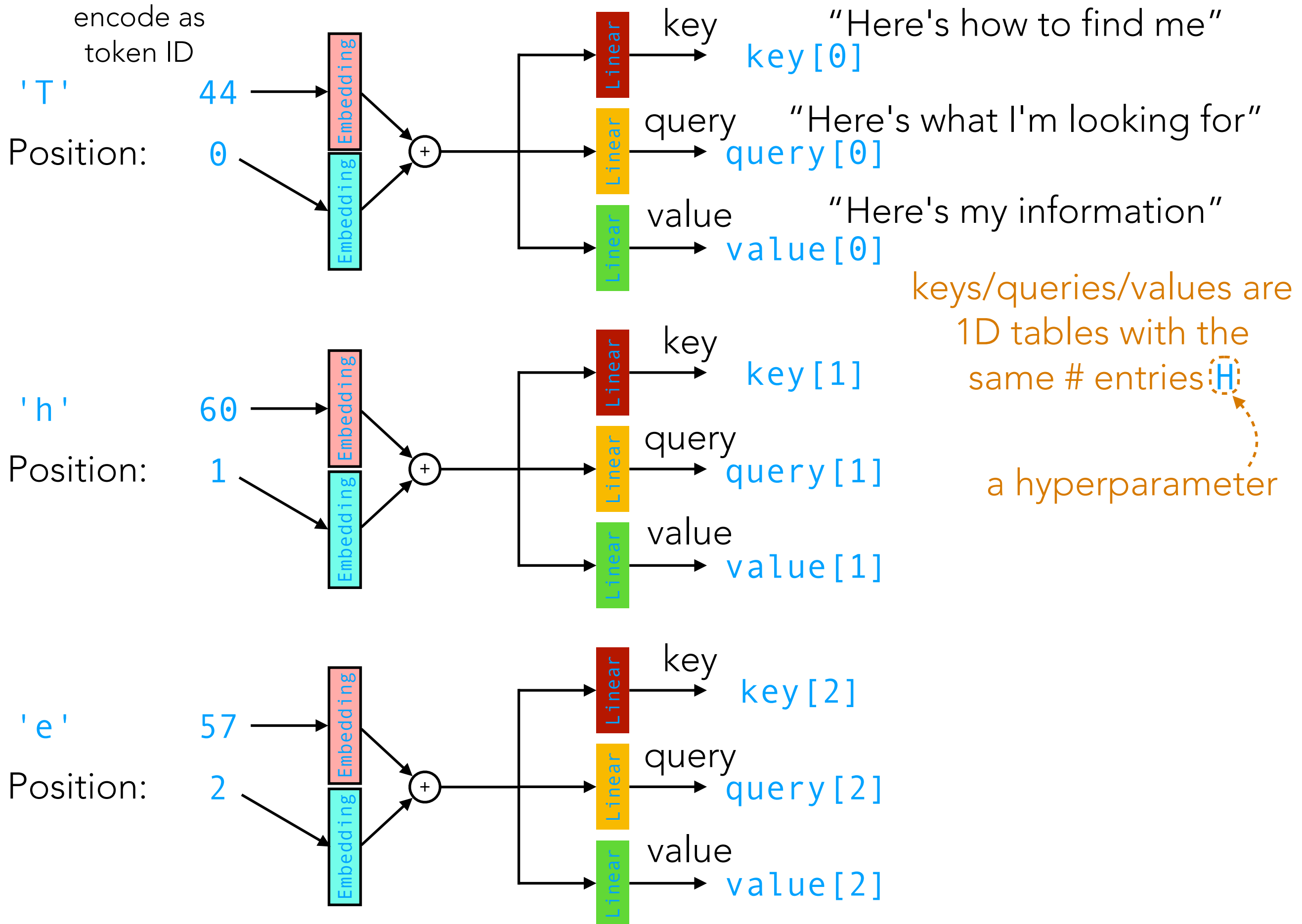
'e'

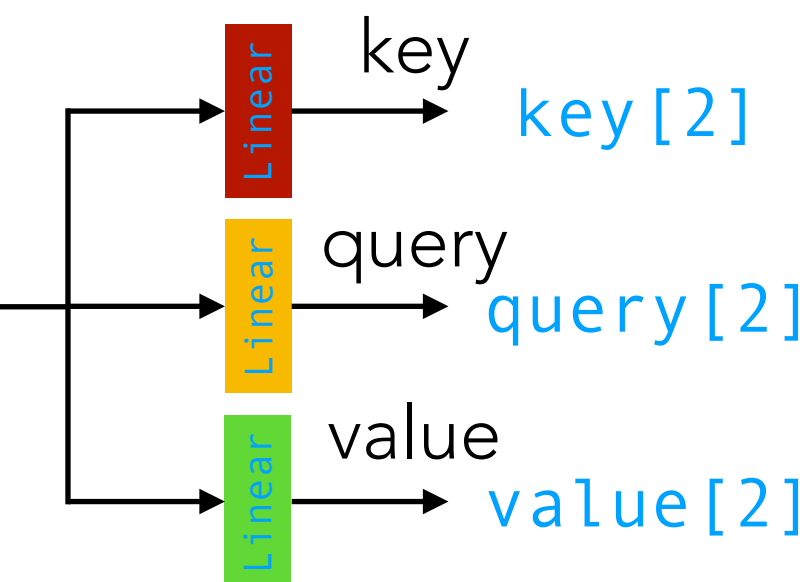
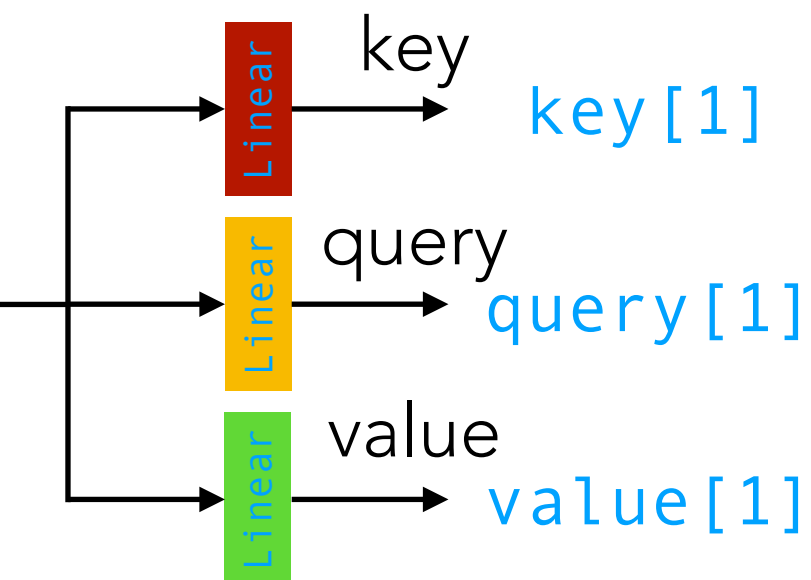
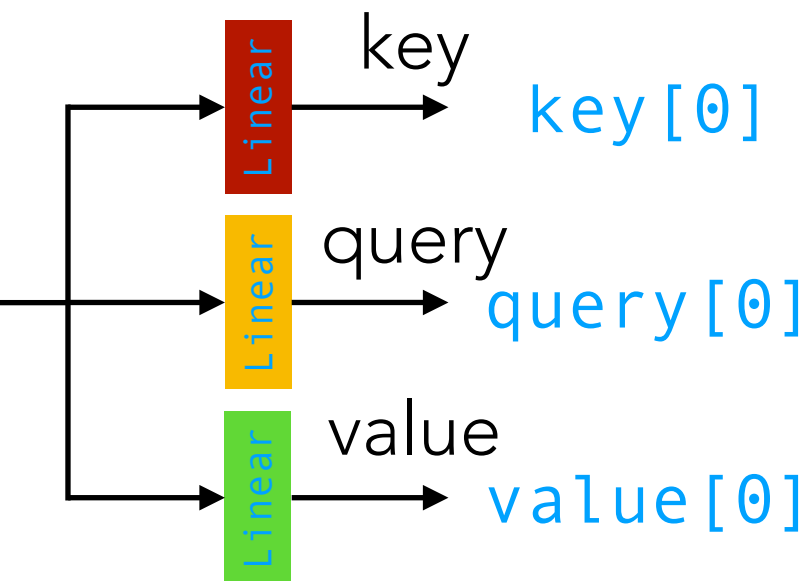
Position:

57

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:

$$w[0] = \text{np.dot}(\text{query}[2], \text{key}[0])$$

How much should time step 1's information contribute?

$$w[1] = \text{np.dot}(\text{query}[2], \text{key}[1])$$

How much should time step 2's information contribute?

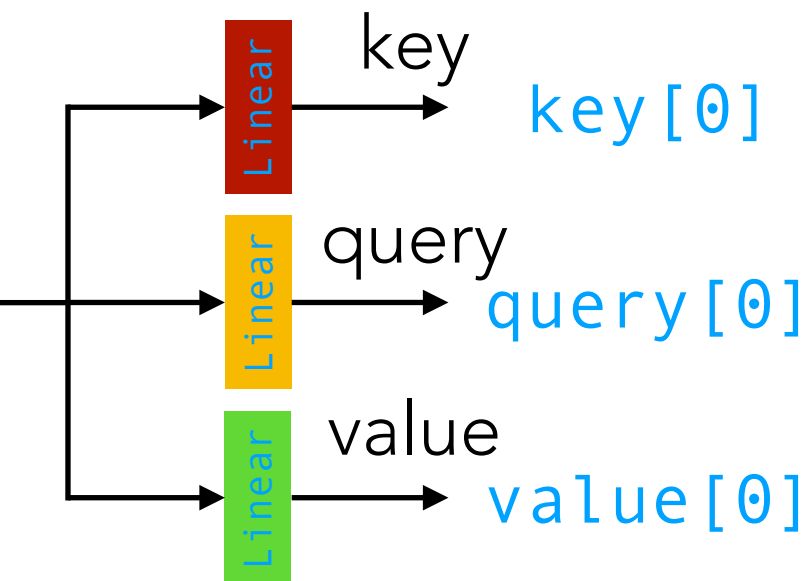
$$w[2] = \text{np.dot}(\text{query}[2], \text{key}[2])$$

Let's normalize the weights so they are probabilities:

$$w_norm = \text{softmax}(w)$$

Output at time step 2:

$$w_norm[0] * \text{value}[0] + w_norm[1] * \text{value}[1] + w_norm[2] * \text{value}[2]$$

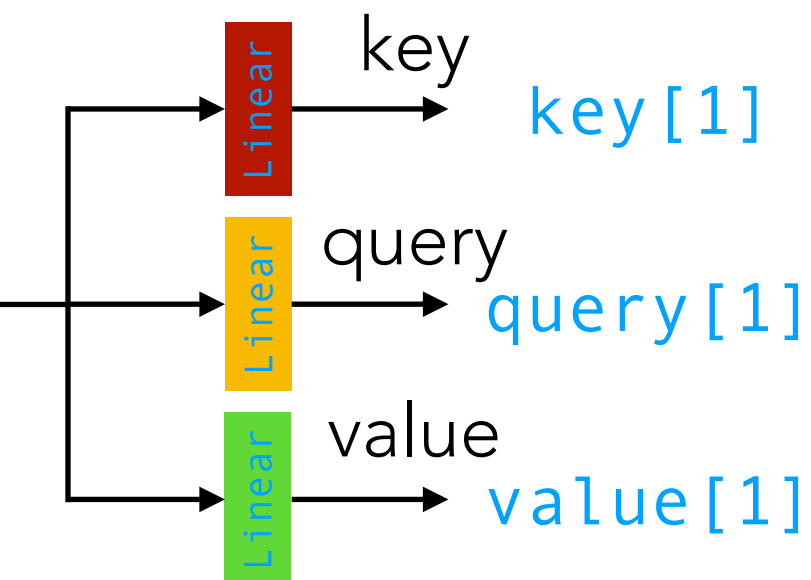


Remember: at this point, we are only computing the output for time step 2

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

Idea: make the contribution amount dependent on:

$$w[0] = \text{np.dot}(\text{query}[2], \text{key}[0])$$

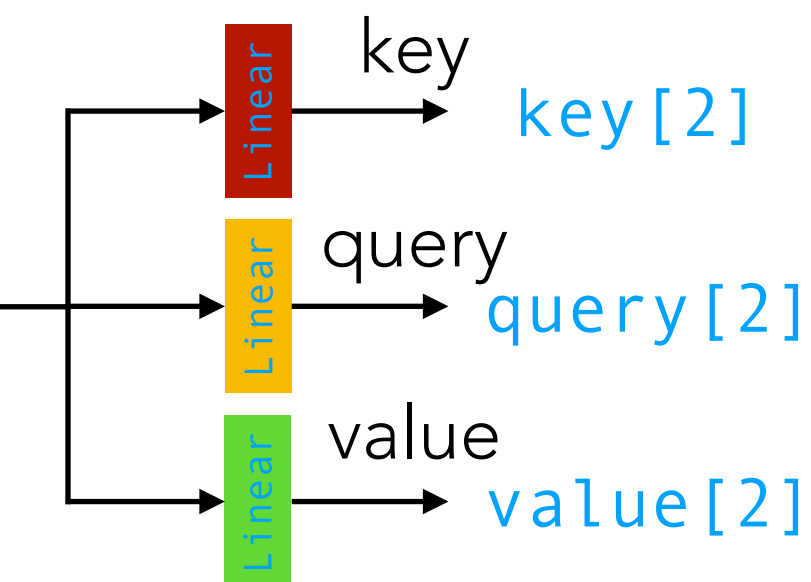


How much should time step 1's information contribute?

$$w[1] = \text{np.dot}(\text{query}[2], \text{key}[1])$$

How much should time step 2's information contribute?

$$w[2] = \text{np.dot}(\text{query}[2], \text{key}[2])$$



Let's normalize the weights so they are probabilities:

$$w_{\text{norm}} = \text{softmax}(w / \text{np.sqrt}(H))$$

In practice: include this division (helps with training)

Output at time step 2:

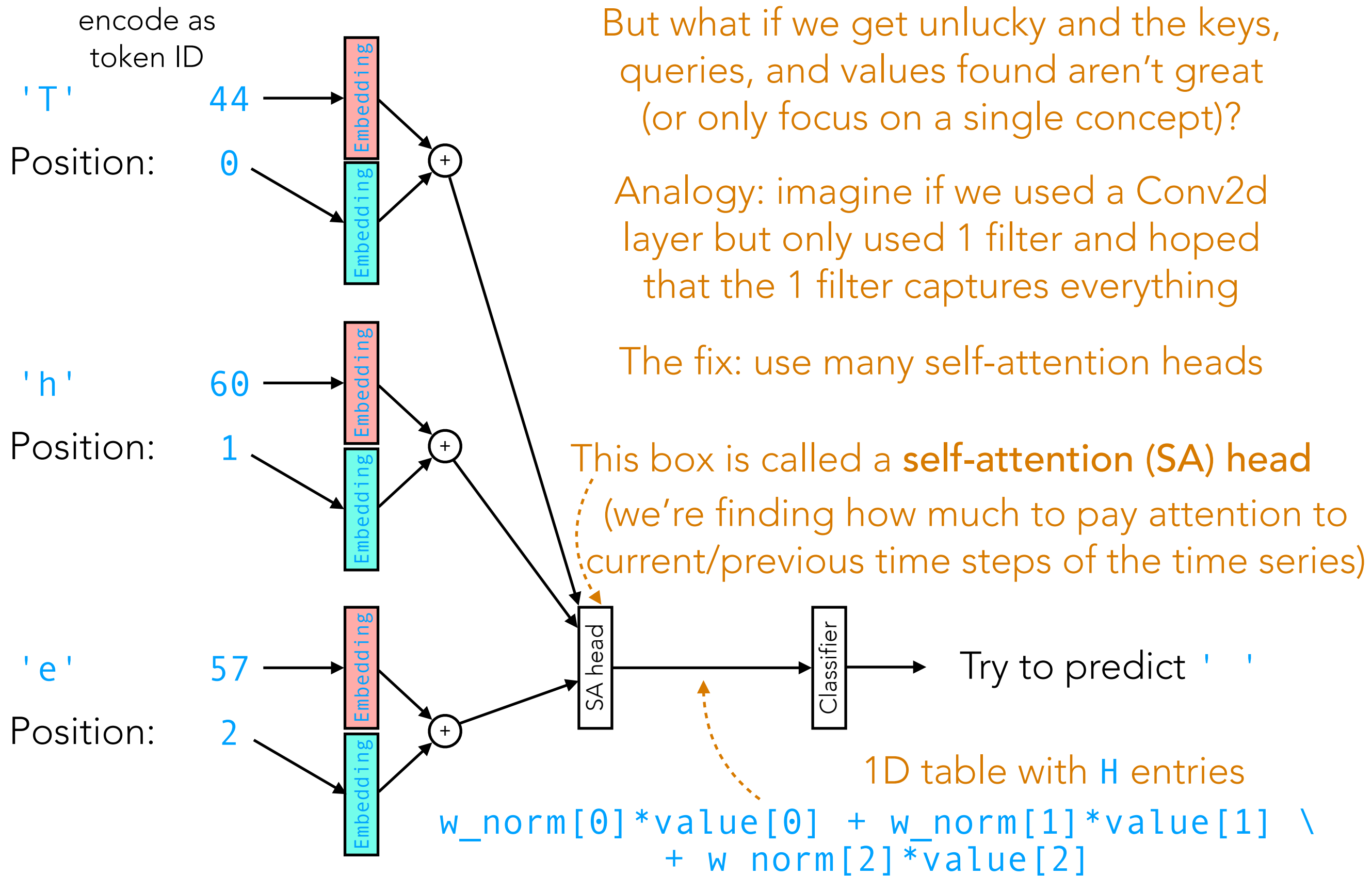
$$w_{\text{norm}}[0] * \text{value}[0] + w_{\text{norm}}[1] * \text{value}[1] + w_{\text{norm}}[2] * \text{value}[2]$$

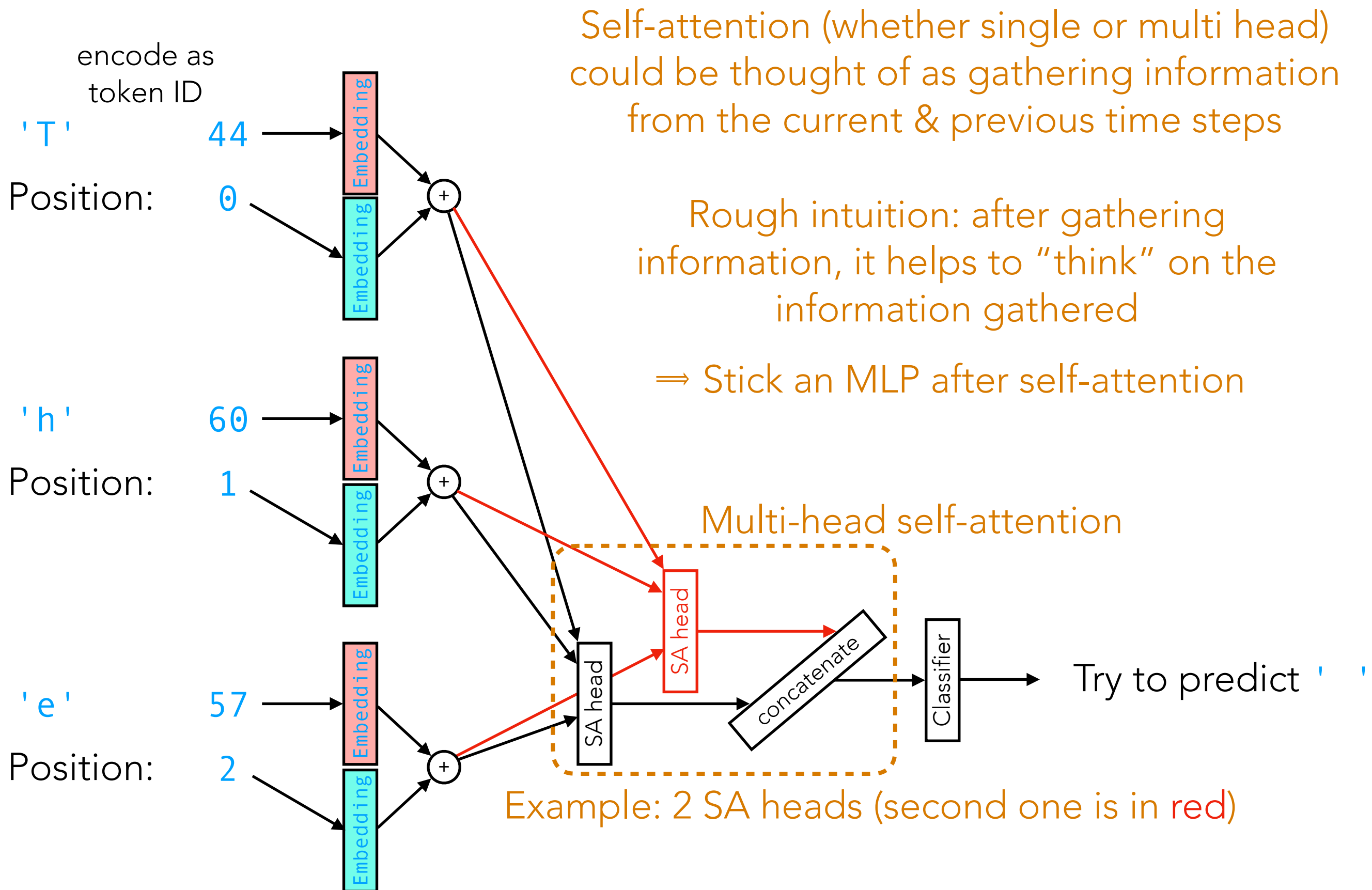
The hope: keys, queries, and values that get learned help with prediction

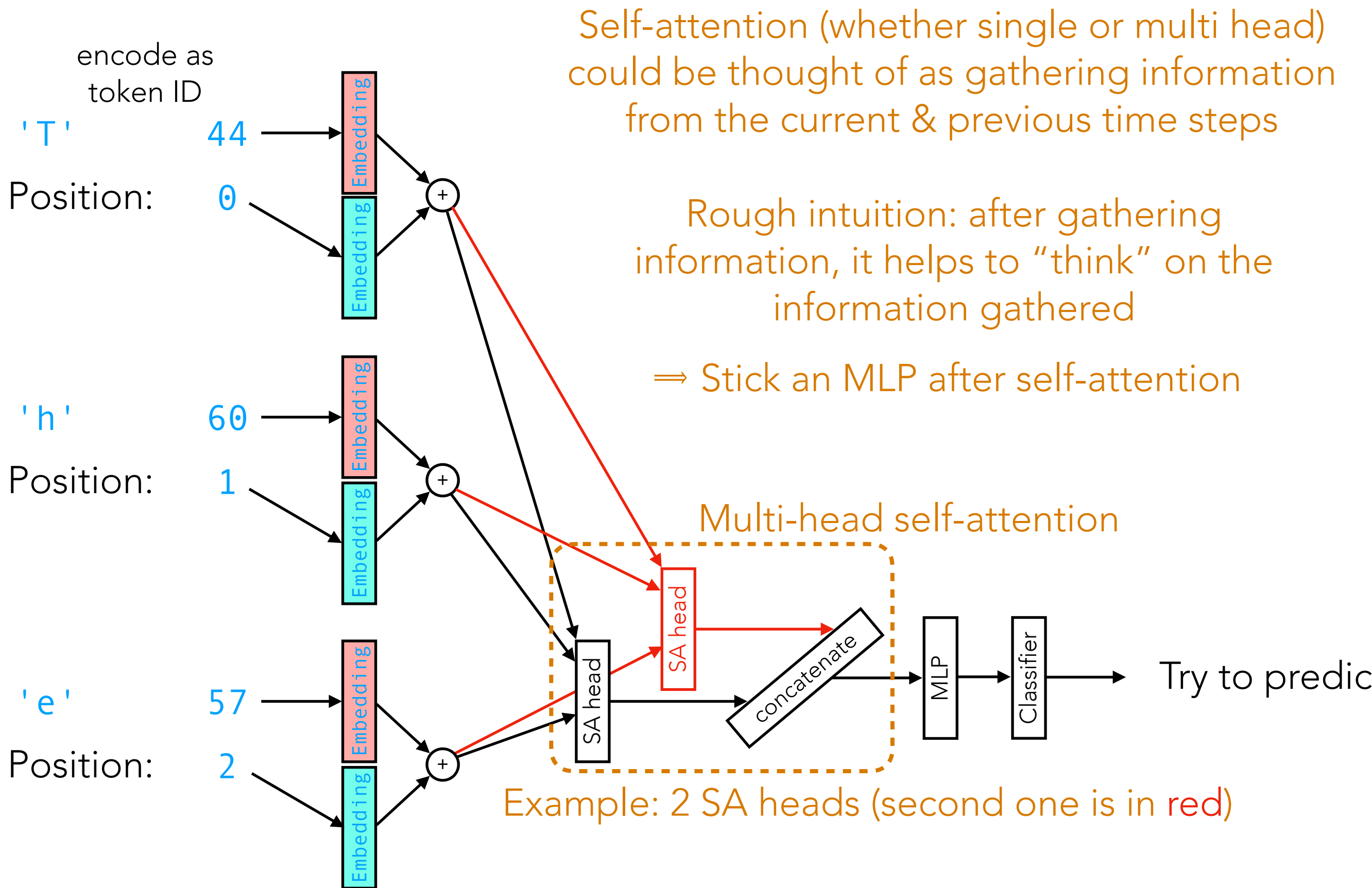
But what if we get unlucky and the keys, queries, and values found aren't great (or only focus on a single concept)?

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





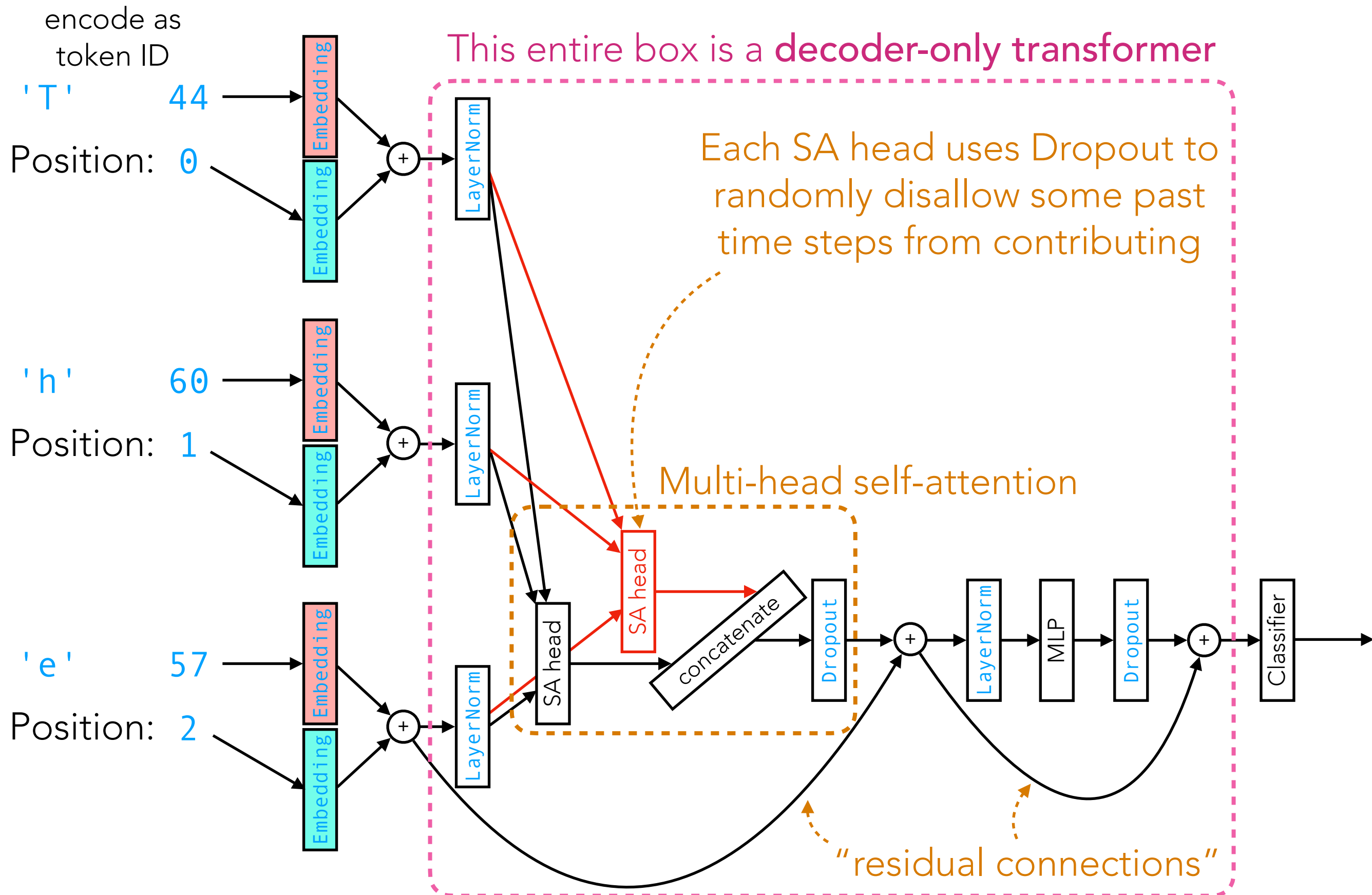


There are a few implementation details

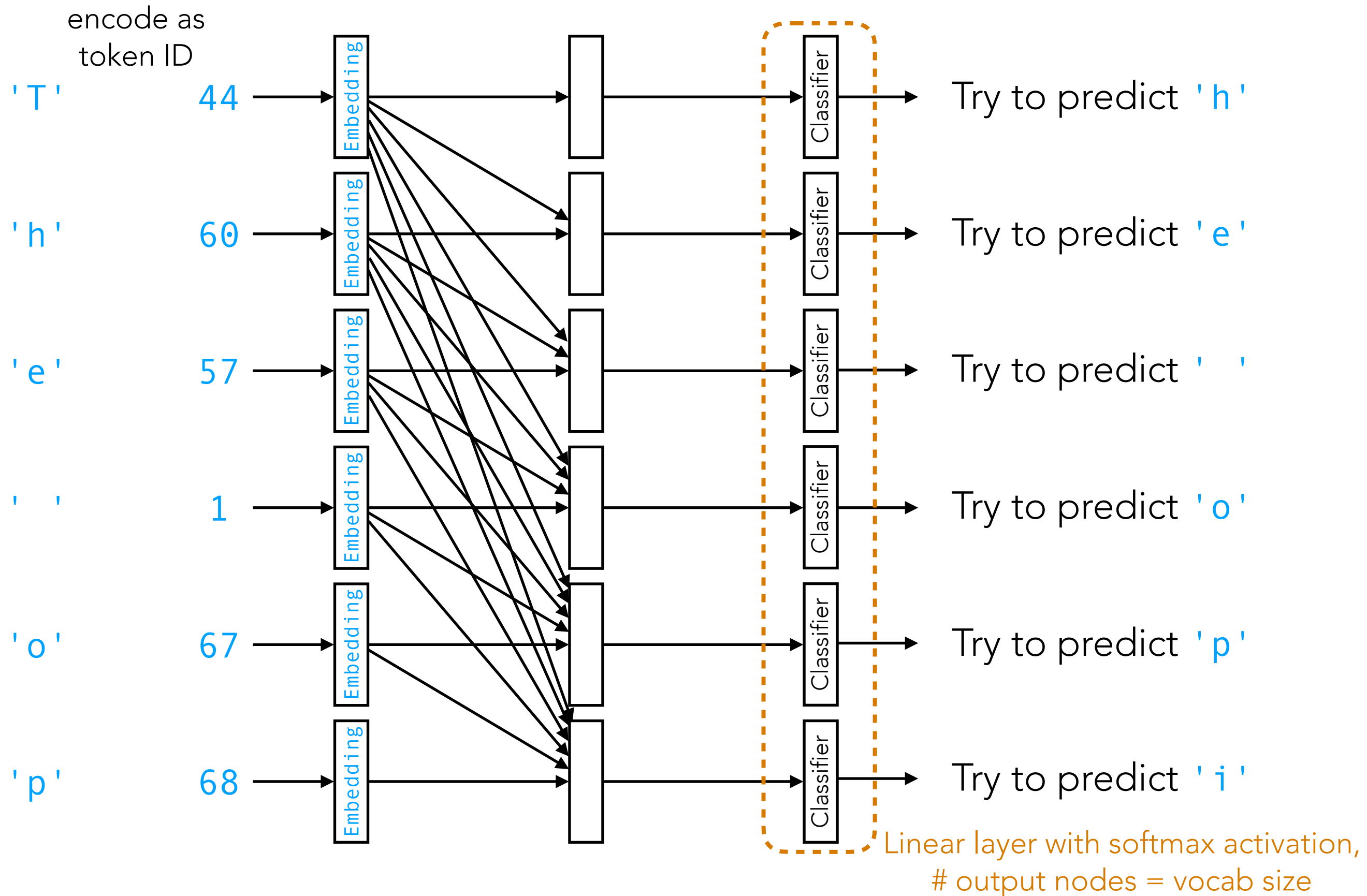
Basically, it turns out that when neural nets get very deep, training can be more difficult without some now-standard tricks (these tricks work with *many* neural net architectures, not just GPTs)

- LayerNorm
- Residual connections
- Dropout

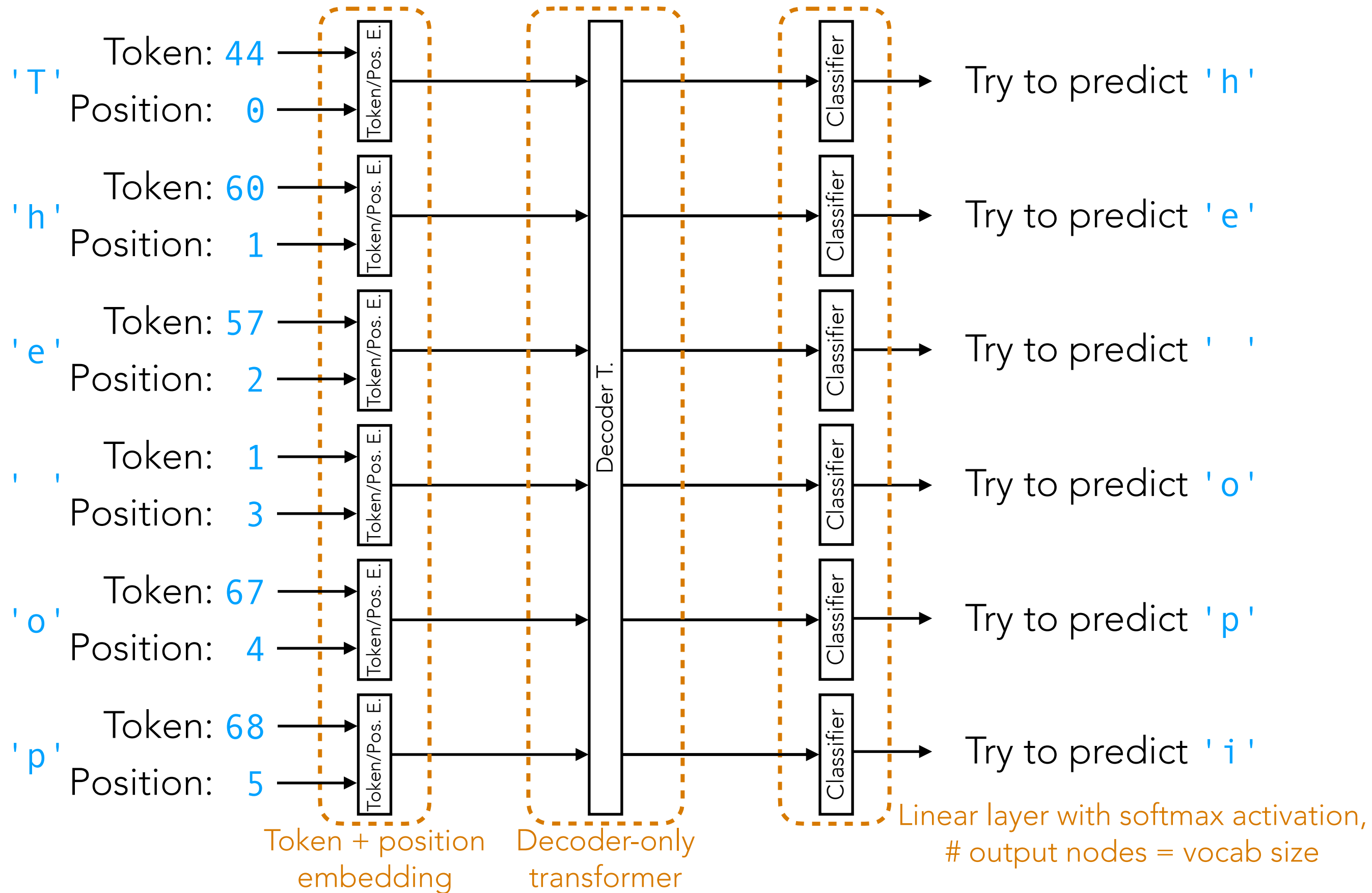
Also, there are some standard strategies for initializing GPT training



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

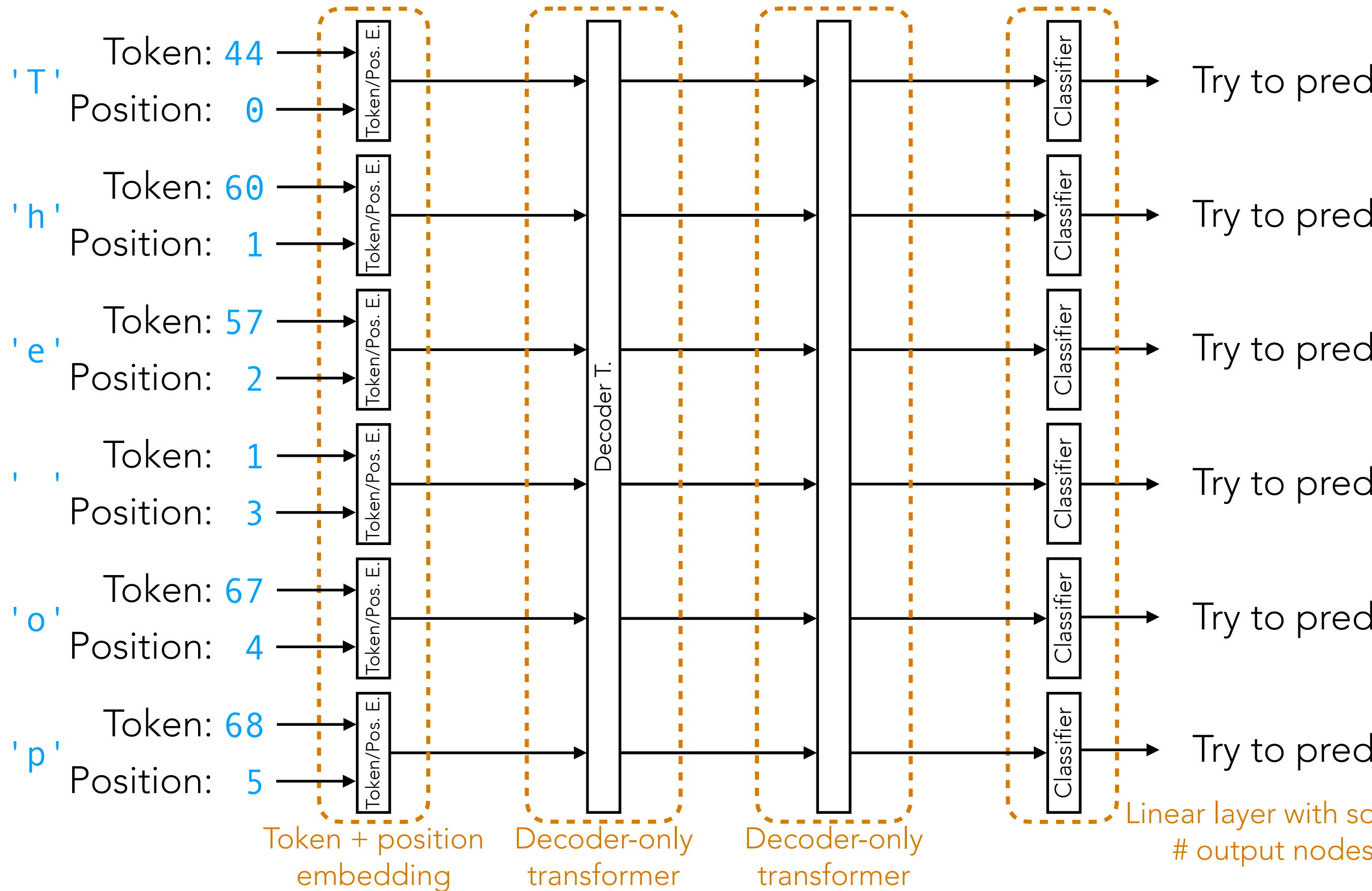


Generative Pre-trained Transformer (GPT)

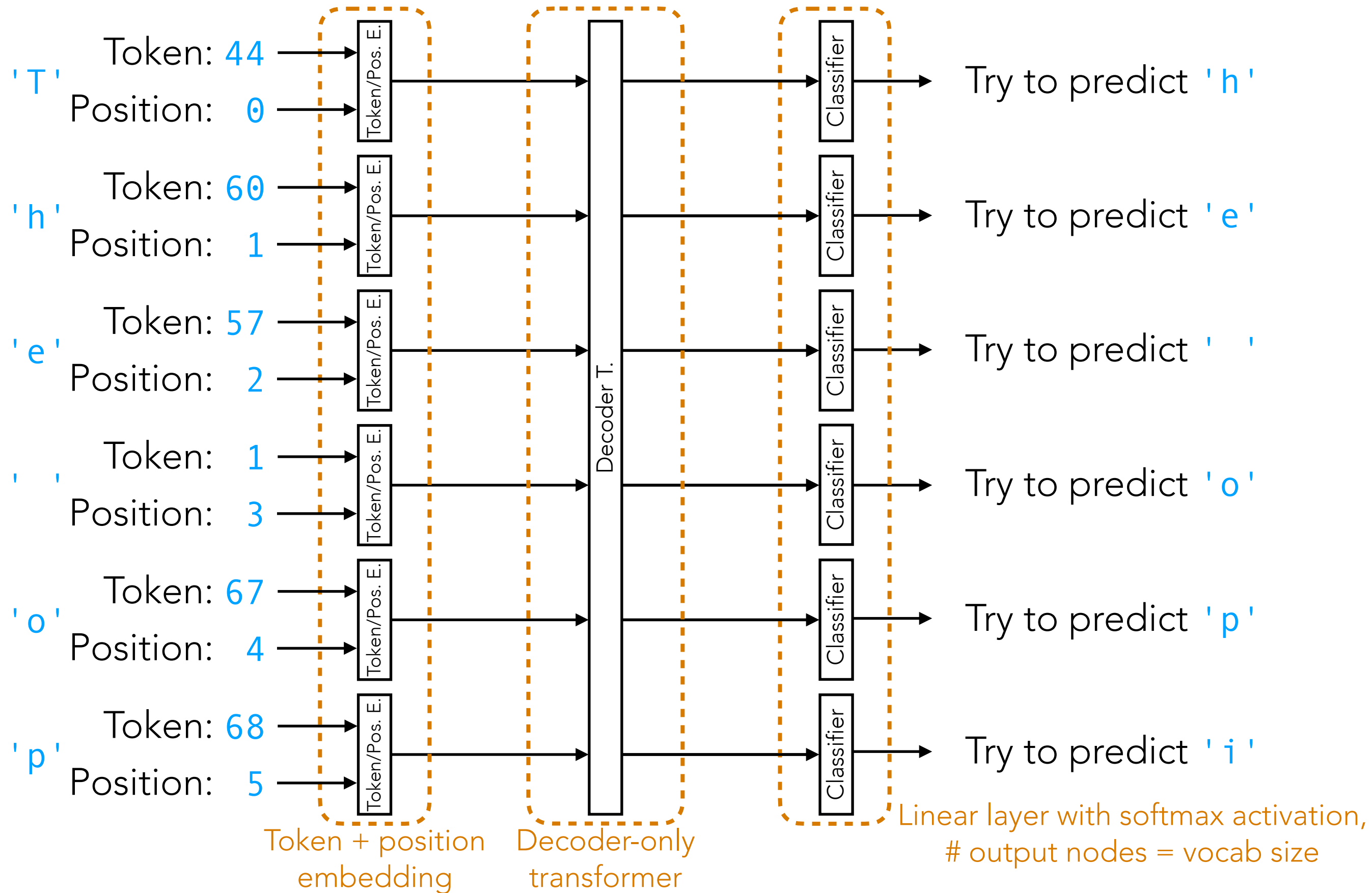


It is possible to stack transformer layers!

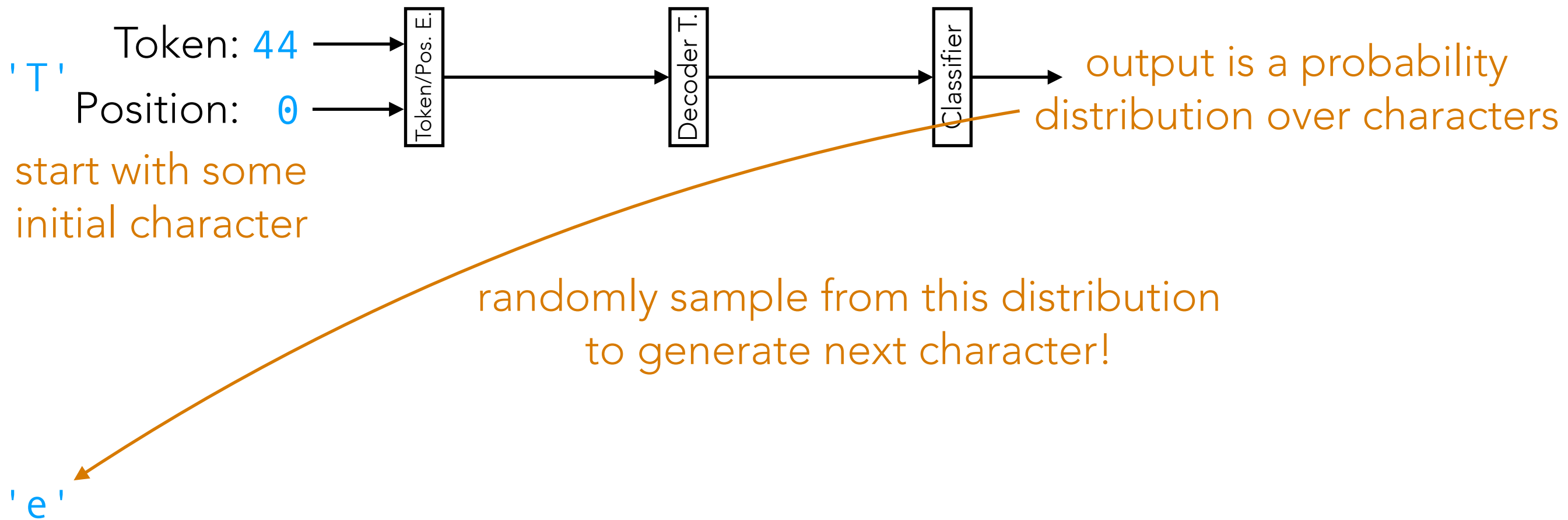
Generative Pre-trained Transformer (GPT)



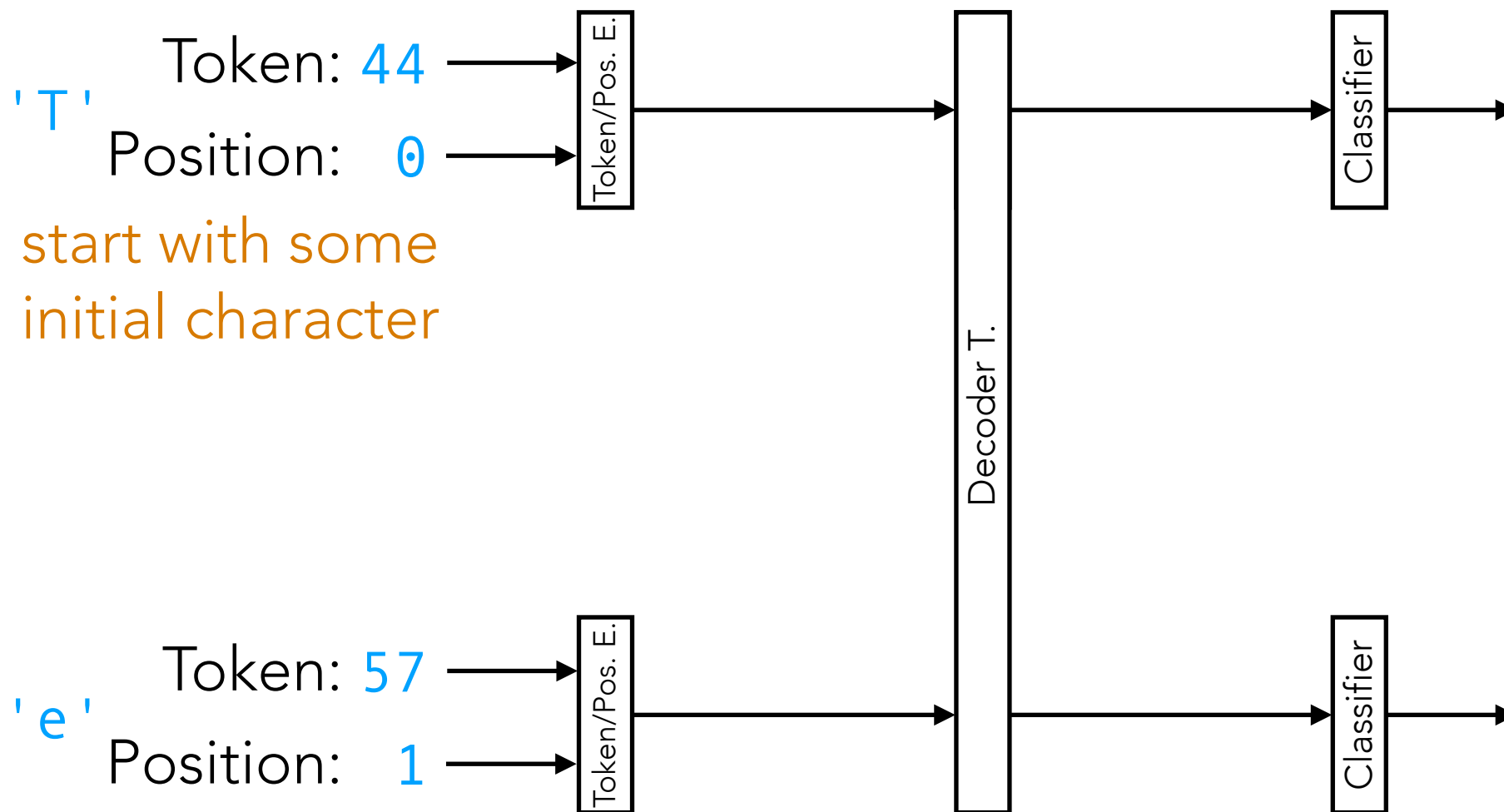
Generative Pre-trained Transformer (GPT)



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 is called “reinforcement learning with human feedback” (RLHF)

One more PyTorch thing...

Constructing PyTorch Models with `nn.Module`

(we'll need this level of detail in the next demo)

```
deeper_model = nn.Sequential(nn.Flatten(),
                             nn.Linear(in_features=784, out_features=512),
                             nn.ReLU(),
                             nn.Linear(in_features=512, out_features=10))
```

Another way to write this:

```
class DeeperModel(nn.Module):
    def __init__(self, num_in_features, num_intermediate_features, num_out_features):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(num_in_features, num_intermediate_features)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(num_intermediate_features, num_out_features)

    def forward(self, inputs):
        flatten_output = self.flatten(inputs)
        linear1_output = self.linear1(flatten_output)
        relu_output = self.relu(linear1_output)
        linear2_output = self.linear2(relu_output)
        return linear2_output

deeper_model = DeeperModel(784, 512, 10)
```


GPT

Demo