Carnegie Mellon University
Heinz College

# 95-865 Unstructured Data Analytics

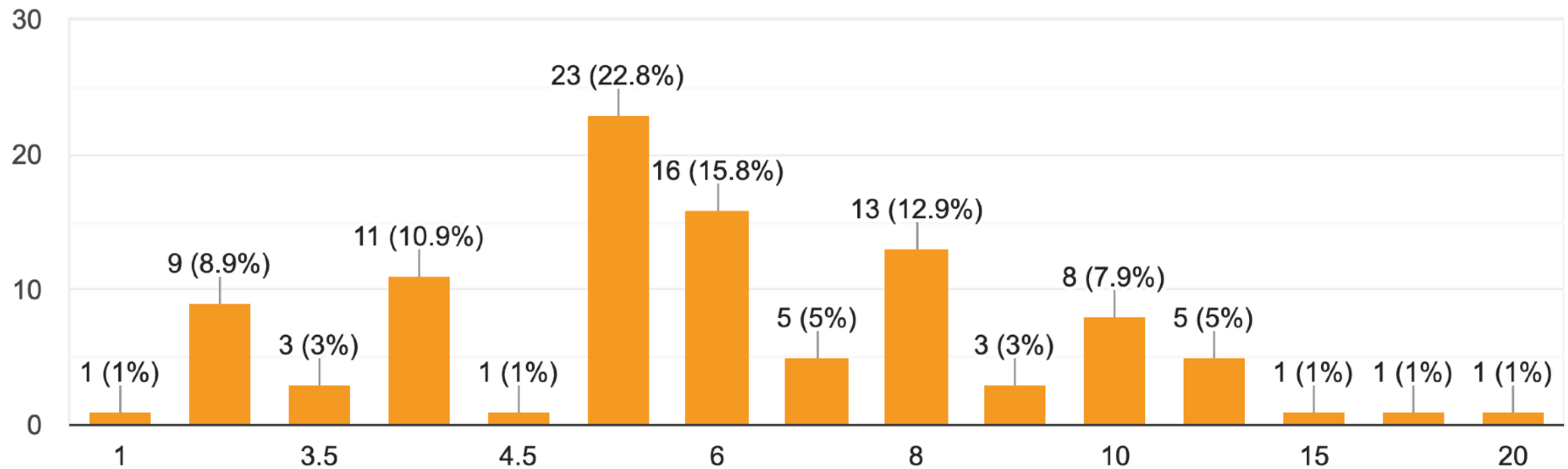
# Last lecture: Text generation with RNNs and generative pre-trained transformers (GPTs); course wrap-up

Slides by George H. Chen

# HW2 Questionnaire

How many hours did you take (roughly) to complete homework 2?

101 responses




What external resources are most useful seems to still be up for debate…

- A number of students mentioned that they find StatQuest helpful
- A number of students mentioned that they don't find StatQuest helpful
- A student pointed out a video on Linear Discriminant Analysis being helpful but this is not covered in 95-865 (Linear Discriminant Analysis is **not** the same as Latent Dirichlet Allocation!!!)

# HW2 Questionnaire

There were some comments about using large language models (LLMs) & why I'm not just teaching you to tell your favorite LLM to do things for you

- I personally think that a major point of this class is for you to understand the fundamental concepts underlying unstructured data analytics
  - I want *you* to understand it, not whether an AI or someone else understands it
- I'm currently not teaching prompt engineering because I think prompt engineering tricks will change over time
  - It's similar to how for search engines, there used to be tricks one would have to learn for how to get better search results, but now for the most part you don't need these tricks anymore
- LLMs and foundation models more generally will continue to get better, but for now when they fail/don't work, we don't fully understand why
  - In other words: sometimes they can give blatantly incorrect answers (and I'd hope that you know how to fact check what they tell you)

Important remark: many people teach neural nets in different ways; for your Quiz 2, for neural net questions that get asked, we will ask them in the context of how neural nets are taught in 95-865

# Today: Text generation as a prediction problem

We’re about to set up a prediction task
**without needing a human to annotate ground truth labels**

This is commonly referred to as *self-supervised learning*

We're setting up a prediction task in an *unsupervised* setting!

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 <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

Given `['T']`, predict next character `'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 <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters (so no need to use spaCy)

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

Given `['T', 'h']`, predict next character `'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 <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters (so no need to use spaCy)

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

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

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

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 <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters
(so no need to use spaCy)

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

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

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

Given `['T', 'h', 'e', ' ']`, predict next character `'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 <u>single data point</u> (a time series of tokens)

For tokenization, let's split by individual characters (so no need to use spaCy)

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

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

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

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

…

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

How to solve this prediction task with an RNN

# Vocabulary

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

```
vocabulary
```

```
array(['\n', ' ', '"', '$', '%', '&', "'", '(', ')', ',', '-', '.', '/',
       '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '?',
       'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
       'N', 'O', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', '[', ']',
       '^', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
       'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
       'z', '|', '–', '—', '‘', '’', '“', '”'], dtype='<U1')
```

```
term_to_idx = {term: idx for idx, term in enumerate(vocabulary)}
```

```
def vocab(text):
    return [term_to_idx[char] for char in text]
```

```
vocab('The opi')
```

```
[44, 60, 57, 1, 67, 68, 61]
```

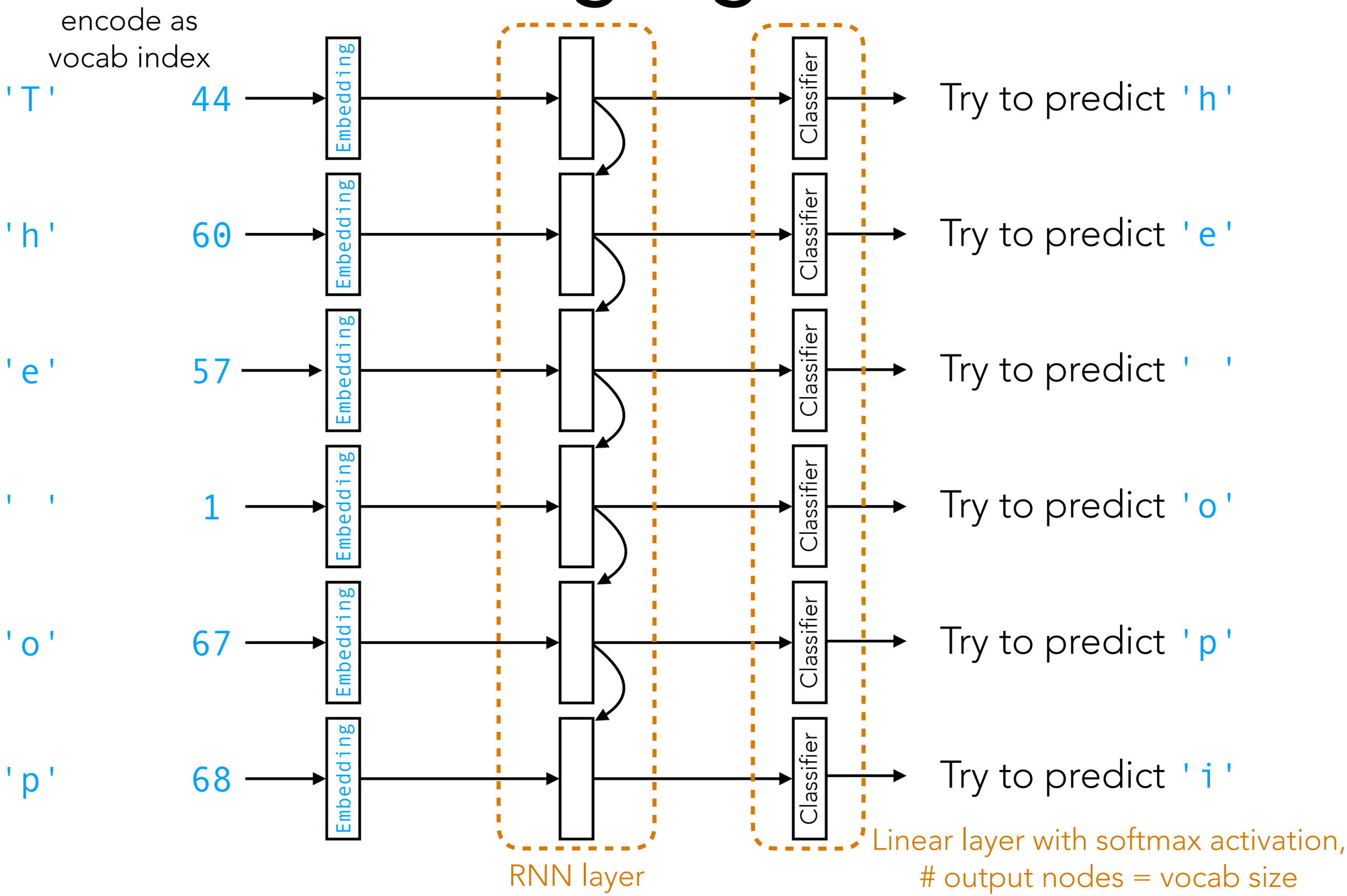
# RNN Language Model

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

length = $L + 1$

$L = 6$ in this example

# RNN Language Model



# RNN Language Model

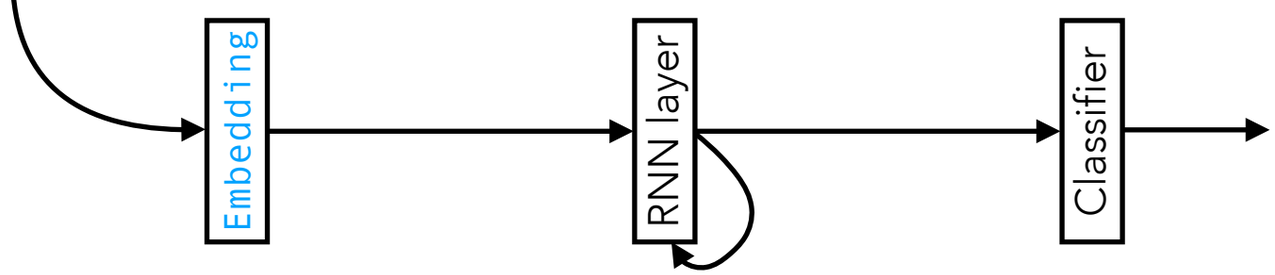
Use each time step's RNN output as the input to the classifier to predict the next time step's character

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

Again, this is a <u>single</u> data point

```
[44, 60, 57, 1, 67, 68]
```

We use the exact same classifier for every time step




Train model the same way we train other neural nets (use cross entropy loss)!

Now we keep every time step's output of the RNN (not just the last time step's)

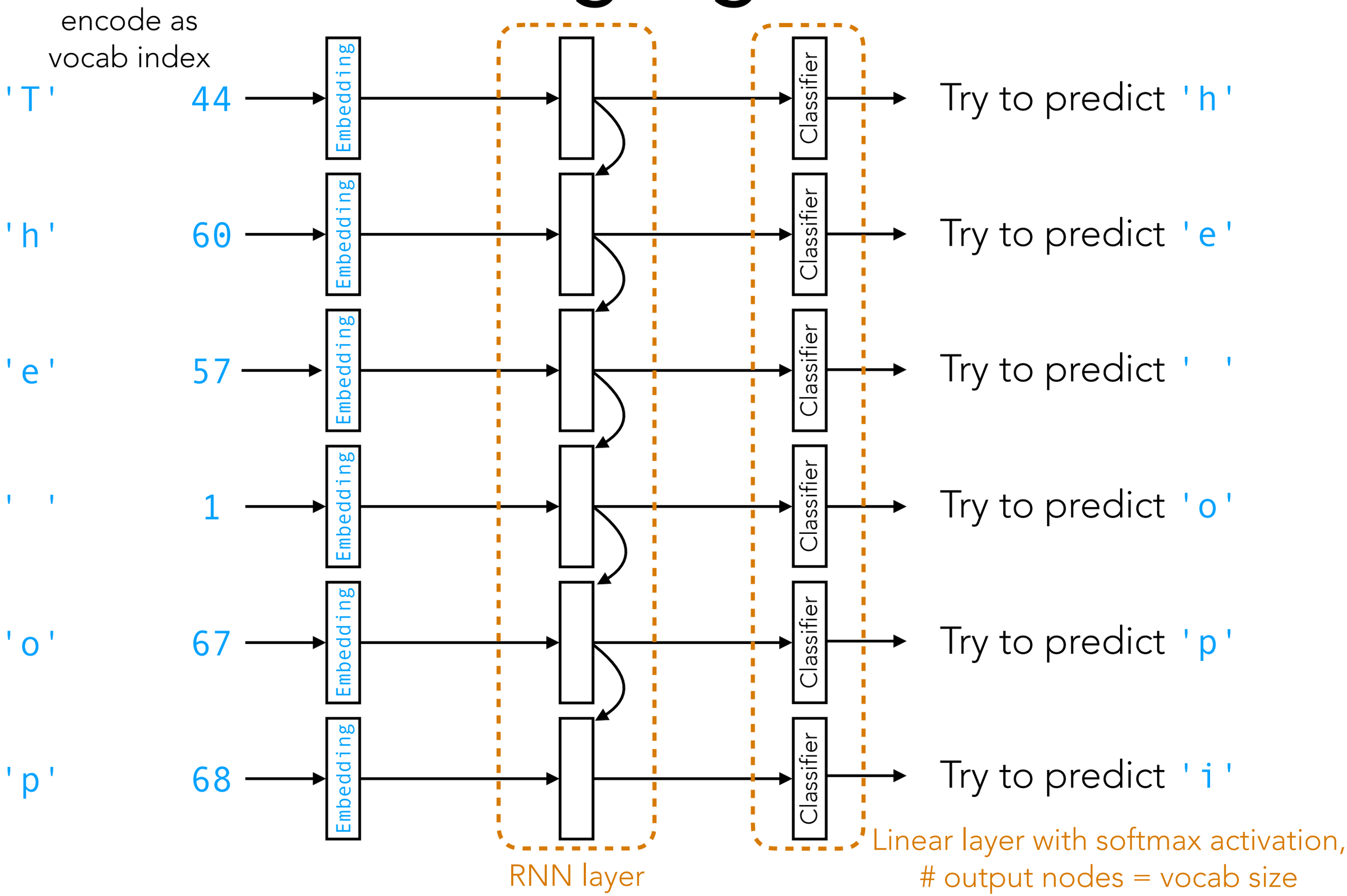
Ground truth target label for this single data point (this target label does not need to be manually annotated by a human):

```
['h', 'e', ' ', 'o', 'p', 'i']
```

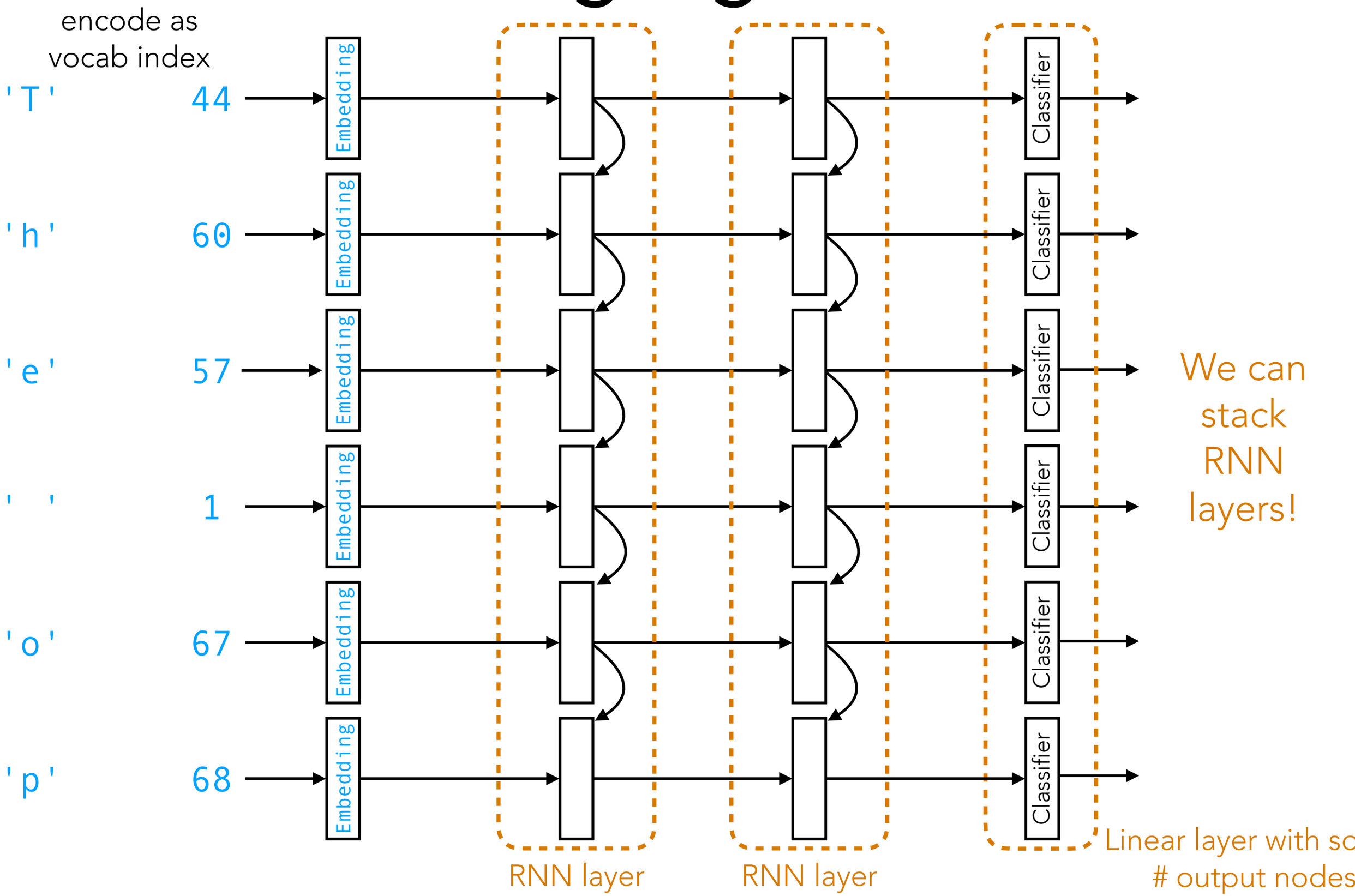
Technically, the actual ground truth is the encoded version:

```
[60, 57, 1, 67, 68, 61]
```

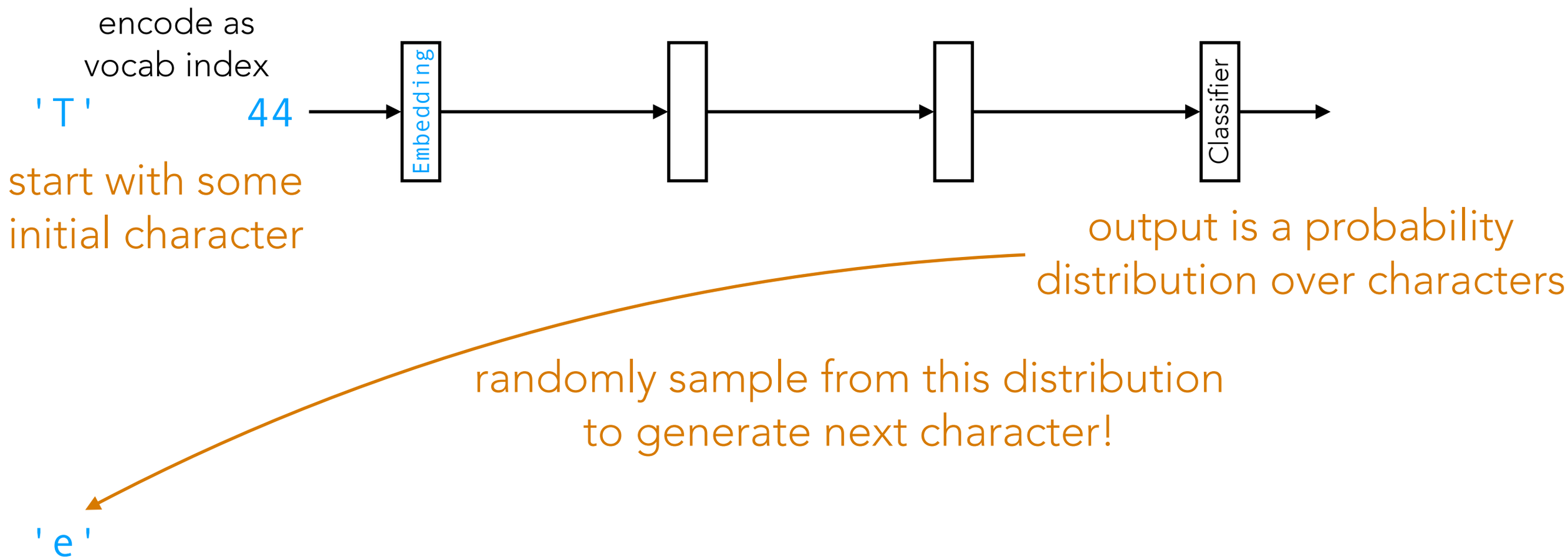
# RNN Language Model



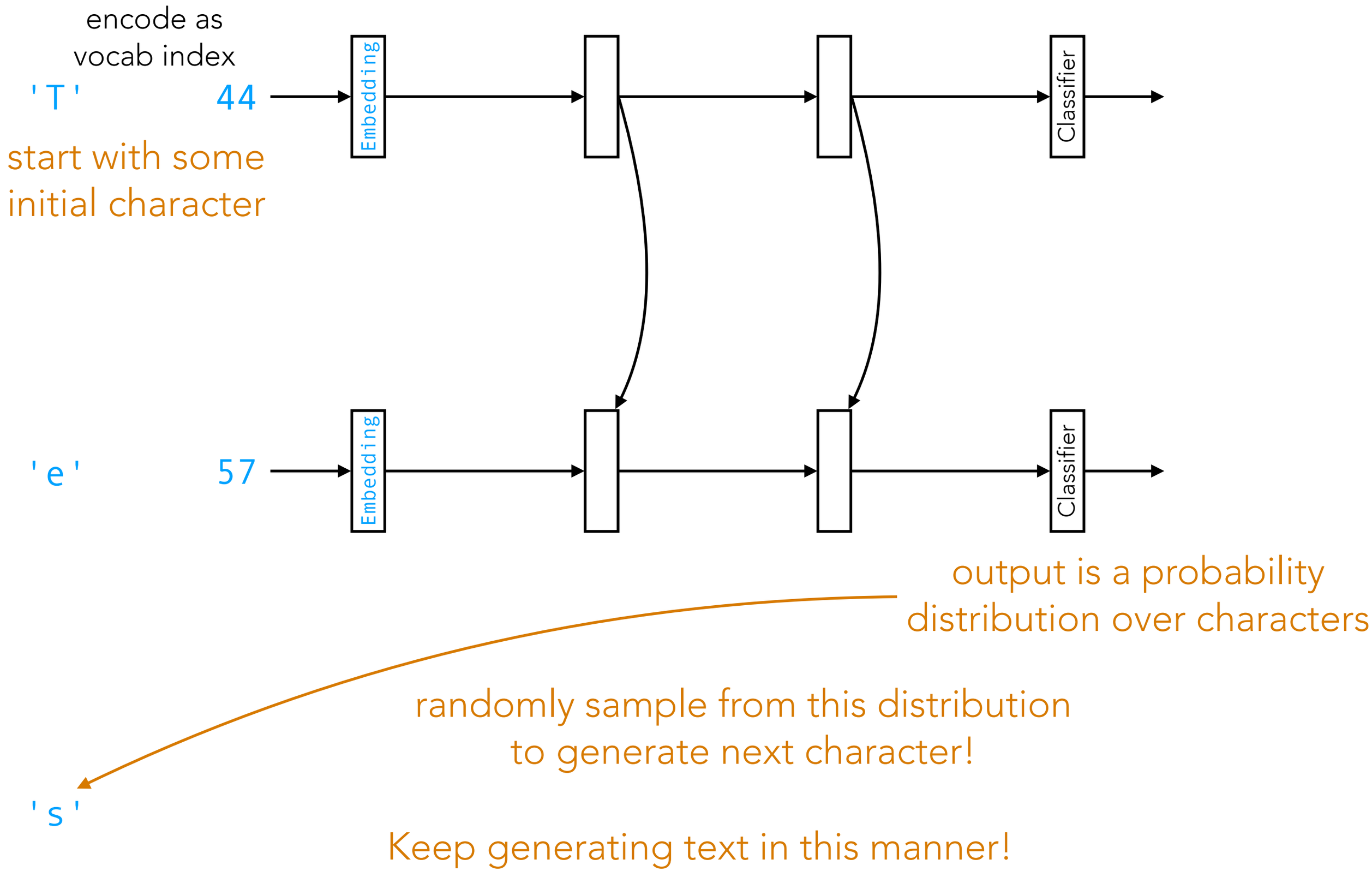
# RNN Language Model



# How to Generate Text After Model Training

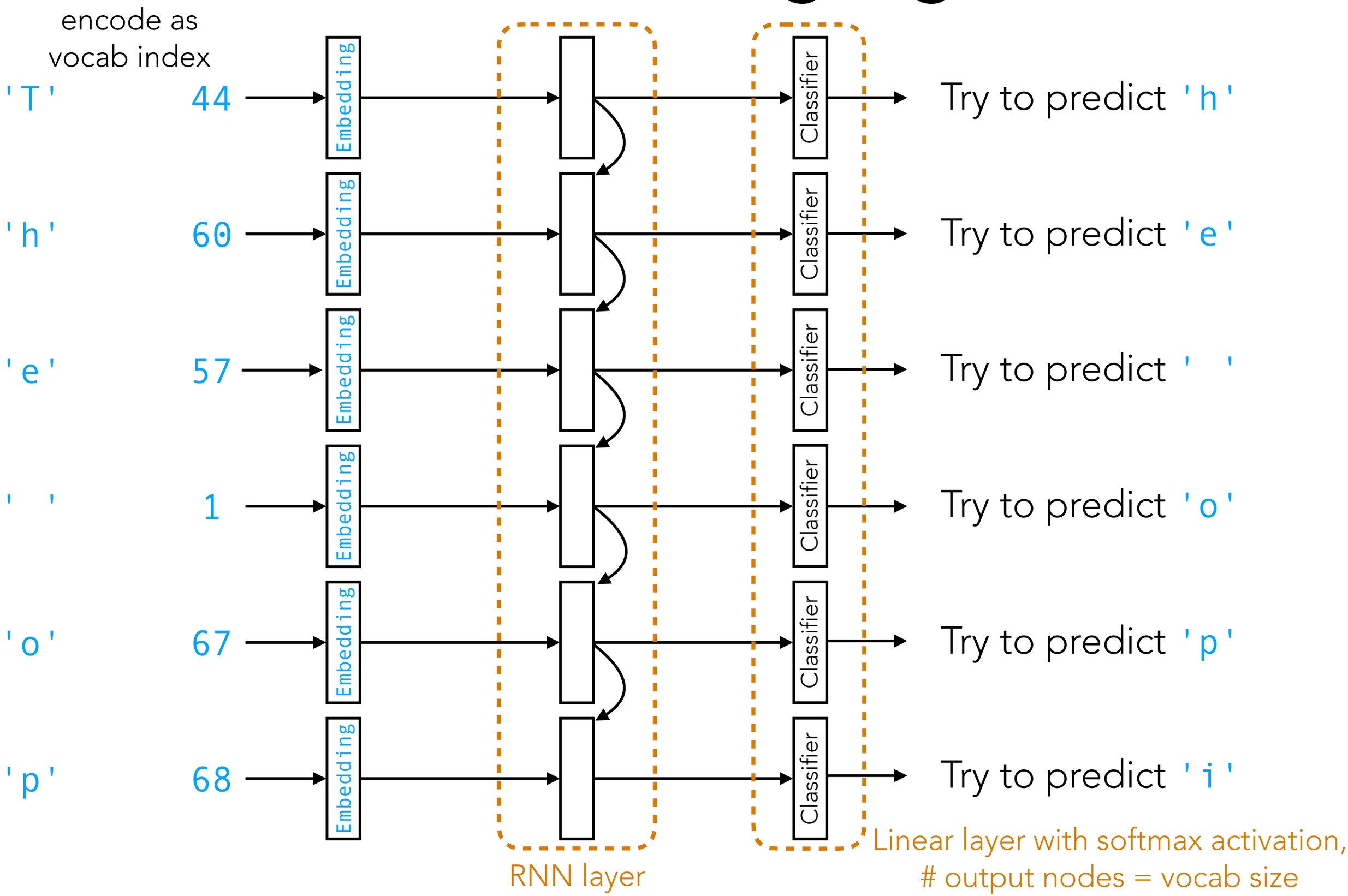


# How to Generate Text After Model Training

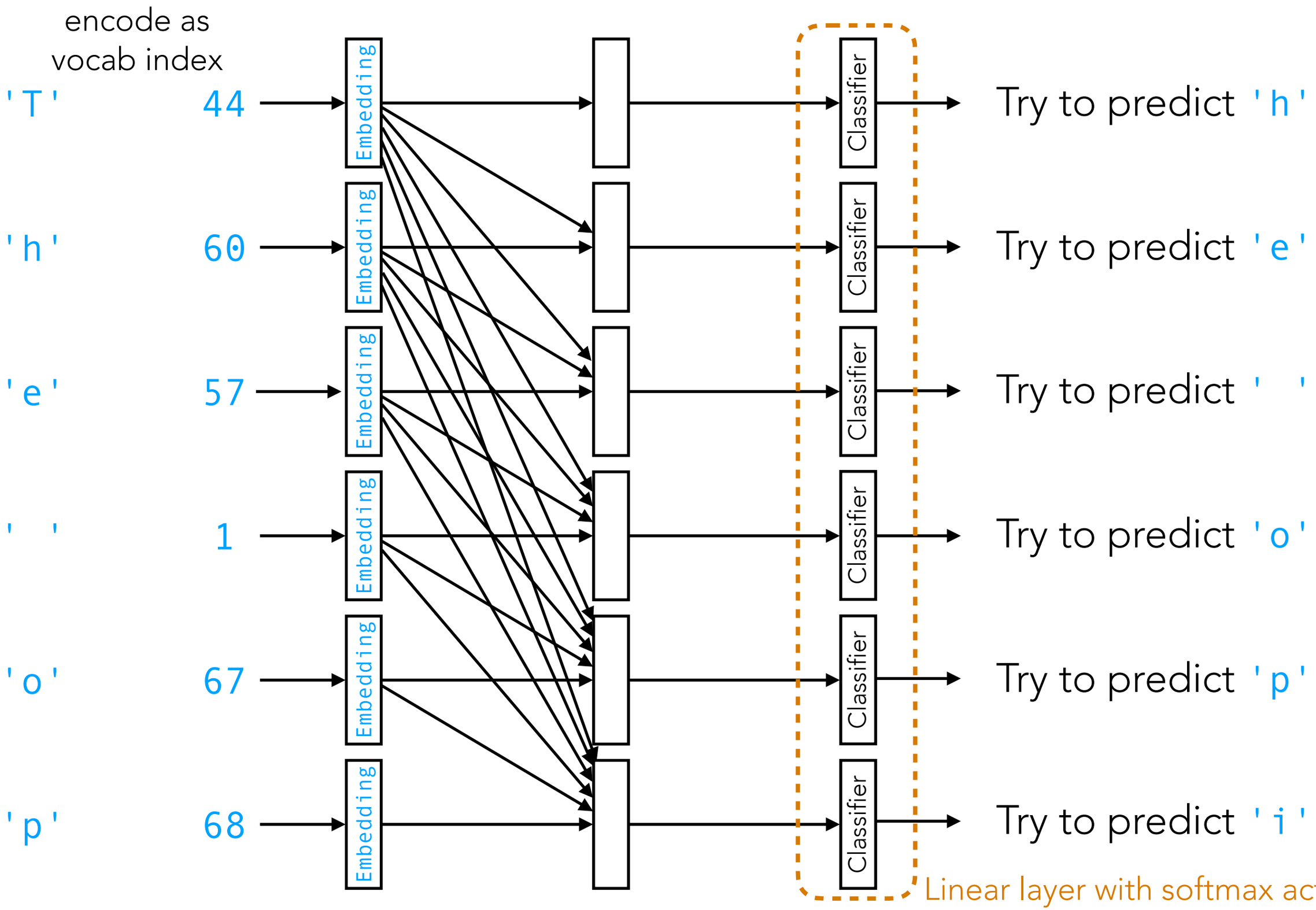


# Generative Pre-trained Transformers (GPTs)

# (Flashback) RNN Language Model

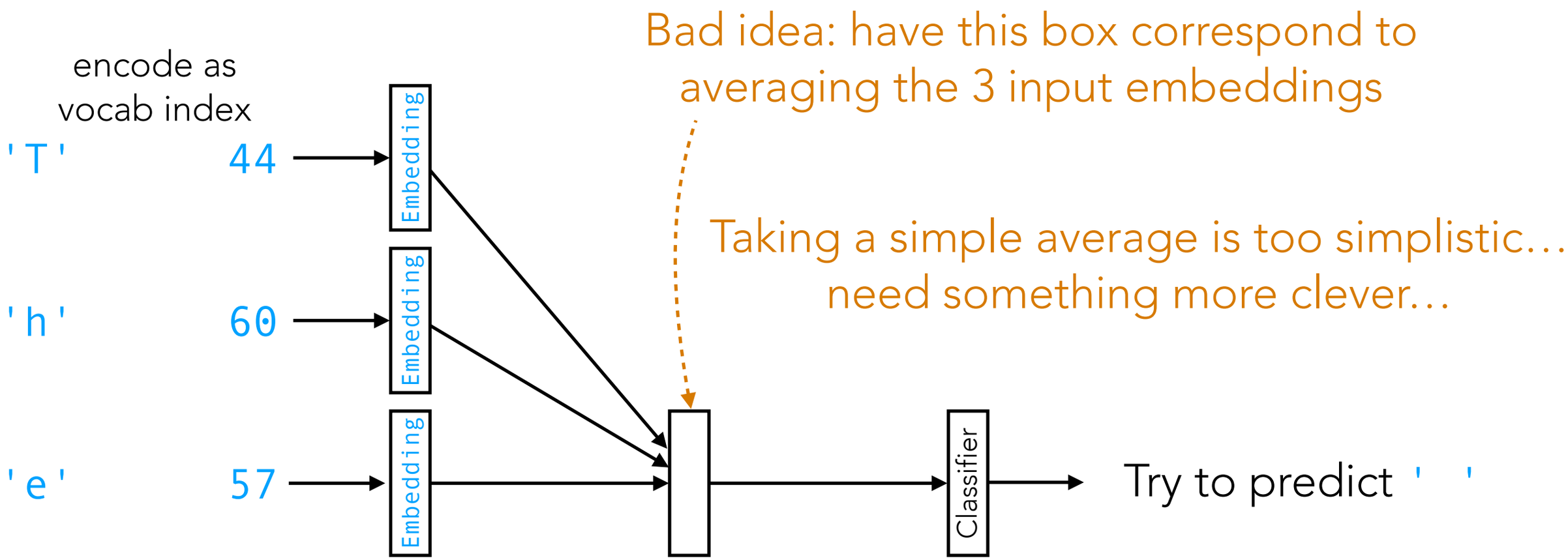


This sort of dependence is "causal": any time step can only depend on its current input and all past inputs (and **not** on future time steps' inputs)




Linear layer with softmax activation, # output nodes = vocab size

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



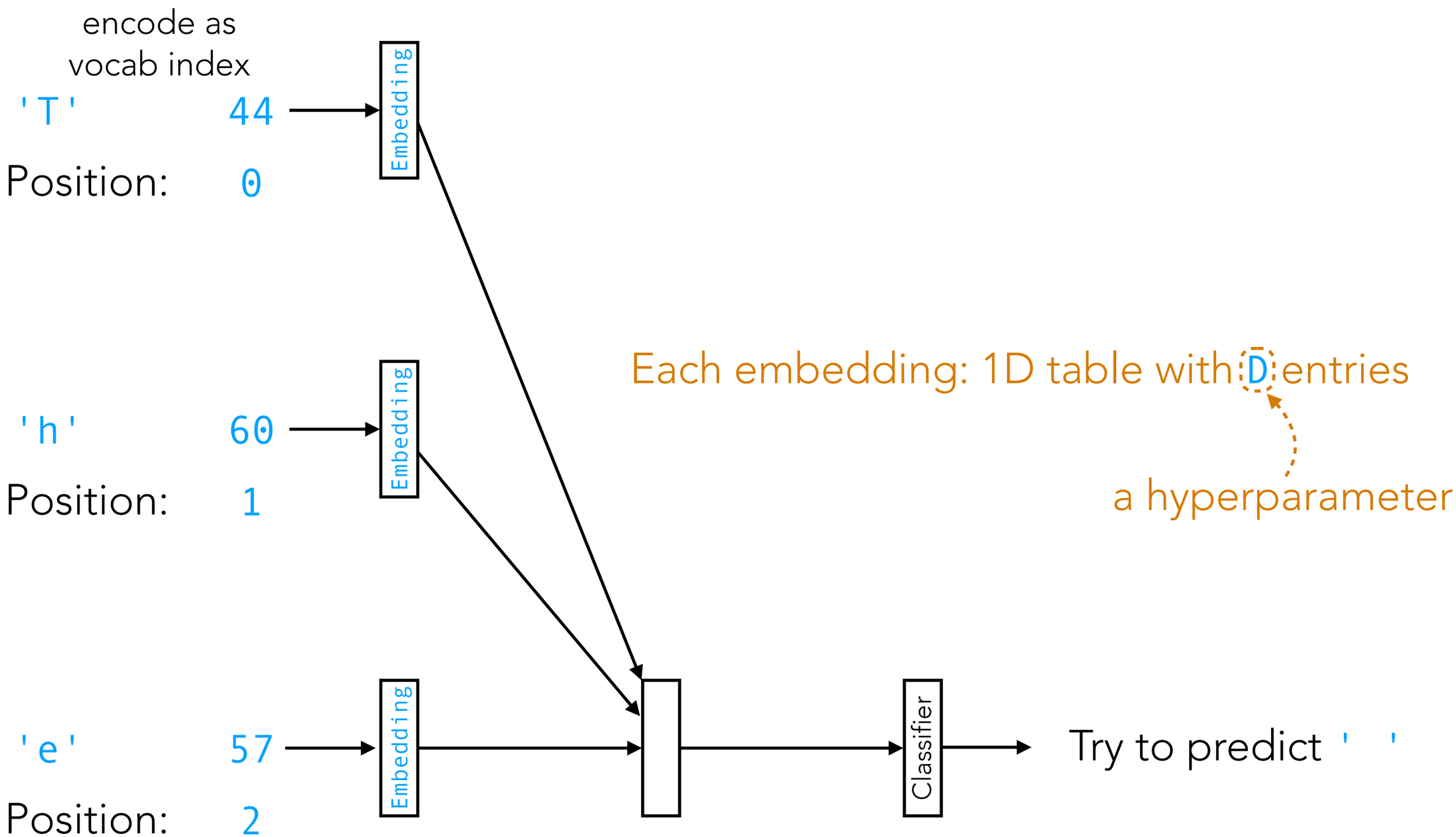

Bad idea: have this box correspond to averaging the 3 input embeddings

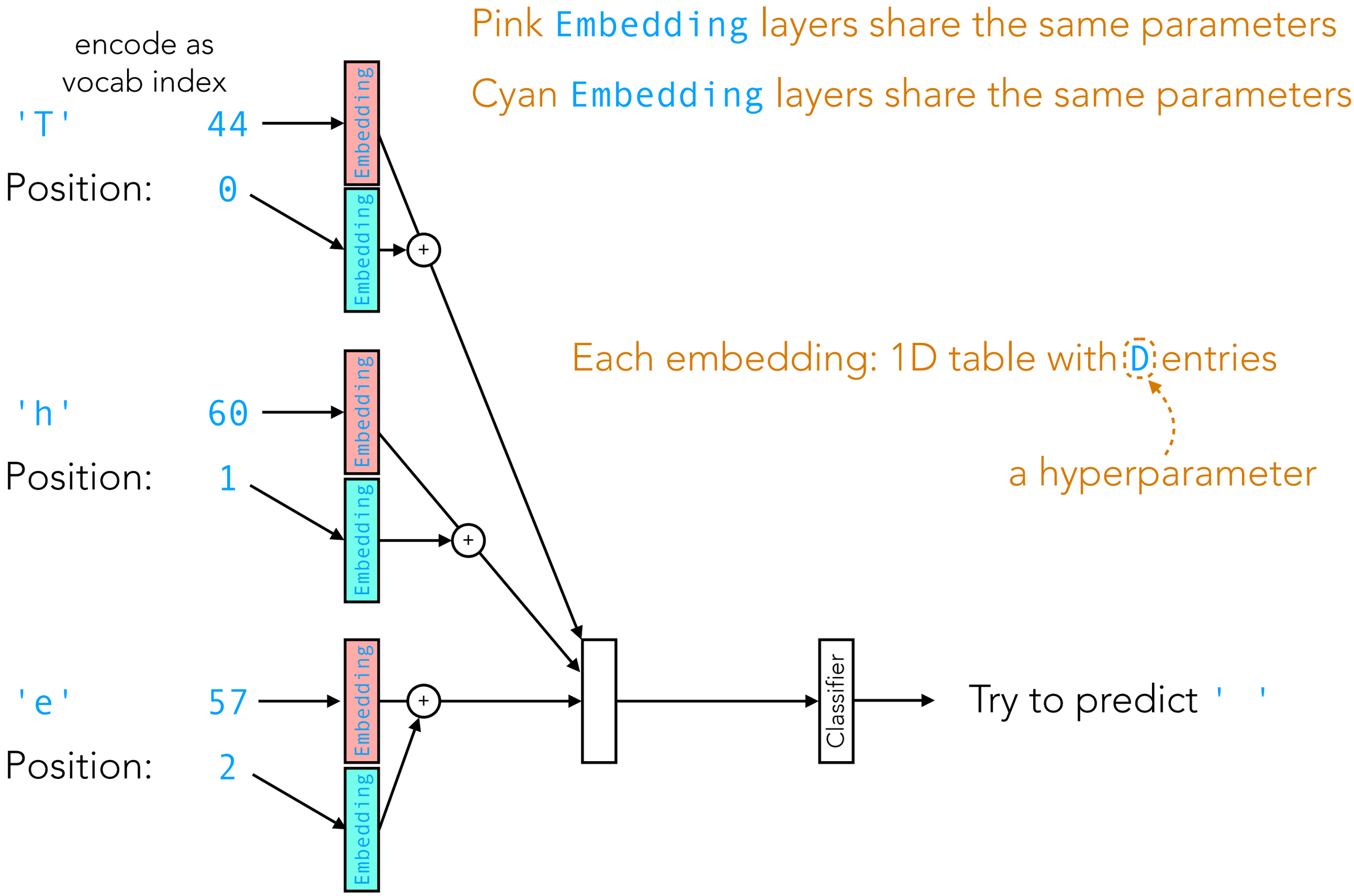
Taking a simple average is too simplistic… need something more clever…

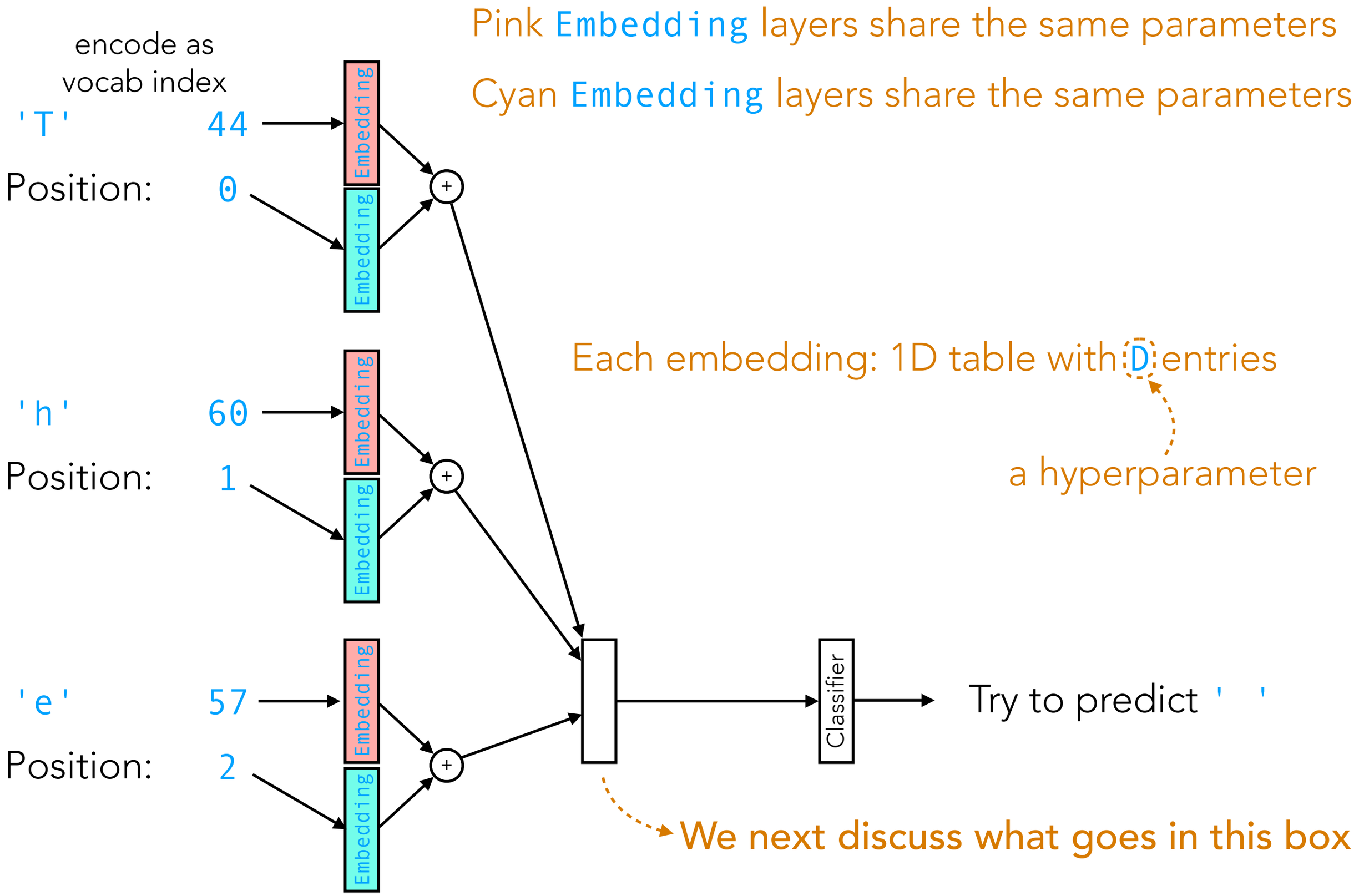
How should we combine information from the input embeddings?

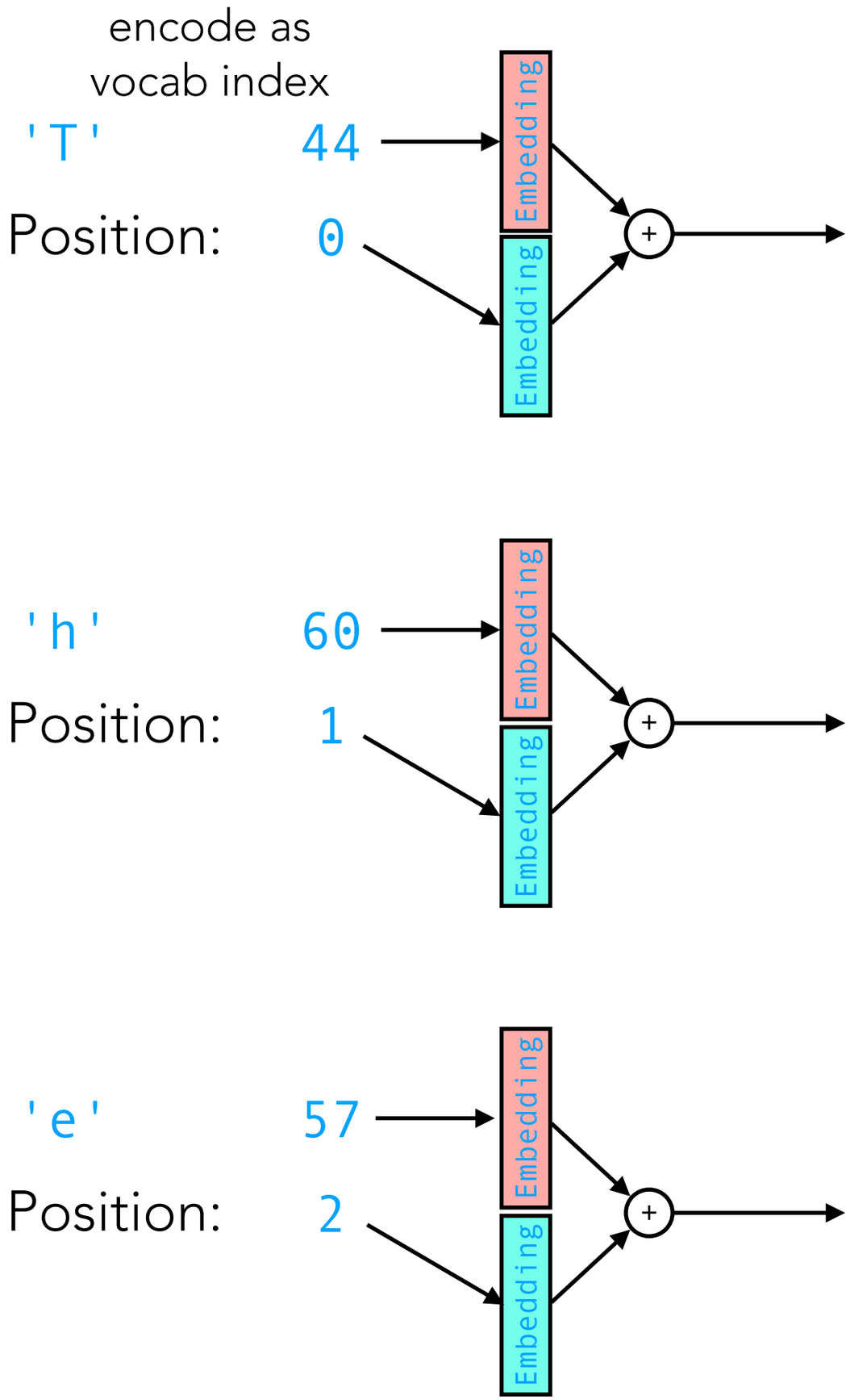
Another issue: the input embeddings by themselves do not contain information about *when* the time steps happened

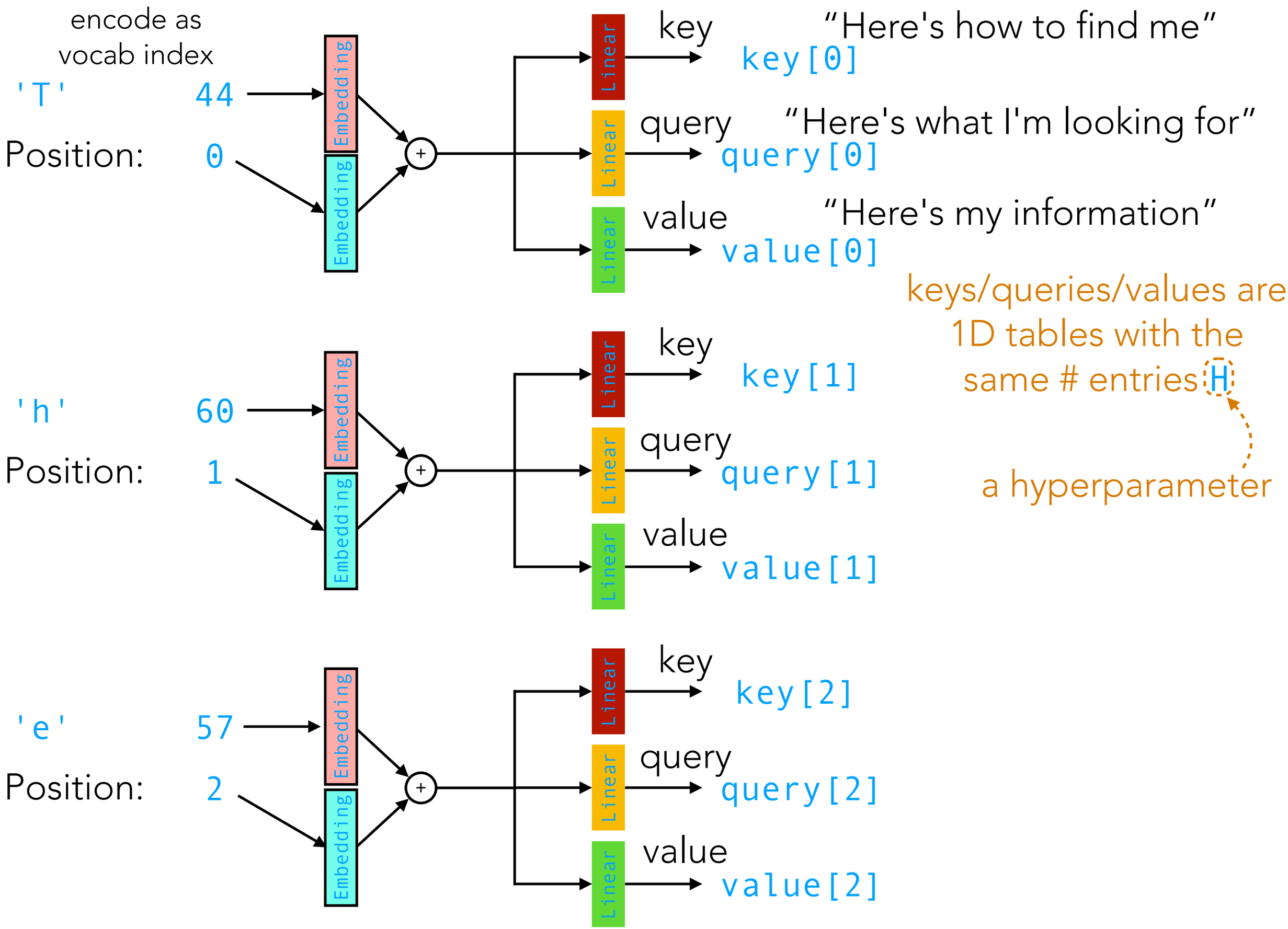
Let's address this issue first

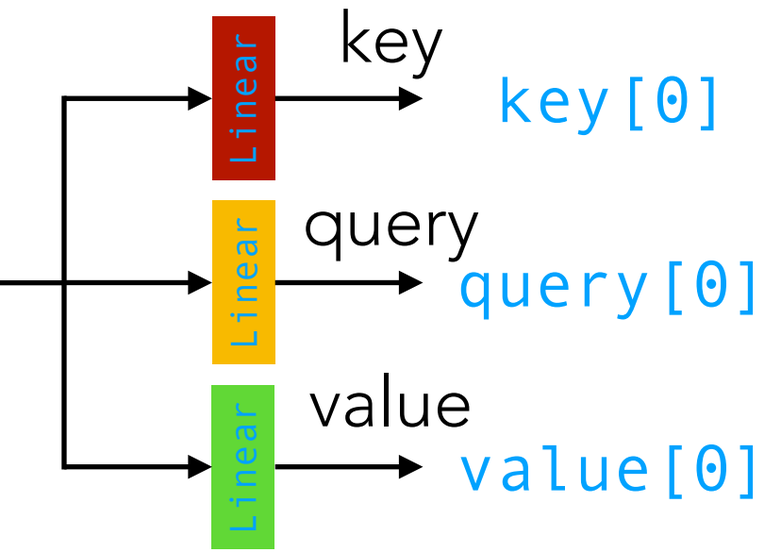

encode as
vocab index
'T'
44
Position:
0
'h'
60
Position:
1
'e'
57
Position:
2
Embedding
Embedding
Embedding
Classifier
Try to predict ' '
Each embedding: 1D table with D entries
a hyperparameter

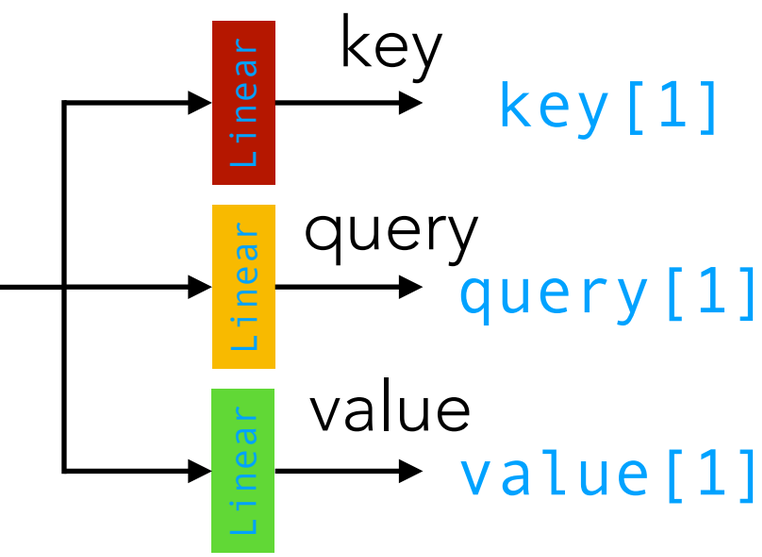

Pink Embedding layers share the same parameters
Cyan Embedding layers share the same parameters
encode as vocab index
'T'
44
Position:
0
'h'
60
Position:
1
'e'
57
Position:
2
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
+
+
+
Classifier
Try to predict ' '
Each embedding: 1D table with D entries
a hyperparameter

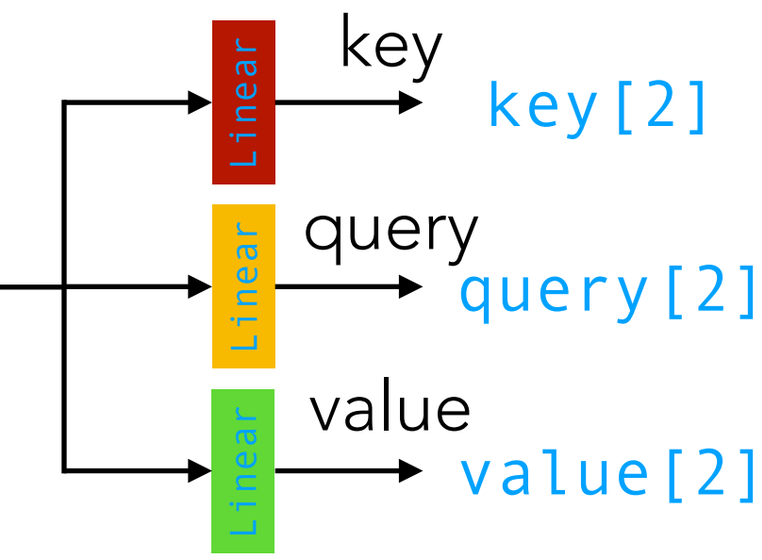

Pink Embedding layers share the same parameters
Cyan Embedding layers share the same parameters
encode as vocab index
'T'
44
Position:
0
'h'
60
Position:
1
'e'
57
Position:
2
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
+
+
+
Classifier
Try to predict ' '
Each embedding: 1D table with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


encode as
vocab index
'T'
44
Position:
0
Embedding
Embedding
+
'h'
60
Position:
1
Embedding
Embedding
+
'e'
57
Position:
2
Embedding
Embedding
+


encode as
vocab index
'T'
44
Position:
0
Embedding
Embedding
Linear
key
"Here's how to find me"
key[0]
Linear
query
"Here's what I'm looking for"
query[0]
Linear
value
"Here's my information"
value[0]
keys/queries/values are
1D tables with the
same # entries
H
a hyperparameter
'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]




**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?

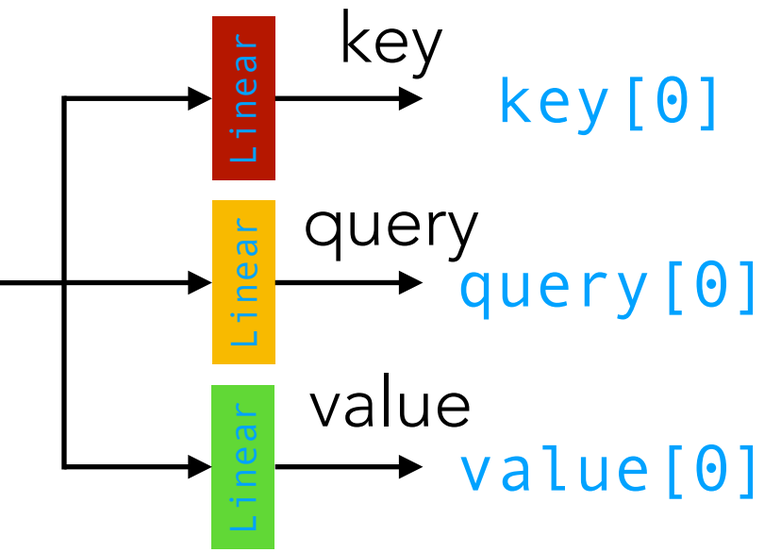
```
w[2] = np.dot(query[2], key[2])
```

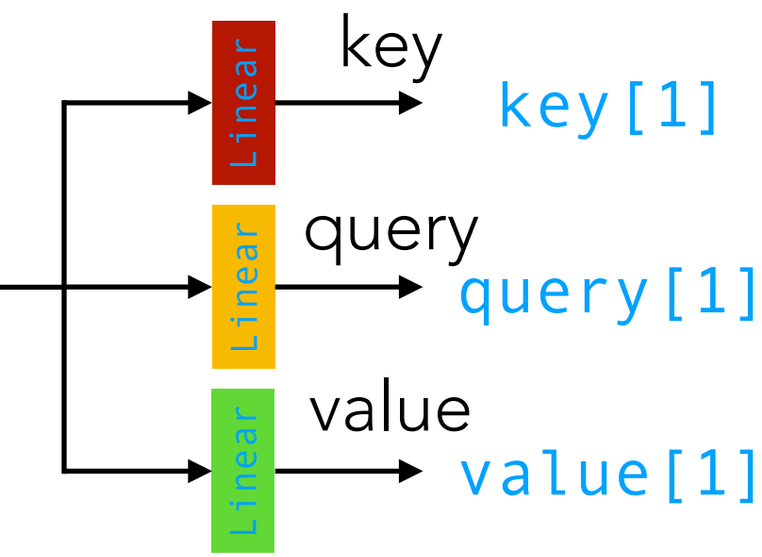
Let's normalize the weights so they are probabilities:

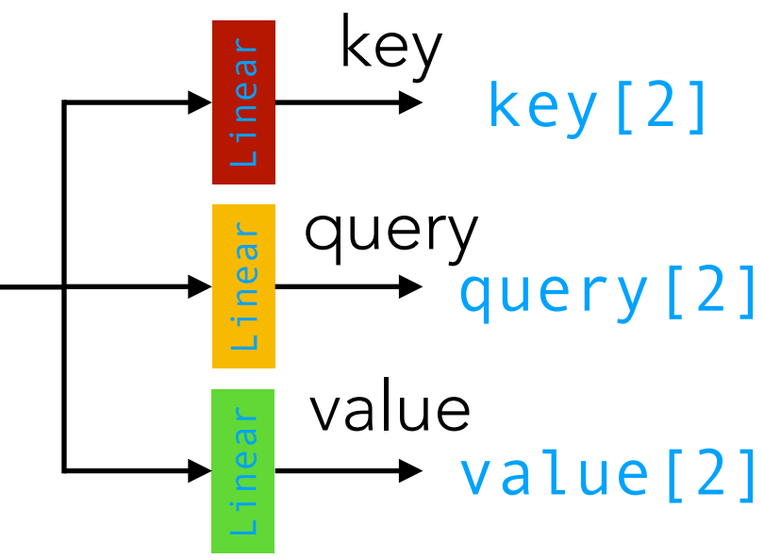
```
w_norm = softmax(w)
```

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])
```

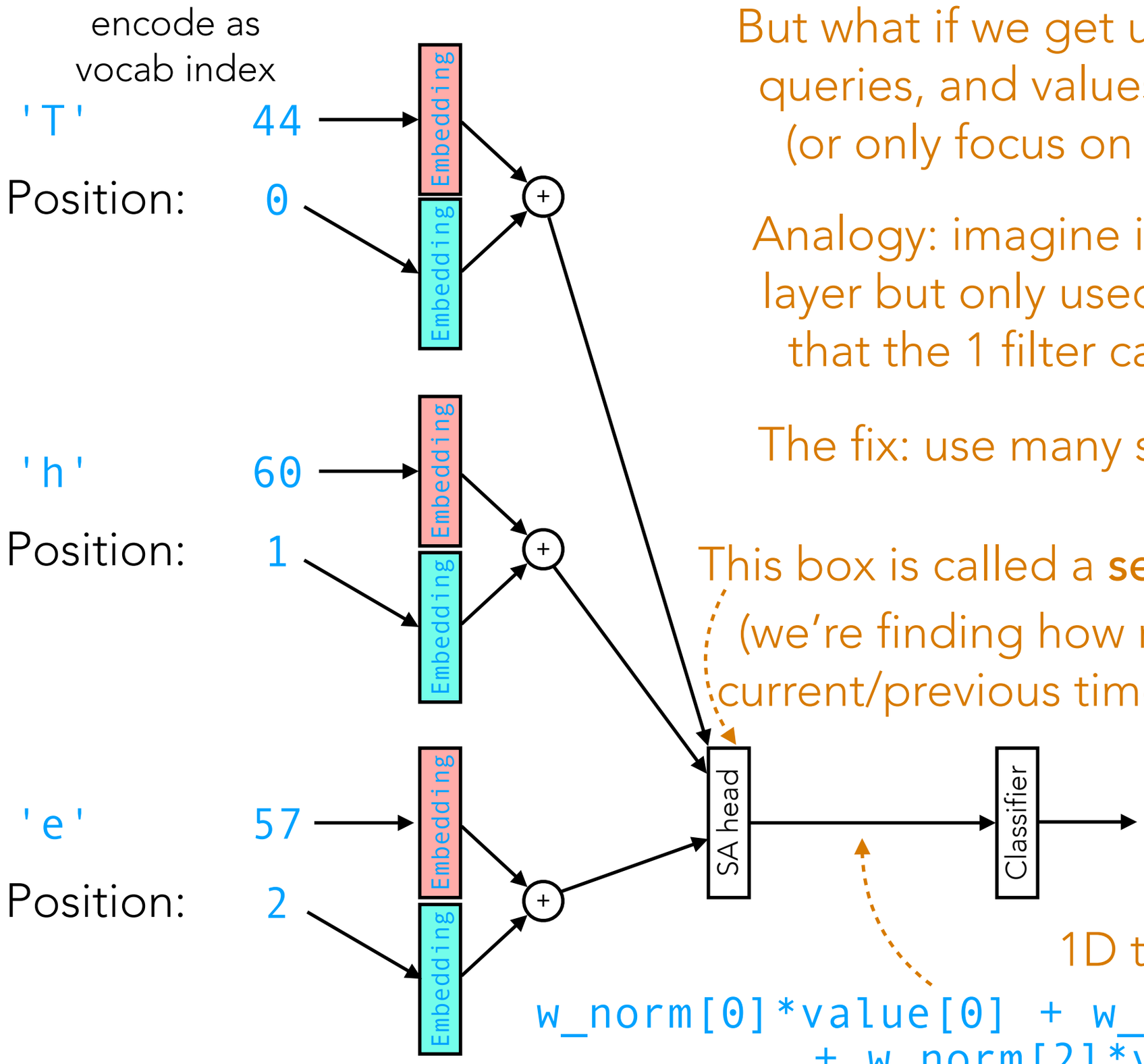
Let's normalize the weights so they are probabilities:

```
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

This box is called a **self-attention (SA) head**

(we're finding how much to pay attention to current/previous time steps of the time series)

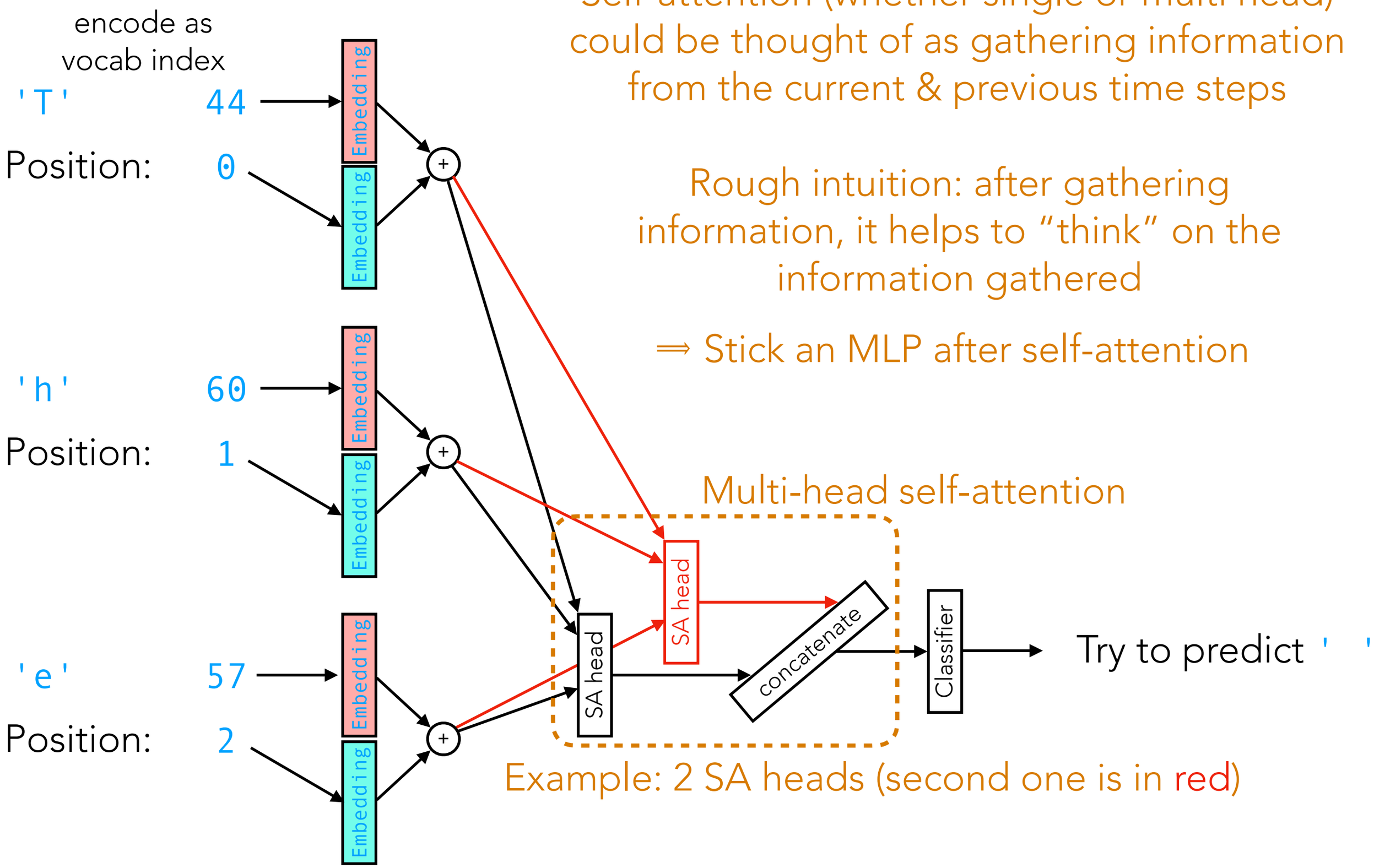
1D table with `H` entries

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

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

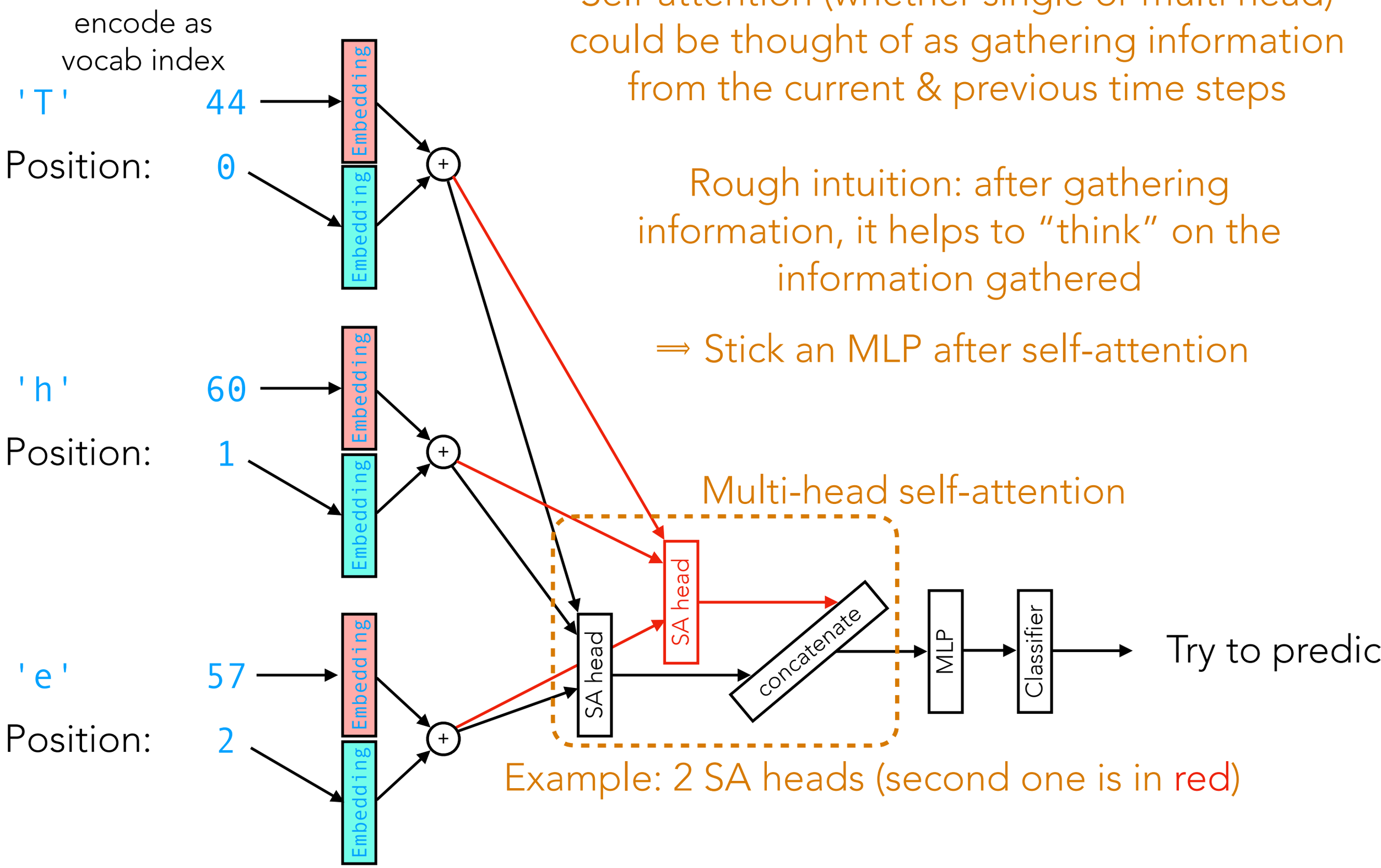
⟹ Stick an MLP after self-attention



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



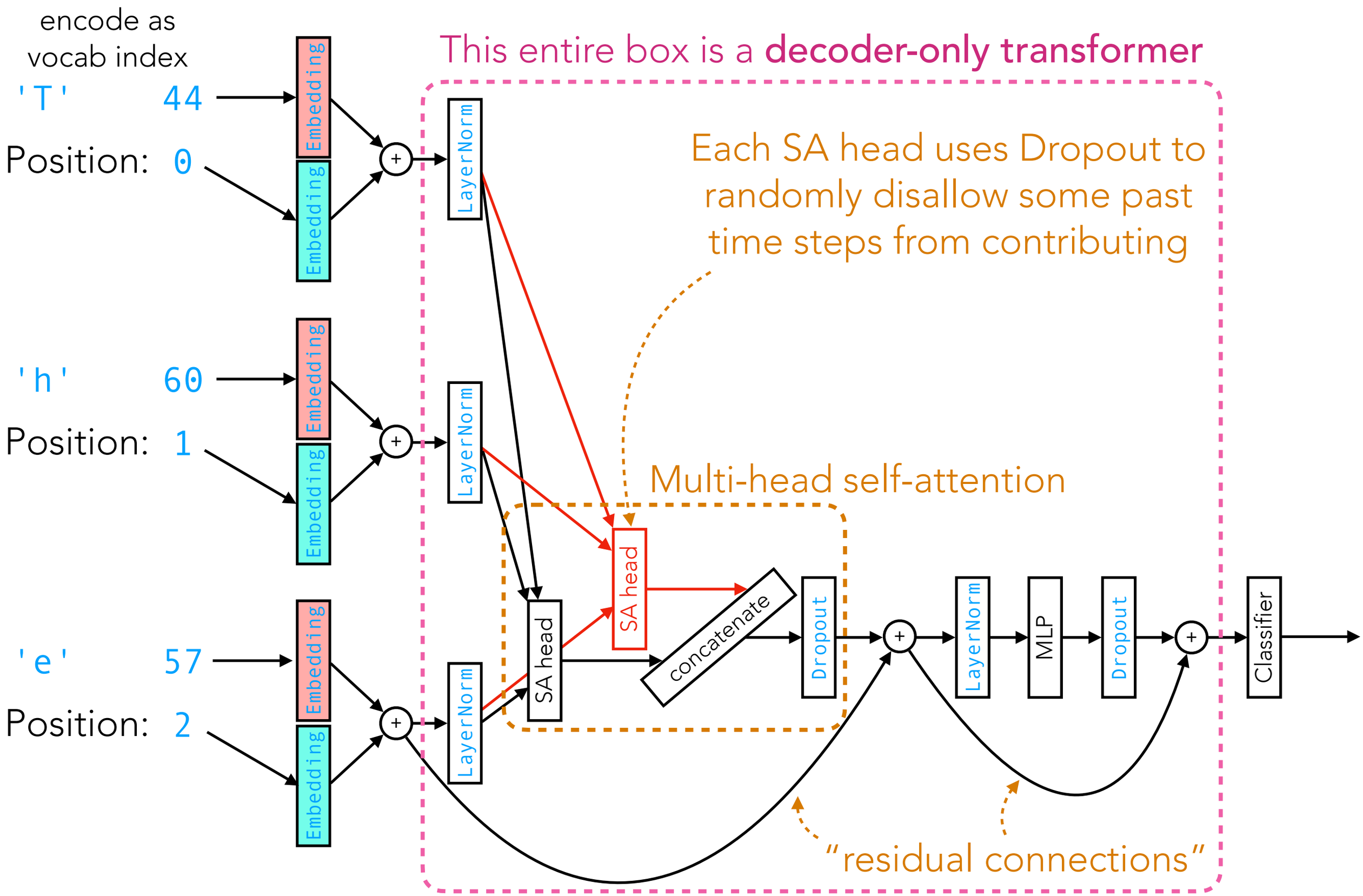
# There are a few implementation details that I won't go over in lecture

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)

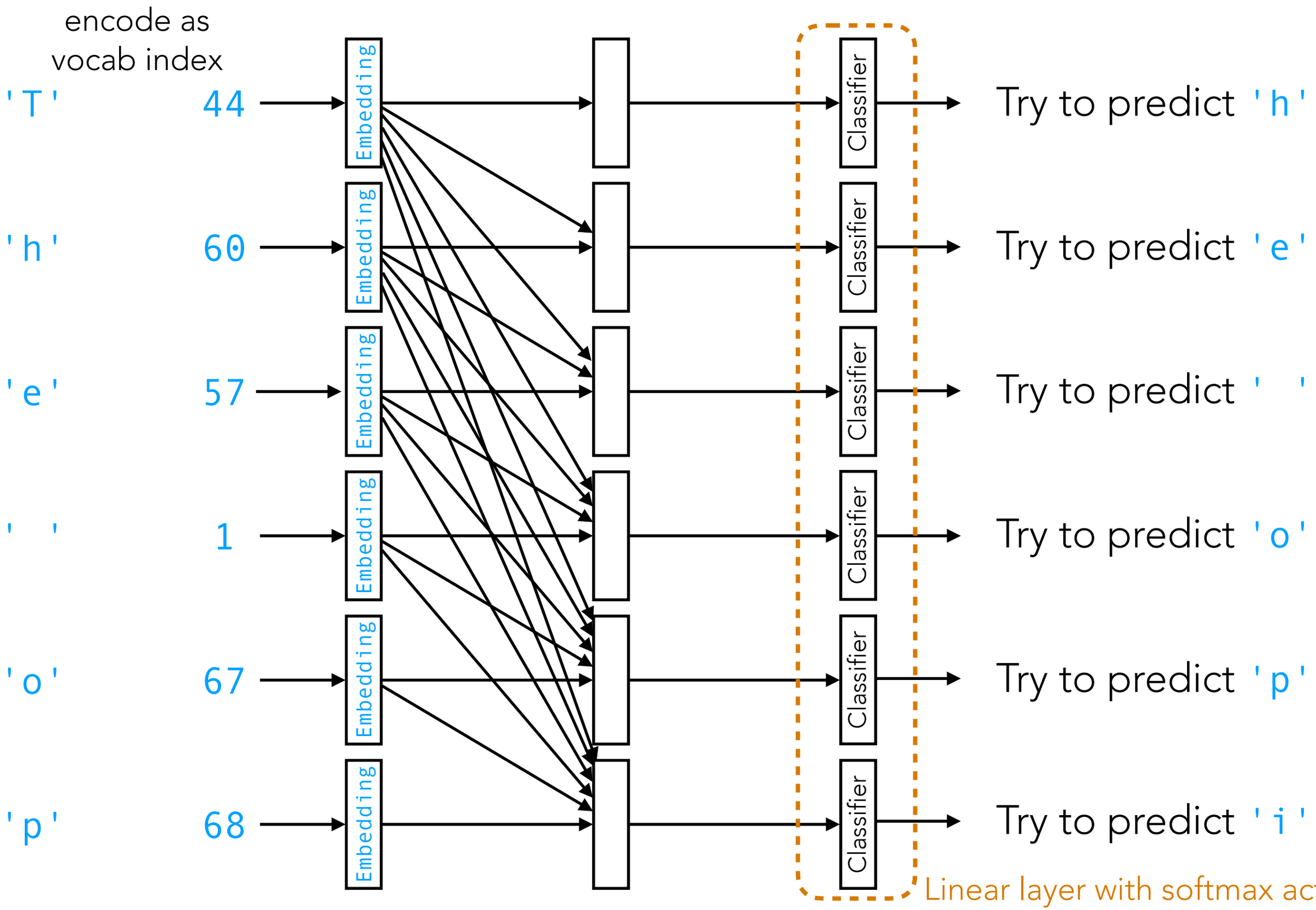
- LayerNorm
- Residual connections
- Dropout

You're not expected to know these technical details

Also, there are some standard strategies for initializing GPT training

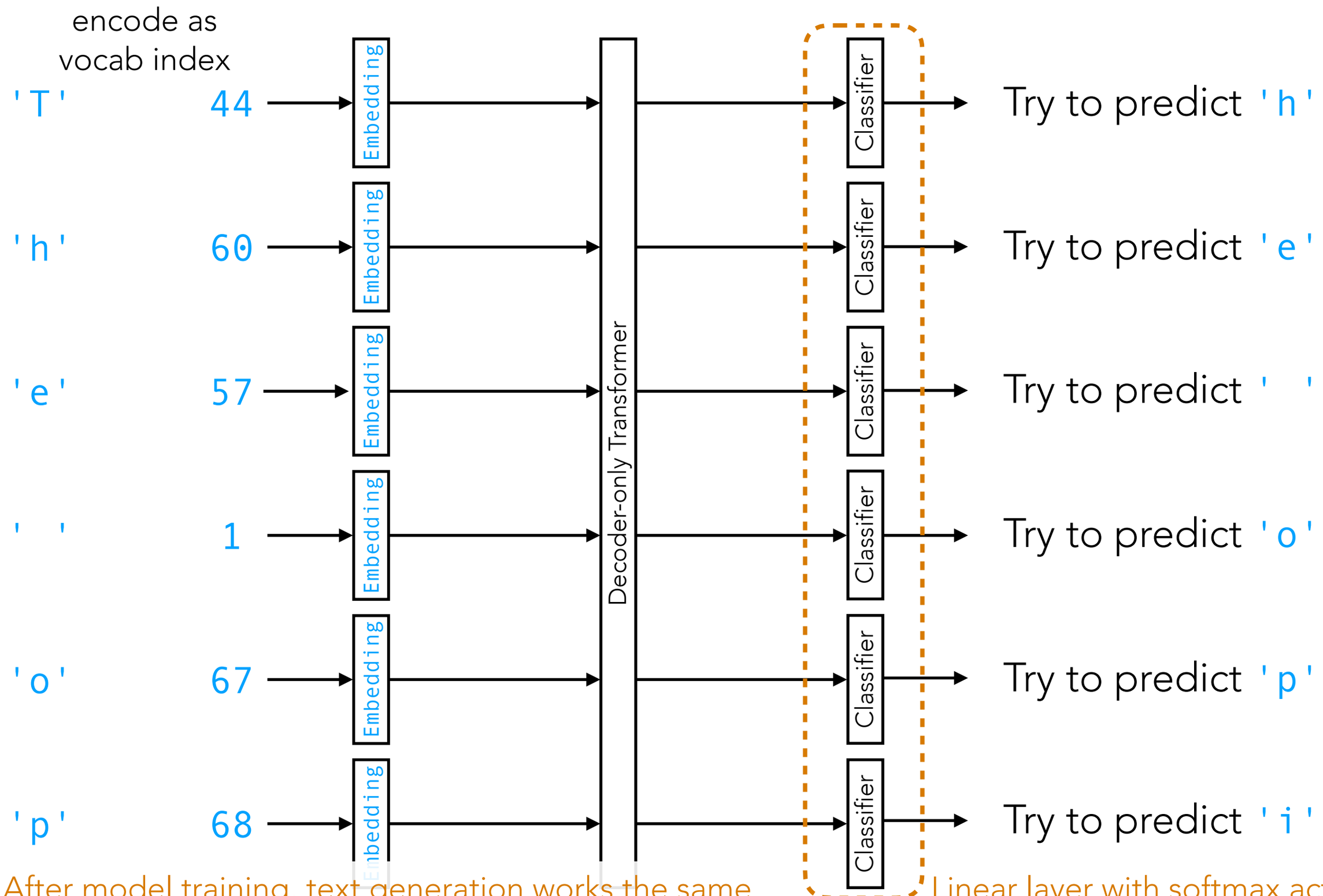

encode as
vocab index
'T' 44
Position: 0
'h' 60
Position: 1
'e' 57
Position: 2
Embedding
Embedding
Embedding
Embedding
Embedding
Embedding
This entire box is a **decoder-only transformer**
Each SA head uses Dropout to randomly disallow some past time steps from contributing
Multi-head self-attention
LayerNorm
LayerNorm
LayerNorm
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




Linear layer with softmax activation, # output nodes = vocab size

# Generative Pre-trained Transformer (GPT)




After model training, text generation works the same as I described for the RNN text generation model

Linear layer with softmax activation, # output nodes = vocab size

# GPT

Demo

# 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)

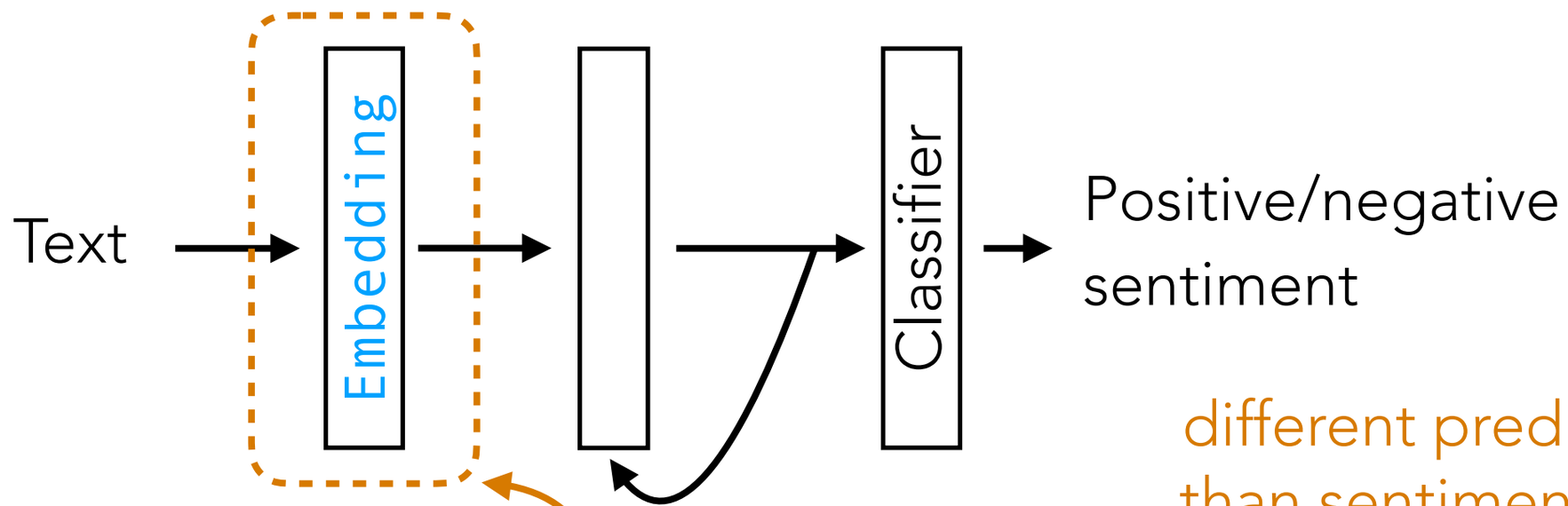
We focused on this first step today

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

# Fine-Tuning

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

**Sentiment analysis RNN demo**



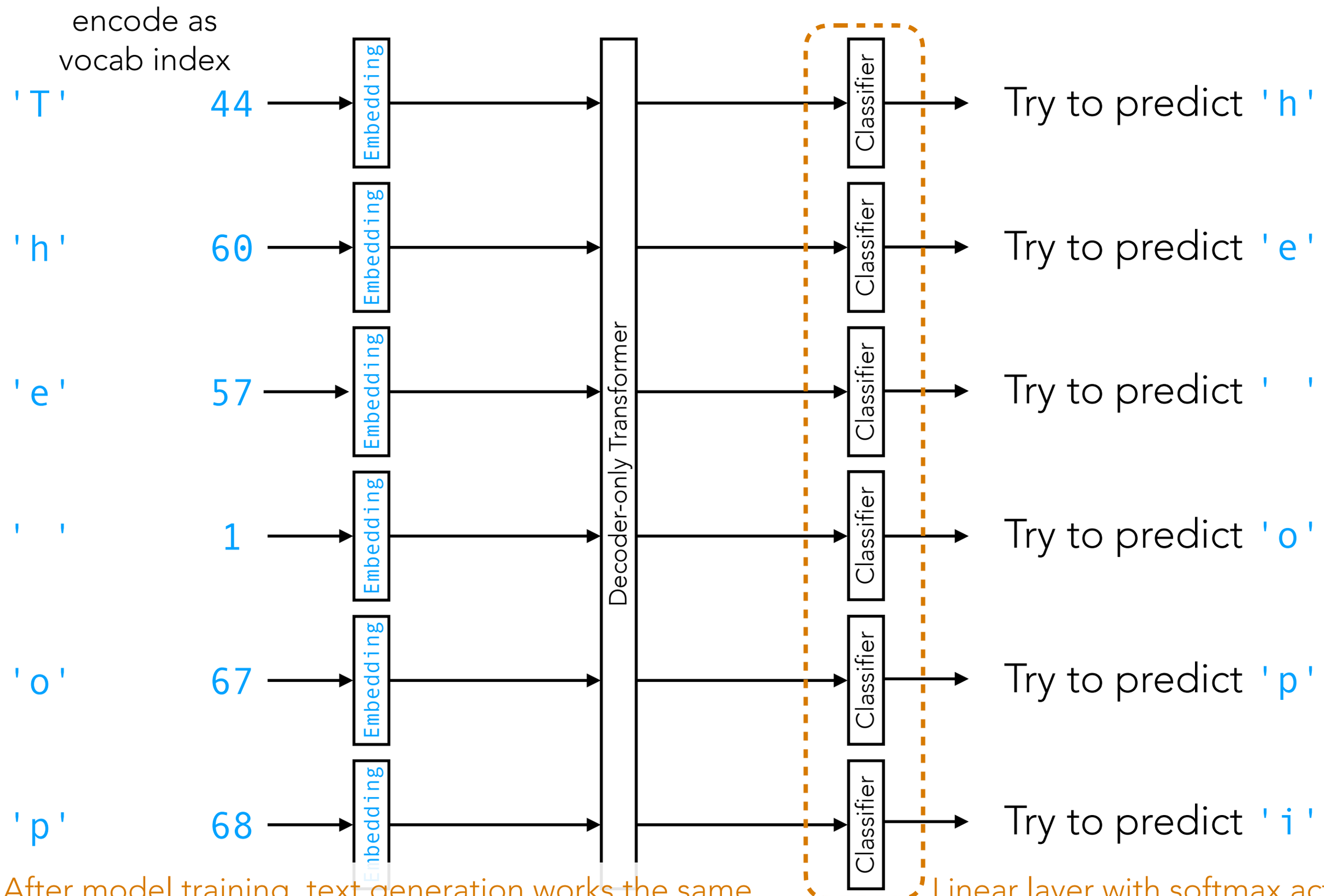

different prediction task than sentiment analysis!

We fixed the weights here to come from pre-trained GloVe word embeddings

Word embeddings are commonly trained to predict words (like our text generation setup except tokenizing based words)

We remove the classifier (for predicting a word) to get word embeddings, which we then re-purpose for a different prediction task

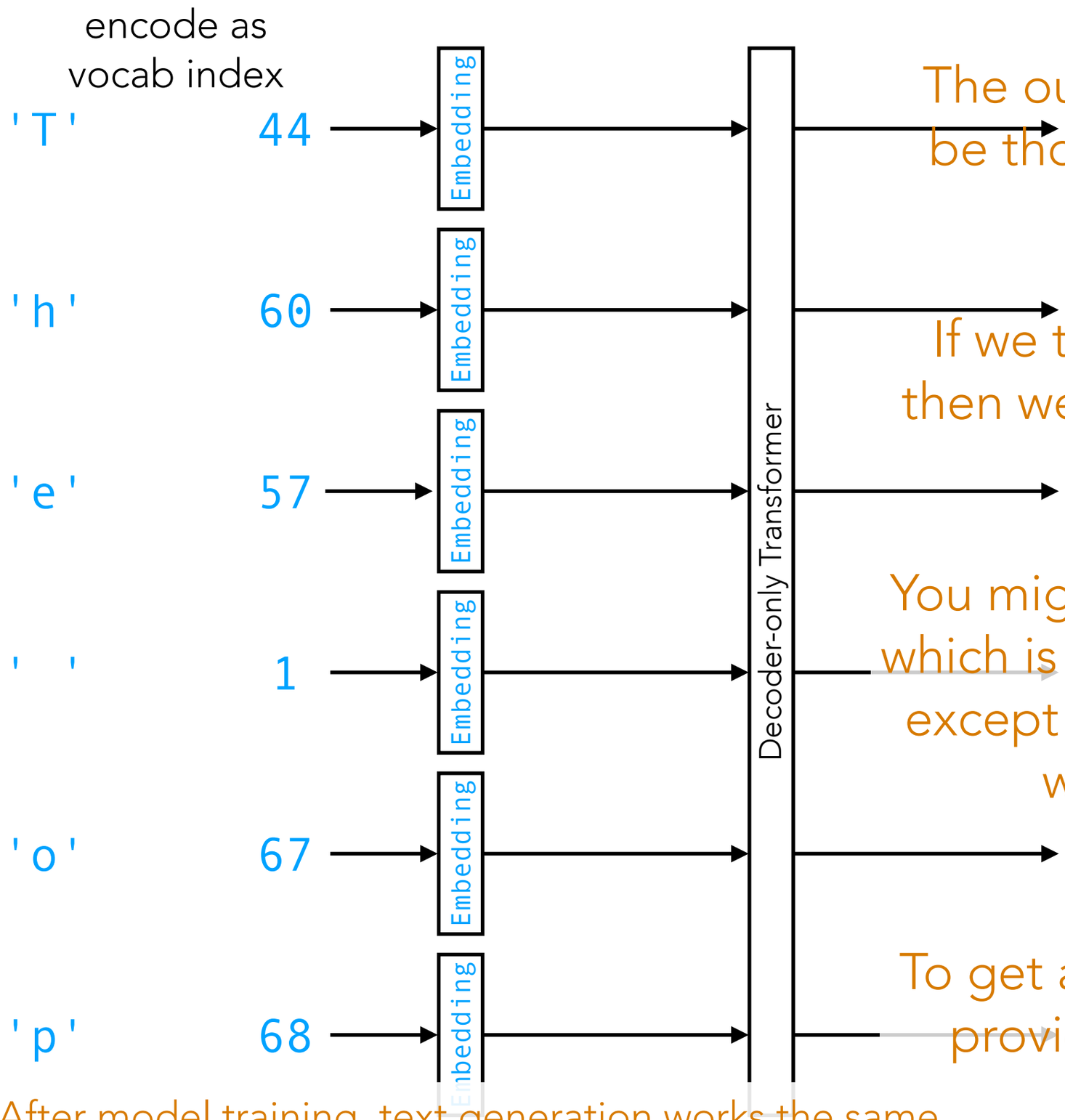
# (Flashback) Generative Pre-trained Transformer (GPT)




After model training, text generation works the same as I described for the RNN text generation model

Linear layer with softmax activation, # output nodes = vocab size

# (Flashback) Generative Pre-trained Transformer (GPT)




The output before the classifier could be thought of as a token embedding

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

You might've heard of the BERT model, which is basically what we showed today except without the causal constraint & with different tokenization

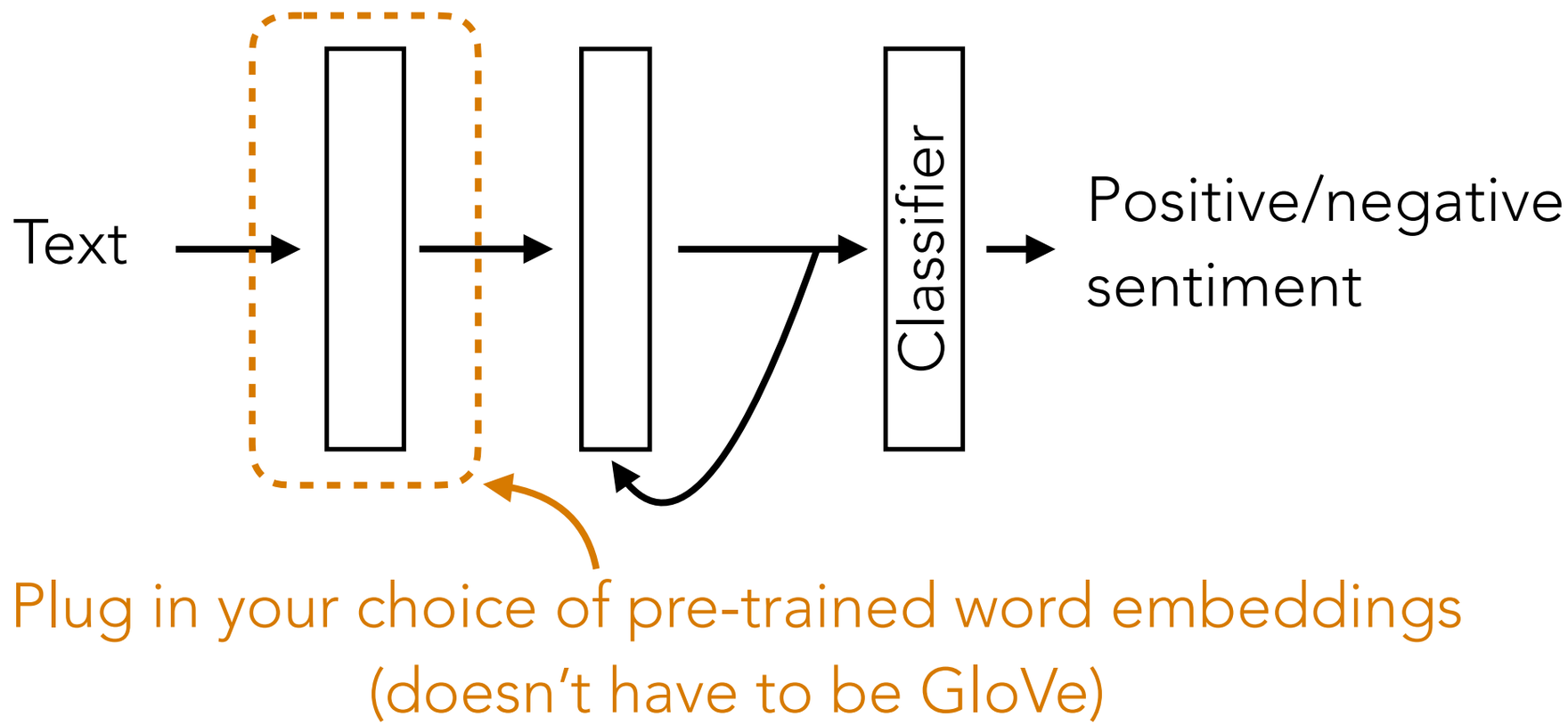
To get a word's BERT embedding, you provide the sentence in which the word is used

After model training, text generation works the same as I described for the RNN text generation model

# Fine-Tuning

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

**Sentiment analysis RNN 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

Sentiment analysis RNN demo:
GloVe vectors pre-trained on massive dataset (Wikipedia + Gigaword)
Fine tune on IMDb review dataset, which is small in comparison (25000 reviews)

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

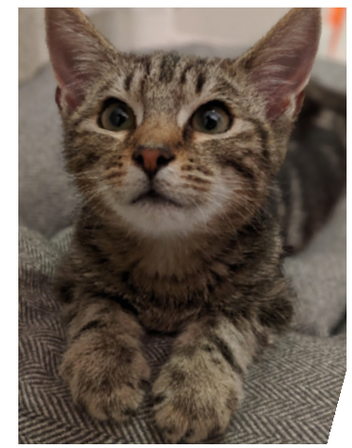
- Fine-tuning has an extremely important application: it allows us to use an existing model trained on a *massive* dataset to help us with a new prediction task where we might only have a *small* dataset
- Another extremely important strategy: **data augmentation (randomly perturb training data to get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

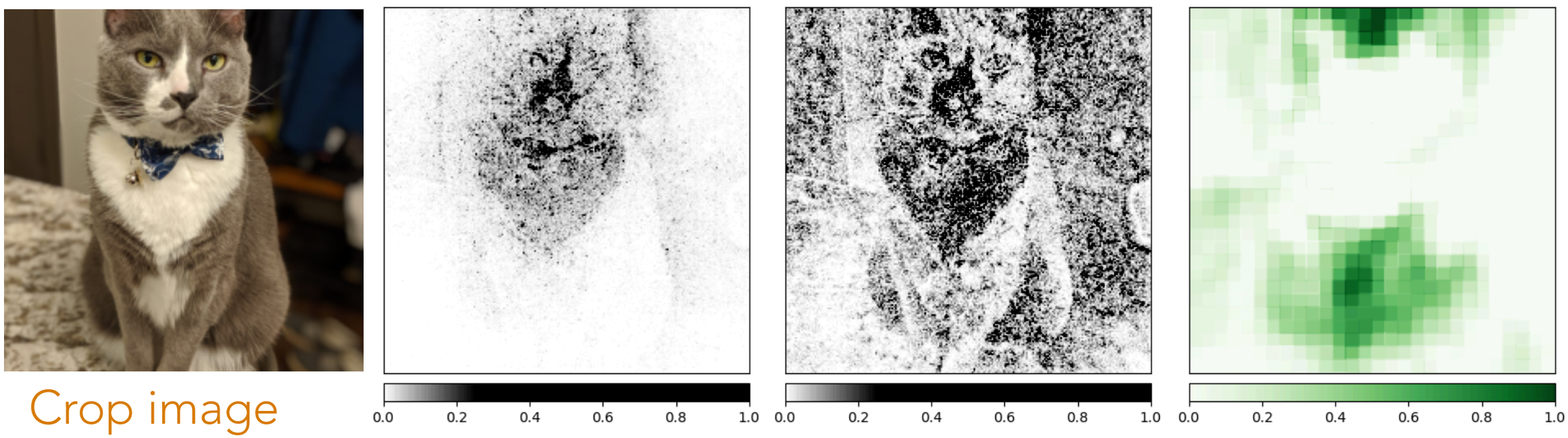
(e.g., for handwritten digits, rotating a 6 or 9 by 180 degrees would be bad)

# Interpretability/Explainability: Current State of Affairs

- There are lots of "explanation" approaches that can be used after learning 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 Analysis

Question → Data → Finding Structure → Insights

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

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

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

Th … ?
by a practitioner? … have to collect … r
more evidence! … analysis … *catchable?*

Answer original question

There isn't always a follow-up prediction problem to solve!

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 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!*