NYU CS-GY 6923 Machine Learning

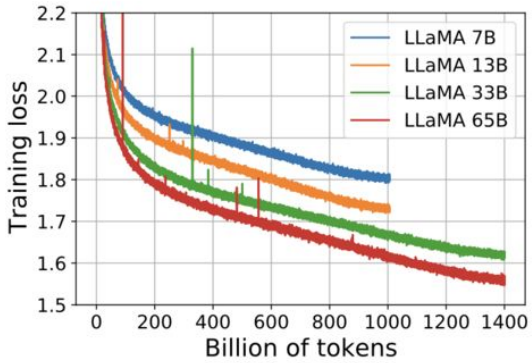
Prof. Pavel Izmailov

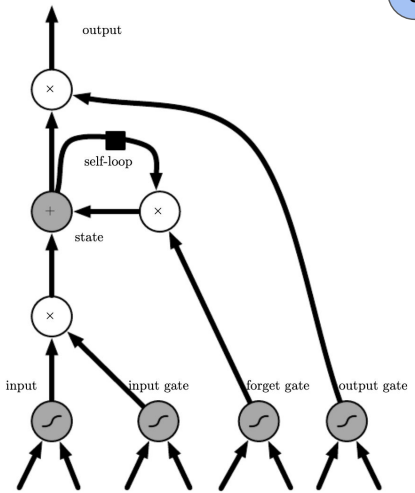
Logistics

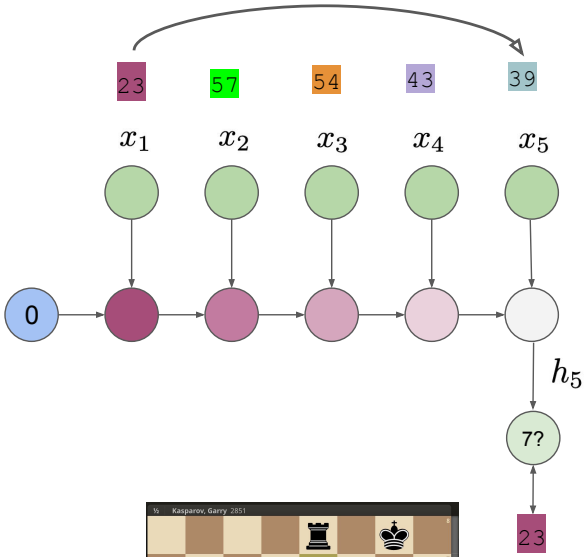
No lecture on thanksgiving week (Nov 26 or 28)!

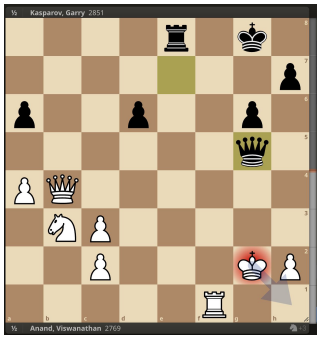
Today

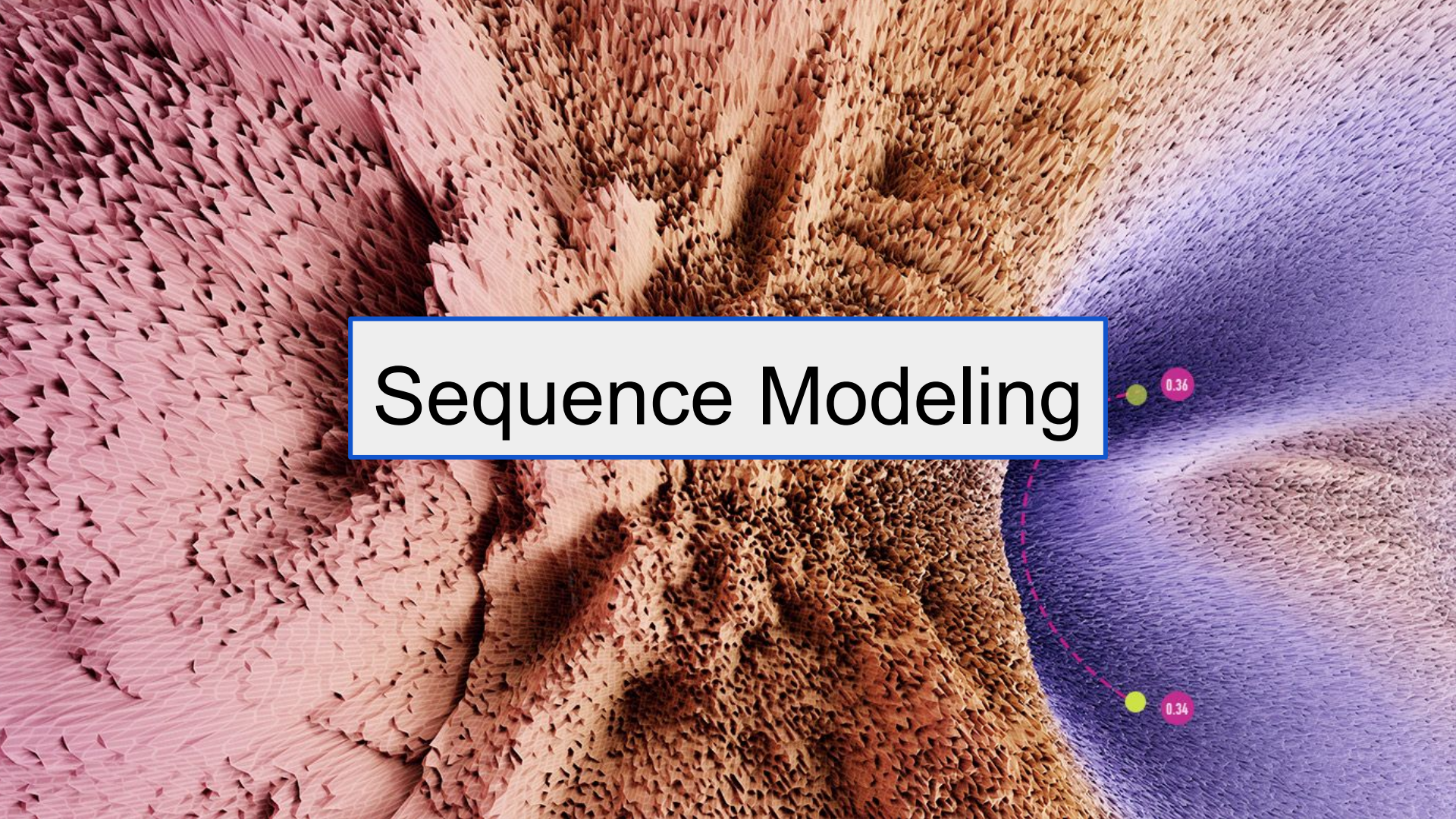
- Sequence modeling
- Recurrent neural networks
- RNN demo
- RNN extensions
- Language model pretraining



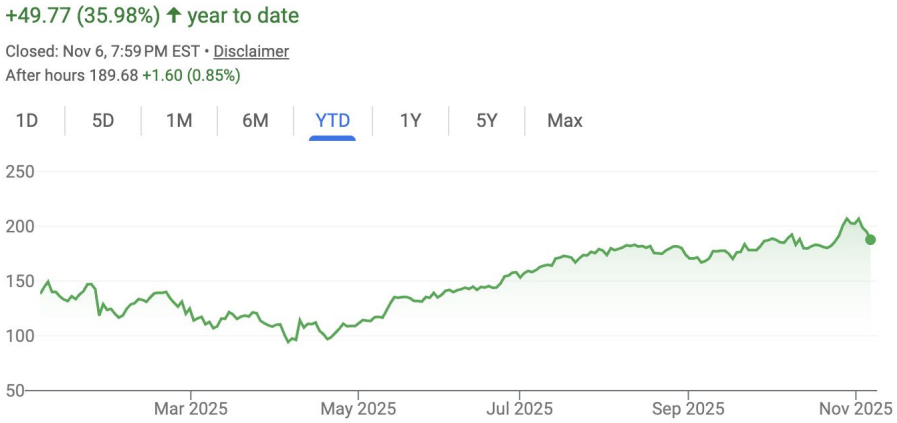


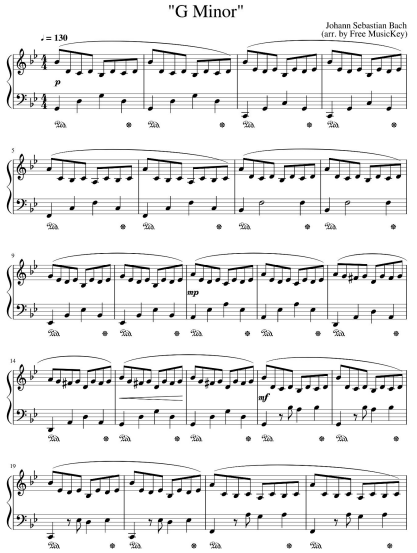


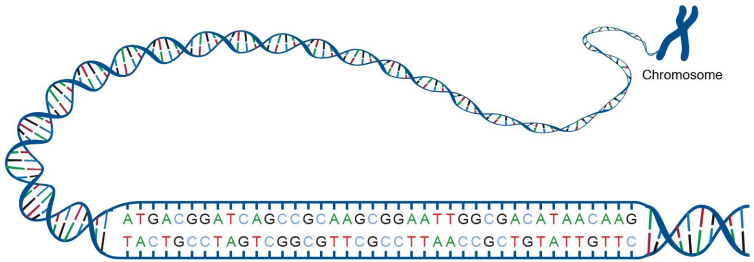



Sequence Modeling
0.36
0.34

Sequence Modeling

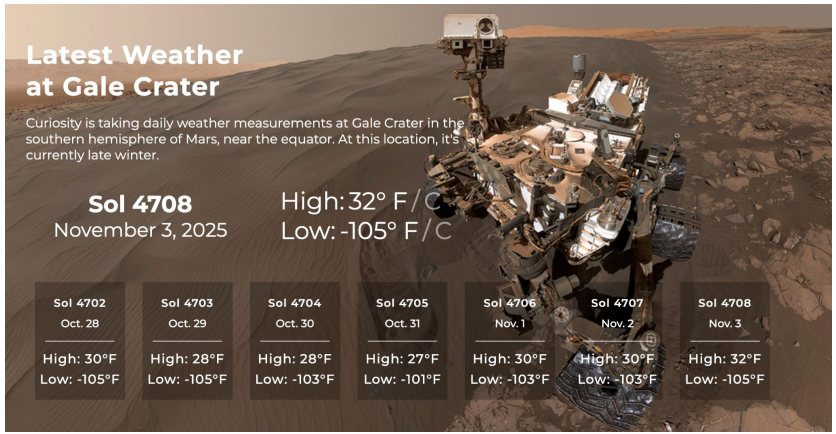






A lot of data in the world can be naturally represented as sequences

Sequence Modeling



master linux / kernel / async.c ↑ Top

Code Blame Raw

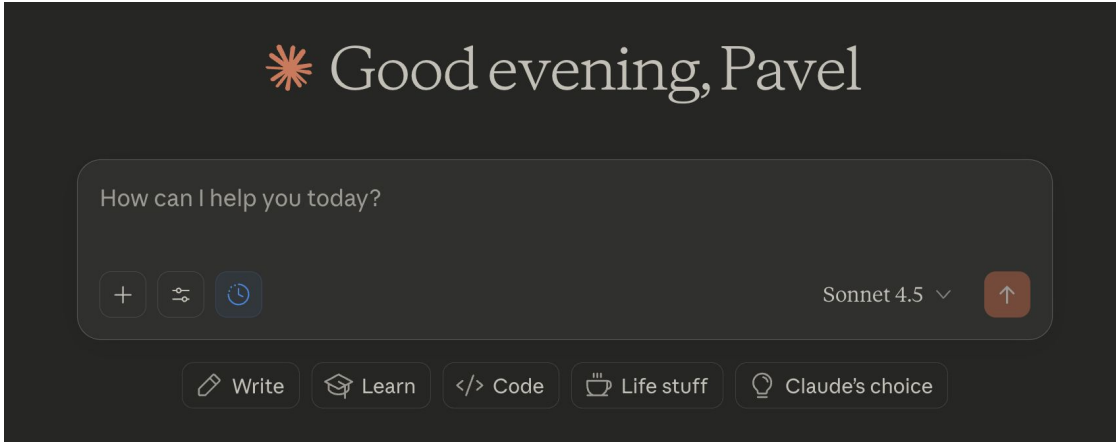
```
82
83  static long long microseconds_since(ktime_t start)
84  {
85          ktime_t now = ktime_get();
86          return ktime_to_ns(ktime_sub(now, start)) >> 10;
87  }
88
89  static async_cookie_t lowest_in_progress(struct async_domain *domain)
90  {
91          struct async_entry *first = NULL;
92          async_cookie_t ret = ASYNC_COOKIE_MAX;
93          unsigned long flags;
94
95          spin_lock_irqsave(&async_lock, flags);
96
97          if (domain) {
98                  if (!list_empty(&domain->pending))
99                          first = list_first_entry(&domain->pending,
100                                                   struct async_entry, domain_list);
101         } else {
102                 if (!list_empty(&async_global_pending))
103                         first = list_first_entry(&async_global_pending,
104                                                  struct async_entry, global_list);
105         }
106
107         if (first)
108                 ret = first->cookie;
109
110         spin_unlock_irqrestore(&async_lock, flags);
111         return ret;
112 }
113
114 /*
115  * pick the first pending entry and run it
116  */
117 static void async_run_entry_fn(struct work_struct *work)
118 {
119         struct async_entry *entry =
120                 container_of(work, struct async_entry, work);
121         unsigned long flags;
122         ktime_t calltime;
123
124         /* 1) run (and print duration) */
```

Sequence Modeling

Even data that doesn’t immediately look like a sequence…

In a computer, everything is a sequence of bits!

Sequence Modeling



Over the next several lectures we will build a basic understanding of how systems like Claude and ChatGPT work.

Data

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to
famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to
the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own
price.
Is't a verdict?
```

Our data will be text. We need to represent it as a sequence of numbers.

Tokenization

```
First Citizen:
Before we proceed any further, hear
me speak.
…
```

Idea: give each symbol a unique number

Tokenization

```
First Citizen:
Before we proceed any further, hear
me speak.
…
```

Idea: give each symbol a unique number

'\n': 0,	'A': 13,	'a': 39,
' ': 1,	'B': 14,	'b': 40,
'!': 2,	'C': 15,	'c': 41,
'$': 3,	'D': 16,	'd': 42,
'&': 4,	'E': 17,	…
…	...	'z': 64

Dictionary

Tokenization

```
First Citizen:
Before we proceed any further, hear
me speak.
…
```

```
18475657581154758476443521011

14434453564316143154565341434342139526314459565846435661464339561

514315754433949 8
…
```

Idea: give each symbol a unique number

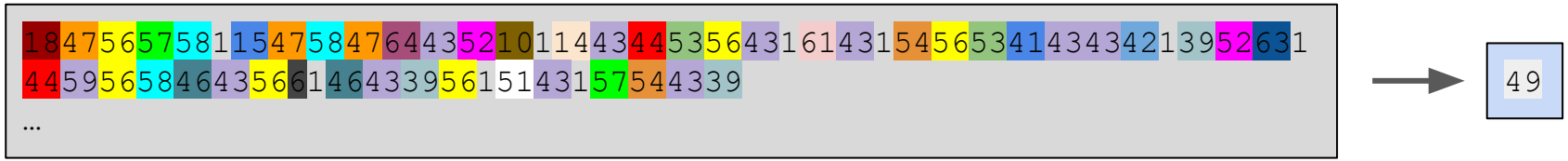
'\n': 0,	'A': 13,	'a': 39,
' ': 1,	'B': 14,	'b': 40,
'!': 2,	'C': 15,	'c': 41,
'$': 3,	'D': 16,	'd': 42,
'&': 4,	'E': 17,	…
…	…	'z': 64

Dictionary

We now converted our string to a list of *tokens*

Next token prediction

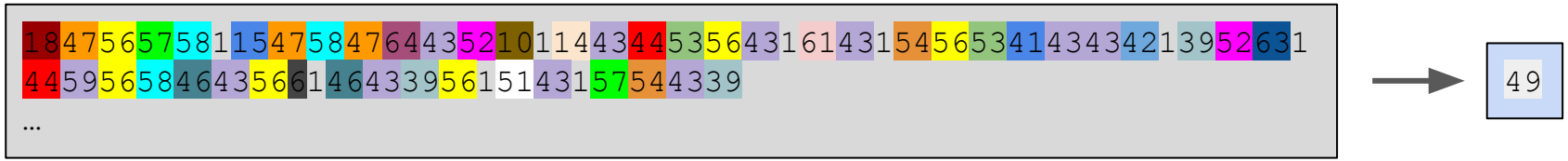
Goal: given previous tokens, predict the next token



- Next token prediction is the objective of (large) language model pretraining
- It may seem trivial, but it is extremely general

Next token prediction

Goal: given previous tokens, predict the next token

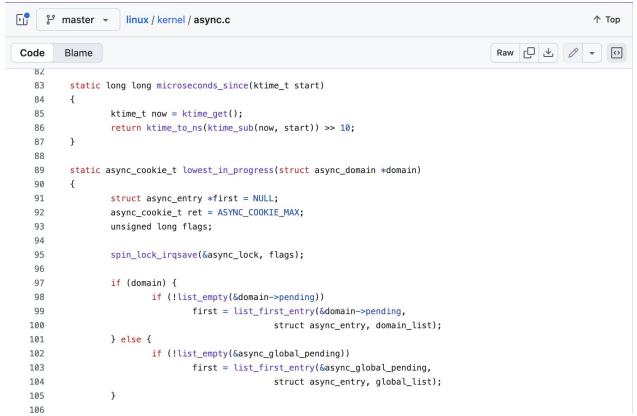


- Next token prediction is the objective of (large) language model pretraining
- It may seem trivial, but it is extremely general

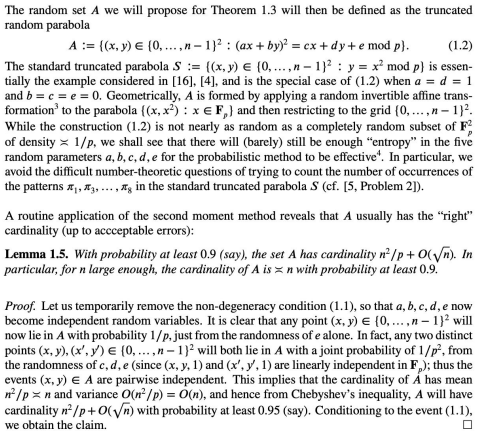
Next token prediction

Good next token prediction requires intelligence

- Predicting the next move in chess
- … the next line of code in the linux kernel
- … the next line of the proof of a math theorem




```
82
83  static long long microseconds_since(ktime_t start)
84  {
85          ktime_t now = ktime_get();
86          return ktime_to_ns(ktime_sub(now, start)) >> 10;
87  }
88
89  static async_cookie_t lowest_in_progress(struct async_domain *domain)
90  {
91          struct async_entry *first = NULL;
92          async_cookie_t ret = ASYNC_COOKIE_MAX;
93          unsigned long flags;
94
95          spin_lock_irqsave(&async_lock, flags);
96
97          if (domain) {
98                  if (!list_empty(&domain->pending))
99                          first = list_first_entry(&domain->pending,
100                                                   struct async_entry, domain_list);
101         } else {
102                 if (!list_empty(&async_global_pending))
103                         first = list_first_entry(&async_global_pending,
104                                                  struct async_entry, global_list);
105         }
106
```



4 TERENCE TAO

The random set A we will propose for Theorem 1.3 will then be defined as the truncated random parabola

$$A := \{(x,y) \in \{0,\dots,n-1\}^2 : (ax+by)^2 = cx+dy+e \bmod p\}. \tag{1.2}$$

The standard truncated parabola $S := \{(x,y) \in \{0,\dots,n-1\}^2 : y = x^2 \bmod p\}$ is essentially the example considered in [16], [4], and is the special case of (1.2) when $a = d = 1$ and $b = c = e = 0$. Geometrically, A is formed by applying a random invertible affine transformation[3] to the parabola $\{(x,x^2) : x \in \mathbf{F}_p\}$ and then restricting to the grid $\{0,\dots,n-1\}^2$. While the construction (1.2) is not nearly as random as a completely random subset of $\mathbf{F}_p^2$ of density $\asymp 1/p$, we shall see that there will (barely) still be enough "entropy" in the five random parameters a,b,c,d,e for the probabilistic method to be effective[4]. In particular, we avoid the difficult number-theoretic questions of trying to count the number of occurrences of the patterns $\pi_1,\pi_{31},\dots,\pi_8$ in the standard truncated parabola S (cf. [5, Problem 2]).

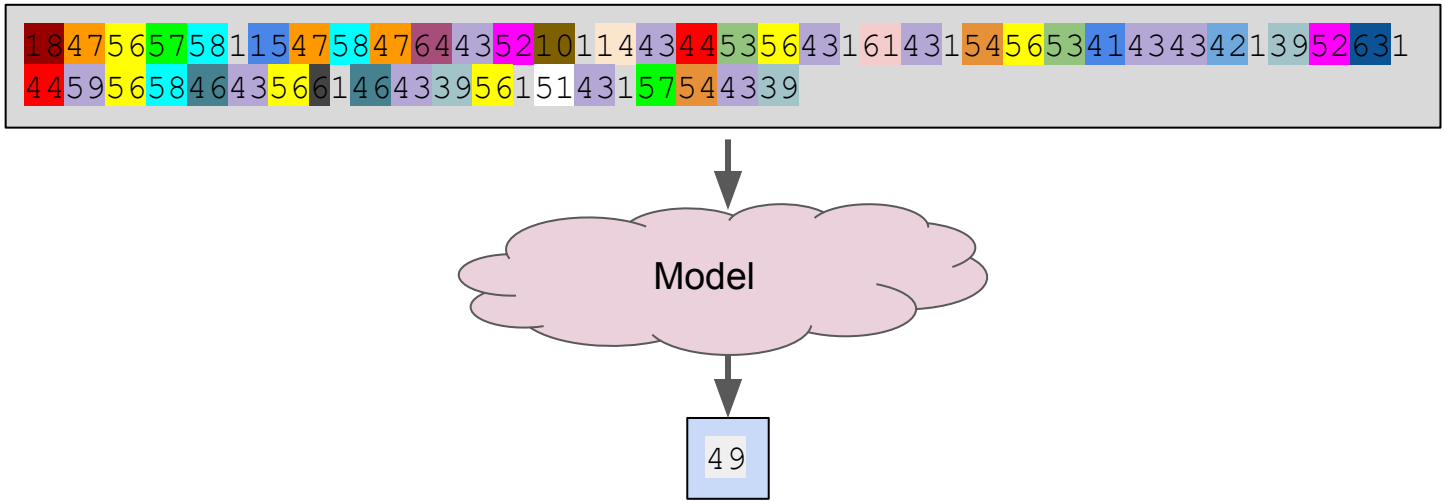
A routine application of the second moment method reveals that A usually has the "right" cardinality (up to accceptable errors):

Lemma 1.5. *With probability at least* 0.9 *(say), the set A has cardinality $n^2/p + O(\sqrt{n})$. In particular, for n large enough, the cardinality of A is $\asymp n$ with probability at least* 0.9.

Proof. Let us temporarily remove the non-degeneracy condition (1.1), so that a,b,c,d,e now become independent random variables. It is clear that any point $(x,y) \in \{0,\dots,n-1\}^2$ will now lie in A with probability $1/p$, just from the randomness of e alone. In fact, any two distinct points $(x,y),(x',y') \in \{0,\dots,n-1\}^2$ will both lie in A with a joint probability of $1/p^2$, from the randomness of c,d,e (since $(x,y,1)$ and $(x',y',1)$ are linearly independent in $\mathbf{F}_p$); thus the events $(x,y) \in A$ are pairwise independent. This implies that the cardinality of A has mean $n^2/p \asymp n$ and variance $O(n^2/p) = O(n)$, and hence from Chebyshev's inequality, A will have cardinality $n^2/p + O(\sqrt{n})$ with probability at least 0.95 (say). Conditioning to the event (1.1), we obtain the claim. □

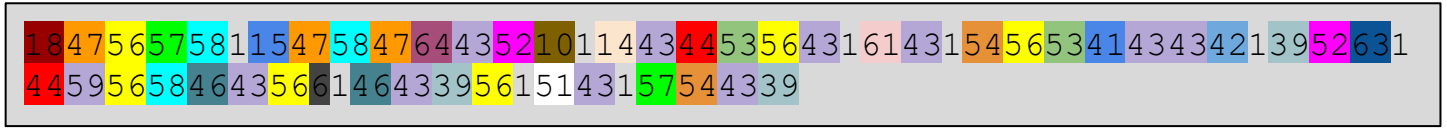
Sequence Modeling

How do we get a model to do next token prediction?



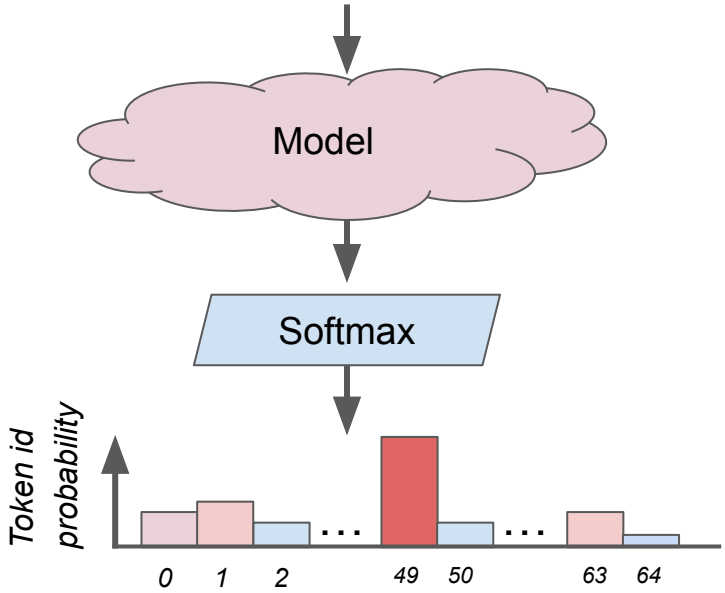
Sequence Modeling

How do we get a model to do next token prediction?

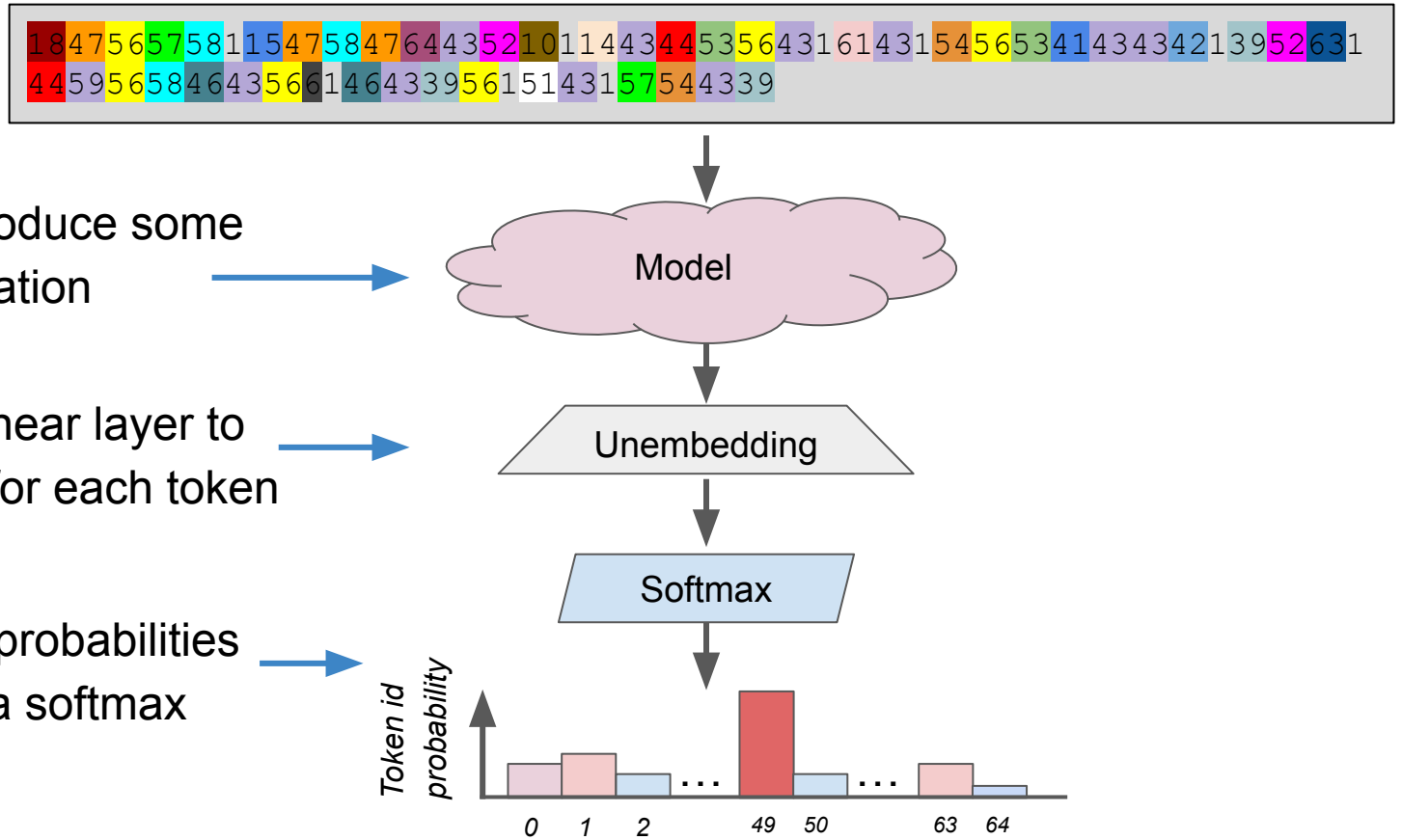


The prediction is a classification problem over the dictionary

'\n': 0,	'A': 13,	'a': 39,
' ': 1,	'B': 14,	'b': 40,
'!': 2,	'C': 15,	'c': 41,
'$': 3,	'D': 16,	'd': 42,
'&': 4,	'E': 17,	…
…	...	'z': 64



Sequence Modeling

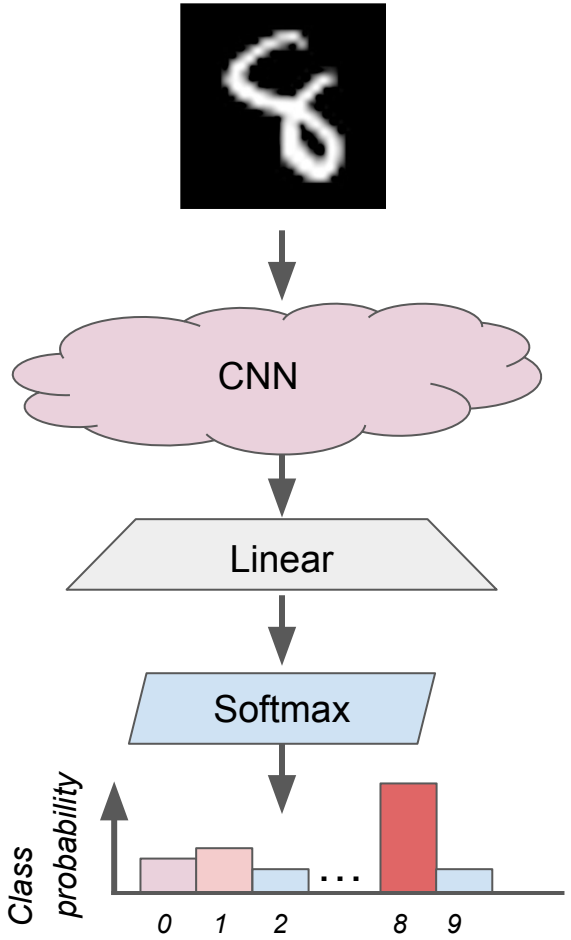


Compare to CNN

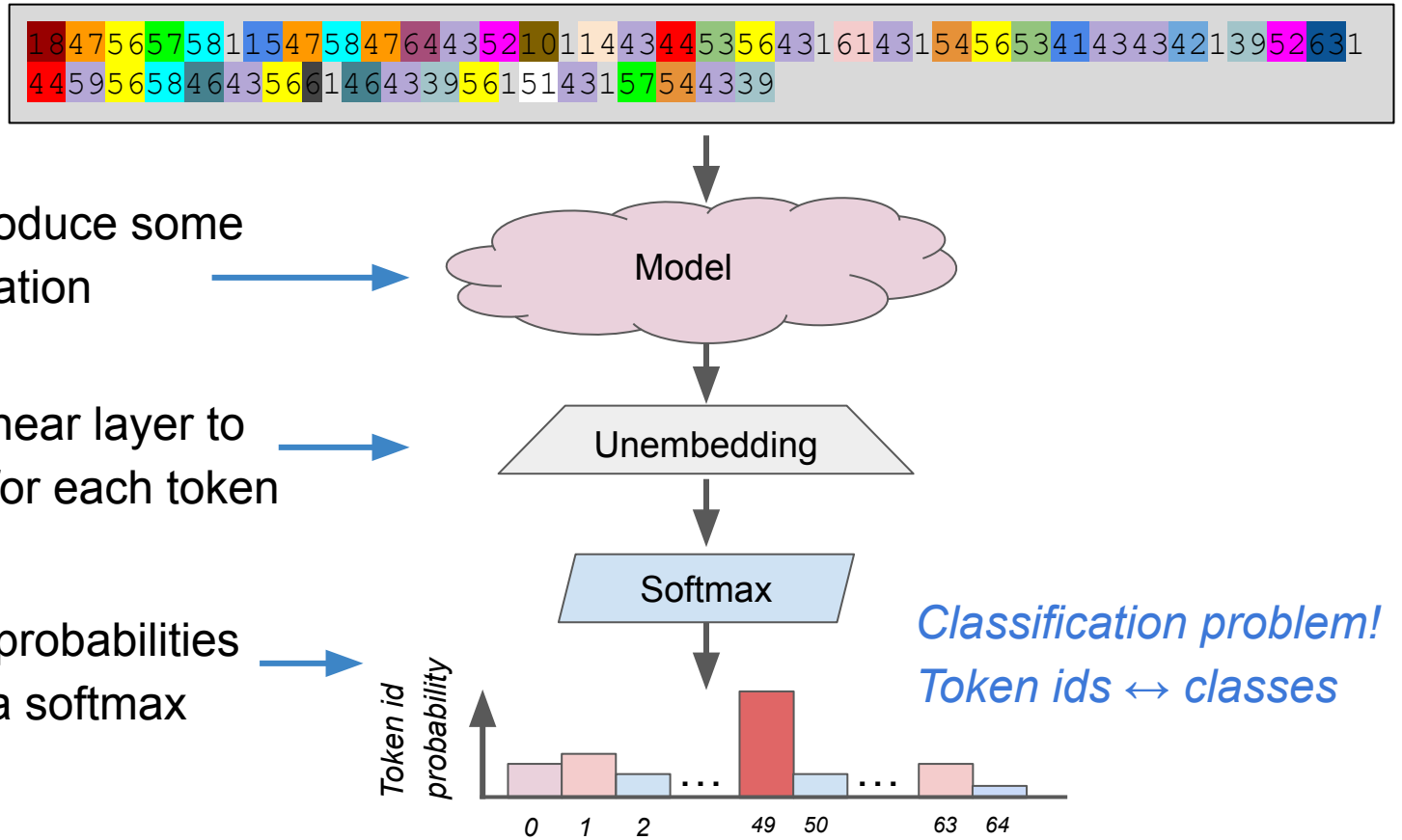
Convolutional layers produce a feature representation

Apply a linear layer to map it to scores for each *class*

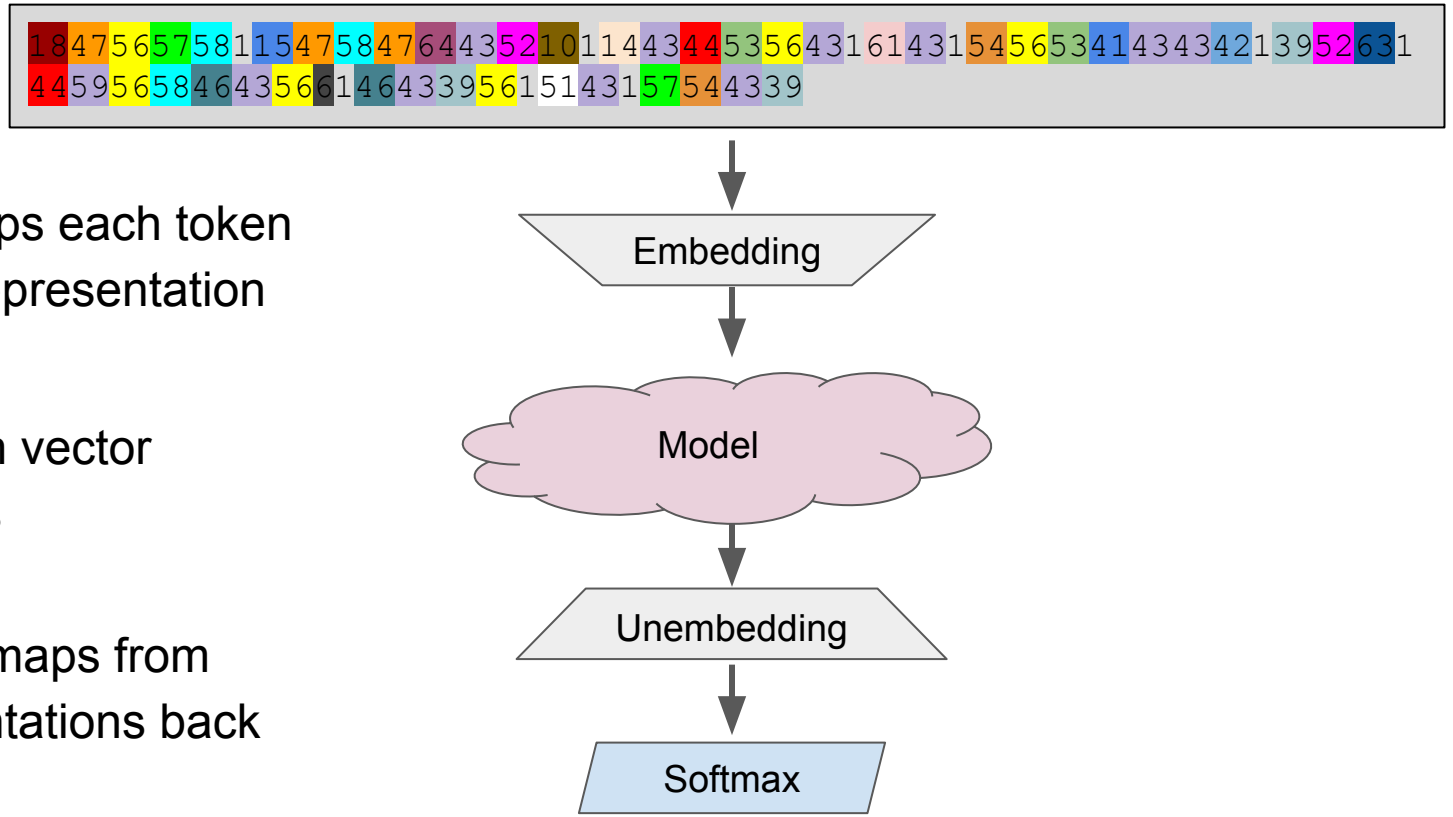
Finally, compute probabilities for each *class* via softmax



Sequence Modeling



Sequence Modeling

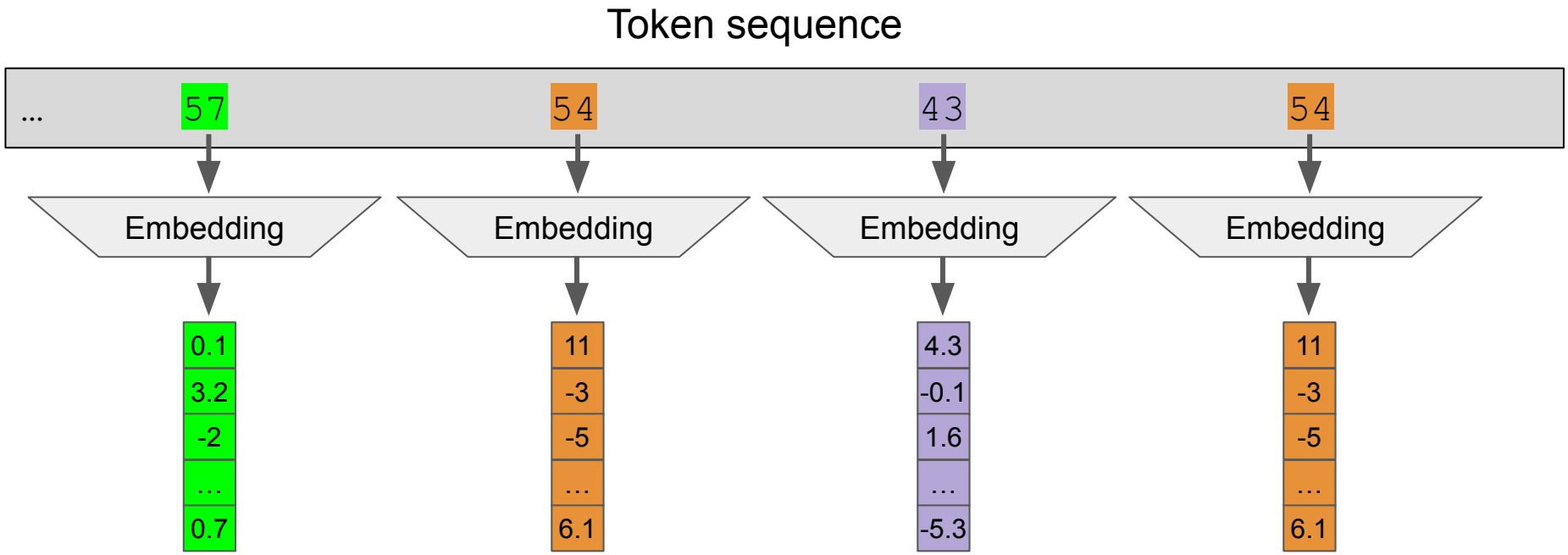


Embedding maps each token id to a vector representation

Model works on vector representations

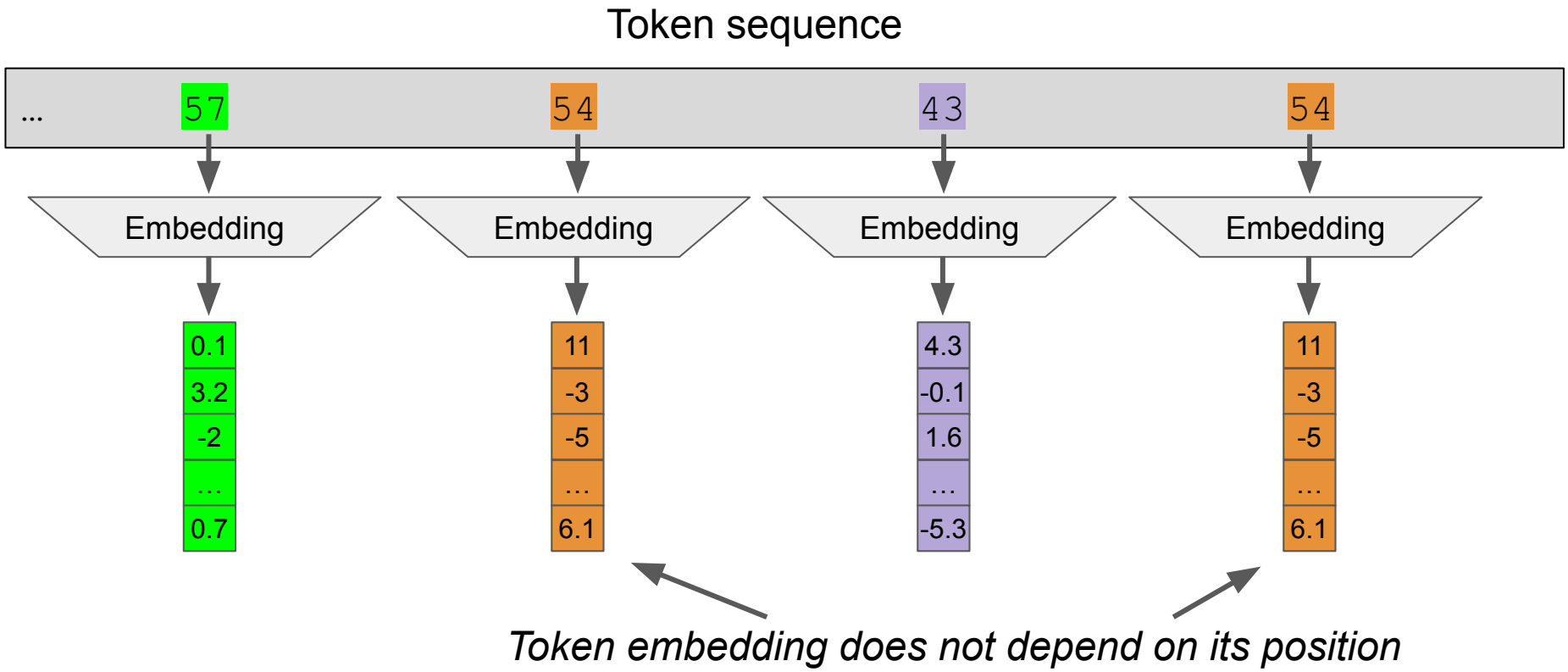
Unembedding maps from vector representations back to token ids

Embeddings



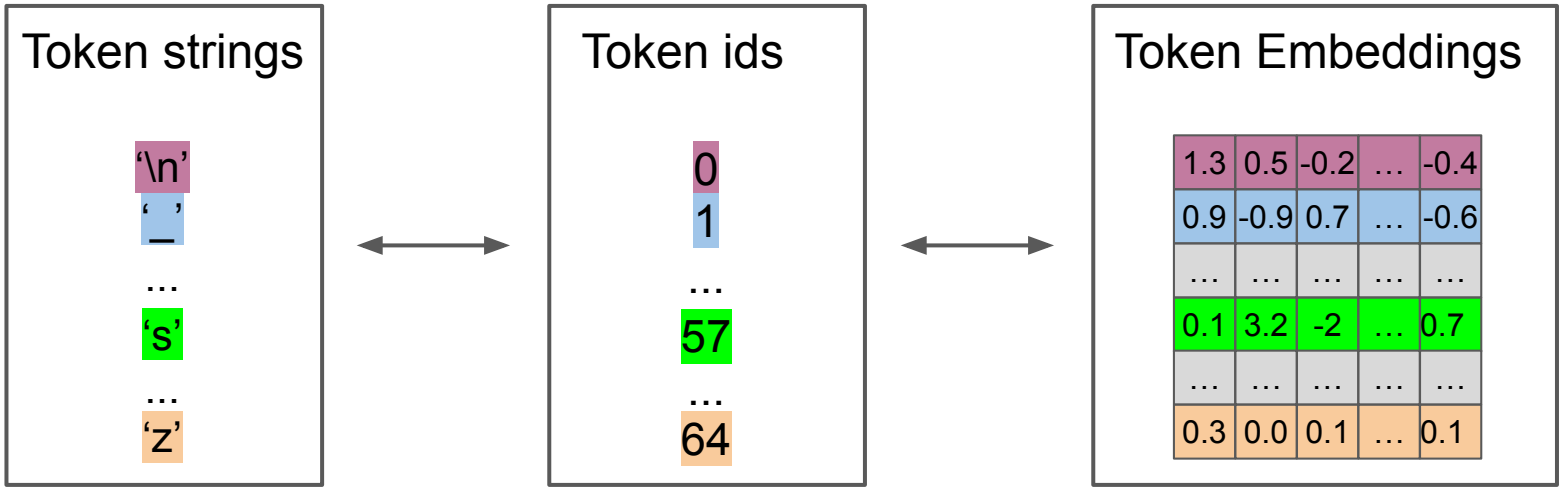
Each token id gets converted to a d-dimensional vector

Embeddings



Each token id gets converted to a d-dimensional vector

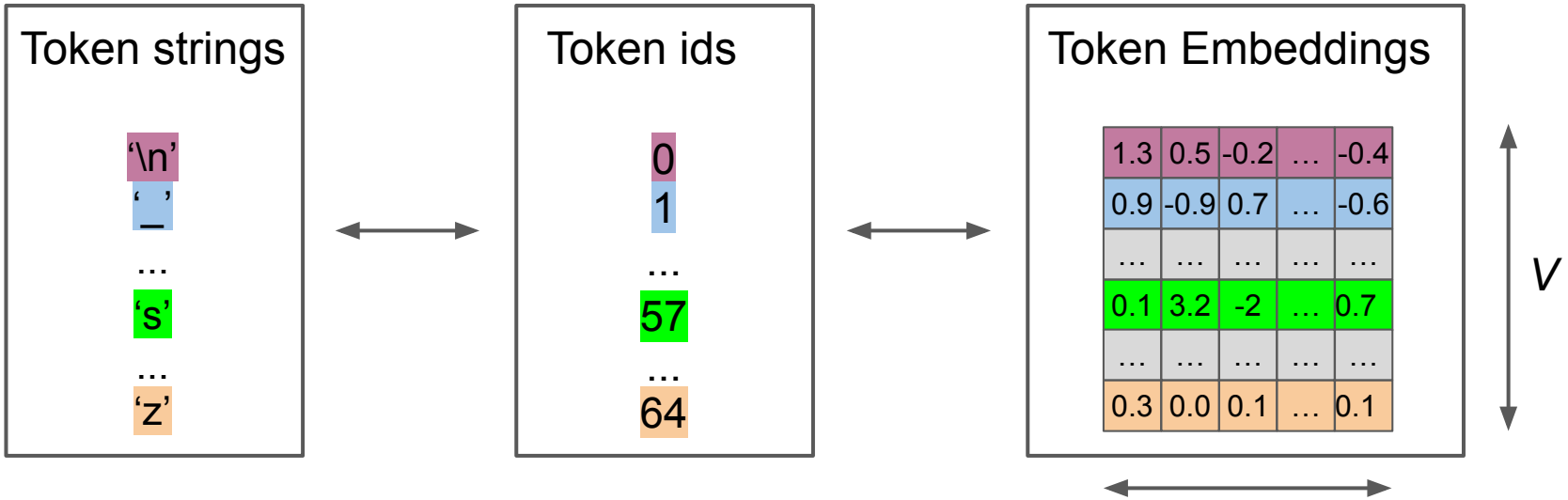
Embedding Layer



The embeddings are learnable d-dimensional vectors

- How many total parameters in the embedding layer?

Embedding Layer

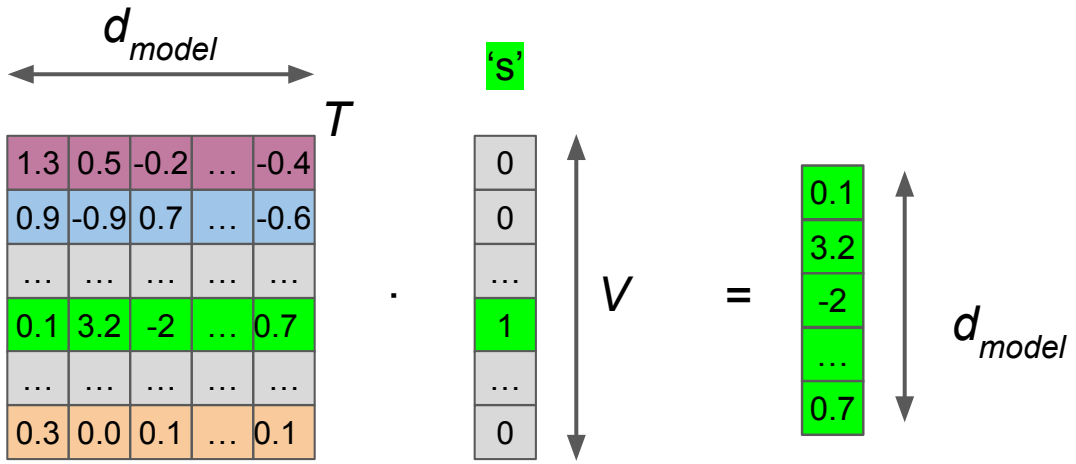


The embeddings are learnable d-dimensional vectors

- How many total parameters in the embedding layer?

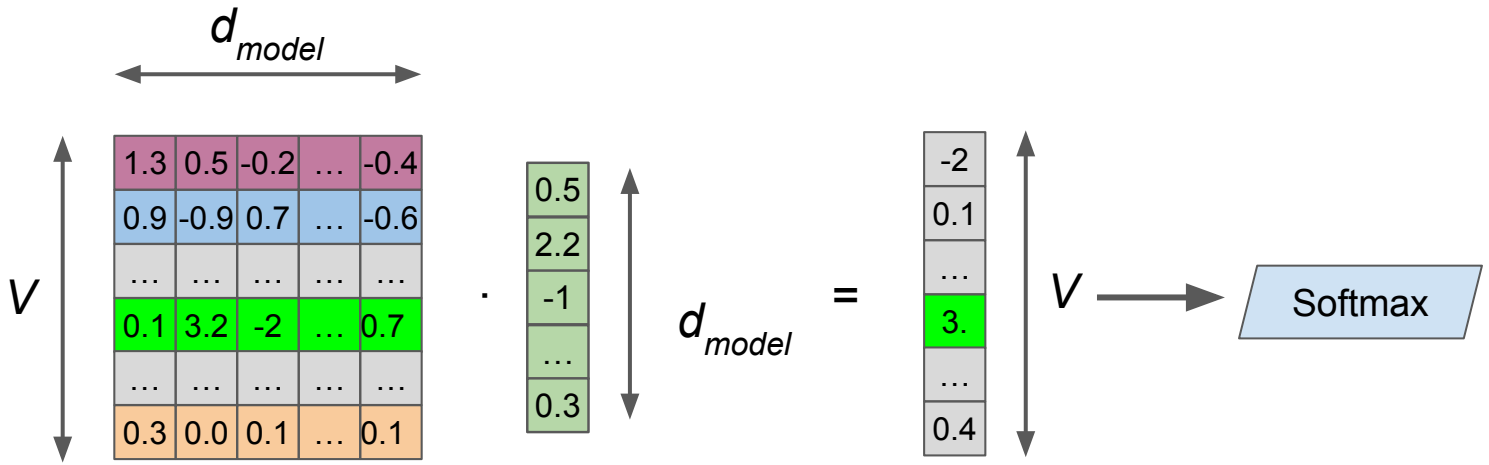
$V \cdot d_{model}$

Embedding Layer



We can get the embeddings for a token by multiplying the embedding matrix with a one-hot encoding for the token id.

Unembedding Layer



The unembedding layer goes in the opposite direction: from embedding (hidden representation) to a score for each token id

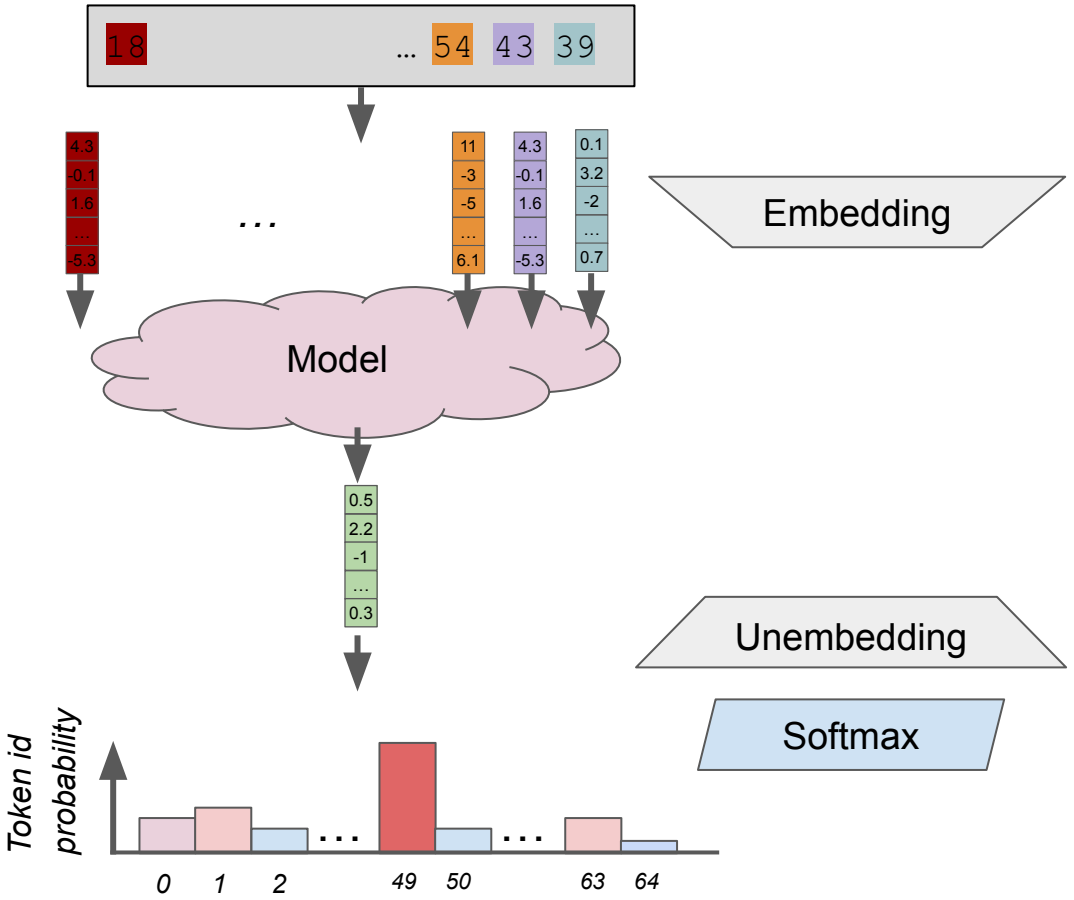
- *Often, the embedding and unembedding parameters are the same!*

Sequence Modeling

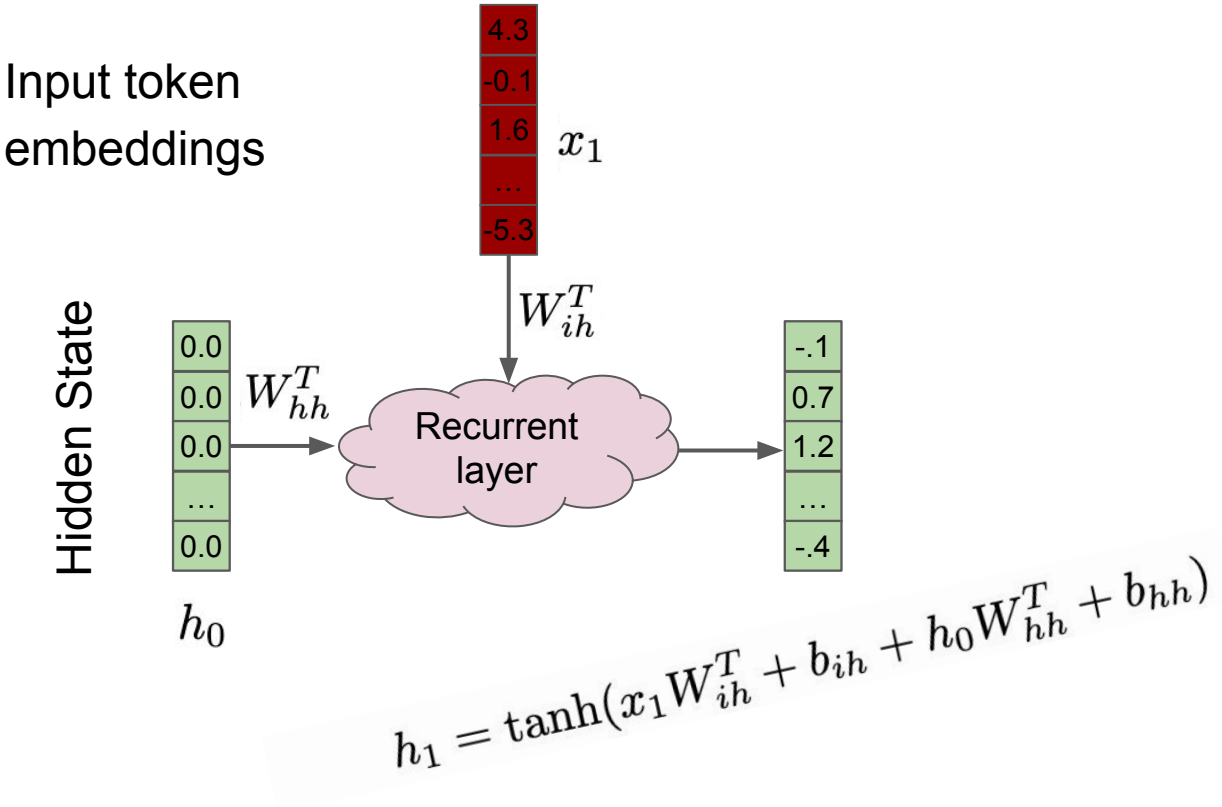
Embed each input token, get a sequence of d_{model} vectors

Model processes them, outputs a d_{model} output vector

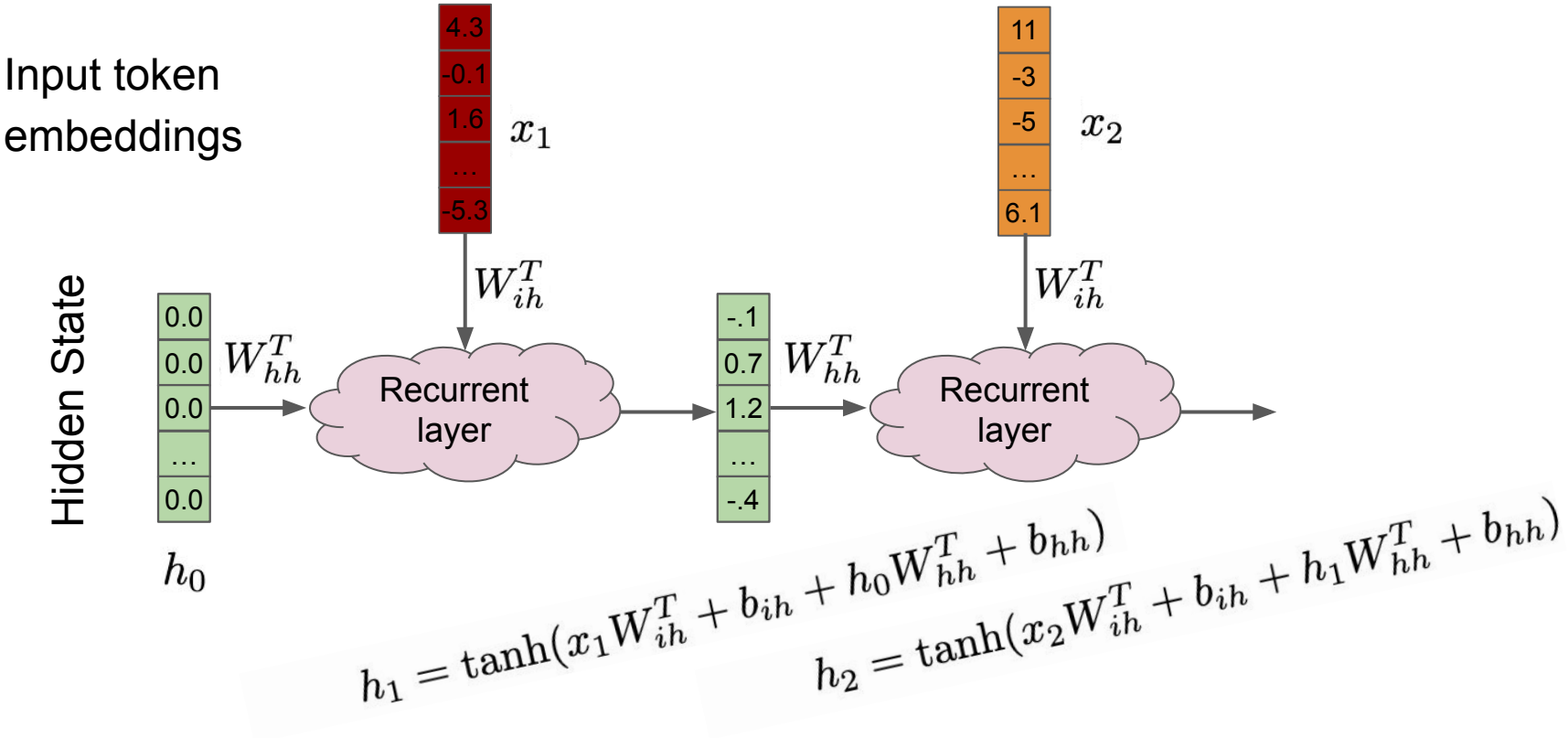
Unembedding + softmax go from the d_{model} vector to a distribution over next token ids



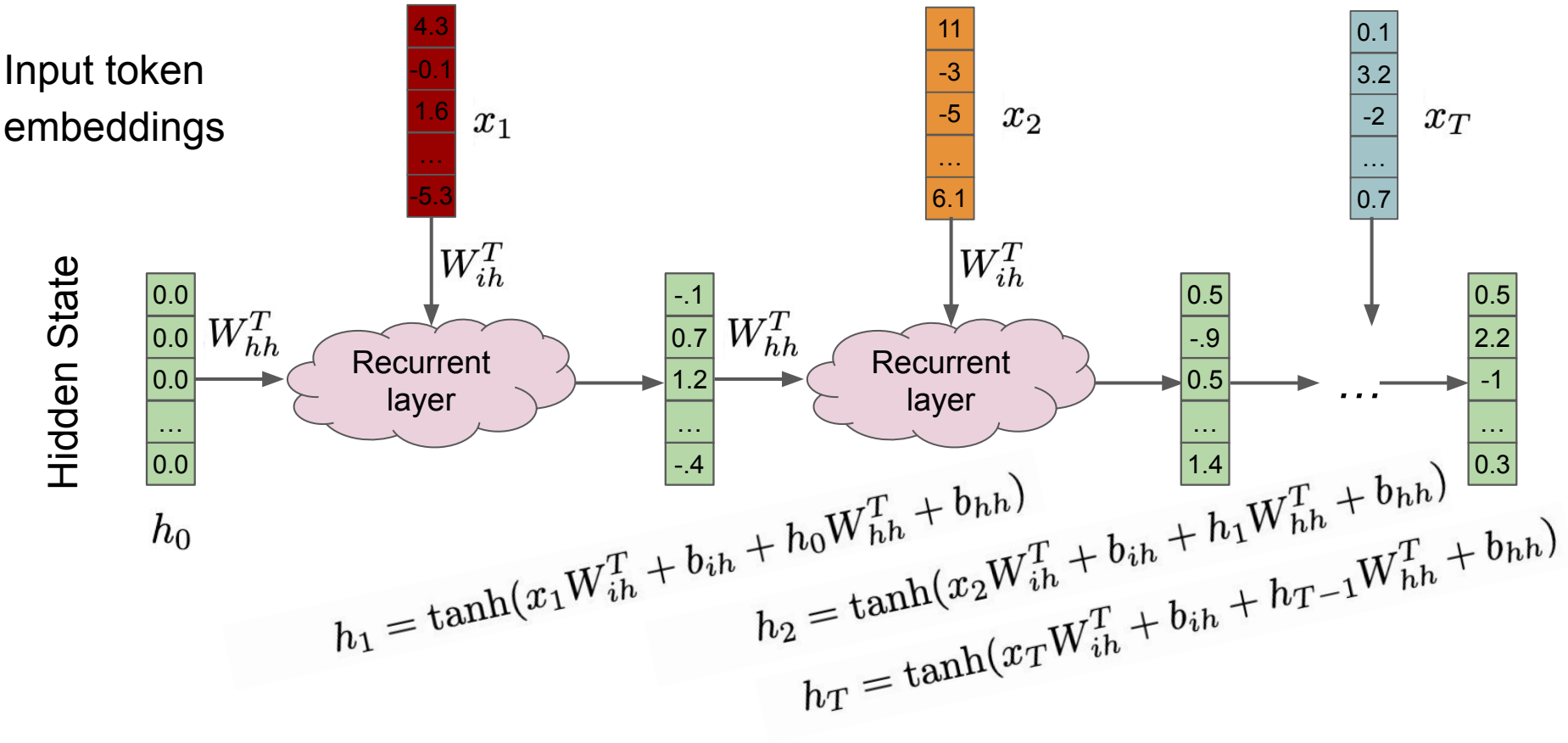
Recurrent Layer



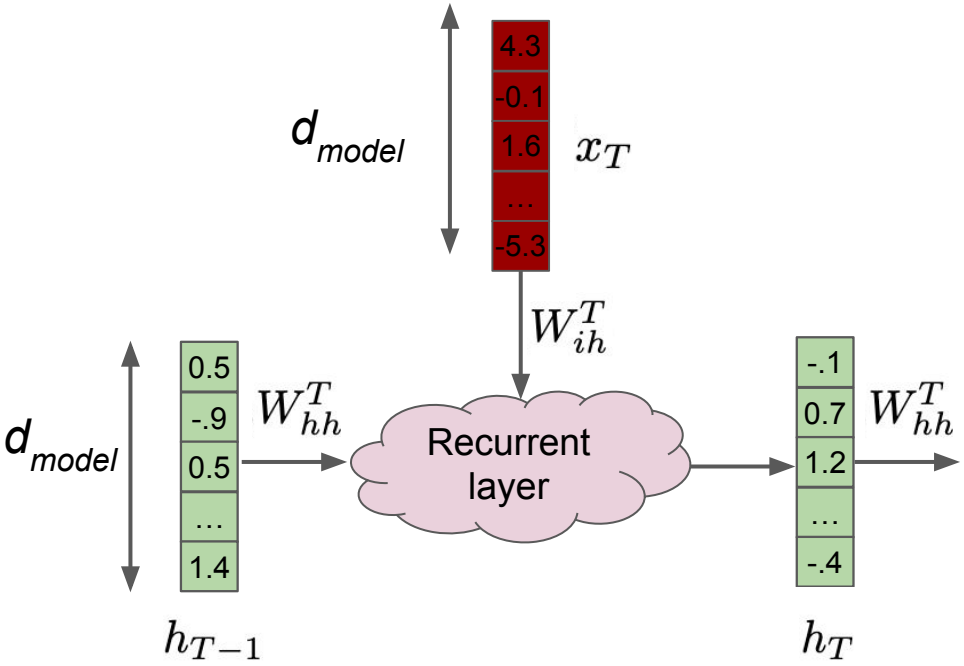
Recurrent Layer



Recurrent Layer



Recurrent Layer

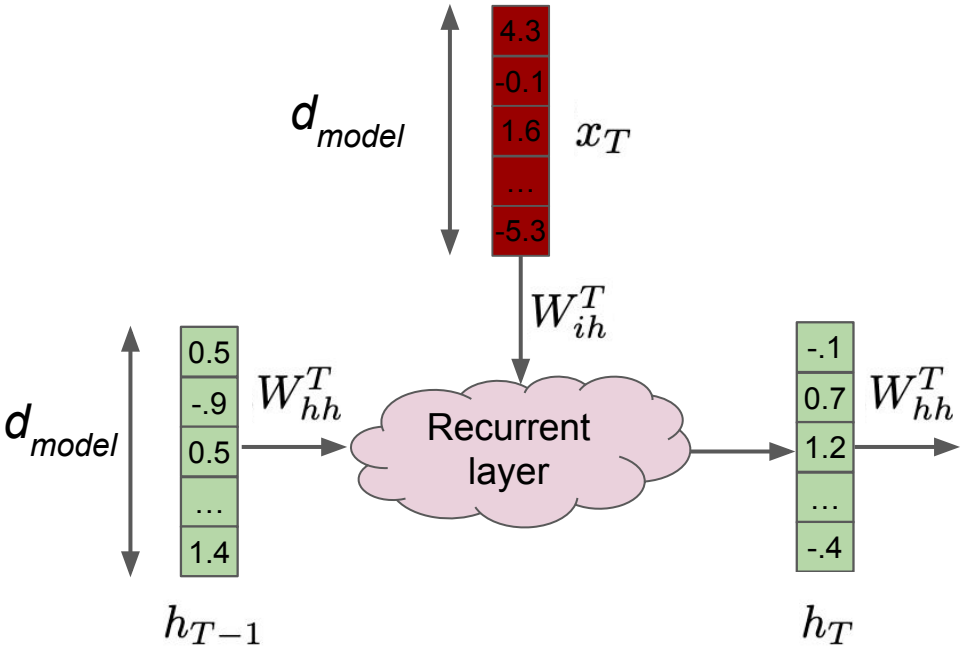


- Linear layer with tanh activation
- Two input vectors: previous hidden state and input embedding
- Hidden state is the summary of the history so far

How many parameters?

$$h_T = \tanh(x_T W_{ih}^T + b_{ih} + h_{T-1} W_{hh}^T + b_{hh})$$

Recurrent Layer



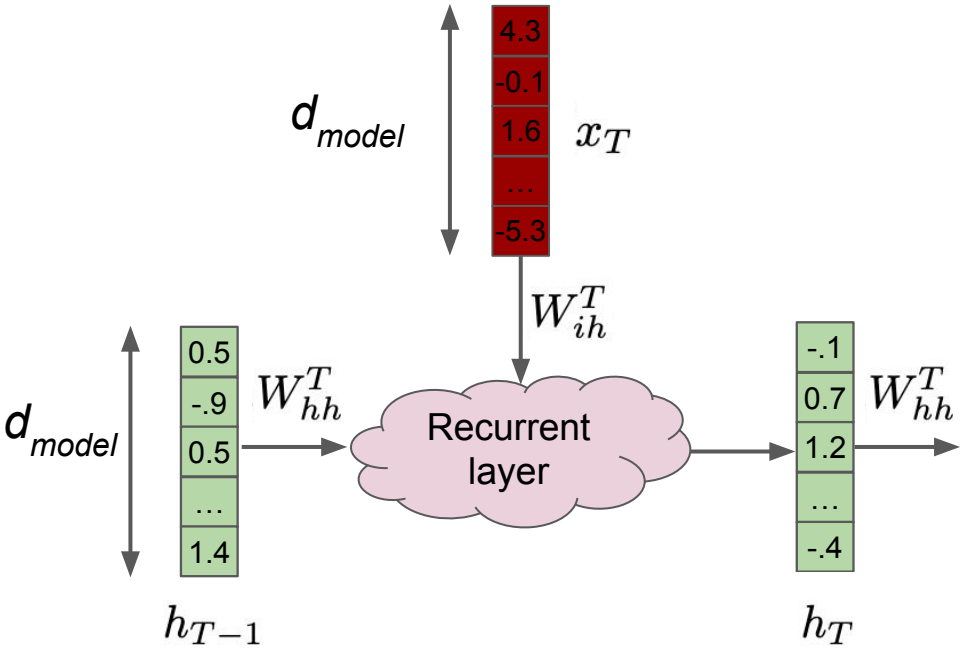
$$h_T = \tanh(x_T W_{ih}^T + b_{ih} + h_{T-1} W_{hh}^T + b_{hh})$$

- Linear layer with tanh activation
- Two input vectors: previous hidden state and input embedding
- Hidden state is the summary of the history so far

How many parameters?

$2 \cdot (d_{model} \cdot d_{model} + d_{model})$

Recurrent Layer



- Linear layer with tanh activation
- Two input vectors: previous hidden state and input embedding
- Hidden state is the summary of the history so far

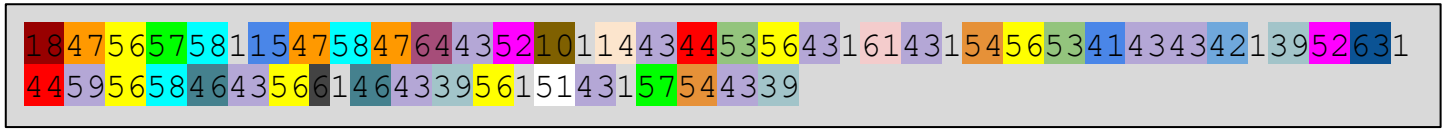
How many parameters?

$2 \cdot (d_{model} \cdot d_{model} + d_{model})$

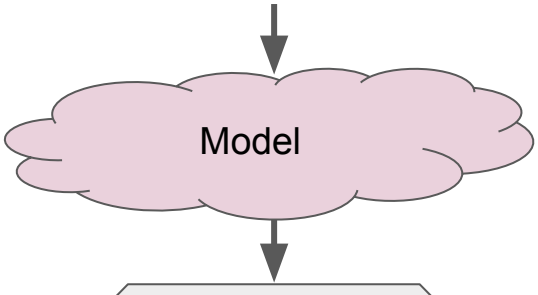
$$h_T = \tanh(x_T W_{ih}^T + b_{ih} + h_{T-1} W_{hh}^T + b_{hh})$$

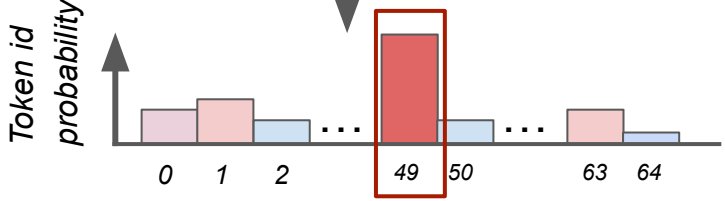
Could use other activations!

Loss



Remember that we are solving a classification problem. We will be using cross-entropy loss

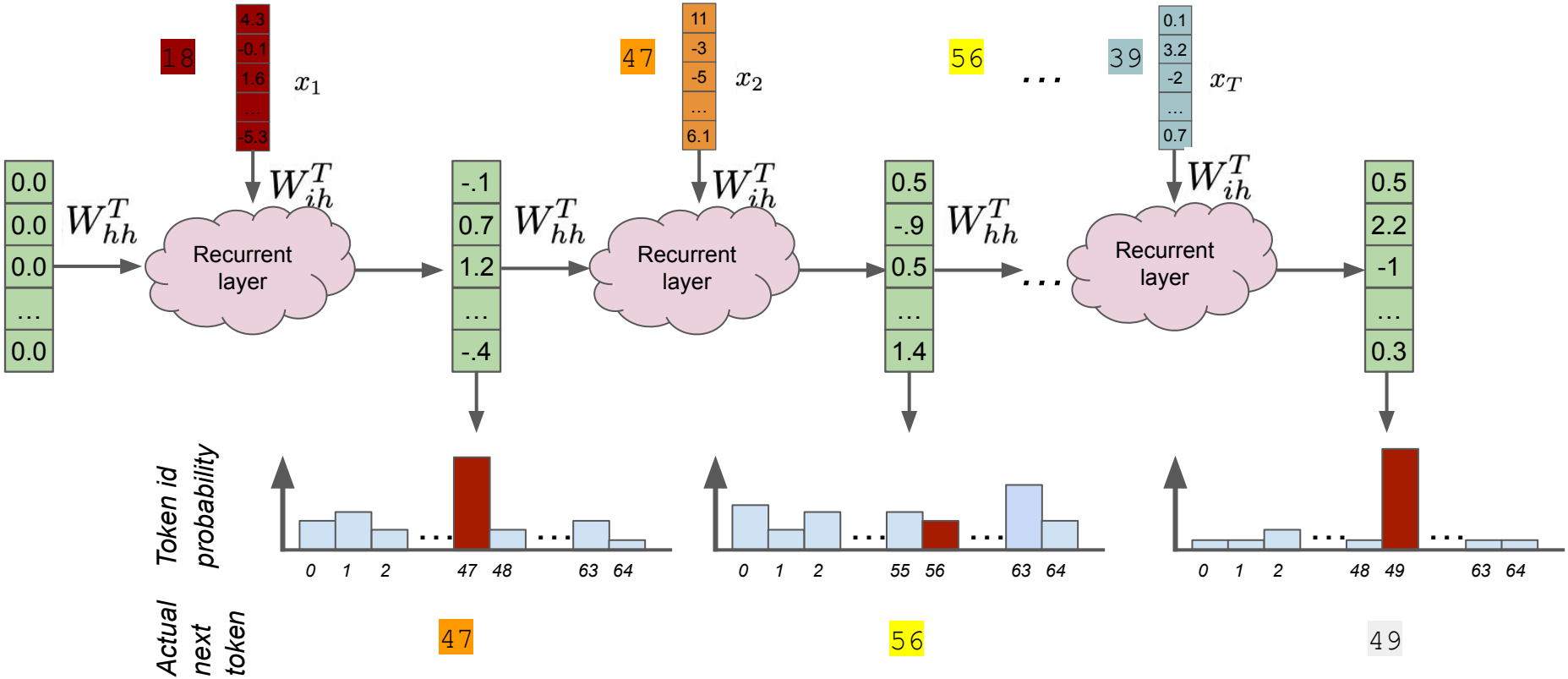




Correct next token id: 49

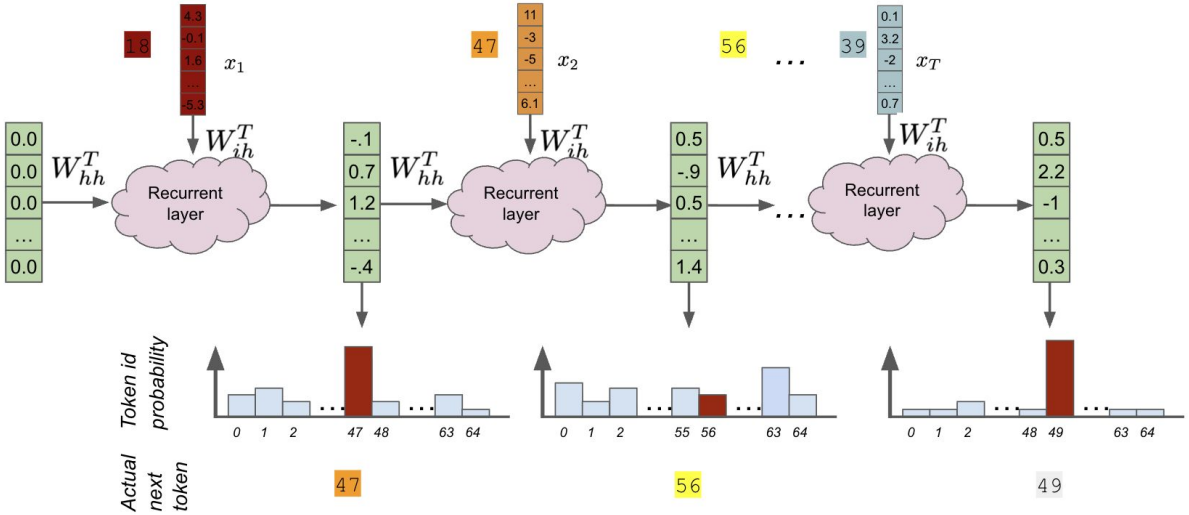
$$\log p_{\text{model}}(x_{T+1}|x_1, \ldots, x_T)$$

Full model



RNN predicts next token at each token position!

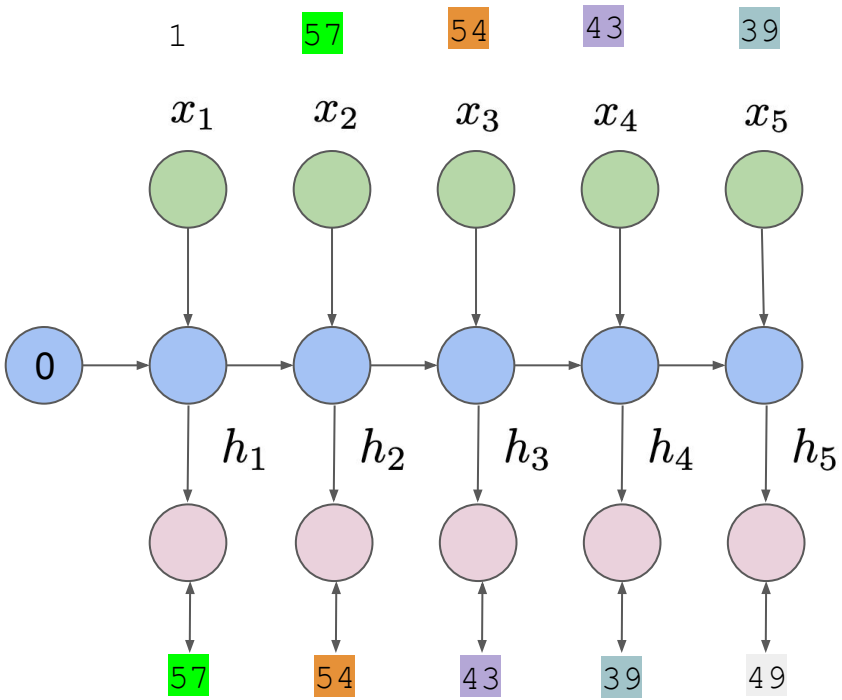
Full model



Full loss is a sum of cross-entropy losses over all tokens:

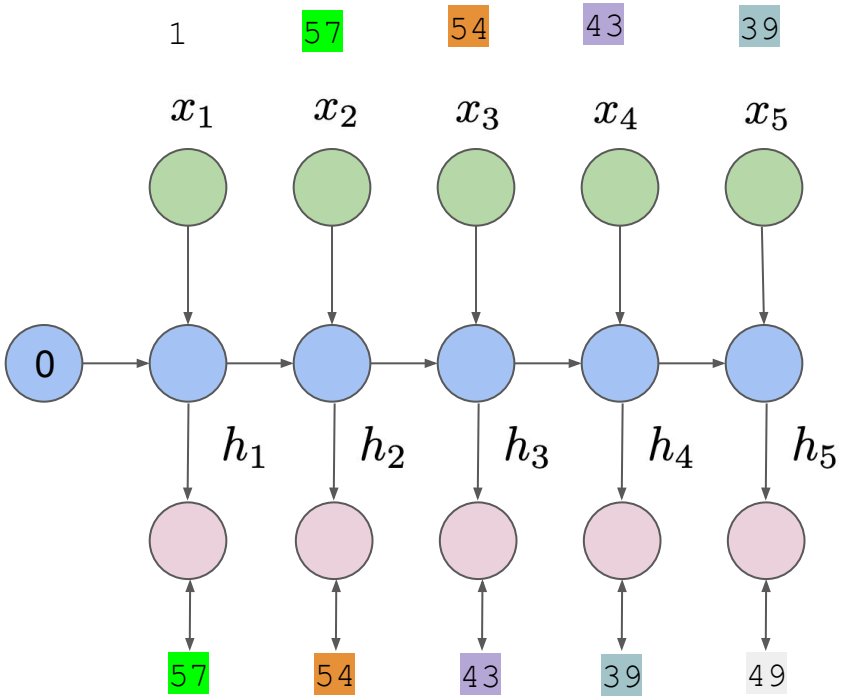
$$L(\text{model}) = \frac{1}{T}\sum_{t=1}^{T} \log p_{\text{model}}(x_{t+1}|x_1, \ldots, x_t)$$

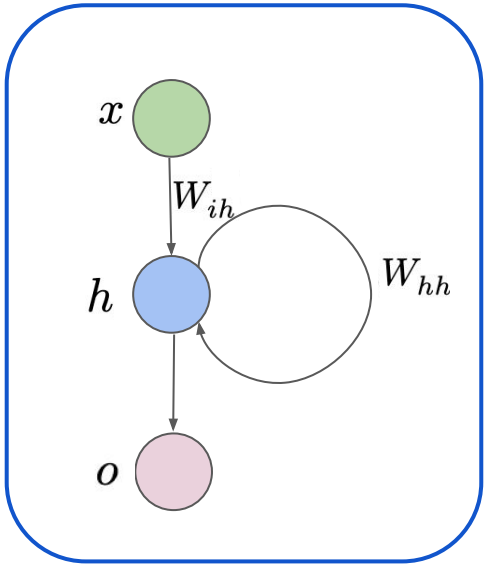
Full model: Minimalistic



$$L(\text{model}) = \frac{1}{T}\sum_{t=1}^{T} \log p_{\text{model}}(x_{t+1}|x_1, \ldots, x_t)$$

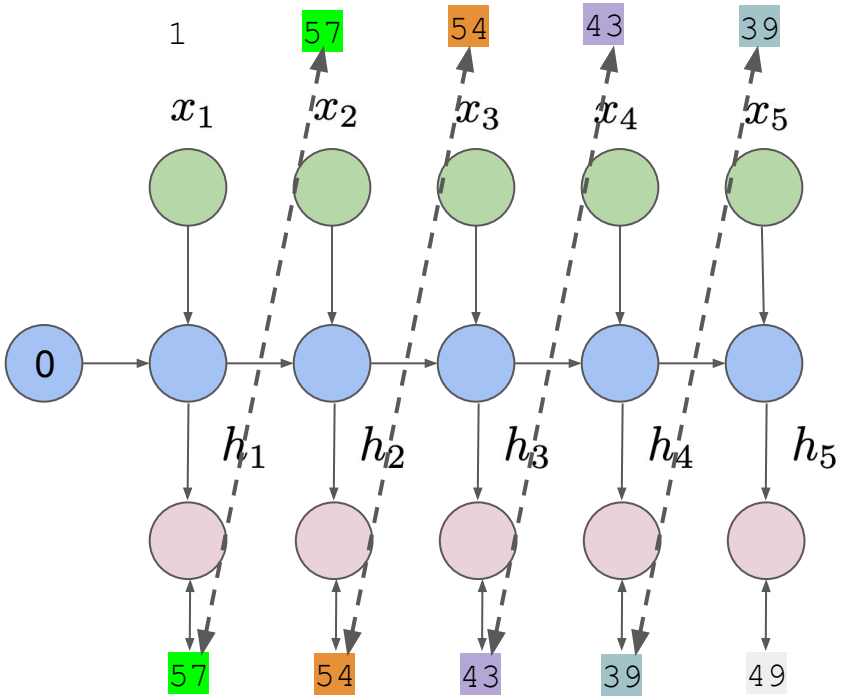
Full model: Minimalistic





$$L(\text{model}) = \frac{1}{T}\sum_{t=1}^{T} \log p_{\text{model}}(x_{t+1}|x_1, \ldots, x_t)$$

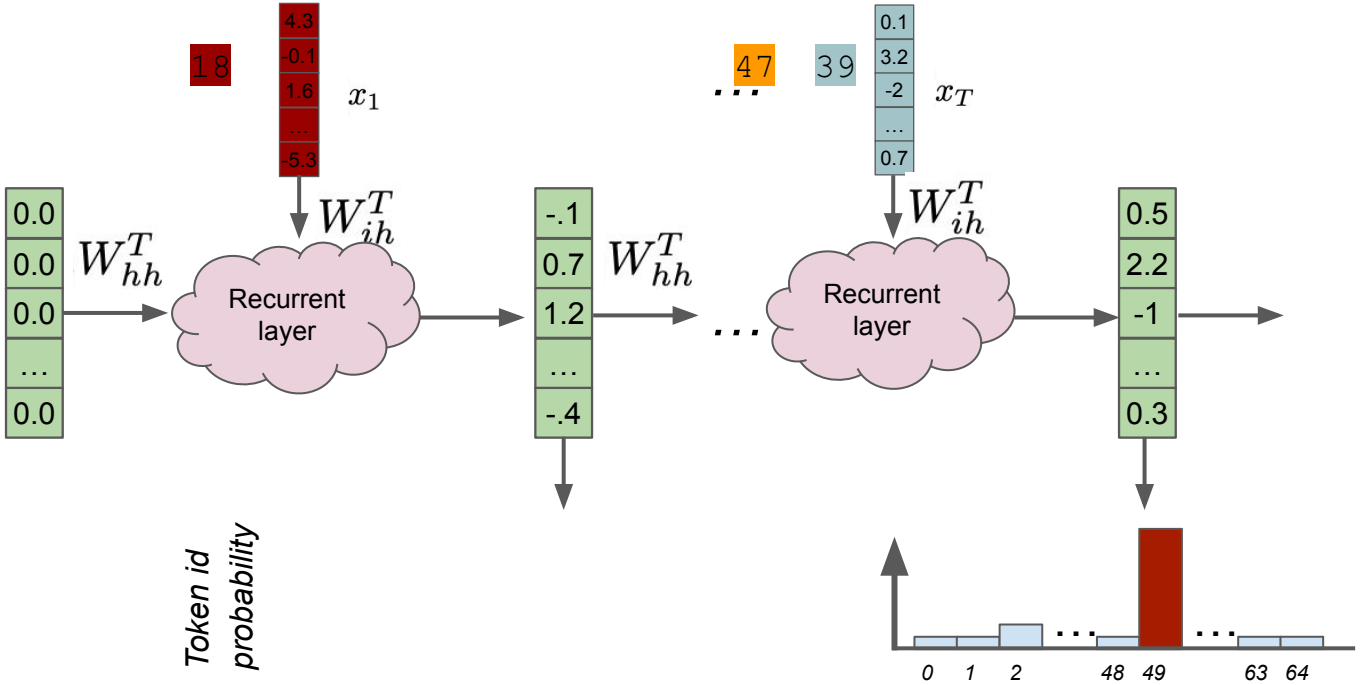
Full model: Minimalistic



$$L(\text{model}) = \frac{1}{T}\sum_{t=1}^{T} \log p_{\text{model}}(x_{t+1}|x_1, \ldots, x_t)$$

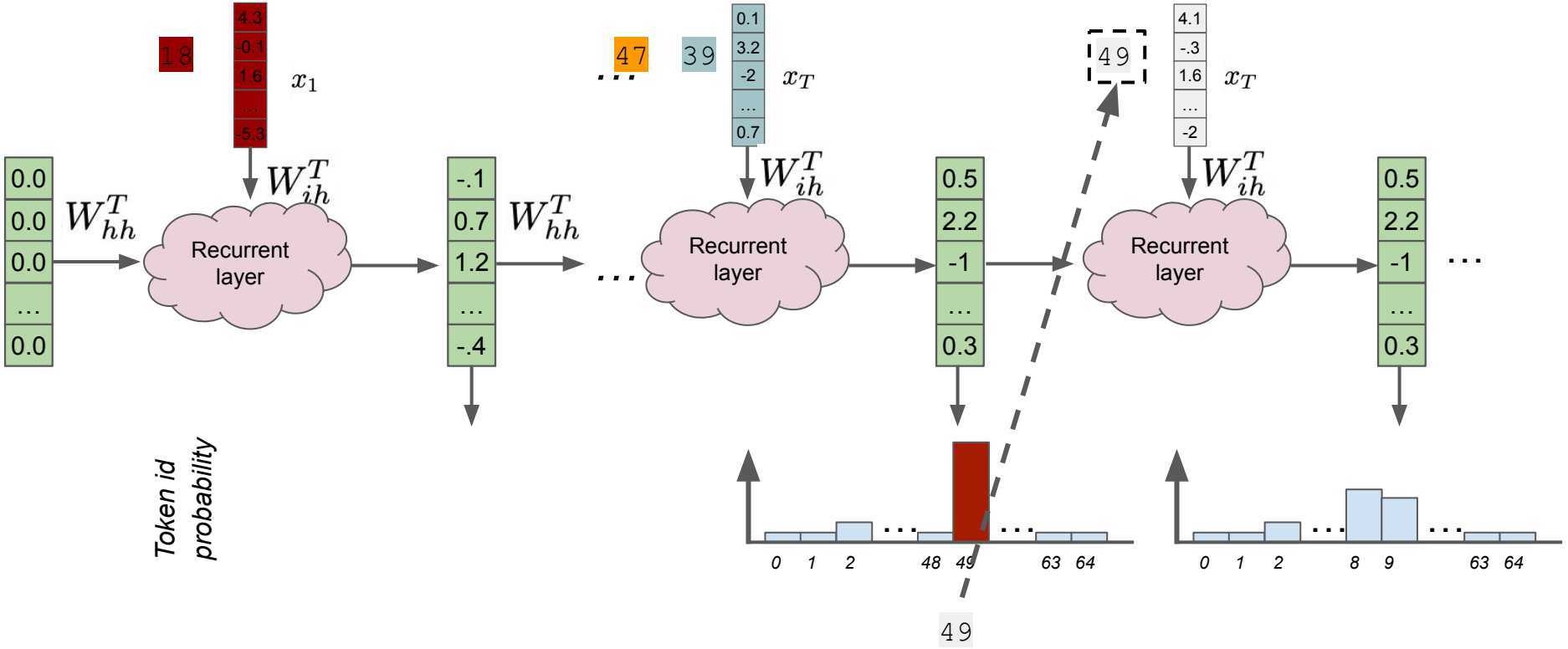
Note: our labels are the same as the next input.

Sampling


18
4.3
-0.1
1.6
...
-5.3
x_1
0.0
0.0
0.0
...
0.0
W_{hh}^T
W_{ih}^T
Recurrent layer
-.1
0.7
1.2
...
-.4
W_{hh}^T
...
47
39
0.1
3.2
-2
...
0.7
x_T
W_{ih}^T
Recurrent layer
0.5
2.2
-1
...
0.3
Token id
probability
0
1
2
48
49
63
64

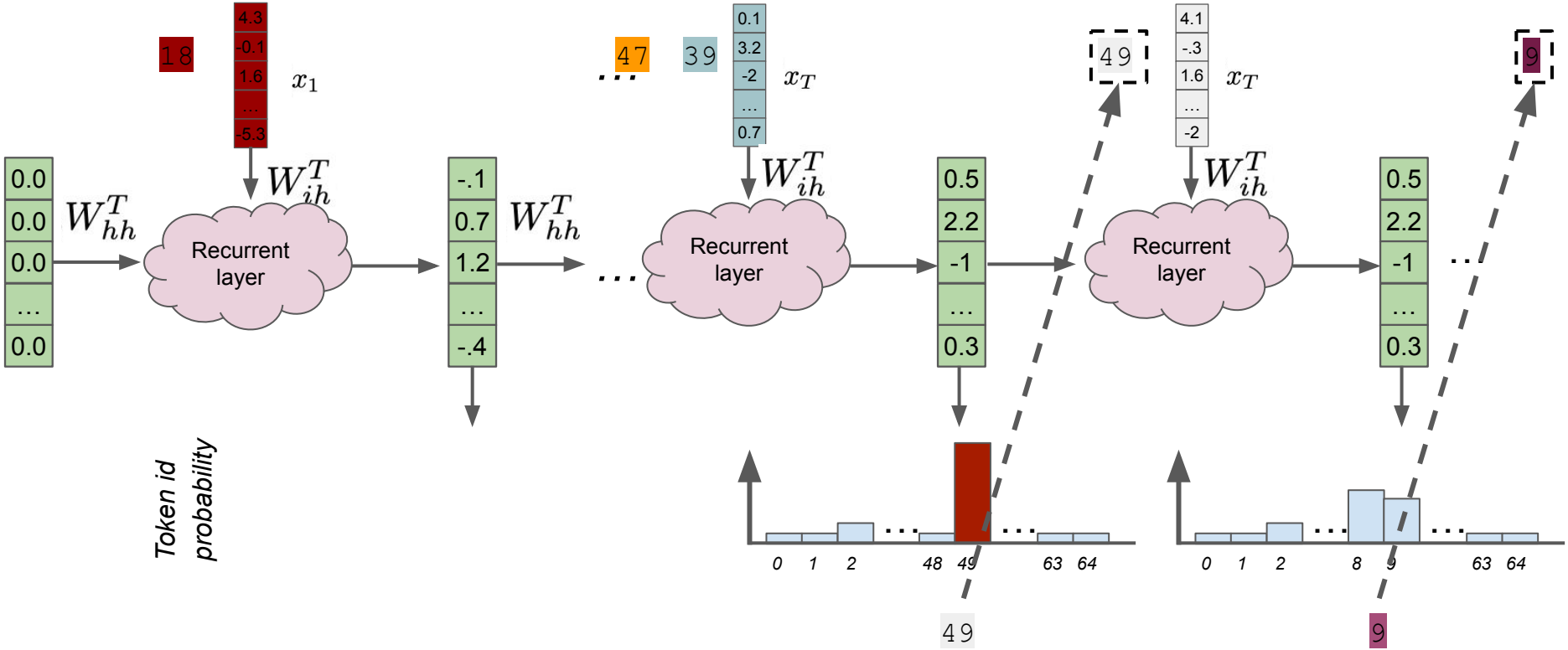
49

Sampling



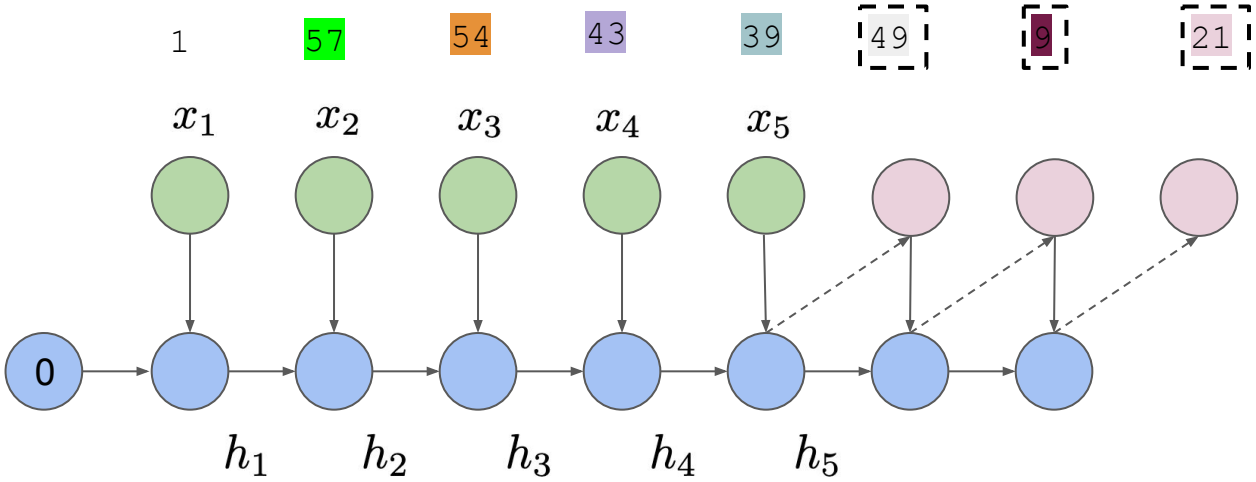
We can sample new tokens from the model, and put them back into the sequence.

Sampling



We can sample new tokens from the model, and put them back into the sequence.

Sampling

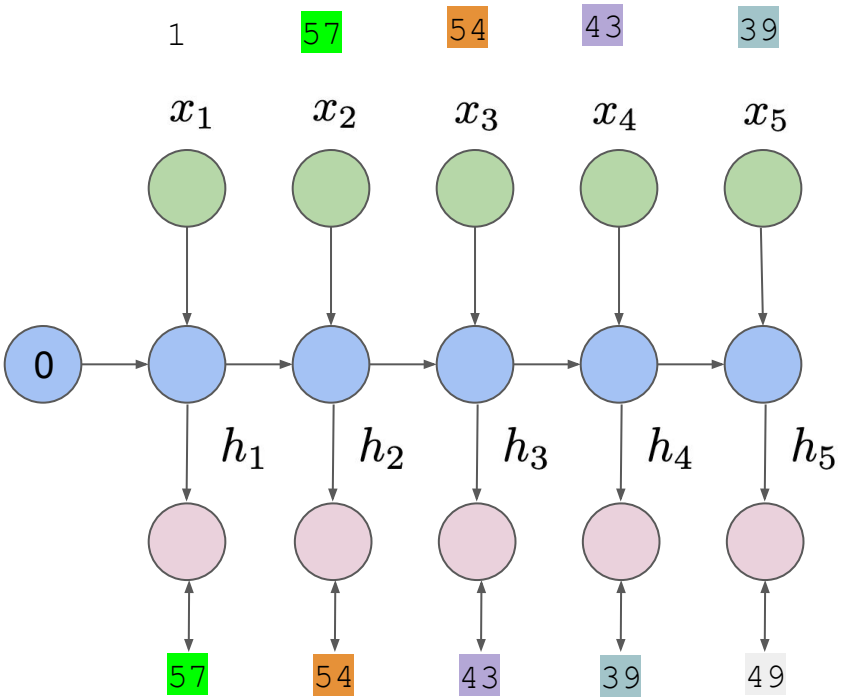


A more minimalistic representation of sampling.

RNN Demo

RNN Demo

Back-Propagation Through Time

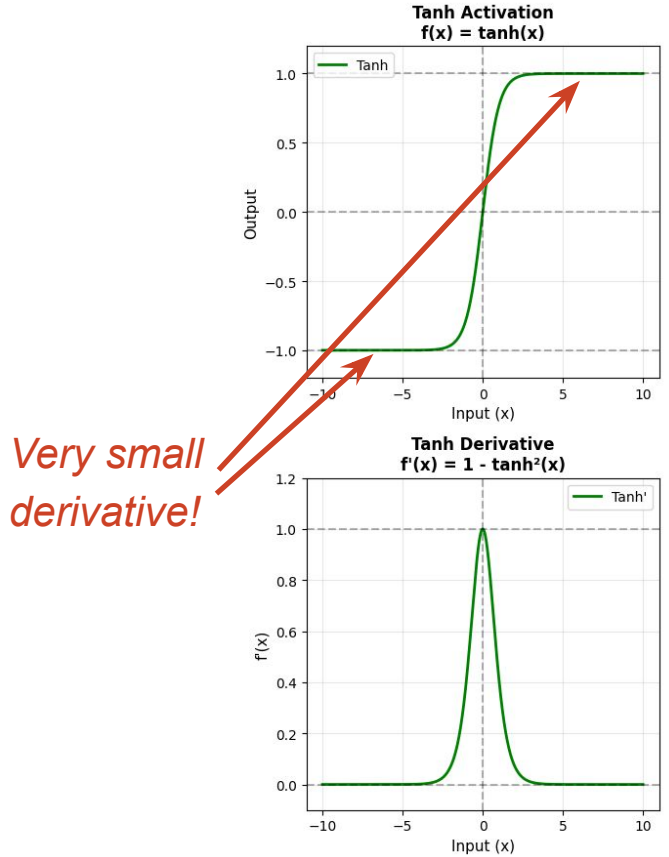


- We can use autograd as always to compute gradients
- The number of layers depends on the token position
- Later tokens ↔ more layers

RNN Issues

Remember our discussion of deepth

- Many layers ⇒ vanishing or exploding gradients
  - RNNs use tanh activations
  - RNNs reuse the same layer many times
- Hard to preserve information about the input
  - RNNs often forget early inputs later in the sequence



Vanishing and Exploding Gradients

Toy model:

- T-step RNN
- Ignore inputs, ignore biases
- Loss computed only on outputs
- 1-d latent

$$h_t = \tanh(w \cdot h_{t-1})$$

$$L(w) = \frac{1}{2}(h_T - y)^2$$

Whiteboard

$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

Vanishing and Exploding Gradients

Whiteboard

$$h_t = \tanh(w \cdot h_{t-1})$$

$$L(w) = \frac{1}{2}(h_T - y)^2$$

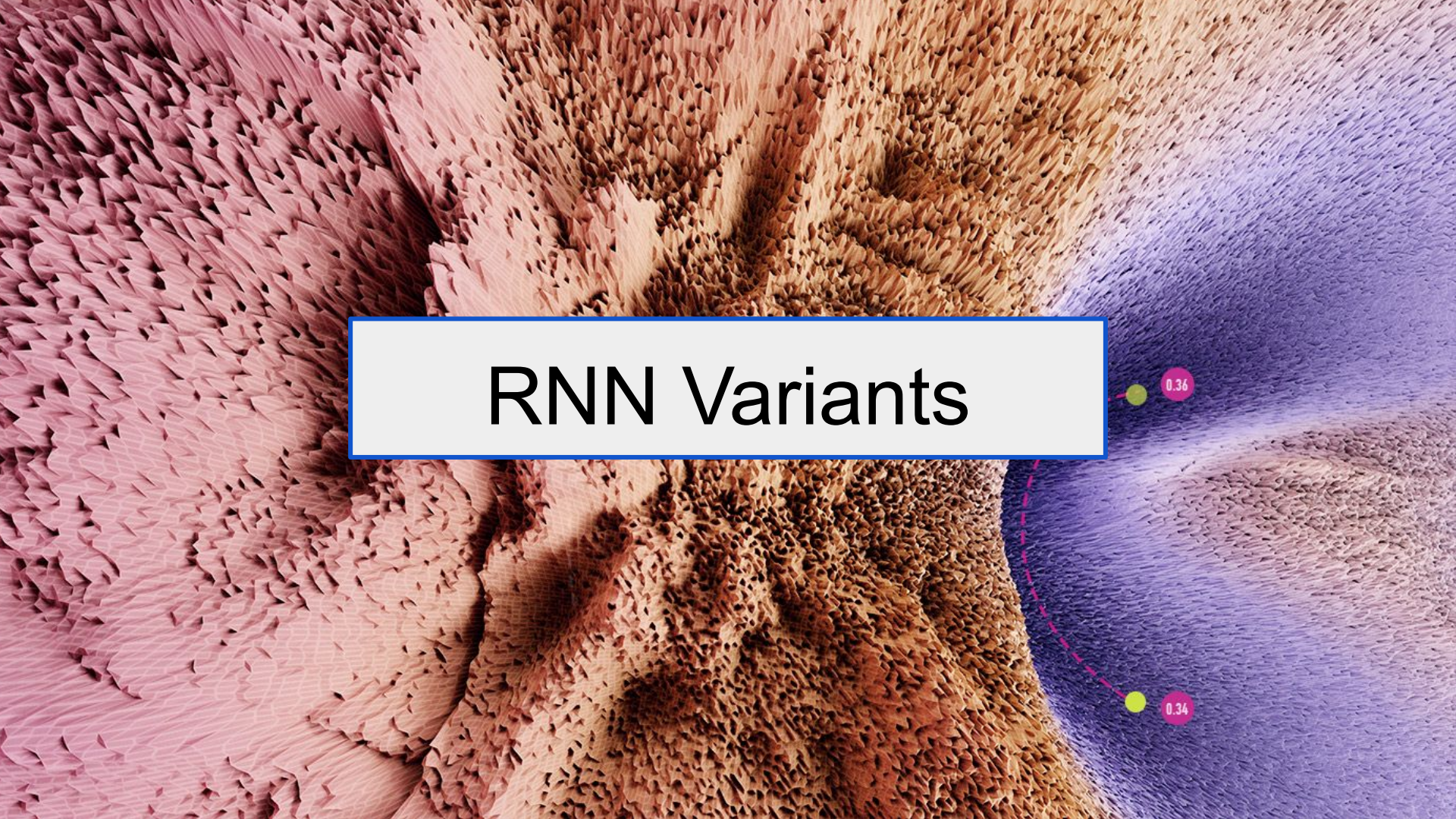
$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

$$\frac{\partial h_T}{\partial h_{T-1}} = \tanh'(w h_{T-1}) \cdot w = (1 - h_T^2)\, w.$$

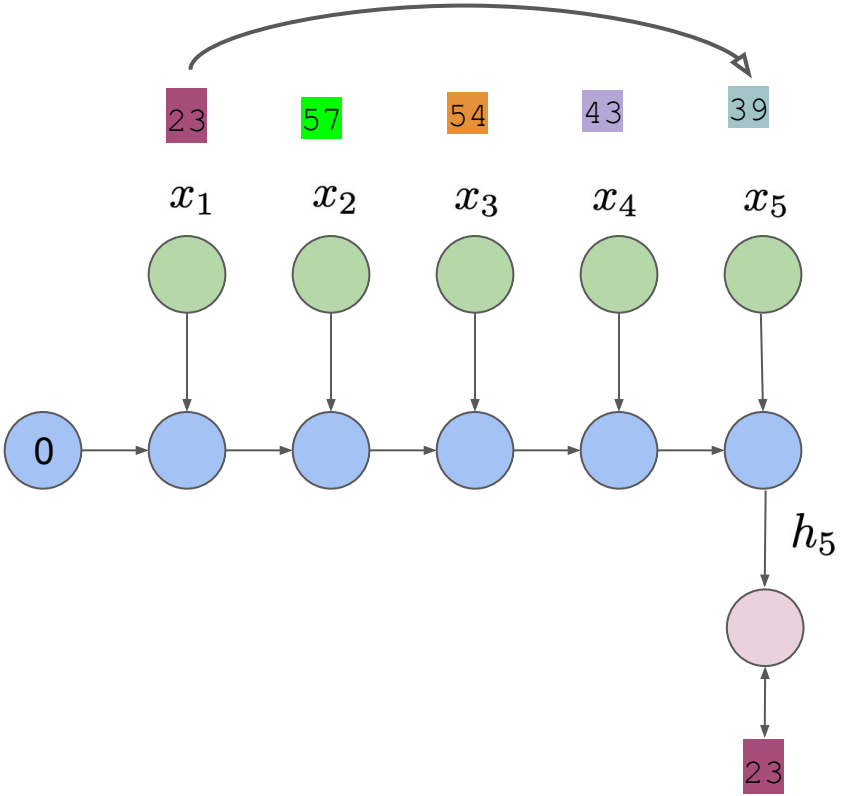
$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} = (h_T - y)\, w^{T-t} \prod_{k=t+1}^{T} (1 - h_k^2)$$

$$\frac{\partial L}{\partial w} = (h_T - y) \sum_{t=1}^{T} w^{T-t} \left(\prod_{k=t}^{T} (1 - h_k^2) \right) h_{t-1}$$

Contributions to the gradient from small t's will vanish if |w| < 1 or explode if |w| > 1


RNN Variants
0.36
0.34

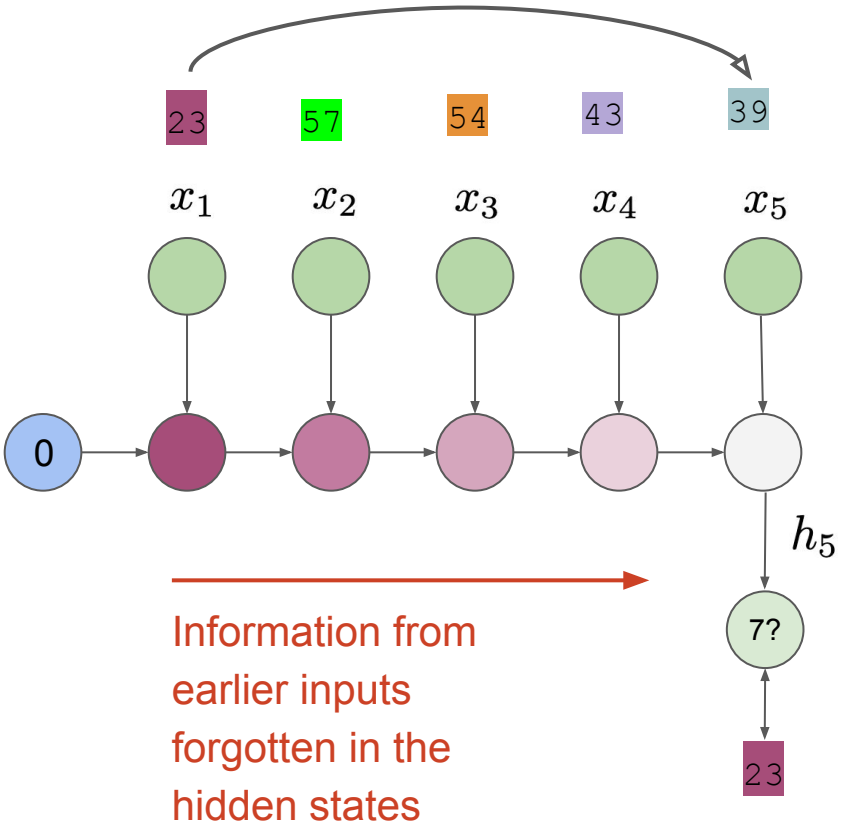
Long-Range Dependencies



We may need information from early tokens to predict later tokens

- E.g. remember name of a character from 10 pages ago in a book
- RNN has to preserve information in its latent across *T* layers

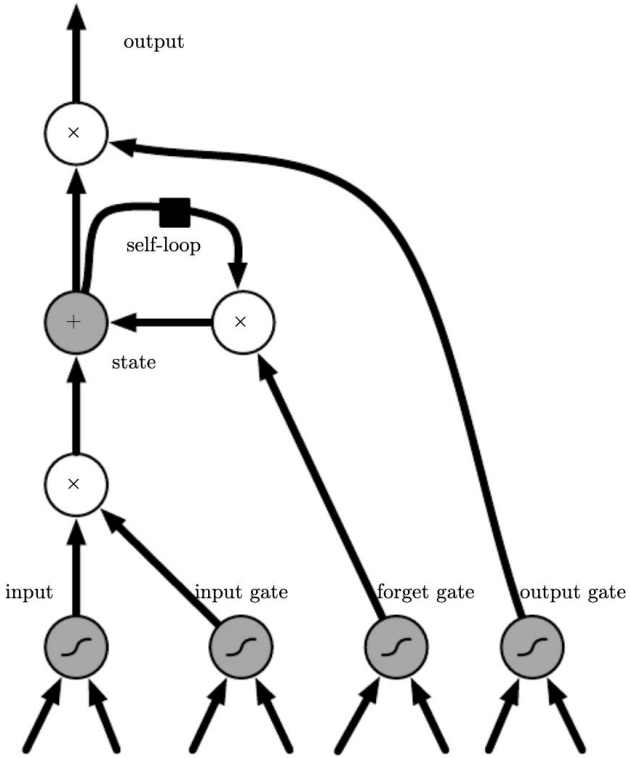
Long-Range Dependencies



We may need information from early tokens to predict later tokens

- E.g. remember name of a character from 10 pages ago in a book
- RNN has to preserve information in its latent across *T* layers
- *Forgetting is a big issue!*

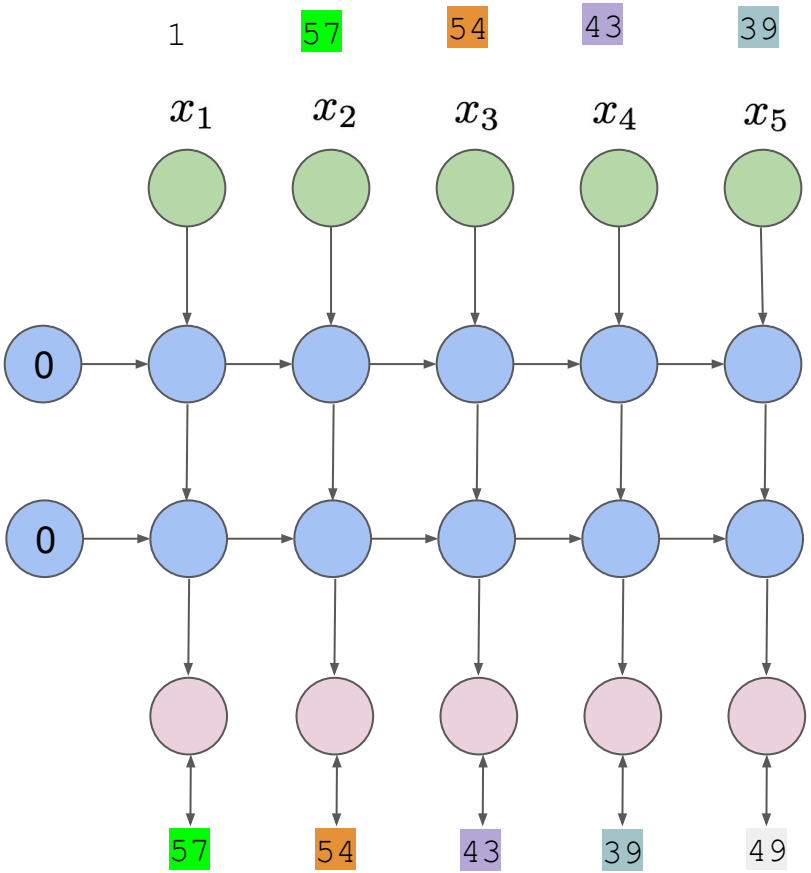
RNN Variants: LSTMs



LSTMs are a complex architecture

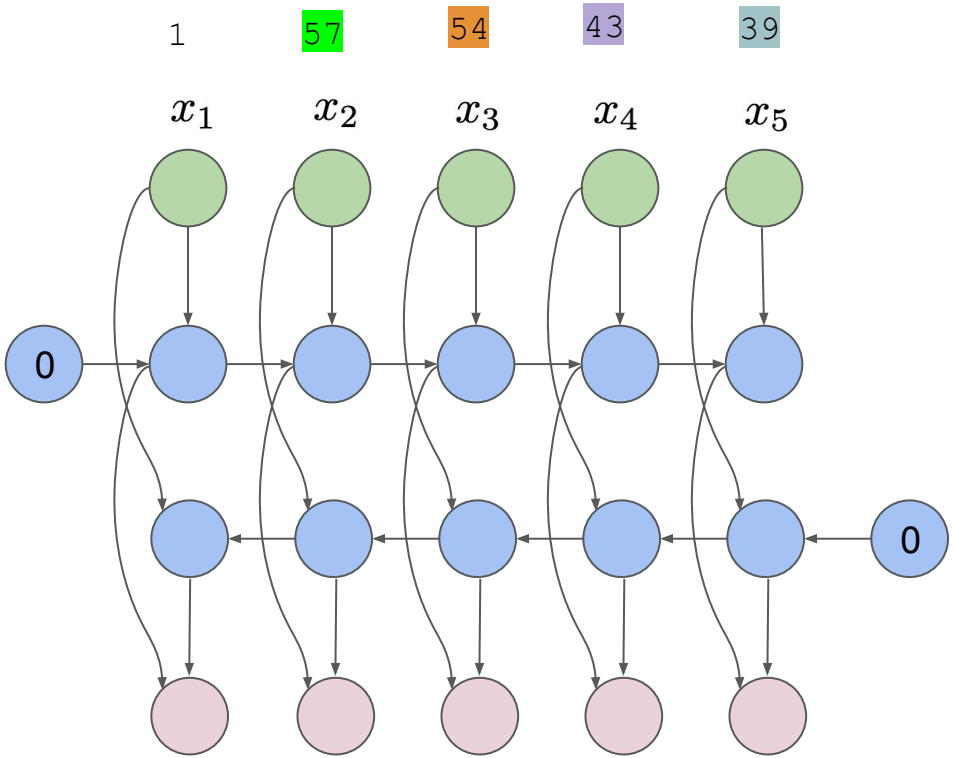
- Idea similar to skip connections
- Tries to address long-range dependence and vanishing / exploding gradient issues
- Default variant of RNN to use

RNN Variants: Multiple Layers



We can stack multiple recurrent layers → Deep RNN

RNN Variants: Bidirectional

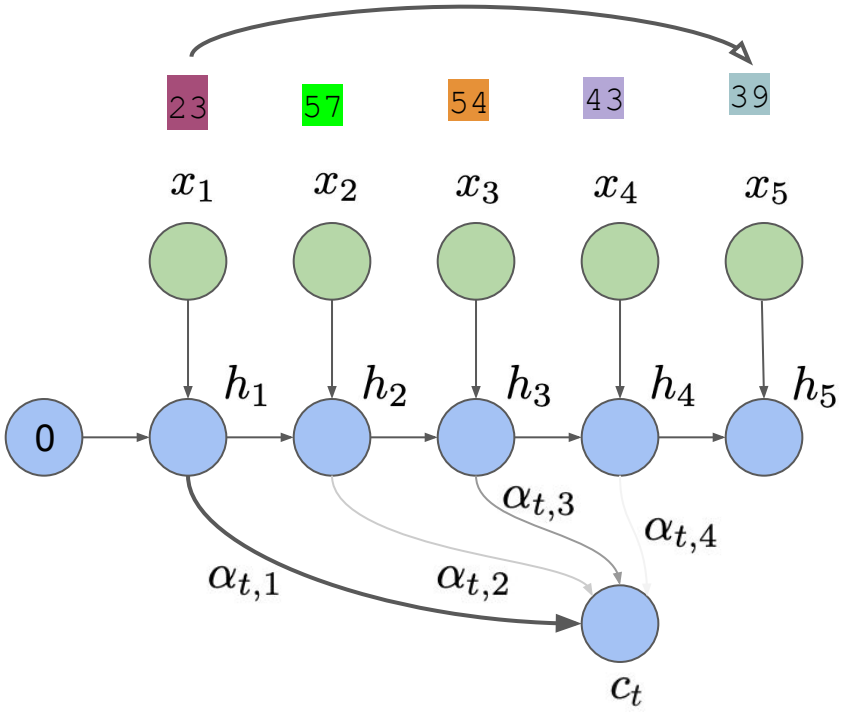


Sometimes we don't want to enforce directionality

- Part of speech tagging
- Speaker labeling in audio

We can use a bi-directional RNN!

RNN Variants: Attention



We will try to fix the long-range dependence issue.

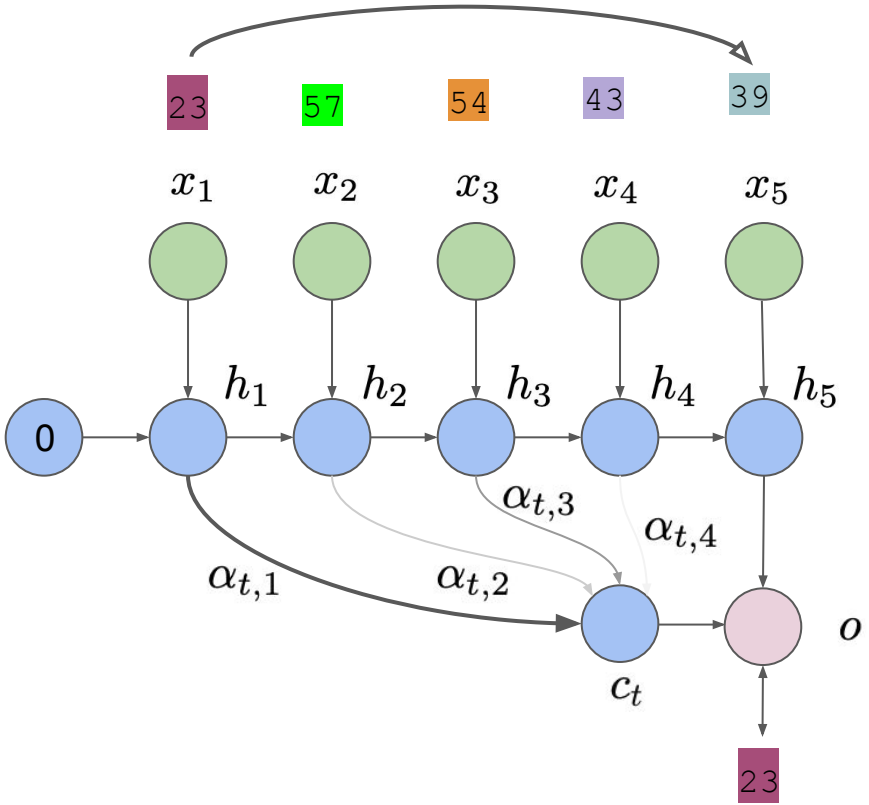
$$e_{t,i} = \text{score}(h_t, h_i), \quad i < t$$

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j<t} \exp(e_{t,j})}$$

Compute *attention weights* for each previous token

$$c_t = \sum_{i<t} \alpha_{t,i} h_i$$

RNN Variants: Attention



We will try to fix the long-range dependence issue.

$$e_{t,i} = \text{score}(h_t, h_i), \quad i < t$$

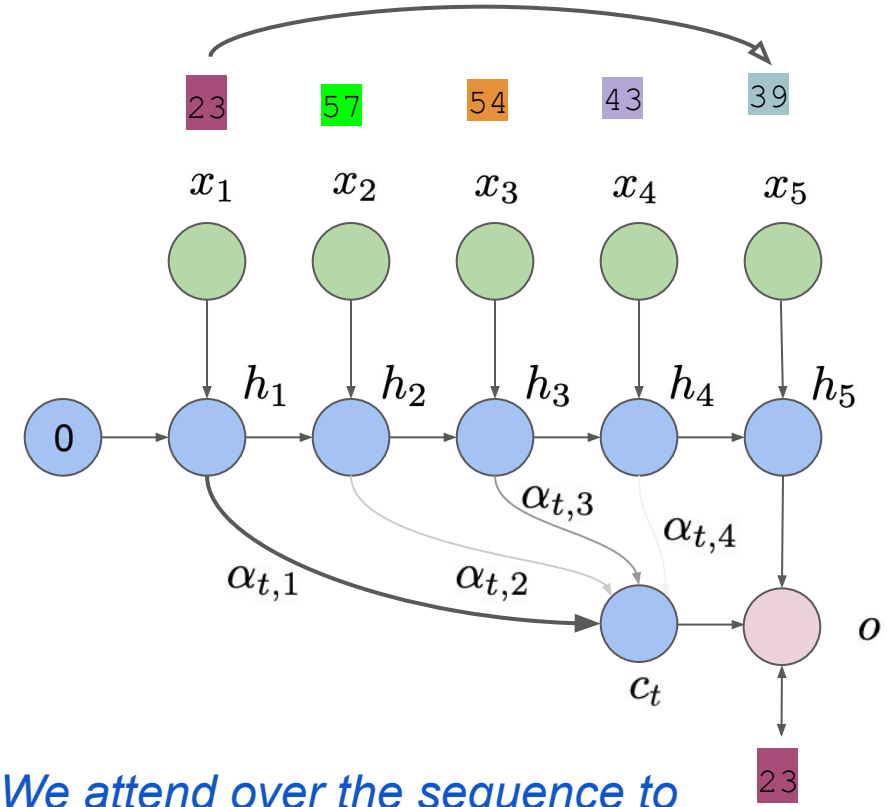
$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j<t} \exp(e_{t,j})}$$

Compute *attention weights* for each previous token

$$c_t = \sum_{i<t} \alpha_{t,i} h_i$$

Use a weighted sum of all representations to make the prediction

RNN Variants: Attention



We attend over the sequence to look for relevant information!

We will try to fix the long-range dependence issue.

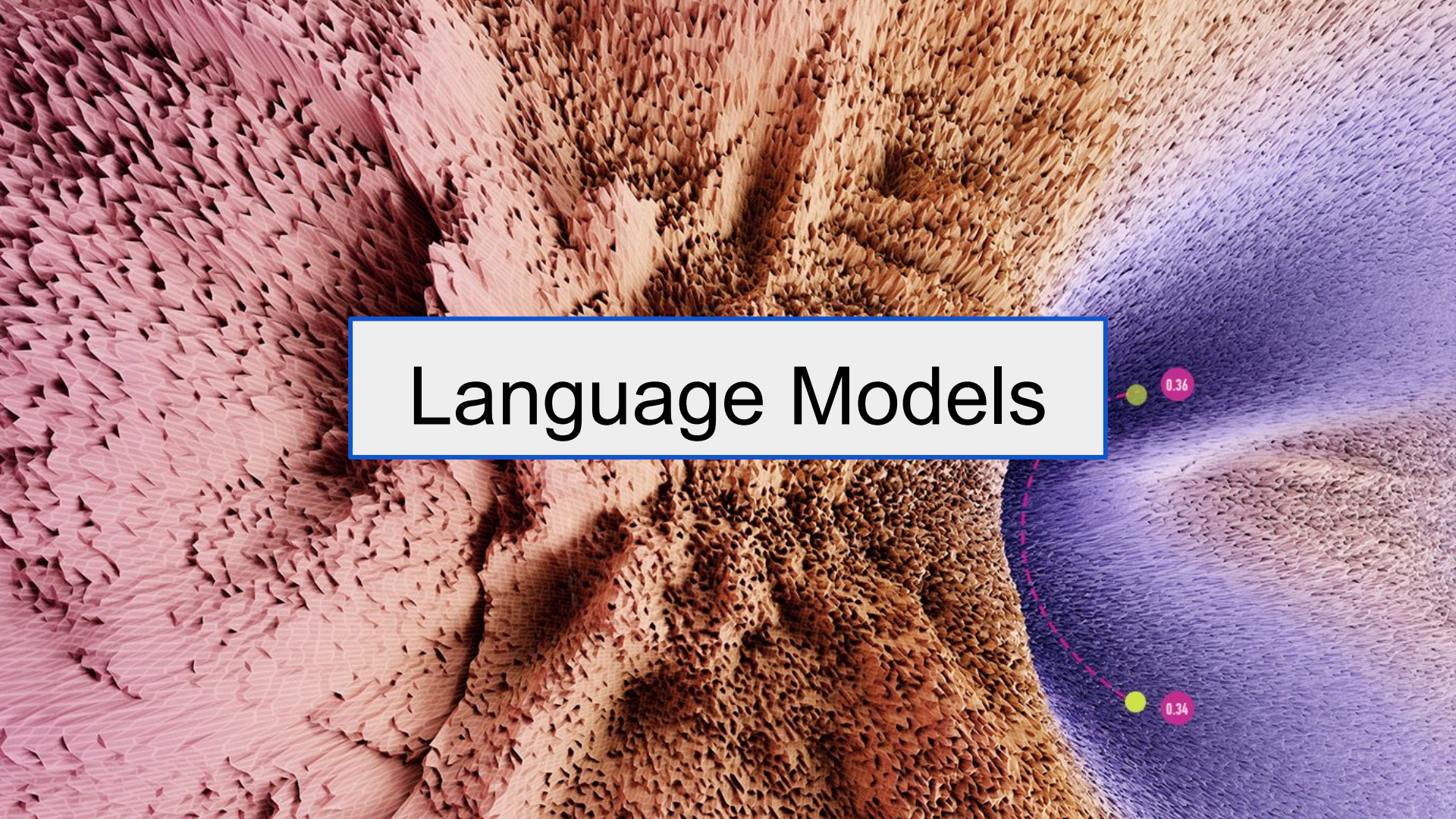
$$e_{t,i} = \text{score}(h_t, h_i), \quad i < t$$

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j<t} \exp(e_{t,j})}$$

Compute *attention weights* for each previous token

$$c_t = \sum_{i<t} \alpha_{t,i} h_i$$

Use a weighted sum of all representations to make the prediction


Language Models
0.36
0.34

More realistic tokenization

Our character tokenizer

```
First Citizen:
Before we proceed any further, hear
me speak.
…
```

```
18475657581154758476443521 01

1443445356431614315456534143434213 95
2631445956584643566146433956 1

514315754433949 8
…
```

60 tokens

Vocabulary size: 65

GPT-4o Tokenizer

```
First Citizen:
Before we proceed any further, hear me
speak.
…
```

```
7127 84479 734
13036 581 18988 1062 6544 11 9598
668 10591 13
…
```

13 tokens

Vocabulary size: ~200k

More realistic tokenization

- Can tokenize any sequence of bits
- Token ids represent common words

  happiness

  bifurcation
- Typical token ~4 characters
- Big vocabulary ⇒ embedding layers are big

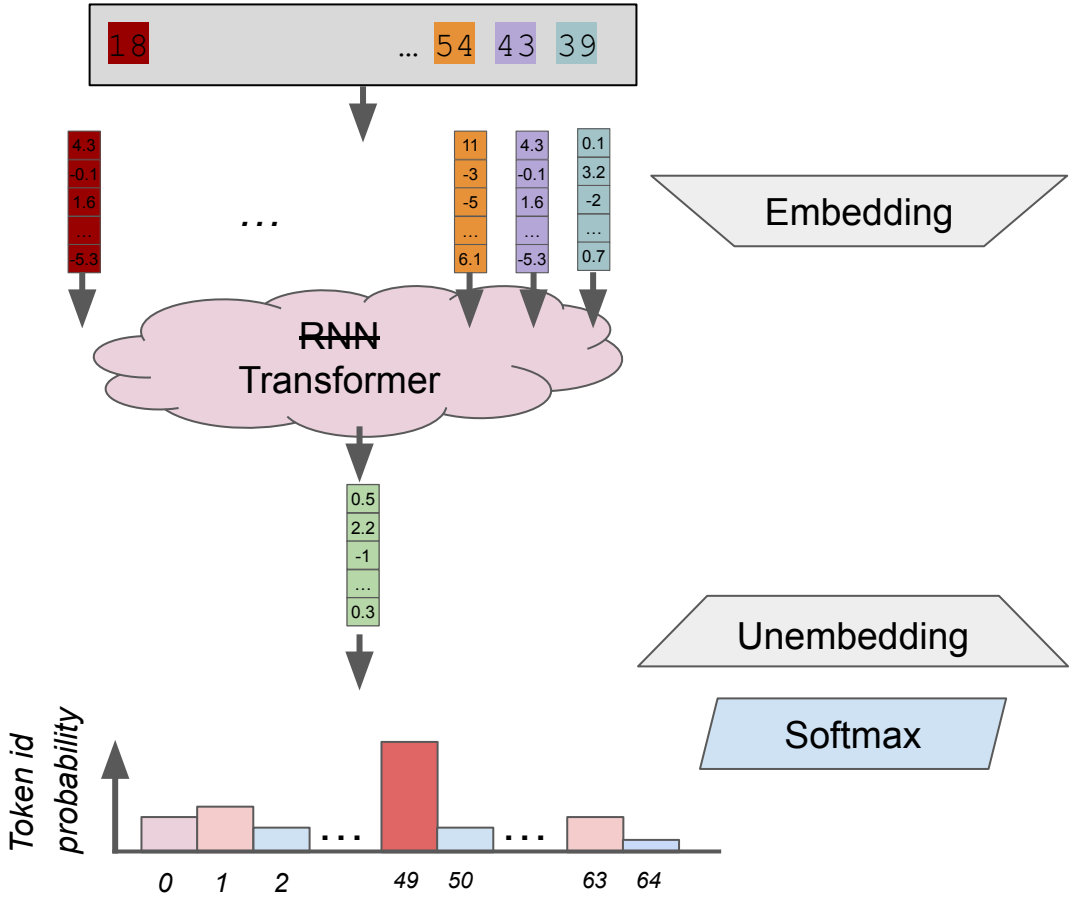
GPT-4o Tokenizer

First Citizen:
Before we proceed any further, hear me speak.
…

7127 84479 734
13036 581 18988 1062 6544 11 9598 668 10591 13
…

From RNNs to Transformers

We will discuss transformers next time, but the big picture is the same.



LLM Pretraining

- Majority of LLM training compute is spent on next token prediction
- Models with hundreds of billions of parameters trained for tens of trillions of tokens
- Costs of LLM training runs estimated $100M +

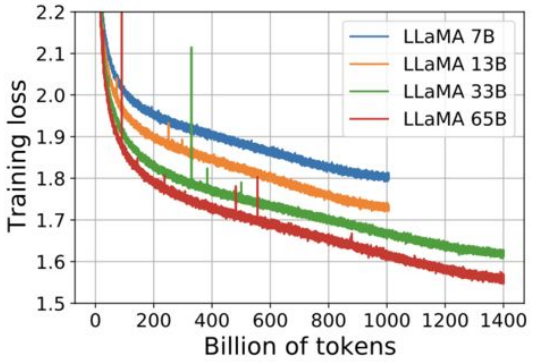


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.


Colossus: Total GPUs: **200,000**
Phase 1: **122 days -** 100k GPUs fully training synchronously. From scratch.
Phase 2: **92 days** to expand to 200K GPUs

Base Language Models

- Next token prediction on the internet does not give us a chatbot
- They continue the text, not respond to your questions
- These are hard to work with

I couldn't find any new interactive base model online!

System Prompt (human-written)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Model Completion (machine-written, 10 tries)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

https://openai.com/index/better-language-models/

Base Language Models

What is 2 + 2?

I've said this before, but the fact is

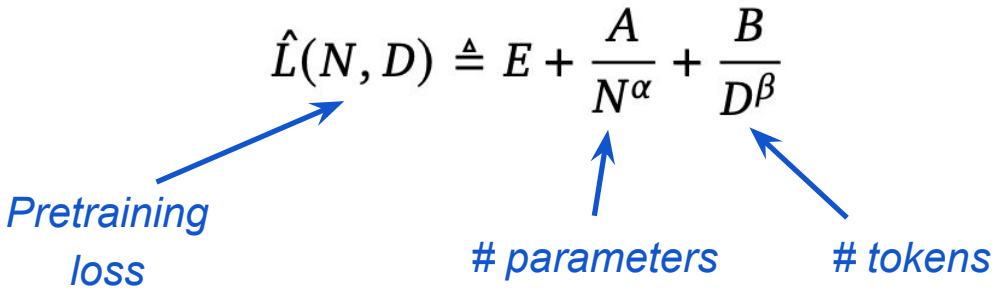
2+2 = 5↵↵How does a

Theorem for the 2 + 2 (multiplying)

It is trying to continue the text, not be useful. We will talk about *post-training* in the next lectures.

https://banana-projects-transformer-autocomplete.hf.space/doc/gpt2-large

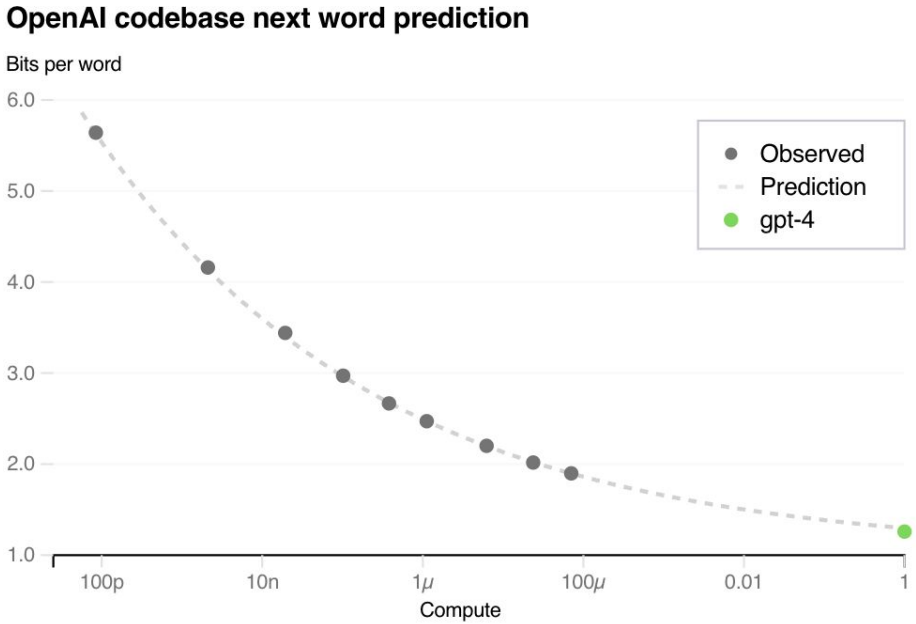
Scaling Laws



Models get predictably better with more data and with bigger size.

[2203.15556] Training Compute-Optimal Large Language Models
[2001.08361] Scaling Laws for Neural Language Models

Scaling Laws

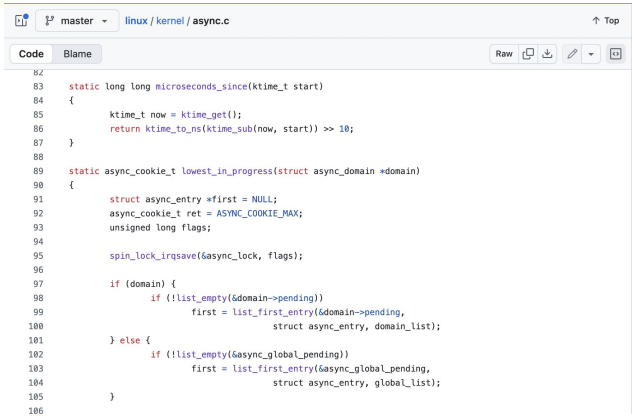


Predictable scaling allows to make better decisions.

https://openai.com/index/gpt-4-research/

Summary

- Lots of things can be presented as a sequence of tokens
- Good next token prediction requires intelligence
- Scaling next token prediction ⇒ modern LLMs



```
master  linux / kernel / async.c                                  Top
Code  Blame                                                       Raw

static long long microseconds_since(ktime_t start)
{
        ktime_t now = ktime_get();
        return ktime_to_ns(ktime_sub(now, start)) >> 10;
}

static async_cookie_t lowest_in_progress(struct async_domain *domain)
{
        struct async_entry *first = NULL;
        async_cookie_t ret = ASYNC_COOKIE_MAX;
        unsigned long flags;

        spin_lock_irqsave(&async_lock, flags);

        if (domain) {
                if (!list_empty(&domain->pending))
                        first = list_first_entry(&domain->pending,
                                        struct async_entry, domain_list);
        } else {
                if (!list_empty(&async_global_pending))
                        first = list_first_entry(&async_global_pending,
                                        struct async_entry, global_list);
        }
```


4 TERENCE TAO

The random set A we will propose for Theorem 1.3 will then be defined as the truncated random parabola

$$A := \{(x,y) \in \{0,\dots,n-1\}^2 : (ax+by)^2 = cx+dy+e \bmod p\}. \tag{1.2}$$

The standard truncated parabola $S := \{(x,y) \in \{0,\dots,n-1\}^2 : y = x^2 \bmod p\}$ is essentially the example considered in [16], [4], and is the special case of (1.2) when $a = d = 1$ and $b = c = e = 0$. Geometrically, A is formed by applying a random invertible affine transformation[3] to the parabola $\{(x,x^2) : x \in \mathbf{F}_p\}$ and then restricting to the grid $\{0,\dots,n-1\}^2$. While the construction (1.2) is not nearly as random as a completely random subset of $\mathbf{F}_p^2$ of density $\asymp 1/p$, we shall see that there will (barely) still be enough "entropy" in the five random parameters a,b,c,d,e for the probabilistic method to be effective[4]. In particular, we avoid the difficult number-theoretic questions of trying to count the number of occurrences of the patterns $\pi_1,\pi_{31},\dots,\pi_8$ in the standard truncated parabola S (cf. [5, Problem 2]).

A routine application of the second moment method reveals that A usually has the "right" cardinality (up to accceptable errors):

Lemma 1.5. *With probability at least* 0.9 *(say), the set A has cardinality $n^2/p + O(\sqrt{n})$. In particular, for n large enough, the cardinality of A is $\asymp n$ with probability at least* 0.9.

Proof. Let us temporarily remove the non-degeneracy condition (1.1), so that a,b,c,d,e now become independent random variables. It is clear that any point $(x,y) \in \{0,\dots,n-1\}^2$ will now lie in A with probability $1/p$, just from the randomness of e alone. In fact, any two distinct points $(x,y),(x',y') \in \{0,\dots,n-1\}^2$ will both lie in A with a joint probability of $1/p^2$, from the randomness of c,d,e (since $(x,y,1)$ and $(x',y',1)$ are linearly independent in $\mathbf{F}_p$); thus the events $(x,y) \in A$ are pairwise independent. This implies that the cardinality of A has mean $n^2/p \asymp n$ and variance $O(n^2/p) = O(n)$, and hence from Chebyshev's inequality, A will have cardinality $n^2/p + O(\sqrt{n})$ with probability at least 0.95 (say). Conditioning to the event (1.1), we obtain the claim. □