Neural sequence generation

Sean Welleck | 03.02.2023

Motivation

Modeling and generating language

- Language encapsulates ideas.
- Factual knowledge
  - *Molly Seidel won the ______ medal in the 2020 Olympic marathon.*
- State of the art GPT-3 **language model**: Today's lecture
  - **I am a highly intelligent question answering bot.**

    Q: Who was president of the United States in 1955?

    A: Dwight D. Eisenhower was president of the United States in 1955.

    Q: Molly Seidel won which medal in the 2020 Olympic marathon?

    A: Molly Seidel won a bronze medal in the 2020 Olympic marathon.

Motivation

Modeling and generating language

- Language encapsulates ideas.
- Common sense
  - *I tipped the bottle. As a result,*
- State of the art GPT-3 language model:
  - **I will continue your sentence based on my common-sense understanding of the world:**

    I tipped the bottle. As a result, the drink spilled out.

Motivation

Modeling and generating language

- Language encapsulates ideas.
- Logical reasoning
  - *Alice purchased three widgets, and Bob purchased three times as many. In total, Alice and Bob purchased* $\boxed{}$ *widgets.*
- State of the art GPT-3 language model:
  - **I can solve numerical reasoning problems.**

    Problem: The dog had four meals every day for three weeks.
    Answer: In total, the dog had 4 * 7 * 3 meals.

    Problem: Two students worked for 8 hours, and a third student worked for 3 hours.
    Answer: In total, the students worked 2 * 8 + 3 hours.

    Problem: Alice purchased three widgets, and Bob purchased three times as many.
    Answer: In total, Alice and Bob purchased 9 widgets.

AI is not solved yet

Motivation

Modeling and generating language

- **Generating** language is **useful**.
- Machine translation
    - **Translate this into 1. French, 2. Spanish and 3. Japanese:**

      What rooms do you have available?

      1. Quels sont les chambres disponibles?
      2. ¿Cuáles son las habitaciones disponibles?
      3. 何とか部屋がありますか?

Motivation

Modeling and generating language

- **Generating** language is **useful**.
- Dialogue
    - **You: What have you been up to?**
      Friend: Watching old movies.
      You: Did you watch anything interesting?
      Friend: Yes, I watched The Omen and Troy.

Motivation

Modeling and generating sequences (text, code, …)

- **Generating** ~~language~~ *sequences* is **useful**.
- Programming assistants

```
# Python 3.7

def randomly_split_dataset(folder, filename, split_ratio=[0.8, 0.2]):
    df = pd.read_json(folder + filename, lines=True)
    train_name, test_name = "train.jsonl", "test.jsonl"
    df_train, df_test = train_test_split(df, test_size=split_ratio[1], random_state=42)
    df_train.to_json(folder + train_name, orient='records', lines=True)
    df_test.to_json(folder + test_name, orient='records', lines=True)
randomly_split_dataset('finetune_data/', 'dataset.jsonl')

# An elaborate, high quality docstring for the above function:
"""
```

```
    Splits a dataset into train and test sets.

    Args:
        folder (str): The folder where the dataset is located.
        filename (str): The name of the dataset file.
        split_ratio (list): The ratio of train/test split.

    Returns:
        None
```

Motivation

Modeling and generating sequences (text, code, …)

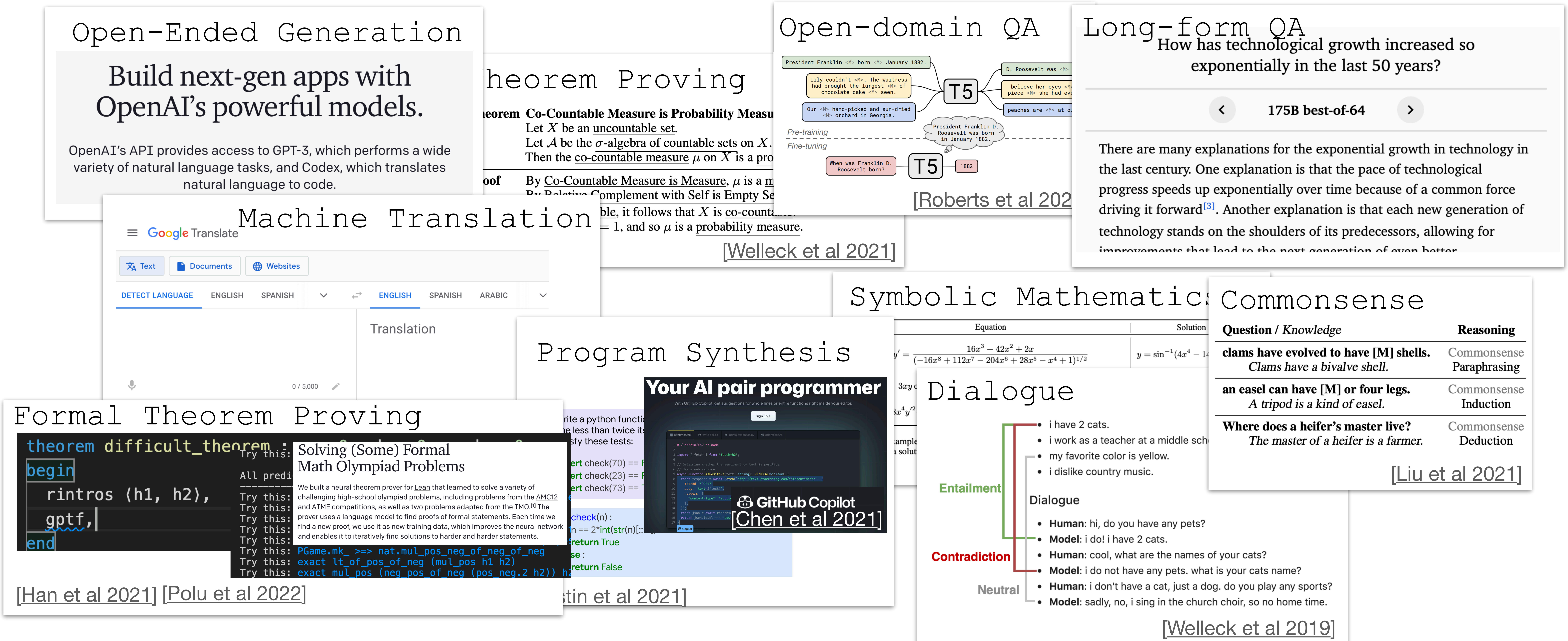
- **Generating** ~~language~~ *sequences* is **useful**.
- Education
  - **I'm an intelligent tutor. Tell me where you're stuck and I'll give you a hint.**

    Q: I'm having trouble proving that the sum of two odd numbers is even.
    A: Make the sum of two odd numbers into the form 2k. Finally, use the definition of an even number.

    Q: I'm having trouble proving that if x is even, x + 5 is odd.
    A: Use a proof by contradiction. Assume that x + 5 is even. This means that x + 5 can be written as 2k for some integer k. Now, subtract 5 from each side of the equation. This gives us x = 2k - 5. But this is a contradiction because x is even and 2k - 5 is odd.

Today’s lecture

A common *language modeling* recipe underlies all of these applications.



Today's lecture
Modeling and generating sequences

- **A common *language modeling* recipe underlies all of these applications.**
- Today's lecture:
  - What is a language model?
  - Generating sequences with a language model.

Modeling	Learning	Decoding

What is a language model?

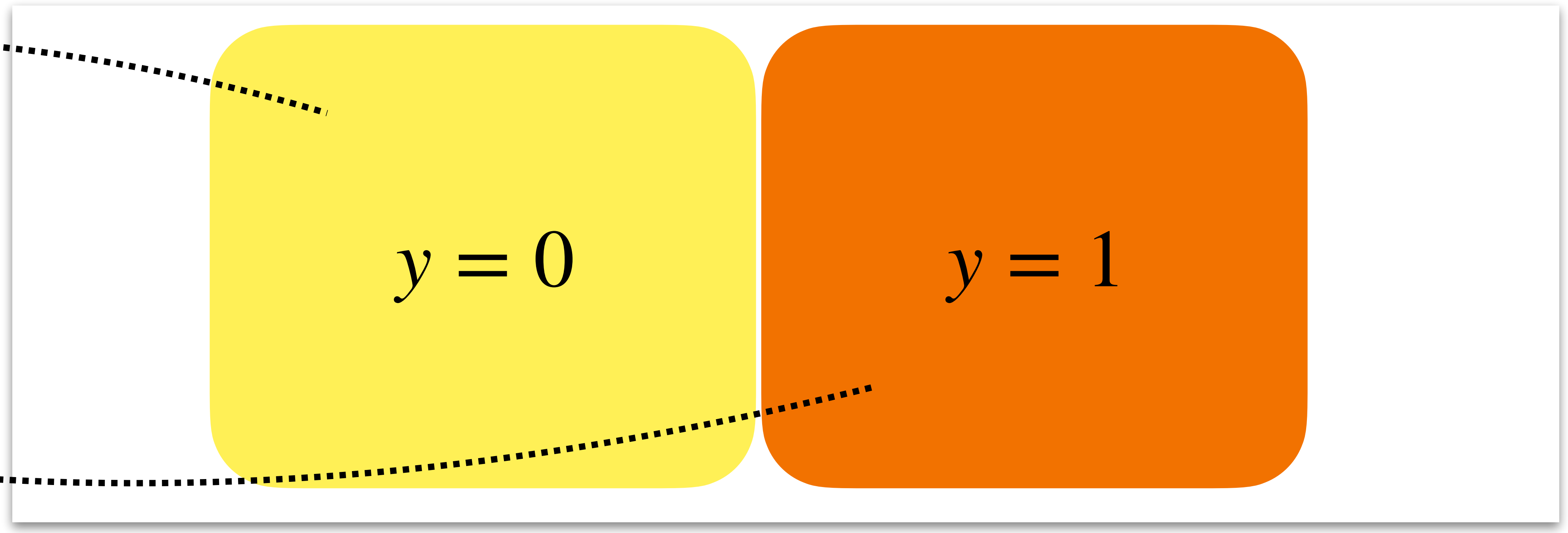
Setup

- $\mathbf{y}_{1:T} = (y_1, y_2, \ldots, y_T)$, where $y_t \in V$
  - $\mathbf{y}_{1:T}$ **sequence** e.g. `the cat sat .` T can vary.
  - y_t **token** e.g. `cat`
  - V **vocabulary** e.g. `{a, b, …, zebra, …}`
- $\mathbf{y} \in \mathcal{Y}$, $\mathcal{Y}$ set of all sequences

What is a language model?

Language model

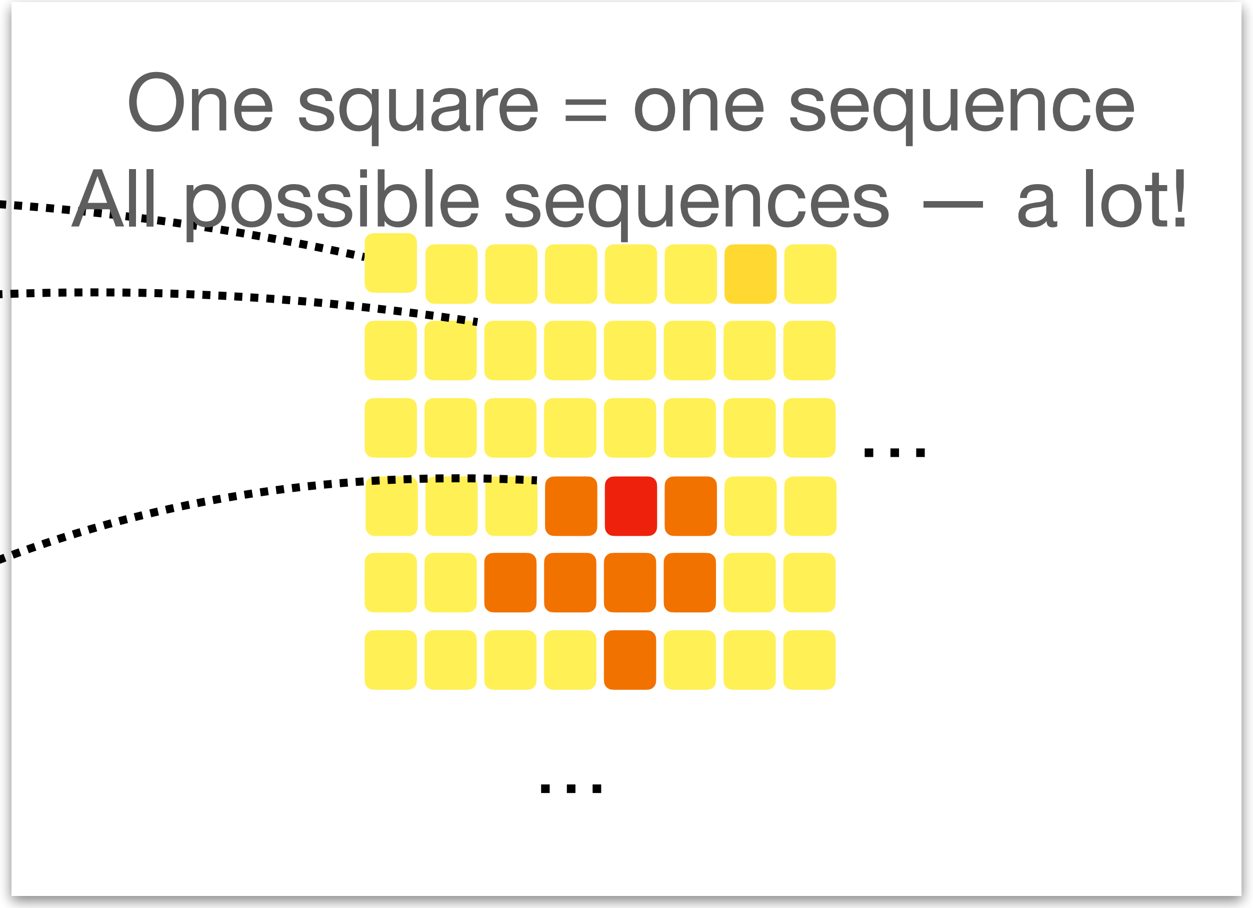
- A language model is a probability distribution over all sequences
    - $p(\mathbf{y})$
- Example probability distribution: **biased coin**
    - $p(y) = \begin{cases} 0.4 & y \text{ is } 0 \\ 0.6 & y \text{ is } 1 \end{cases}$



What is a language model?

Language model

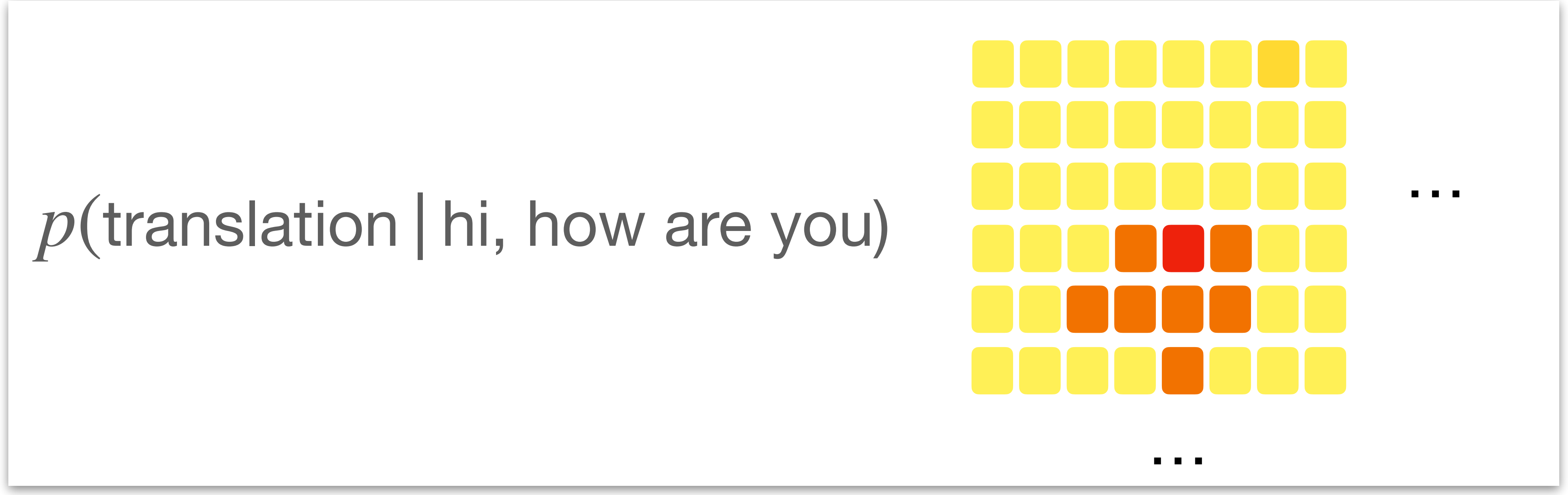
- A language model is a probability distribution over all sequences
    - $p(\mathbf{y})$
- Example language model
    - $p(\mathbf{y}) =$ 0.000013 if $\mathbf{y}$ is `a.`
      0.000001 if $\mathbf{y}$ is `aa.`
      …
      0.019100 if $\mathbf{y}$ is `a cat sat.`
      …

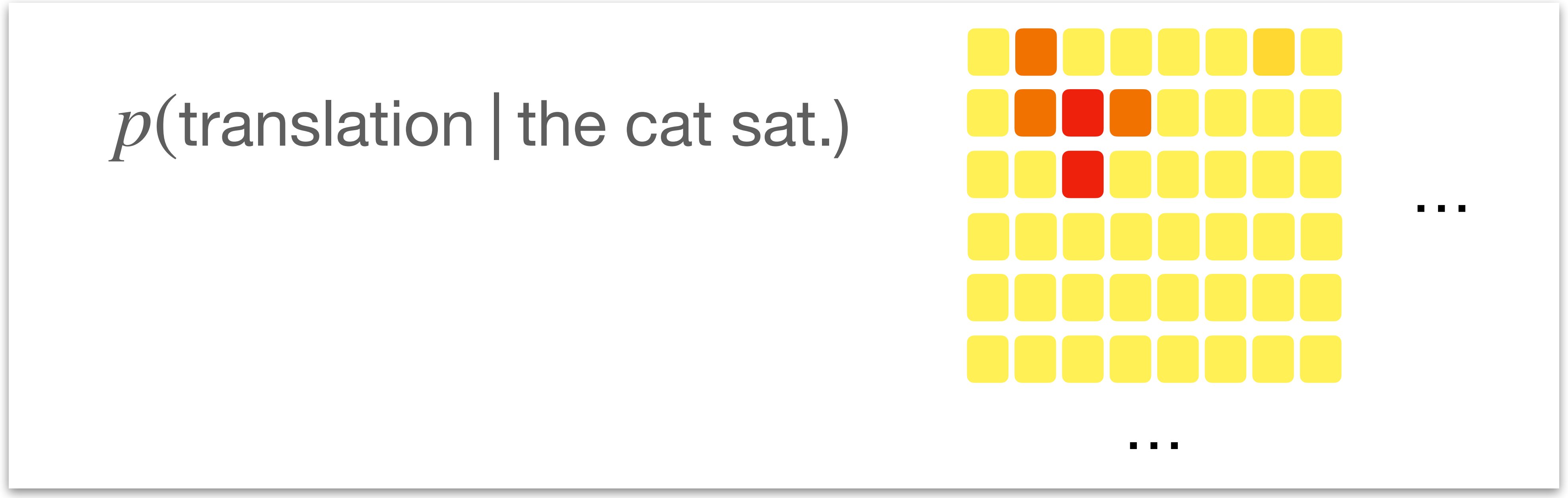


What is a language model?

Sequence-to-sequence with a language model

- A language model can accept an **input** by conditioning on an input prefix ('prompt'):
  - $p(\mathbf{y}_{k+1:T} | \mathbf{y}_{1:k})$
  - Machine translation:
    - Prefix : sentence in English
      Continuation : sentence in Japanese
  - General task:
    - Prefix : instructions, examples, start of output
      Continuation: output





What is a language model?

Sequence-to-sequence with a language model

- A language model can accept an **input** by conditioning on an input prefix :
  - $p(\mathbf{y}_{k+1:T} \mid \mathbf{y}_{1:k})$
  - Machine translation:
    - Prefix : sentence in English
      Continuation : sentence in Japanese
  - General task:
    - Prefix : instructions, examples, start of output
      Continuation: output

Translate this into 1. French, 2. Spanish and 3. Japanese:

What rooms do you have available?

1. Quels sont les chambres disponibles?
2. ¿Cuáles son las habitaciones disponibles?
3. 何とか部屋がありますか?

- How do we *learn* a language model from data?
- How do we *generate* text from a language model?

Modeling

The building blocks | Modeling

Autoregressive language model

- First step: use the chain rule of probability:

- $$p(\mathbf{y}_{1:T}) = \prod_{t=1}^{T} p(y_t \mid \mathbf{y}_{<t})$$

  - $p(\texttt{the cat sat <end>}) =$
    $p(\texttt{the})\,p(\texttt{cat}\mid\texttt{the})\,p(\texttt{sat}\mid\texttt{the cat})\,p(\texttt{<end>}\mid\texttt{the cat sat})$

Modeling

The building blocks | Modeling

Autoregressive language model

- Language modeling is reduced to **classification**

- $$p(\mathbf{y}_{1:T}) = \prod_{t=1}^{T} p(\underbrace{y_t}_{\text{Next Token}} \mid \underbrace{\mathbf{y}_{<t}}_{\text{Previous Tokens}})$$

  - $p(\texttt{the cat sat <end>}) = p(\texttt{the})p(\texttt{cat} \mid \texttt{the})p(\texttt{sat} \mid \texttt{the cat})p(\texttt{<end>} \mid \texttt{the cat sat})$

- Sequence probability =
  product of **next-token** probabilities

Modeling

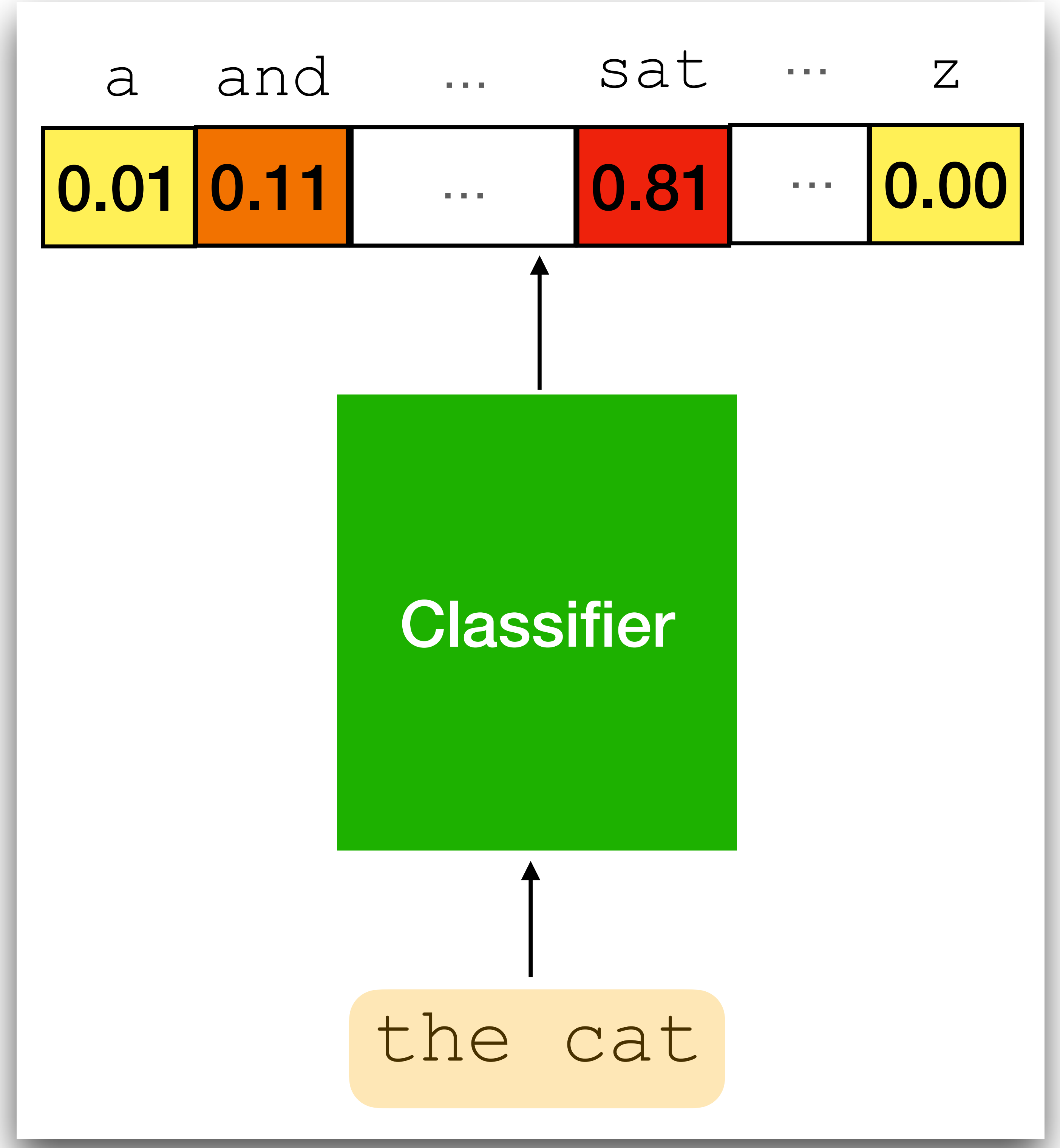
The building blocks | Modeling

Autoregressive language model

- Language modeling is reduced to **classification**

- $$p(\mathbf{y}_{1:T}) = \prod_{t=1}^{T} p(\underbrace{y_t}_{\text{Next Token}} \mid \underbrace{\mathbf{y}_{<t}}_{\text{Previous Tokens}})$$

- $p(y_t \mid \mathbf{y}_{<t})$
  - Input: $\mathbf{y}_{<t} \in V \times V \times \ldots V$
    Output: probability distribution over V
  - Target: $y_t \in V$



The building blocks | Modeling

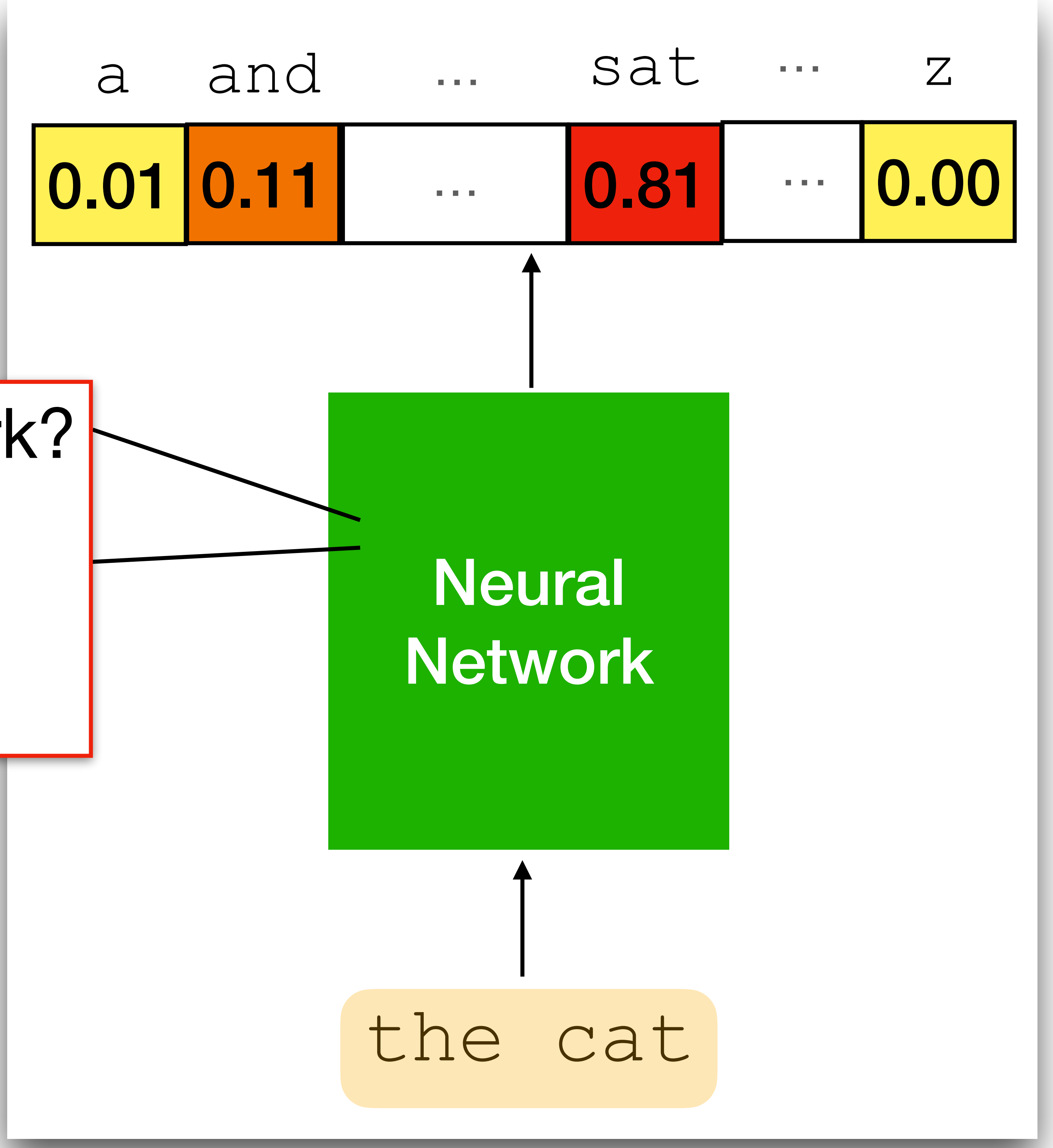
Neural autoregressive language model

- Use a neural network for language modeling

- $$p_\theta(\mathbf{y}_{1:T}) = \prod_{t=1}^{T} p_\theta(\underbrace{y_t}_{\text{Next Token}} \mid \underbrace{\mathbf{y}_{<t}}_{\text{Previous Tokens}})$$

- $p_\theta(y_t \mid \mathbf{y}_{<t})$
    - Input: $\mathbf{y}_{<t} \in V \times V \times \ldots V$
      Output: probability distribution over V
    - Target: $y_t \in V$

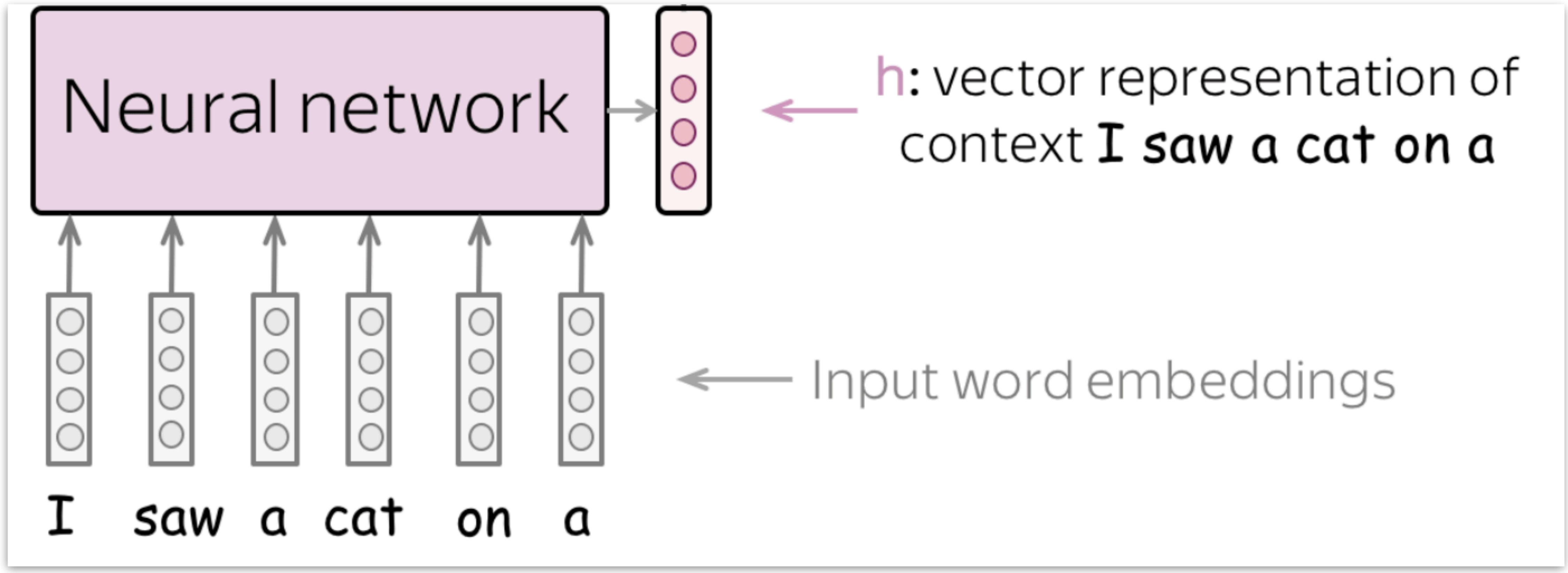
- What kind of neural network?
- How do we learn the parameters θ?

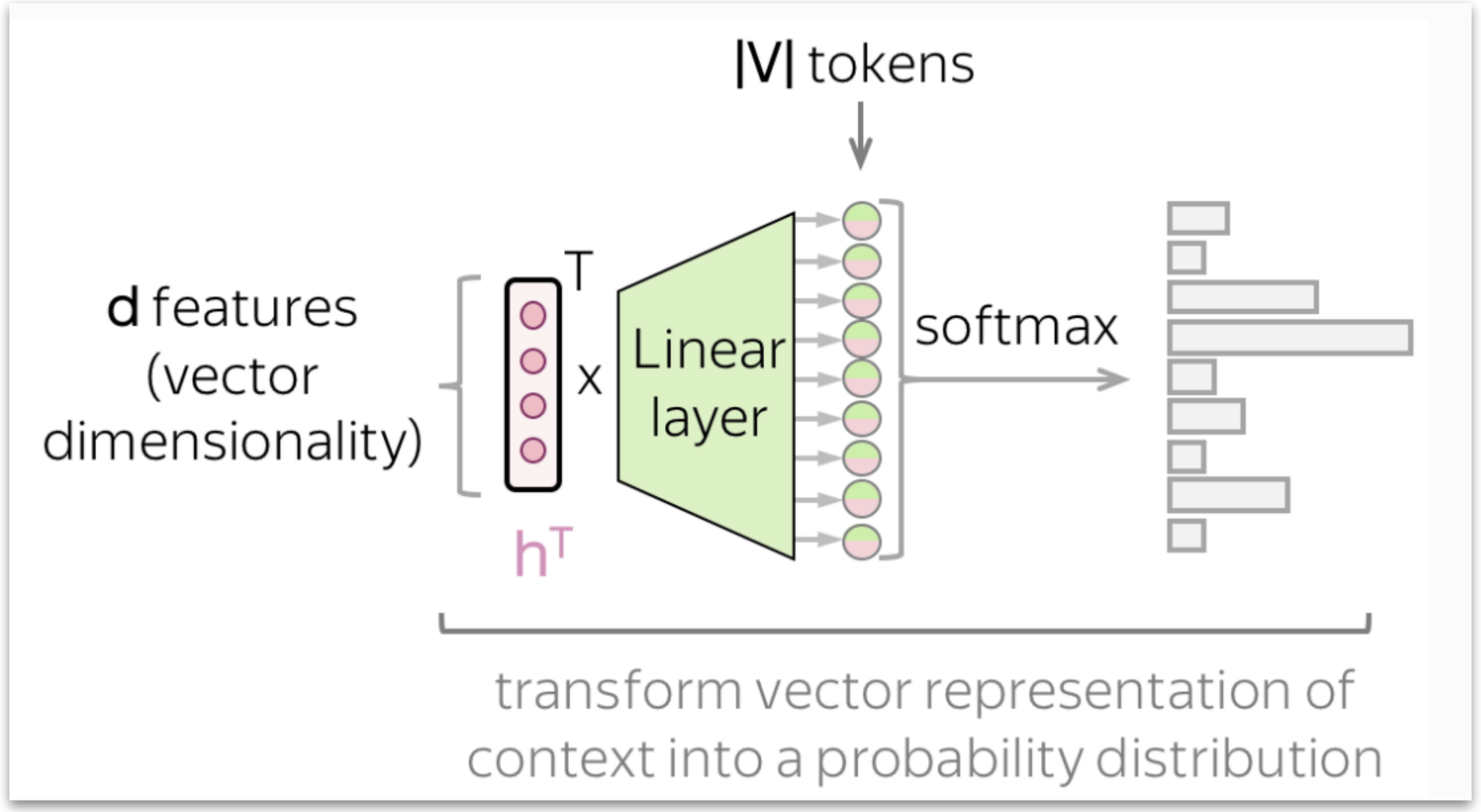


The building blocks | Modeling

What kind of neural network?

- Want: $p_\theta(y_t \mid y_1, \ldots, y_{t-1})$
  - Encode context into a vector:
    - $h_t = f_\theta(y_1, \ldots, y_{t-1}),\, h_t \in \mathbb{R}^d$
  - Transform into |V| token scores:
    - $s_t = Eh_t$, where $s_t \in \mathbb{R}^{|V|}, E \in \mathbb{R}^{(|V| \times d)}$
  - Take the softmax to get a probability vector
    - $p_\theta(\cdot \mid y_1, \ldots, y_{t-1}) = \text{softmax}(s_t)$



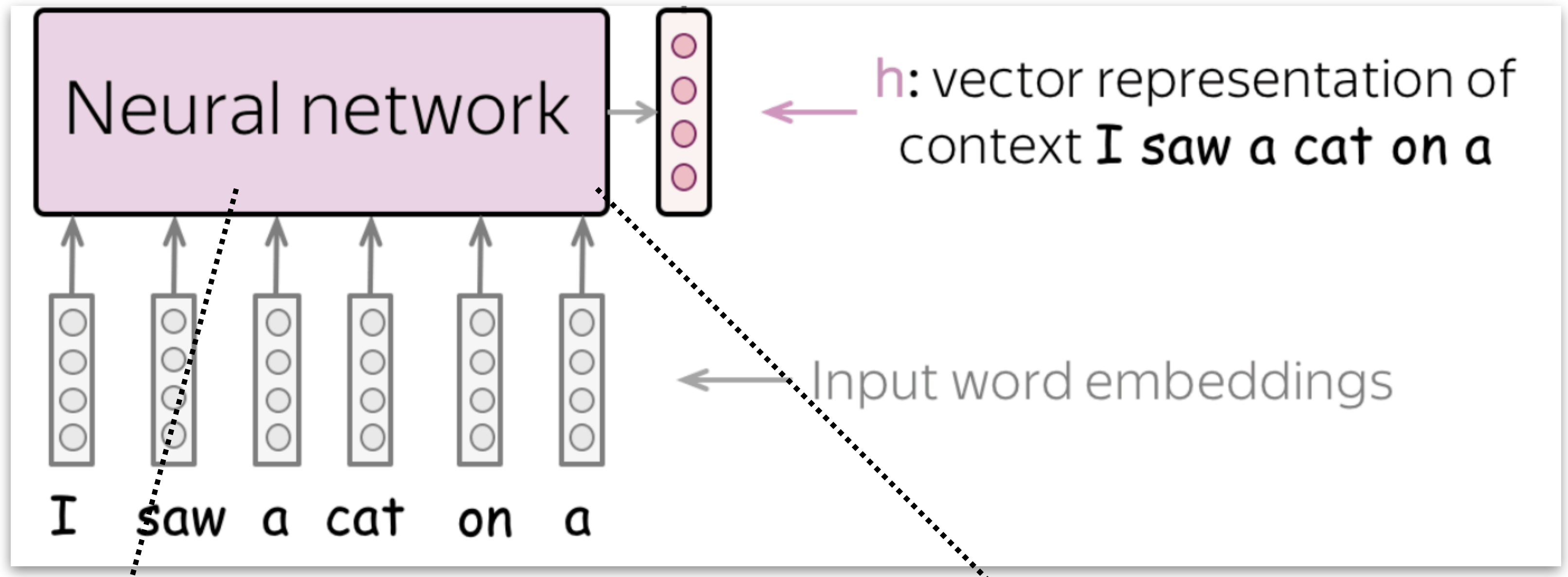


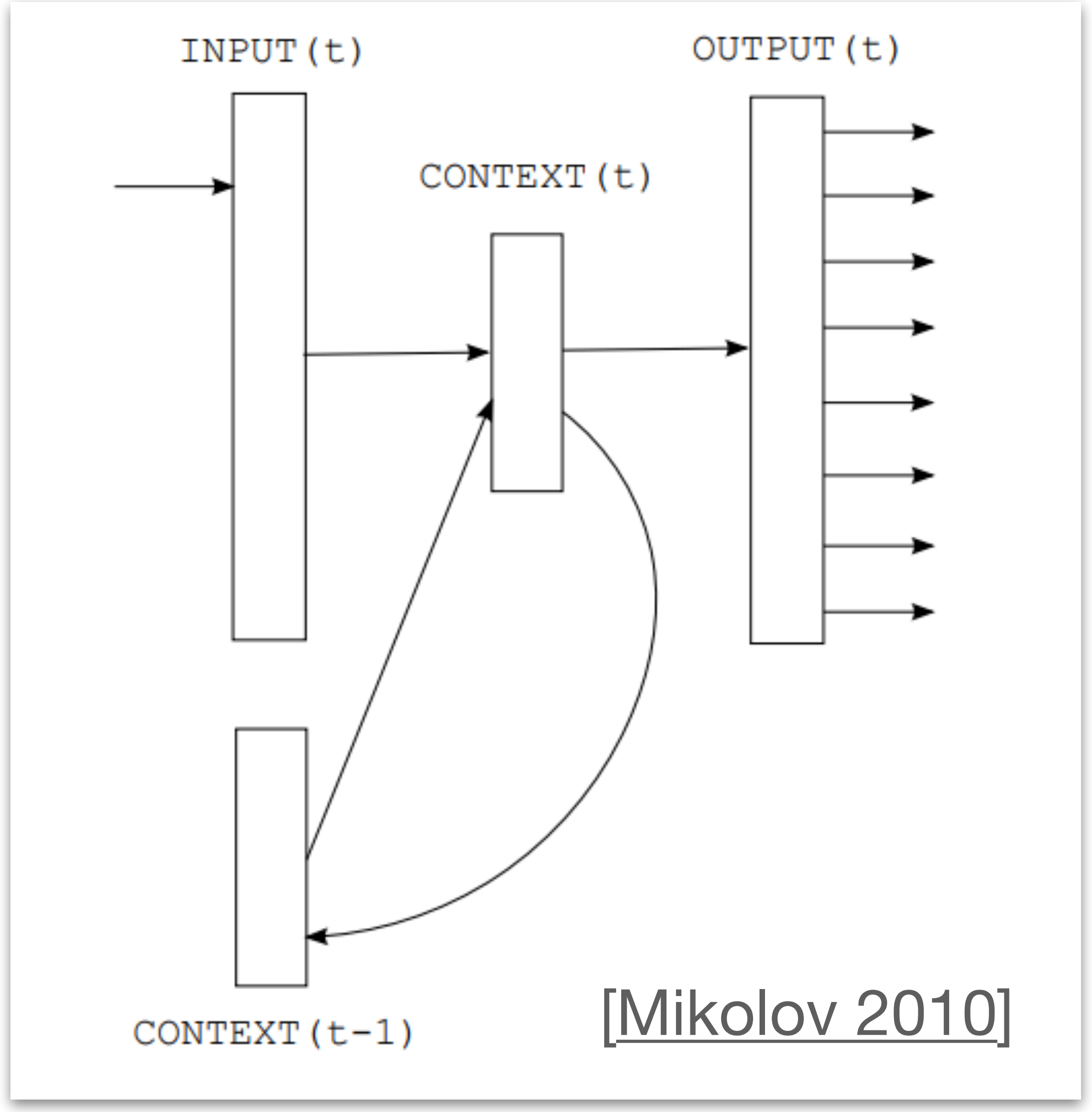
Diagrams: https://lena-voita.github.io/nlp_course/language_modeling.html

The building blocks | Modeling

What kind of neural network?

- Common choices for the neural network:
    - Recurrent neural network
    - Feedforward + attention (transformer)
- *Further details are out of scope for this lecture*!





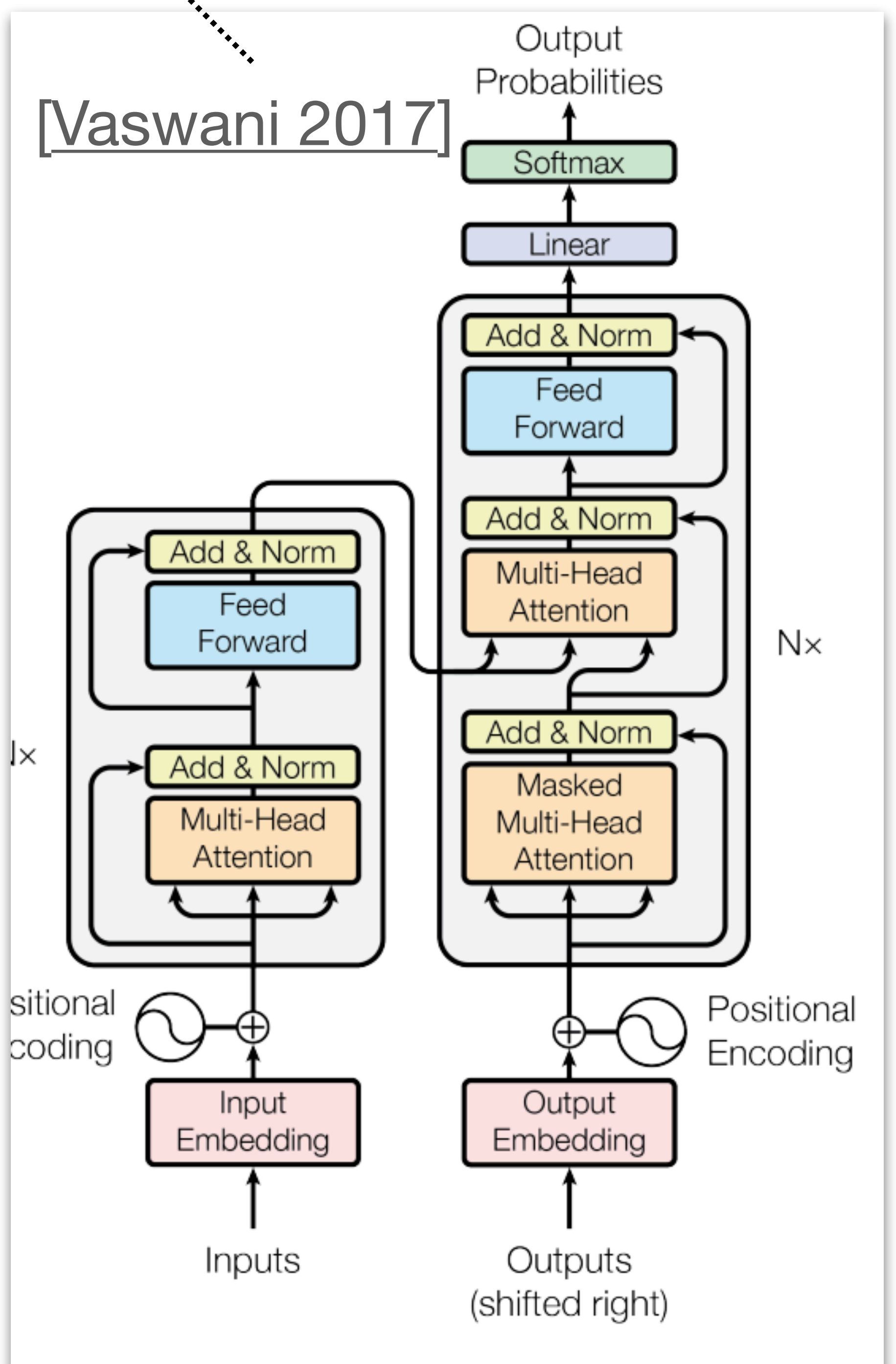


Figure 1: The Transformer - model architecture.

The building blocks | Learning

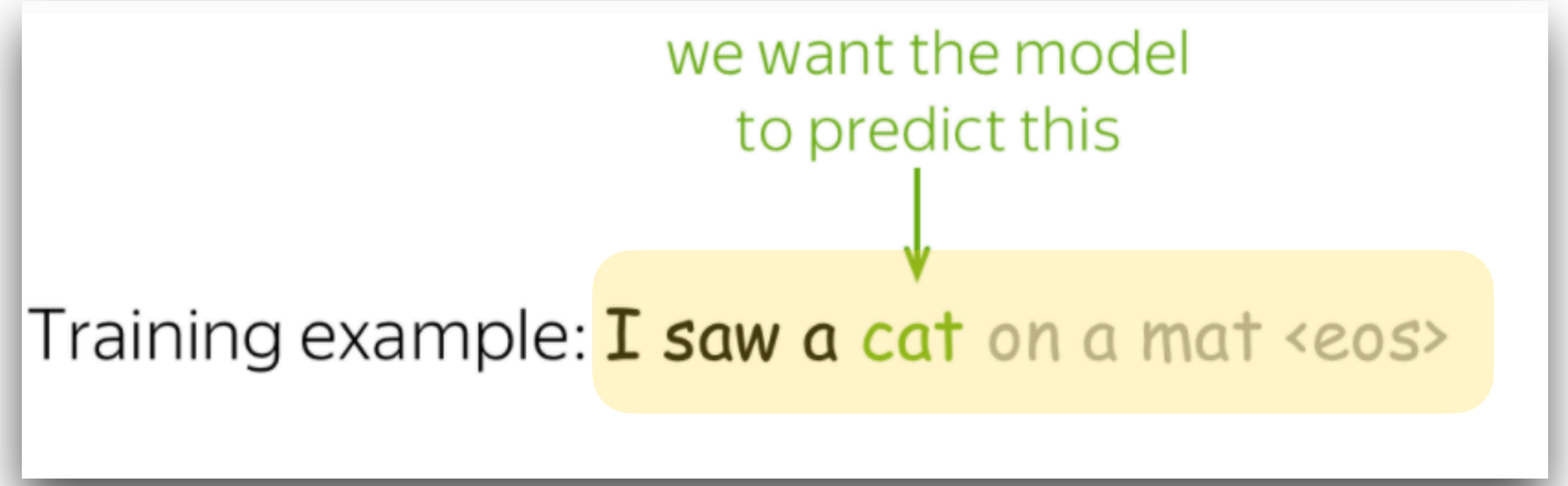
How do we learn the parameters θ?

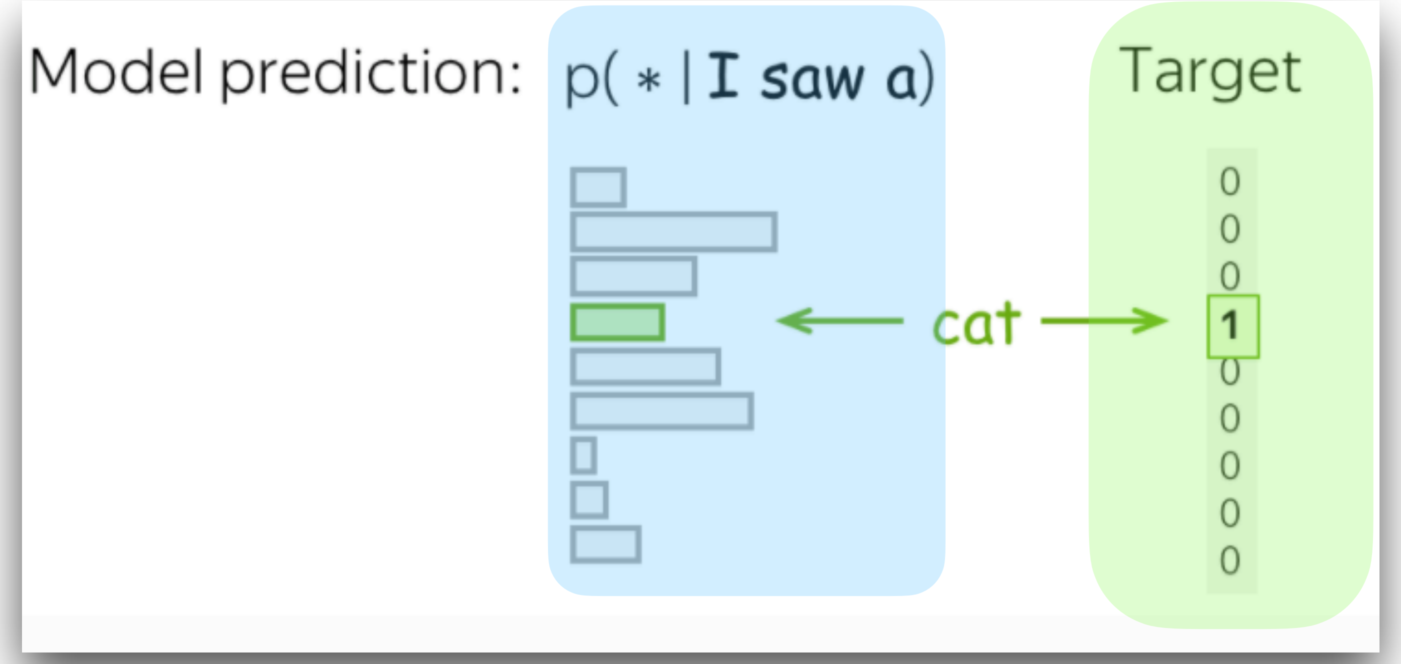
- Collect a dataset of sequences $D = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$
  - D: a book
  - D: all text on the internet
  - …
- Tokenize each sequence, $\mathbf{y}_i = (y_1, \ldots, y_{T_i})$
  - We'll see this concretely in the lab!

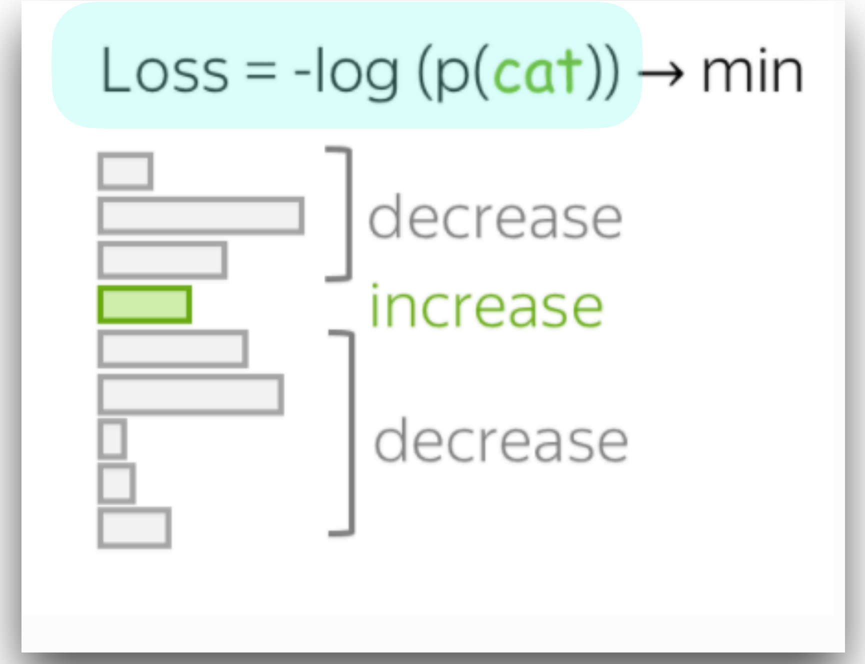
The building blocks | Learning

How do we learn the parameters θ?

- For each training sequence $\mathbf{y} = (y_1, \ldots, y_T)$ and step t:
  - Model predicts $p_\theta^t(\cdot \mid \mathbf{y}_{<t}) \in \Delta^V$
  - Target is $p_*^t = \begin{cases} 1 & y_t \\ 0 & \text{otherwise} \end{cases} \in \Delta^V$
  - Use **cross-entropy loss**:
  - $$\mathscr{L}_t = -\sum_{y \in V} p_*^t(y) \log p_\theta^t(y \mid \mathbf{y}_{<t})$$ $$= -\log p_\theta^t(y_t \mid \mathbf{y}_{<t})$$







Diagrams: https://lena-voita.github.io/nlp_course/language_modeling.html

The building blocks | Learning

Why cross-entropy loss?

- **Classifier view:**
  - We've used cross-entropy loss to train classifiers previously in the course…
- **Estimation view:** Loss summed over the entire dataset:

$$\min_{\theta} - \sum_{\mathbf{y} \in D} \sum_{t} \log p_{\theta}(y_t \mid \mathbf{y}_{<t})$$

  - $\equiv \max_{\theta} \sum_{\mathbf{y} \in D} \log p_{\theta}(\mathbf{y})$
  - Finds parameters that make the observed data *D* most probable; i.e. **maximum likelihood estimation**

Learning

The building blocks | Learning

Why cross-entropy loss? | Distribution matching view

- Makes p_θ match an underlying 'true' distribution $p_*(\mathbf{y})$ E.g. distribution that generated all internet text…

$$\min_\theta D_{KL}(p_* || p_\theta) = \min_\theta - \sum_{\mathbf{y}\in\mathcal{Y}} p_*(\mathbf{y}) \log \frac{p_\theta(\mathbf{y})}{p_*(\mathbf{y})}$$

$$\equiv \min_\theta - \sum_{\mathbf{y}\in\mathcal{Y}} p_*(\mathbf{y}) \log p_\theta(\mathbf{y}) + \text{const}$$

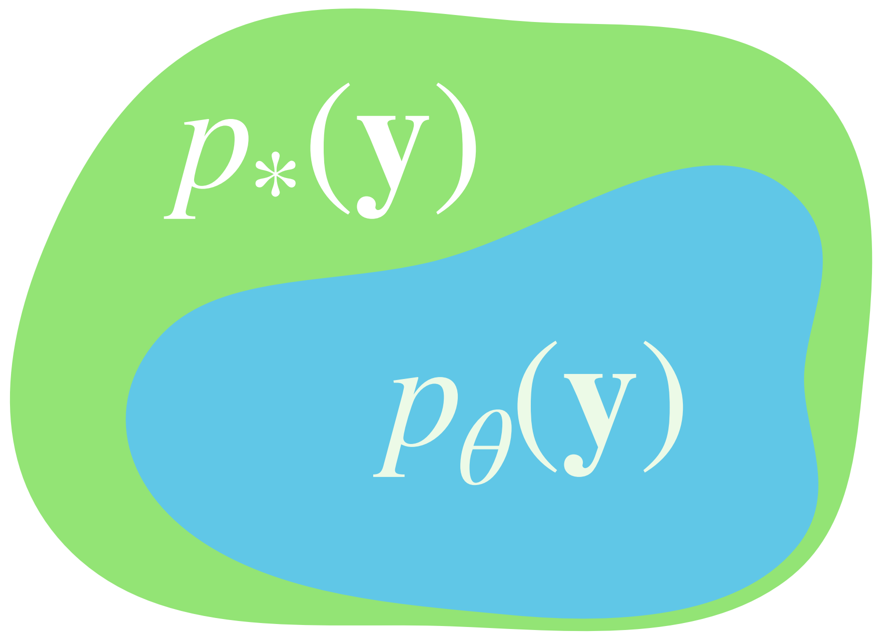
$$= -\mathbb{E}_{\mathbf{y}\sim p_*} \log p_\theta(\mathbf{y})$$

Definition of expected value

$$\approx \min_\theta - \frac{1}{|D|} \sum_{\mathbf{y}\in D} \log p_\theta(\mathbf{y})$$

"Monte-Carlo" approximation of expected value

$$\equiv \max_\theta \sum_{\mathbf{y}\in D} \log p_\theta(\mathbf{y})$$



The building blocks | Learning

Why cross-entropy loss?

- Scaling laws: more {compute, data, parameters} $\implies$ better loss

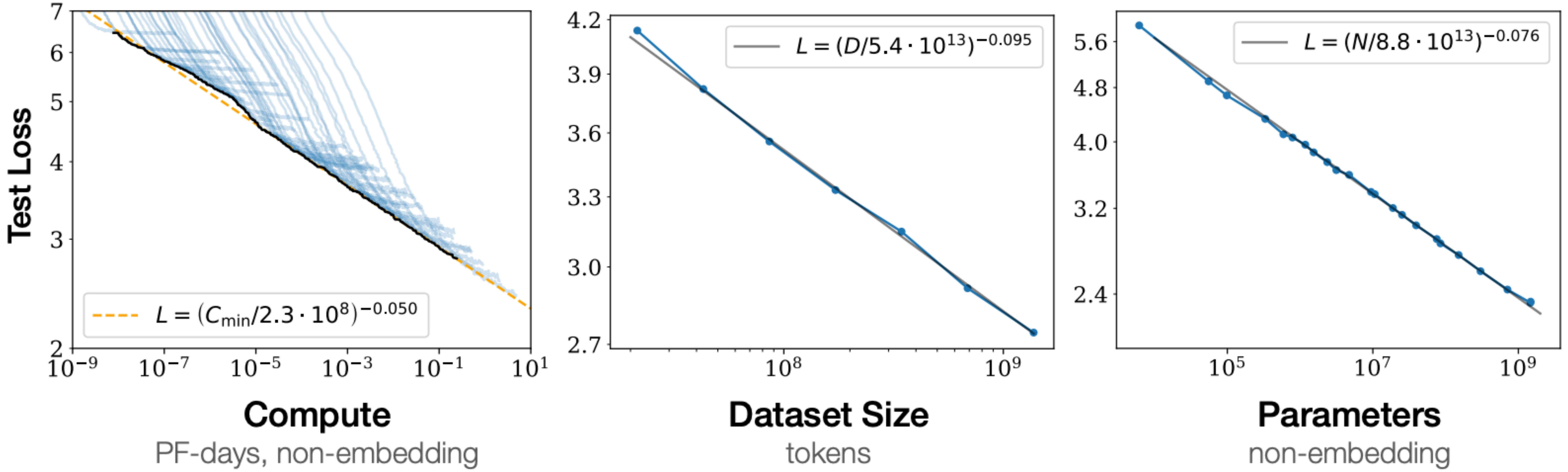


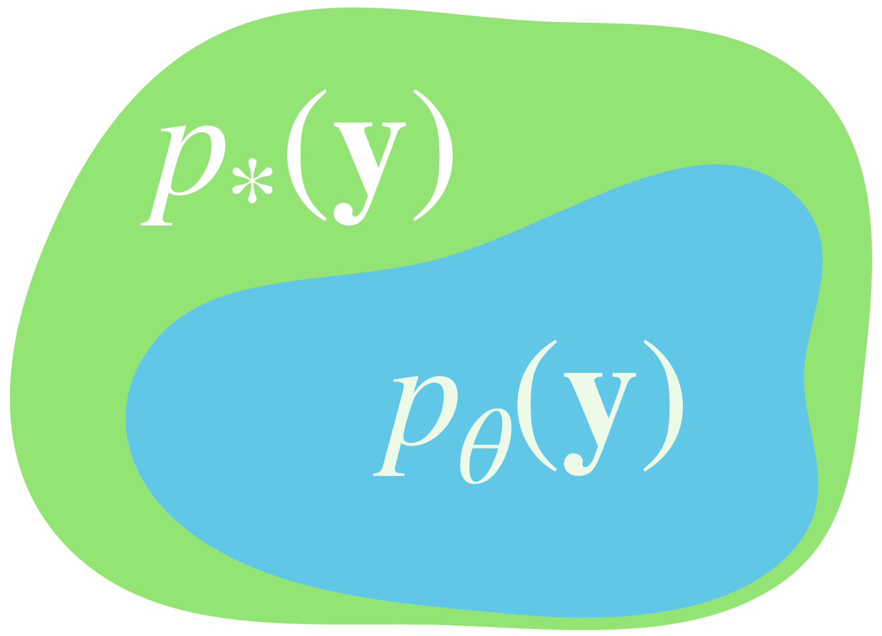
Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

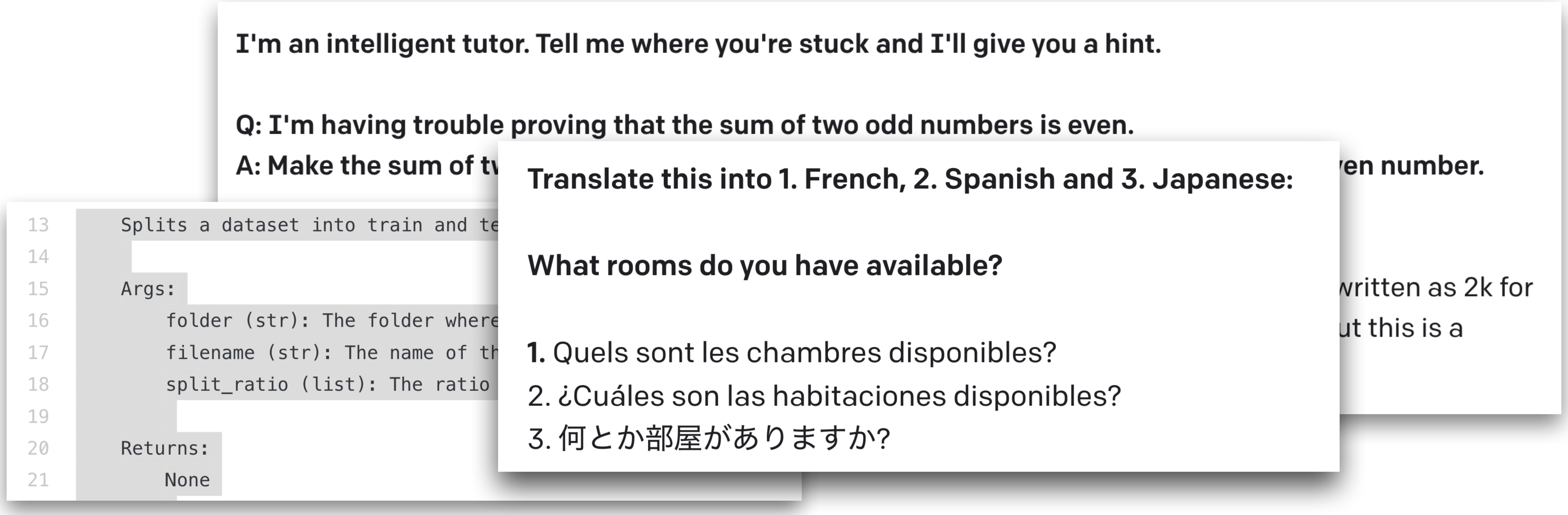
[Kaplan et al 2020]

The building blocks | Recap

Recap

- We've now *learned* a neural language *model* p_θ from data.
  - We have a distribution over all sequences.
- Next: To **generate** text, we use a *decoding* algorithm.





Building blocks | Decoding

Generating sequences from our model

- **Goal**: generate a continuation $\mathbf{y}$ given a model p_θ
- We want to generate $\mathbf{y} = (y_1, \ldots, y_T)$, starting from $y_0 = \langle start \rangle$
  - We generate one-token, feed it into the model, and repeat:
    - $y_1 =$ generate$(p_\theta(y \mid y_0))$
    - $y_2 =$ generate$(p_\theta(y \mid y_0, y_1))$
    - $y_3 =$ generate$(p_\theta(y \mid y_0, y_1, y_2))$
    - … => $(y_1, \ldots, y_T)$

Building blocks | Decoding

Generating sequences from our model

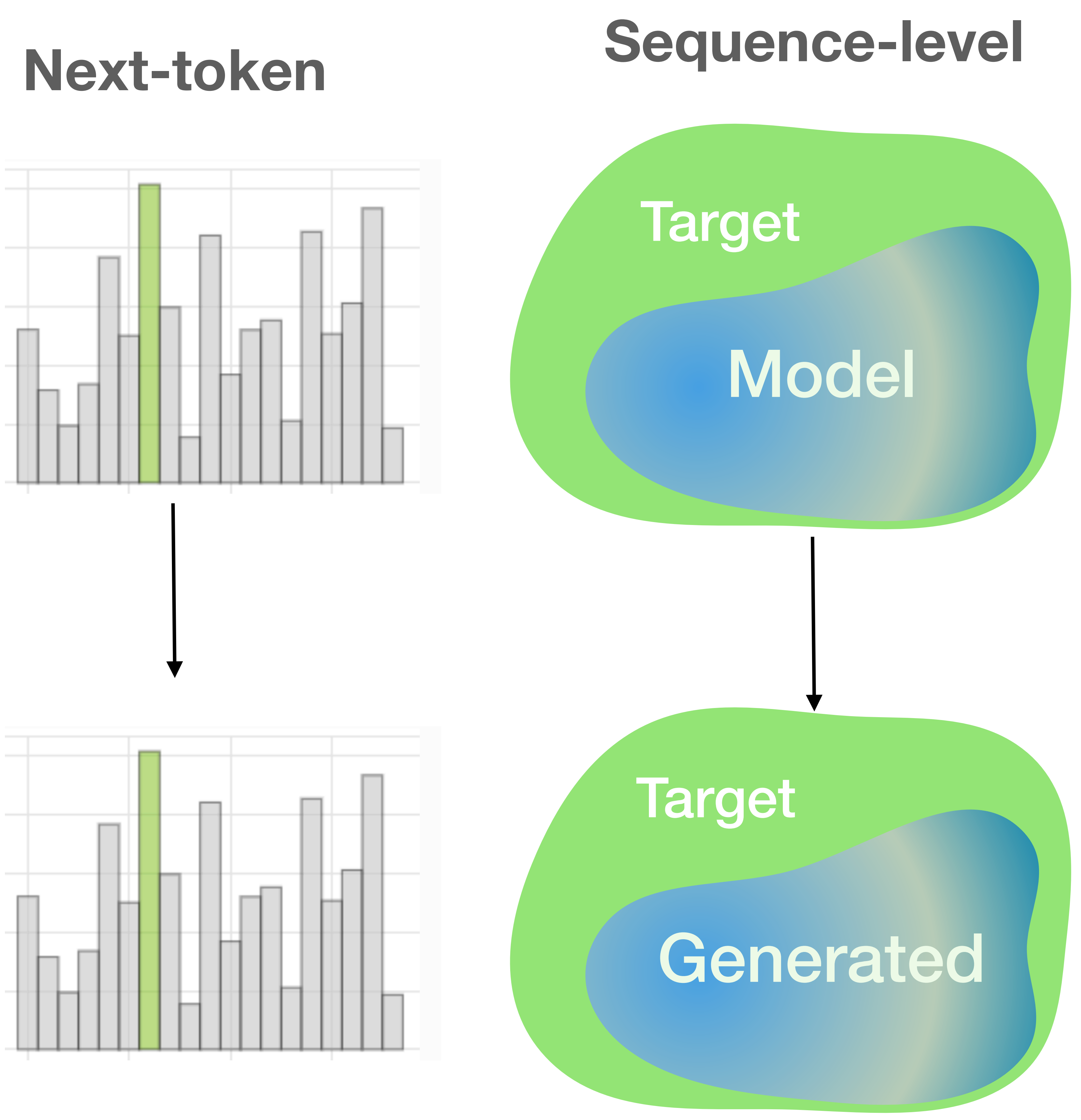
- **Goal**: generate a continuation $\mathbf{y}$ given a model p_θ and prefix $\mathbf{x}$
  - Sampling
    - $\mathbf{y} \sim p_\theta(\cdot \mid \mathbf{x})$
  - Mode-seeking
    - $\mathbf{y} = \arg\max_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})$

Building blocks | Decoding

Generating sequences from our model

Decoding

- **Ancestral sampling:** sample from the model's distribution
  - Until $y_t = \langle end \rangle$:
    - $y_t \sim p_\theta(\cdot \mid \mathbf{y}_{<t})$
- $\mathbf{y}$ is a sample from $p_\theta(\mathbf{y})$, since
  - $p_\theta(\mathbf{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \mathbf{y}_{<t})$

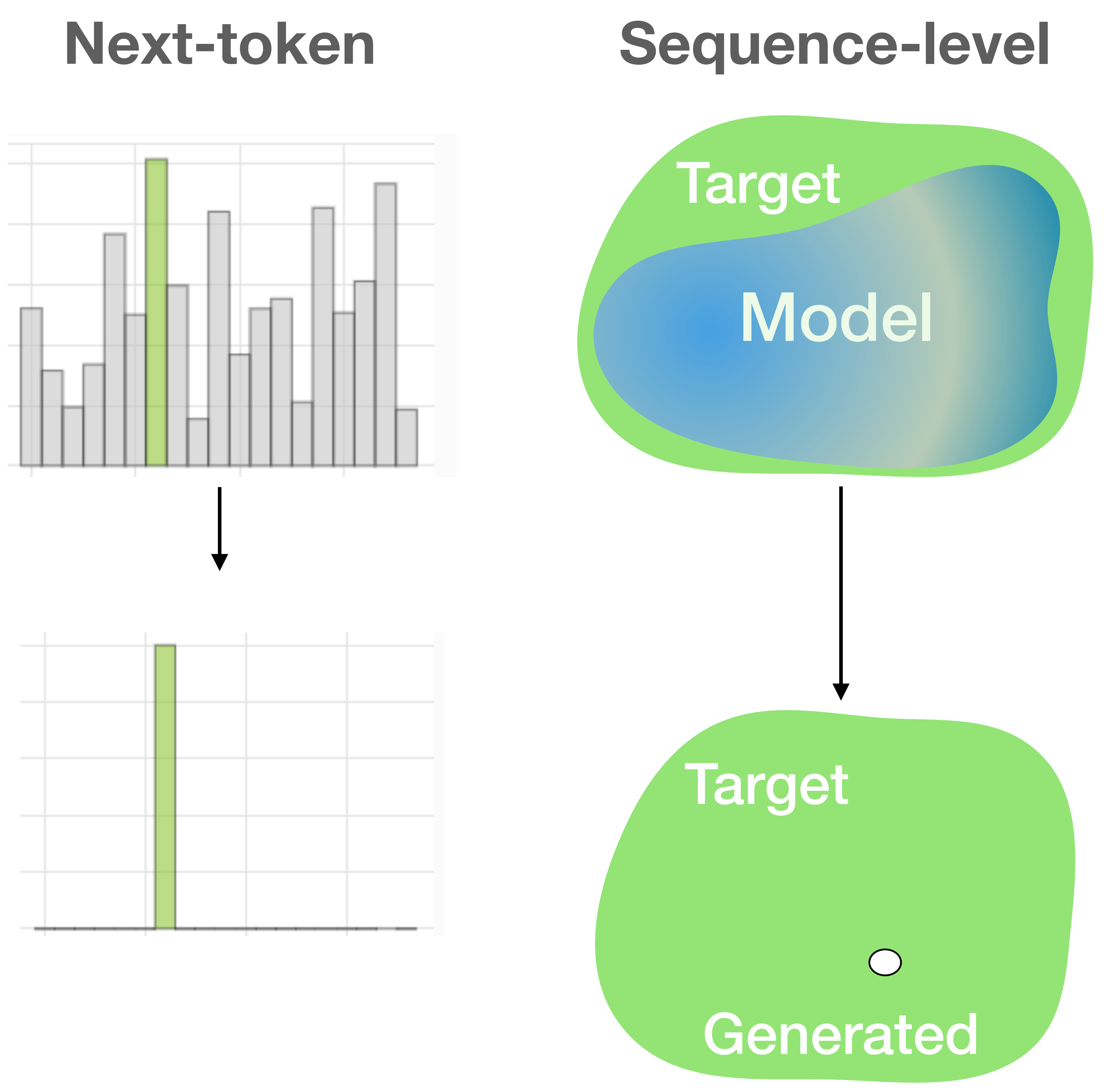


Next-token: https://lena-voita.github.io/nlp_course/language_modeling.html

Building blocks | Decoding

Generating sequences from our model

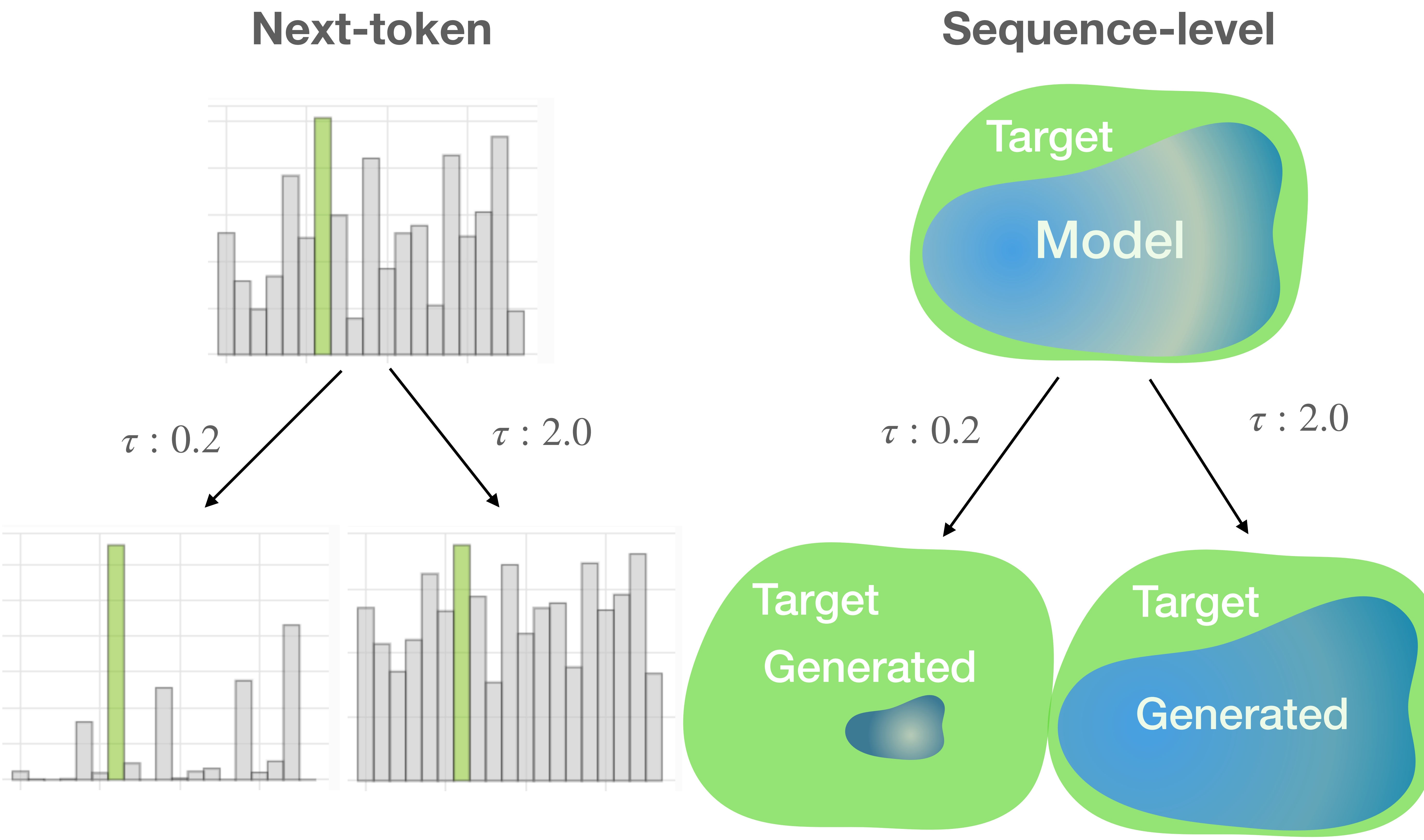
- **Greedy decoding:** select the most-probable token at each step
  - Until $y_t = \langle end \rangle$:
    - $y_t = \arg\max_{y \in V} p_\theta(\cdot \mid \mathbf{y}_{<t}, \mathbf{x})$
- $\mathbf{y}$ is a (naive) approximation of
  - $\arg\max_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})$



Building blocks | Decoding

Generating sequences from our model

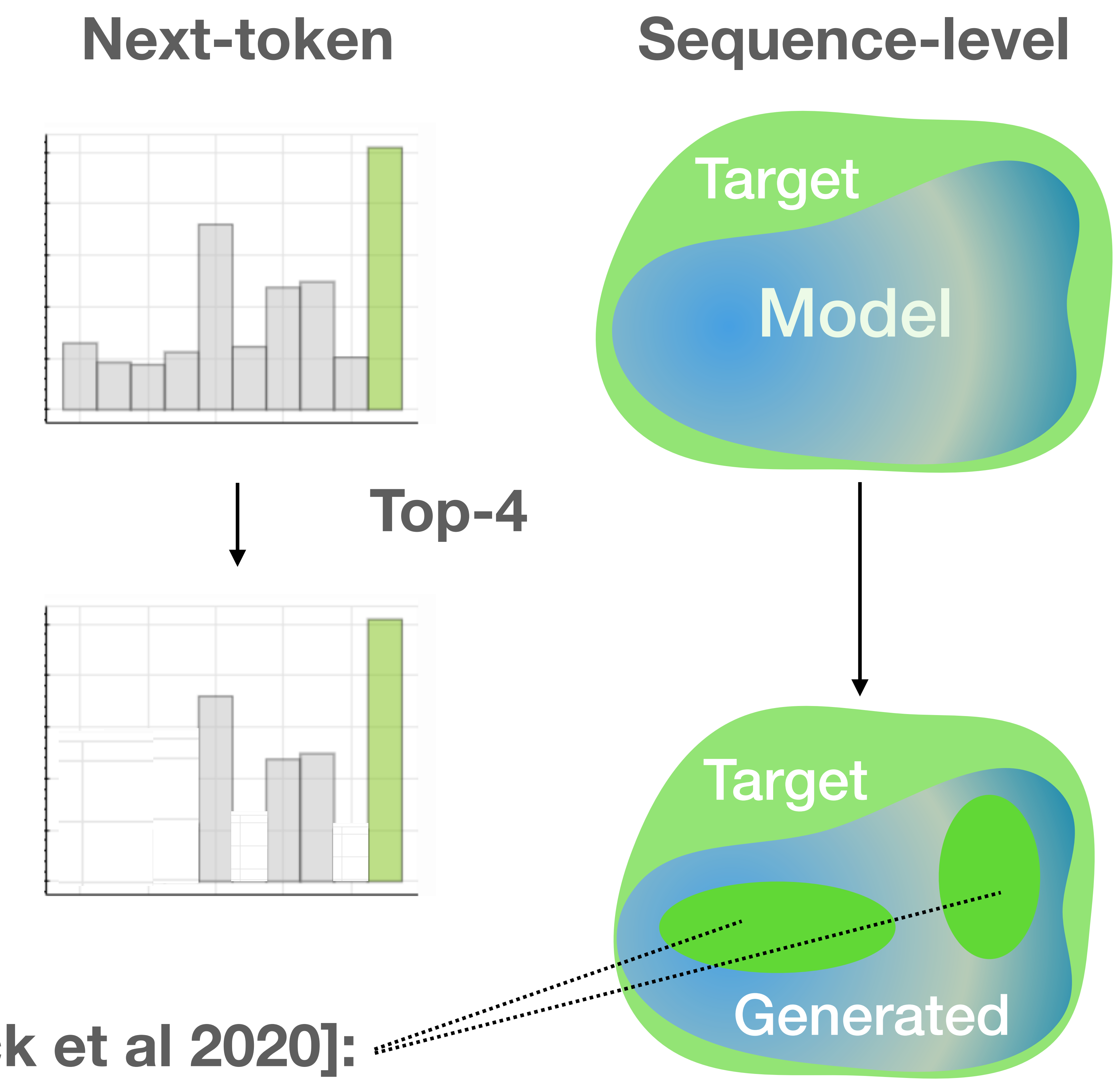
- **Temperature Sampling:** adjust each distribution & sample
  - Until $y_t = \langle end \rangle$:
    - $y_t \sim p_\theta^\tau(\cdot \mid \mathbf{y}_{<t})$
  - Where $p_\theta^\tau(\cdot \mid \ldots) = \text{softmax}(s_t/\tau)$, $\tau \in \mathbb{R}_{>0}$
- τ small: "sharpens" the distribution
  - $\tau \to 0$: greedy decoding
- τ big: "flattens" the distribution
  - $\tau \to \infty$: uniform distribution



Building blocks | Decoding

Generating sequences from our model

- **Top-k sampling:** sample from the k-most-probable tokens
    - $y_t \sim \propto \begin{cases} p_\theta(y \mid \mathbf{y}_{<t}) & y \in \text{top-k} \\ 0 & \text{otherwise} \end{cases}$
- k small: only sample from highly-ranked tokens
    - k=1: greedy decoding
    - k=|V|: ancestral sampling

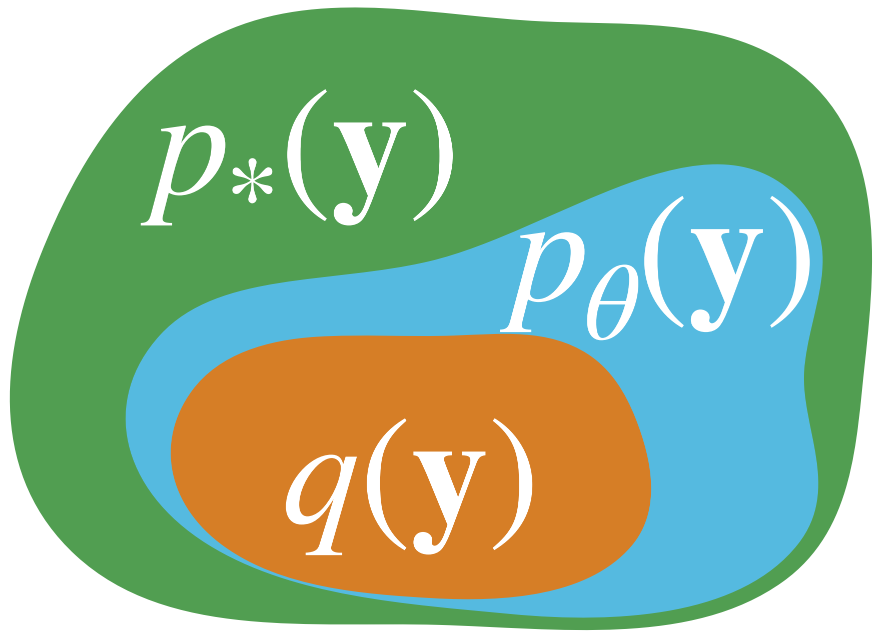


Yes- these holes are problematic, see [Welleck et al 2020]:
[Consistency of a Recurrent Language Model With Respect to Incomplete Decoding]

Building blocks | Decoding

Generating sequences from our model

- What is going on? **Distributional view**
  - Using a decoding algorithm gives us a *new generation distribution* $q(\mathbf{y} \mid p_\theta)$
    - In practice, we do this with new *per-step distributions,* $q^{(t)}(y_t \mid p_\theta, \mathbf{y}_{<t})$.
  - Varying the decoding algorithm varies the generation distribution q.
  - Generating means sampling from q.

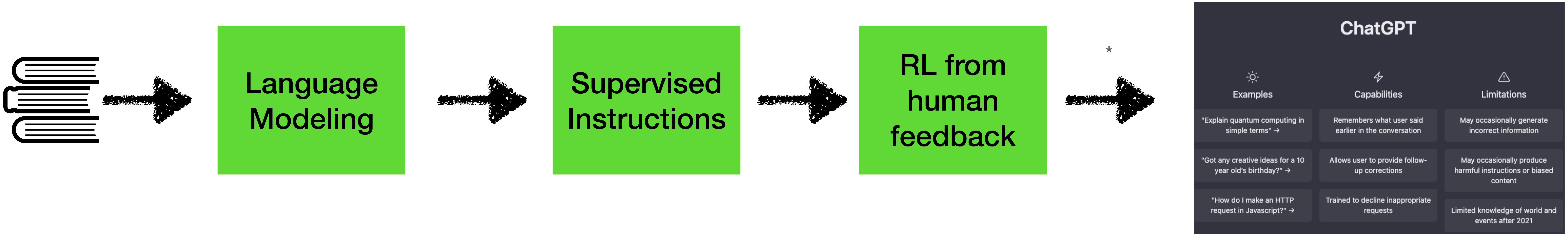


Recap
Modeling and generating sequences

- Today’s language models consist of three building blocks:
  - An *autoregressive model* that reduces language modeling to classification.
  - *Learning* the model’s parameters by maximum likelihood.
  - Generating with a *decoding* algorithm.

From distributions to assistants

- Language models are not directly trained to perform tasks
- One current solution: fine-tune them to perform tasks!

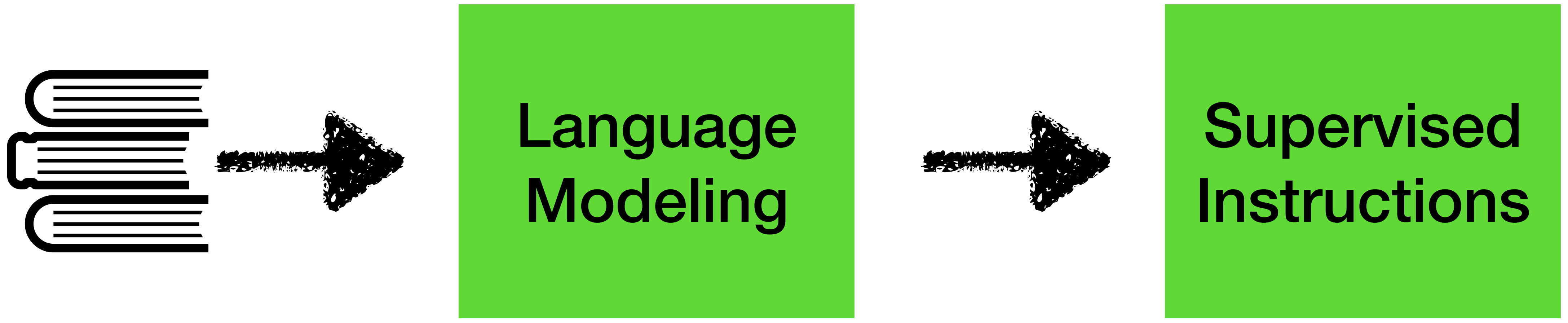


[Training language models to follow instructions with human feedback]

* we don't know the actual learning procedure for chat-gpt

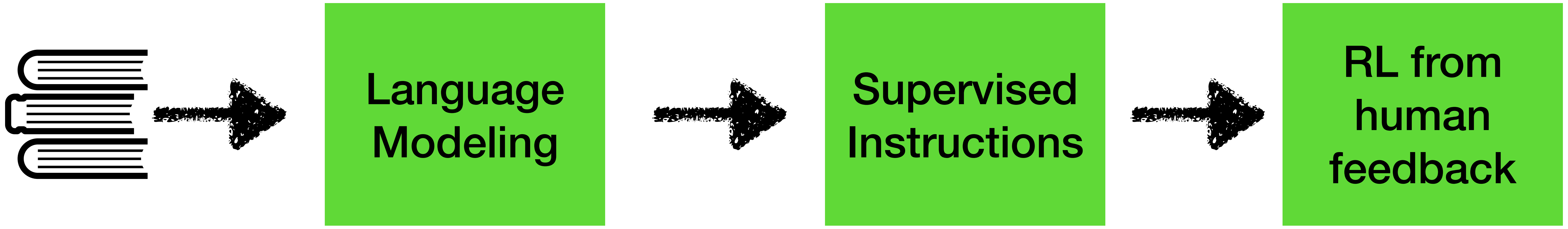
From distributions to assistants

- Collect (prompt, output) pairs written by humans for various tasks
    - E.g. “summarize this document….”, “translate this sentence…”
- Fine-tune a large language model on the pairs
    - I.e. maximize $p(\text{output} \mid \text{prompt})$

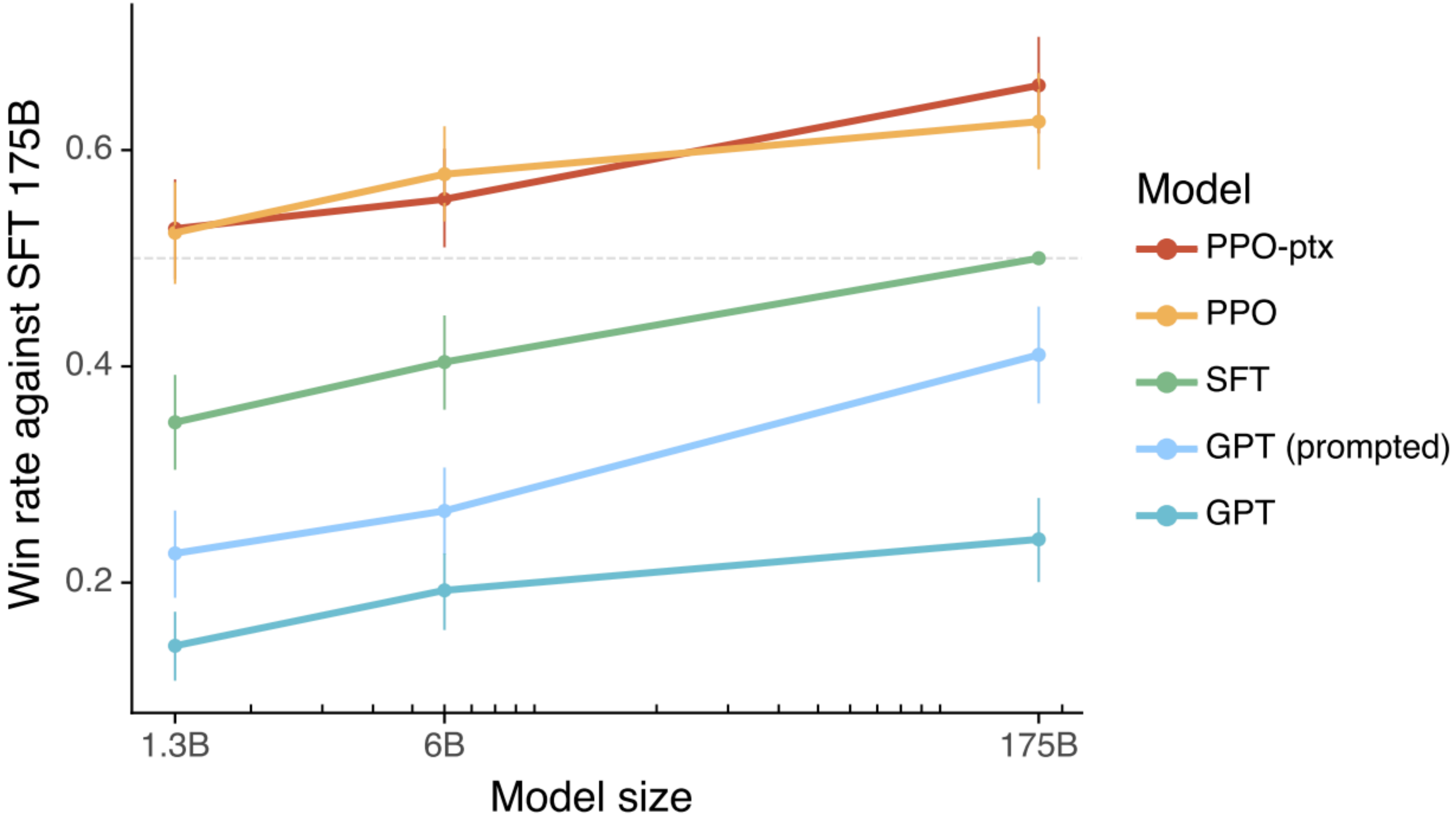


From distributions to assistants

- Generate multiple outputs for a given prompt.
- Learn a neural network that scores $p(\text{output}_i > \text{output}_j) \in [0,1]$
  - "Reward model". Pair orderings are determined by humans.
- Use a reinforcement learning algorithm to optimize the reward



From distributions to assistants



Training language models to follow instructions with human feedback

From distributions to assistants

Write a summary of the Wikipedia page on Jupyter:

https://en.wikipedia.org/wiki/Jupyter

Write a summary of the Wikipedia page on Jupyter:

https://en.wikipedia.org/wiki/Jupyter

Write a summary of the Wikipedia page on Jupyter:

https://en.wikipedia.org/wiki/Jupyter

Write a summary of the Wikipedia page on Jupyter:

https://en.wikipedia.org/wiki/Jupyter

Write a summary of the Wikipedia page on Jupyter:

https://en.wikipedia.org/wiki/Jupyter

Write a summary of the Wikipedia page on Jupyter:

Jupyter is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It supports over 40 programming languages, including Python, R, Julia, and Scala. Jupyter is used by data scientists, researchers, and educators to explore and analyze data, create interactive documents, and collaborate with others. It is also used for teaching and learning, as it allows users to create interactive notebooks that can be shared with others. Jupyter is a popular tool for data science and machine learning, and is used by many organizations, including Google, Microsoft, IBM, and NASA.

Thanks for your attention!