Neural sequence generation
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:

  • 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 [ ] 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 $4 \times 7 \times 3$ meals.

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

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

Open-Ended Generation

Build next-gen apps with OpenAI’s powerful models.

OpenAI’s API provides access to GPT-3, which performs a wide variety of natural language tasks, and Codex, which translates natural language to code.

Theorem Proving

Welleck et al 2021

Lample & Charton 2019

Open-domain QA

Roberts et al 2020

Open-Ended Generation

Lample & Charton 2019

Symbolic Mathematics

Welleck et al 2019

Dialogue

Liu et al 2021

Commonsense

Formal Theorem Proving

Han et al 2021

[Polu et al 2022]

Program Synthesis

Austin et al 2021

Chen et al 2021

Machine Translation

Welleck et al 2021

Long-form QA

How has technological growth increased so exponentially in the last 50 years?

> 175B best-of-64

There are many explanations for the exponential growth in technology in the last century. One explanation is that the pace of technological progress speeds up exponentially over time because of a common force driving it forward. Another explanation is that each new generation of technology stands on the shoulders of its predecessors, allowing for improvements that lead to the next generation of even better...
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.
What is a language model?

Setup

- $y_{1:T} = (y_1, y_2, \ldots, y_T)$, where $y_t \in V$
  
  - $y_{1:T}$ sequence e.g. the cat sat. $T$ can vary.
  
  - $y_t$ token e.g. cat
  
  - $V$ vocabulary e.g. $\{a, b, \ldots, \text{zebra}, \ldots\}$
  
  - $y \in \mathcal{Y}$, $\mathcal{Y}$ set of all sequences
What is a language model?

Language model

- A language model is a probability distribution over all sequences
  - $p(y)$
- 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

  • \( p(y) \)

• Example language model

  • \( p(y) = 0.000013 \) if \( y \) is \( a \).
  • \( 0.000001 \) if \( y \) is \( aa \).
  • \( \ldots \)
  • \( 0.019100 \) if \( y \) is \( a \ cat \ sat \).
  • \( \ldots \)
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’):
  
  • $p(y_{k+1:T} | y_{1:k})$

• 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:
  
  \[ p(y_{k+1:T} \mid y_{1:k}) \]

• 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?
The building blocks | Modeling

Autoregressive language model

- First step: use the chain rule of probability:

\[ p(y_{1:T}) = \prod_{t=1}^{T} p(y_t | y_{<t}) \]

- \( p(\text{the cat sat <end>}) = p(\text{the})p(\text{cat}|\text{the})p(\text{sat}|\text{the cat})p(<\text{end}>|\text{the cat sat}) \)
The building blocks | Modeling

Autoregressive language model

• Language modeling is reduced to classification

\[ p(y_{1:T}) = \prod_{t=1}^{T} p \left( y_t \mid y_{<t} \right) \]

• \( p(\text{the cat sat <end}>)= \]
  \[ p(\text{the})p(\text{cat | the})p(\text{sat | the cat})p(\text{<end> | the cat sat}) \]

• Sequence probability =
  product of **next-token** probabilities
The building blocks | Modeling

Autoregressive language model

- Language modeling is reduced to **classification**

\[
p(y_{1:T}) = \prod_{t=1}^{T} p(y_t | y_{<t})
\]

- \( p(y_t | y_{<t}) \)
  - Input: \( y_{<t} \in V \times V \times \ldots \times V \)
  - 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

\[ p_\theta(y_{1:T}) = \prod_{t=1}^{T} p_\theta(y_t | y_{<t}) \]

• \( p_\theta(y_t | y_{<t}) \)
  - Input: \( y_{<t} \in V \times V \times \ldots \times V \)
  - Output: probability distribution over \( V \)
  - Target: \( y_t \in V \)

• What kind of neural network?
• How do we learn the parameters \( \theta \)?
The building blocks | Modeling

What kind of neural network?

- Want: $p_\theta(y_t | y_1, \ldots, y_{t-1})$
- 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
  - $p_\theta(\cdot | y_1, \ldots, y_{t-1}) = \text{softmax}(s_t)$

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!

[Image of neural network diagram]
The building blocks | Learning

How do we learn the parameters $\theta$?

- Collect a dataset of sequences $D = \{y_1, \ldots, y_N\}$
  - D: a book
  - D: all text on the internet
  - ...
- Tokenize each sequence, $y_i = (y_1, \ldots, y_{T_i})$
  - We’ll see this concretely in the lab!
The building blocks | Learning

How do we learn the parameters $\theta$?

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

\[
\min_{\theta} - \sum_{y \in D} \sum_{t} \log p_{\theta}(y_t | y_{<t})
\]

\[
\equiv \max_{\theta} \sum_{y \in D} \log p_{\theta}(y)
\]

• Finds parameters that make the observed data \(D\) most probable;
  i.e. maximum likelihood estimation
The building blocks | Learning

Why cross-entropy loss? | Distribution matching view

- Makes $p_\theta$ match an underlying ‘true’ distribution $p_*(y)$ .............. E.g. distribution that generated all internet text...

\[
\min_{\theta} D_{KL}(p_* | p_\theta) = \min_{\theta} - \sum_{y \in \mathcal{Y}} p_*(y) \log \frac{p_\theta(y)}{p_*(y)} \\
\equiv \min_{\theta} - \sum_{y \in \mathcal{Y}} p_*(y) \log p_\theta(y) + \text{const} \\
= - \mathbb{E}_{y \sim p_*} \log p_\theta(y) \quad \text{Definition of expected value} \\
\approx \min_{\theta} - \frac{1}{|D|} \sum_{y \in D} \log p_\theta(y) \quad \text{“Monte-Carlo” approximation of expected value} \\
\equiv \max_{\theta} \sum_{y \in D} \log p_\theta(y)
\]
The building blocks | Learning

Why cross-entropy loss?

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

**Figure 1** Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute 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]
We’ve now learned a neural language model $p_\theta$ 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 $y$ given a model $p_\theta$

- We want to generate $y = (y_1, \ldots, y_T)$, starting from $y_0 = \langle \text{start} \rangle$
  - We generate one-token, feed it into the model, and repeat:
    - $y_1 = \text{generate}(p_\theta(y \mid y_0))$
    - $y_2 = \text{generate}(p_\theta(y \mid y_0, y_1))$
    - $y_3 = \text{generate}(p_\theta(y \mid y_0, y_1, y_2))$
    - $\ldots \Rightarrow (y_1, \ldots, y_T)$
Building blocks | Decoding

Generating sequences from our model

- **Goal**: generate a continuation $y$ given a model $p_\theta$ and prefix $x$
  - Sampling
    - $y \sim p_\theta(\cdot | x)$
  - Mode-seeking
    - $y = \arg\max_y p_\theta(y | x)$
Building blocks | Decoding
Generating sequences from our model

- **Ancestral sampling**: sample from the model’s distribution

- Until $y_t = \langle \text{end} \rangle$:
  - $y_t \sim p_\theta(\cdot | y_{<t})$
  - $y$ is a sample from $p_\theta(y)$, since

$$p_\theta(y) = \prod_{t=1}^{T} p_\theta(y_t | y_{<t})$$
Building blocks | Decoding
Generating sequences from our model

• **Greedy decoding:** select the most-probable token at each step
  
  • Until $y_t = \langle \text{end} \rangle$:
    
    $y_t = \arg \max_{y \in V} p_\theta(y \mid y_{<t}, x)$
  
  • $y$ is a (naive) approximation of
    
    $\arg \max_y p_\theta(y \mid x)$
Building blocks | Decoding
Generating sequences from our model

- **Temperature Sampling**: adjust each distribution & sample
  - Until \( y_t = \langle \text{end} \rangle \):
    - \( y_t \sim p_{\theta}^\tau(\cdot | y_{<t}) \)
    - Where \( p_{\theta}^\tau(\cdot | \ldots) = \text{softmax}(s_i/\tau) \), \( \tau \in \mathbb{R}_{>0} \)
  - \( \tau \) small: “sharpens” the distribution
    - \( \tau \to 0 \): greedy decoding
  - \( \tau \) 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
  - $y_t \sim \alpha \begin{cases} p_\theta(y | y_{<t}) & y \in \text{top-k} \\ 0 & \text{otherwise} \end{cases}$
  - $k$ 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 $q(y \mid p_\theta)$
    - In practice, we do this with new per-step distributions, $q^{(t)}(y_t \mid p_\theta, y_{<t})$.
  - 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 \( p(\text{output} | \text{prompt}) \)
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:

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!