10-423/10-623 Generative AI

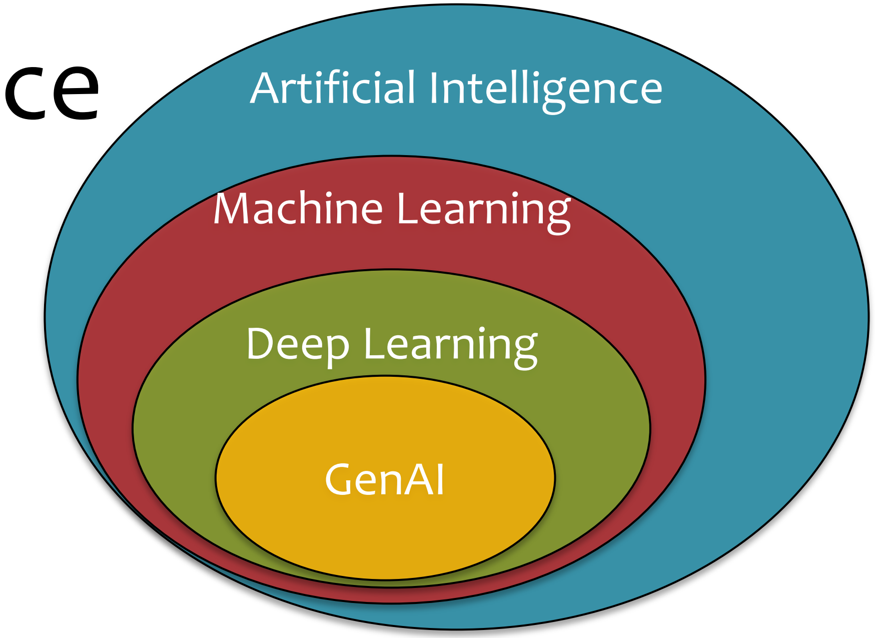
Machine Learning Department
School of Computer Science
Carnegie Mellon University

Course Overview + AutoDiff + RNN-LMs

Matt Gormley
Lecture 1
Jan. 17, 2024

WHAT IS GENERATIVE AI?

Artificial Intelligence

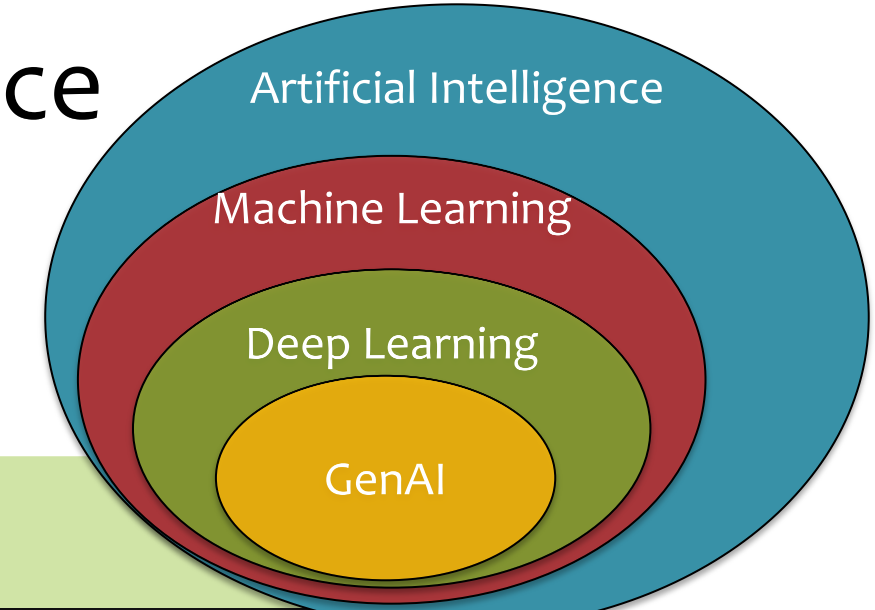


The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

Artificial Intelligence

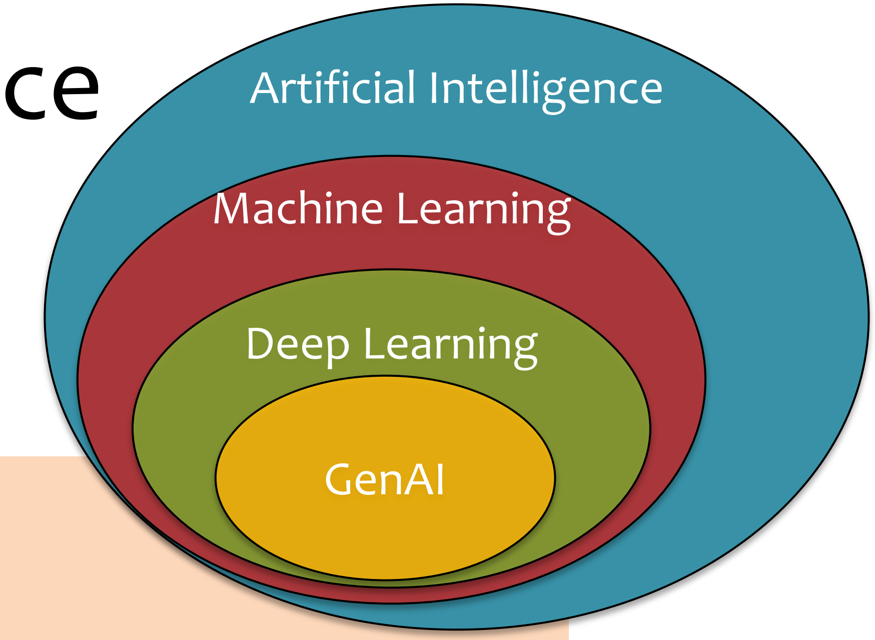


The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

Artificial Intelligence

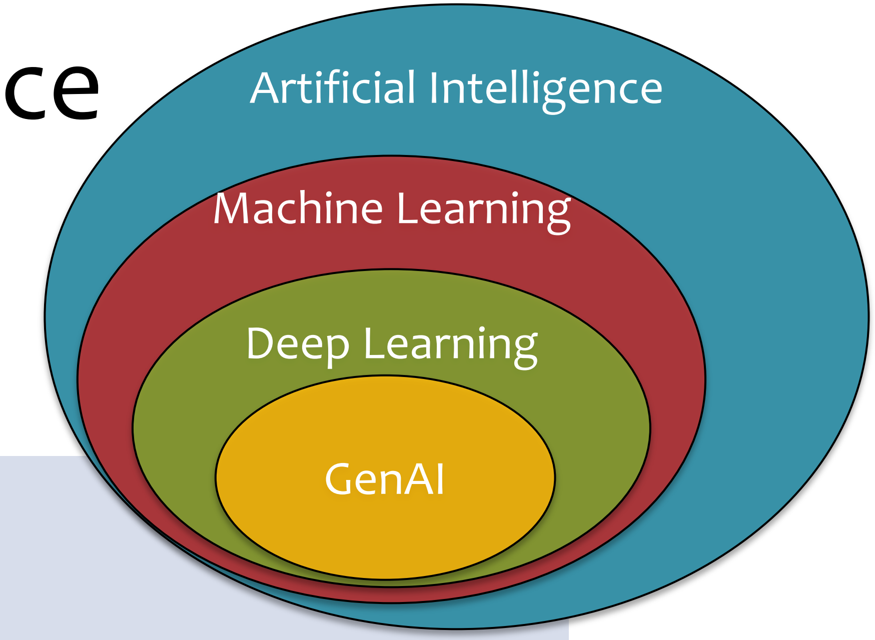


The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

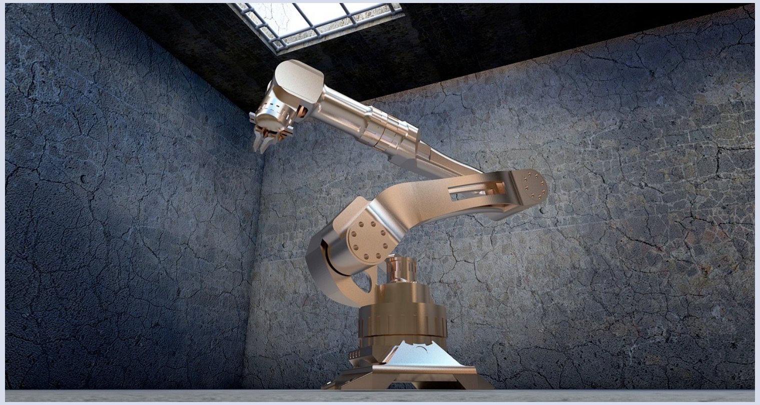
Artificial Intelligence



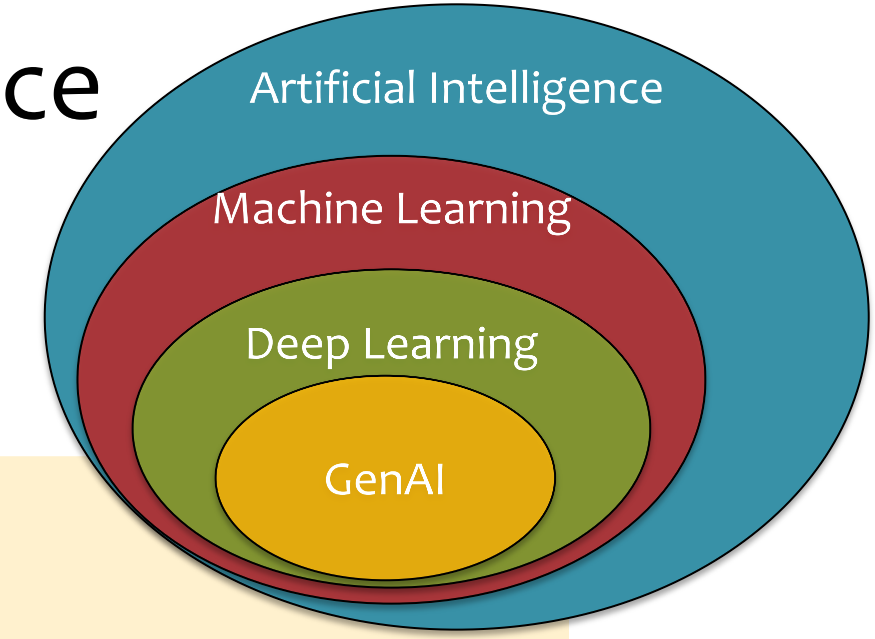
The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning



Artificial Intelligence

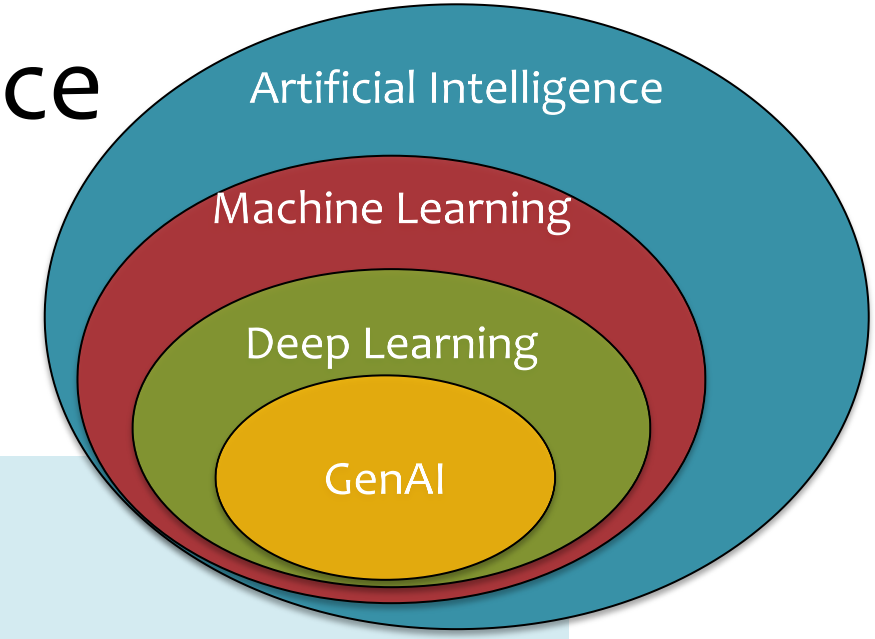


The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

Artificial Intelligence

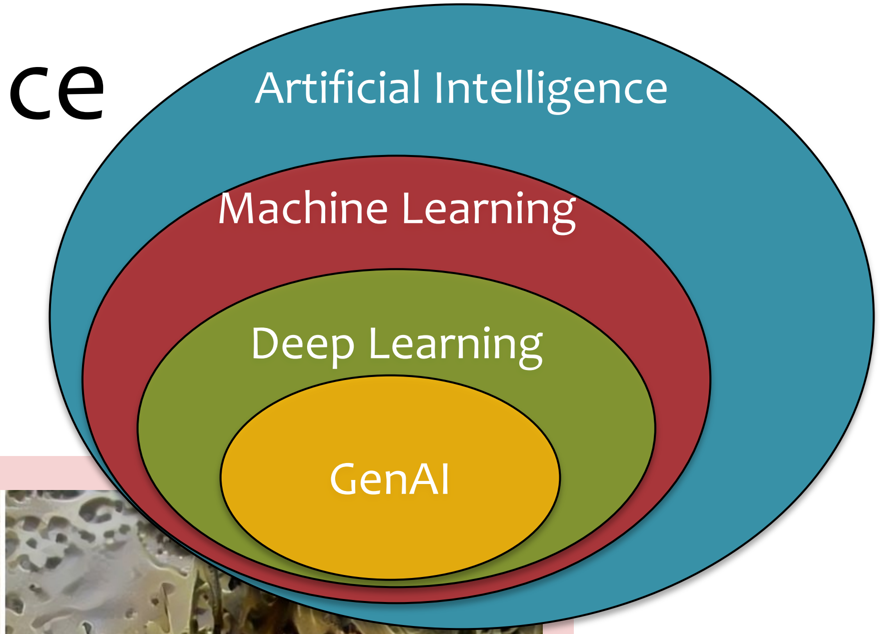

Artificial Intelligence
Machine Learning
Deep Learning
GenAI

The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

Artificial Intelligence



Artificial Intelligence

Machine Learning

Deep Learning

GenAI

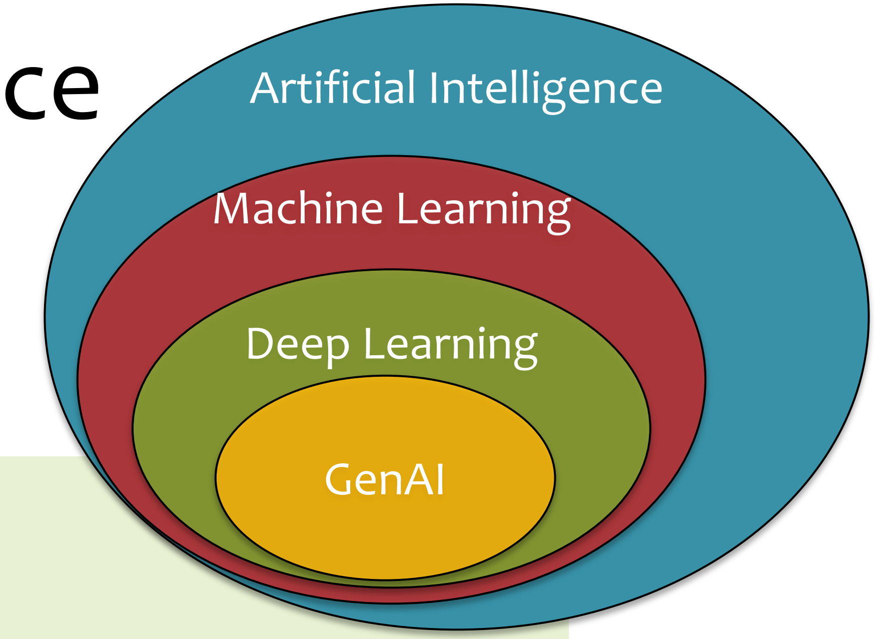
The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

“Deep Style” from https://deepdreamgenerator.com/#gallery

Artificial Intelligence

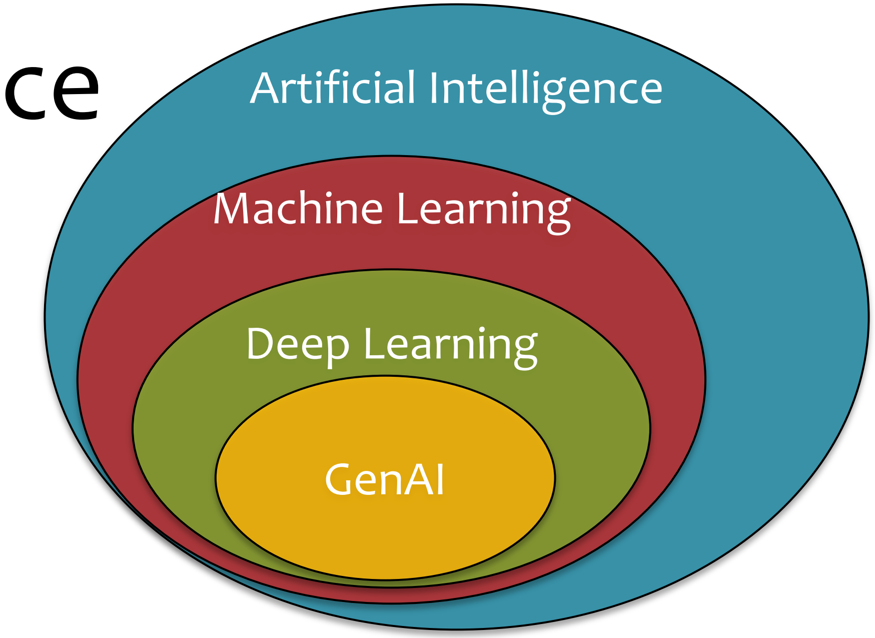


The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

Artificial Intelligence



The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

❑ Q: What does Generative AI have to do with **any of these goals**?

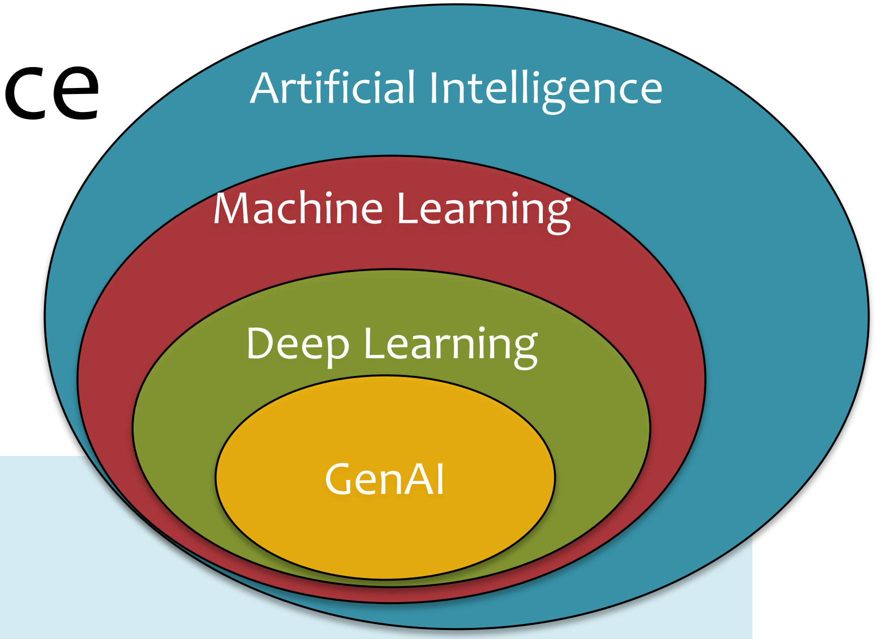
❑ A: It's making in-roads into **all of them.**

Artificial Intelligence

The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

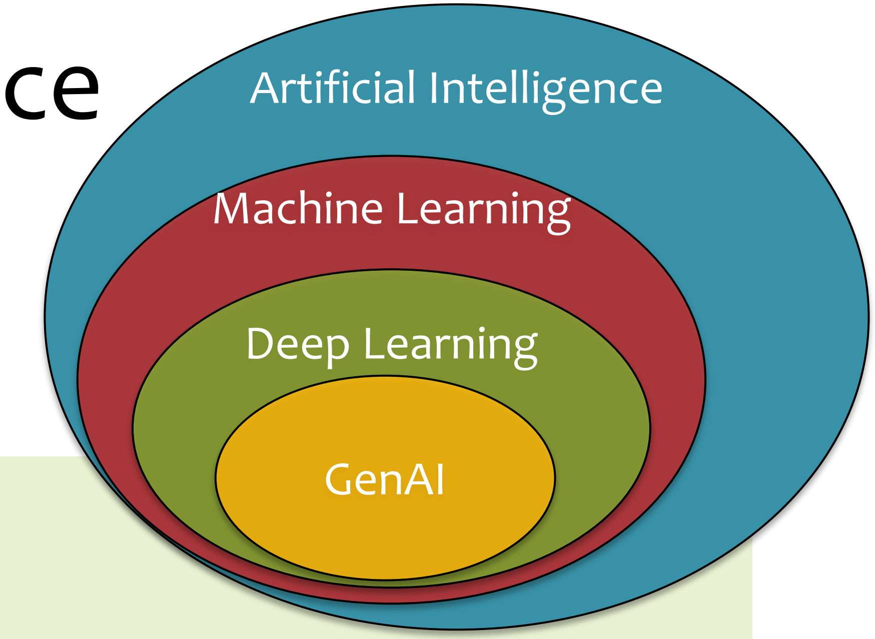


❑ Communication comprises the comprehension and generation of human language.

❑ Large language models (LLMs) excel at both

❑ (Even though they are most often trained autoregressively, i.e. to **generate** a next word, given the previous ones)

Artificial Intelligence



The basic goal of AI is to develop intelligent machines.

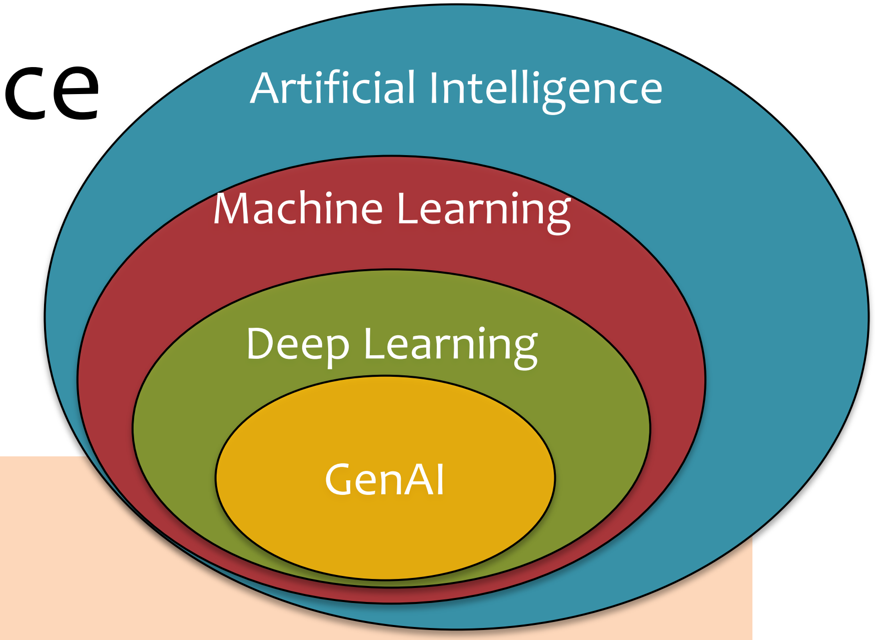
This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

❑ The traditional way of learning in ML is via **parameter estimation**

❑ But **in-context learning** (i.e. providing training examples as context at test time) shows that learning can also be done via **inference**

Artificial Intelligence



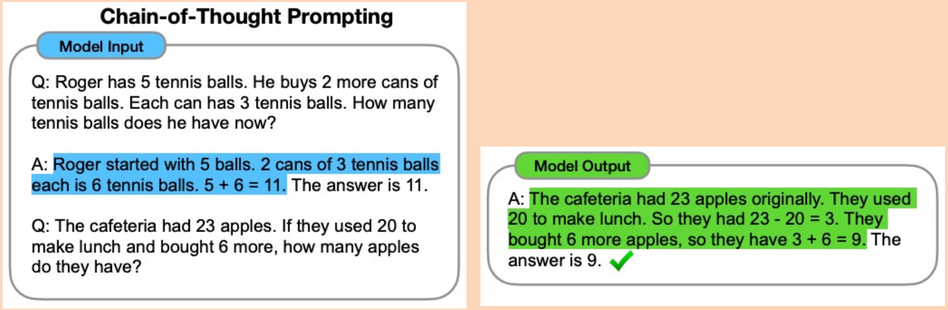
The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

☐ LLMs are also (unexpectedly) good at certain reasoning tasks

☐ cf. Chain-of-Though Prompting (an ex. of in-context learning)



Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

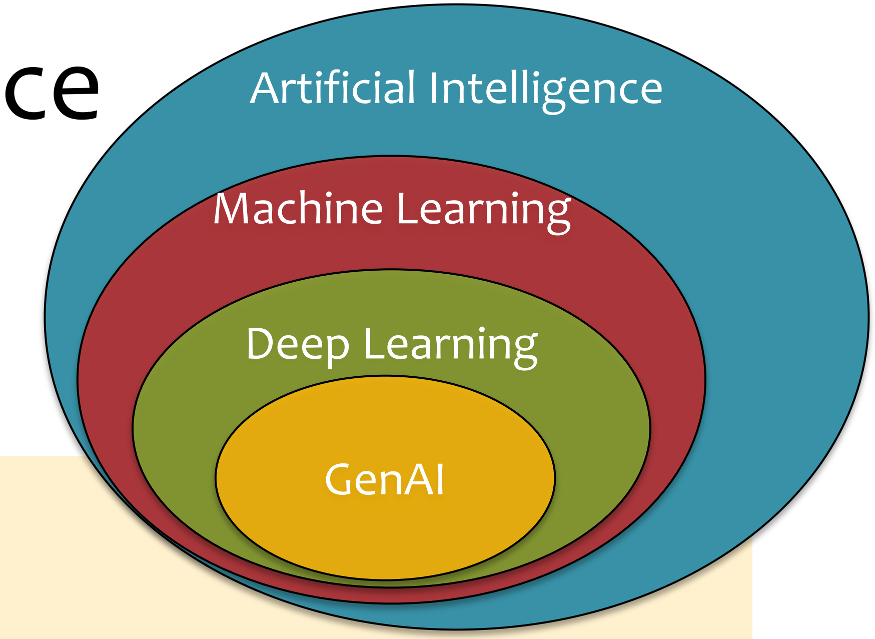
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Artificial Intelligence



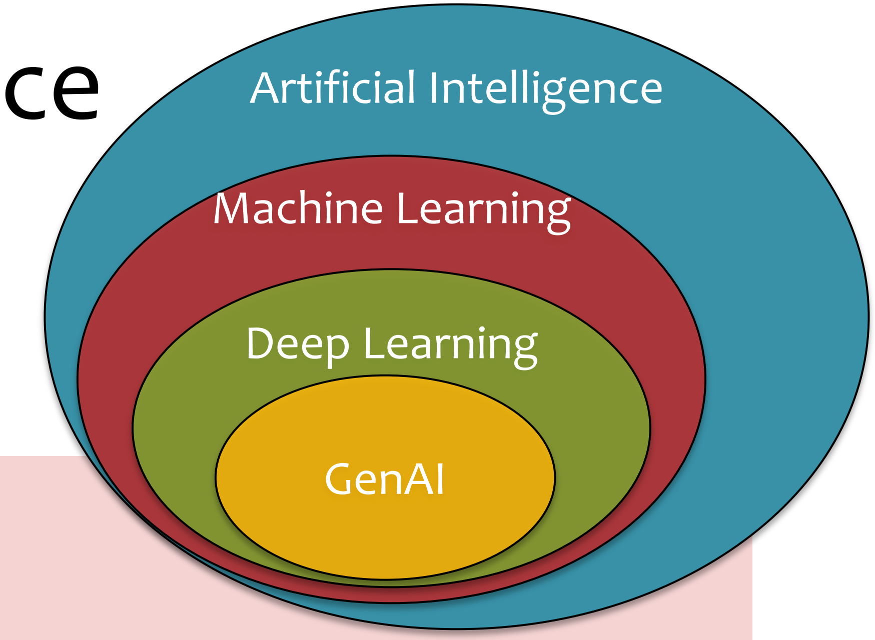
The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

❑ LLMs are already being used for grounded planning for embodied agents, c.f. LLM-Planner

Artificial Intelligence



The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

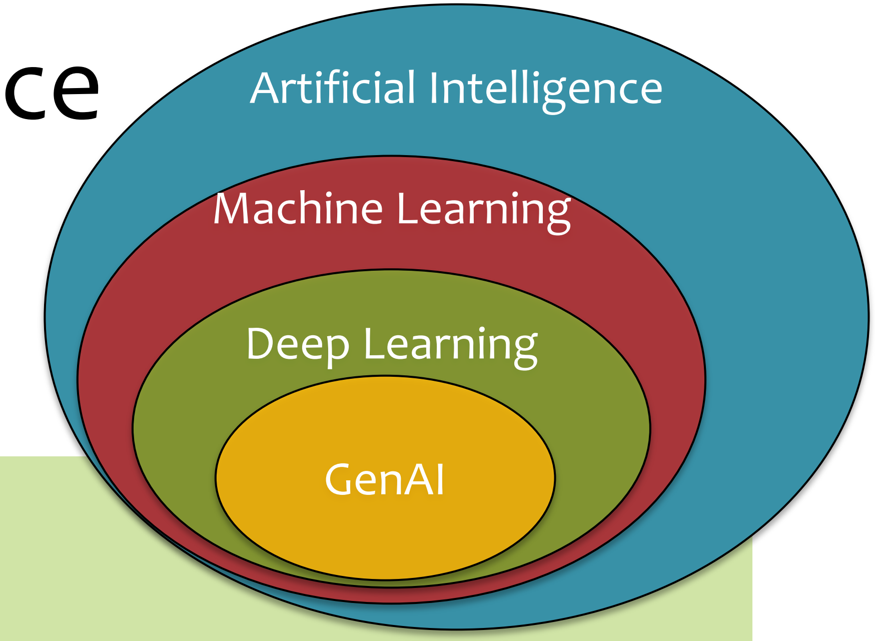
- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

☐ Text-to-image models [Midjourney's Discord server has 18 million members (1.7 million were online this morning)]

☐ Text-to-music models [MusicGen capable of conditioning on text and audio sample]

"Deep Style" from https://deepdreamgenerator.com/#gallery

Artificial Intelligence



The basic goal of AI is to develop intelligent machines.

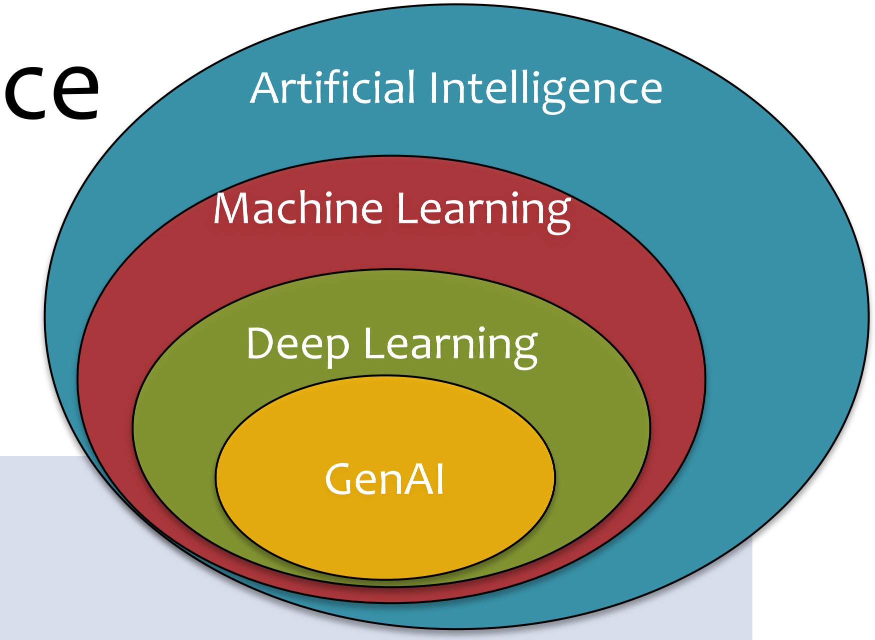
This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

❑ Multimodal foundation models learn to answer questions about images (and text in images)

❑ Diffusion models can be used as zero-shot classifiers

Artificial Intelligence



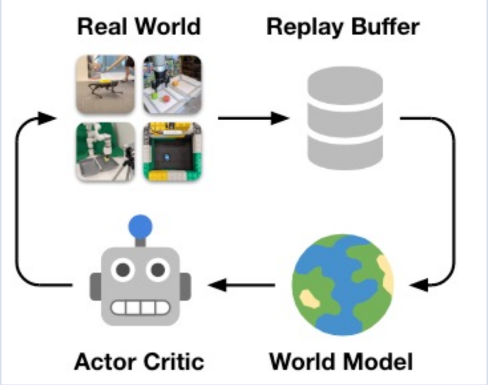
The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

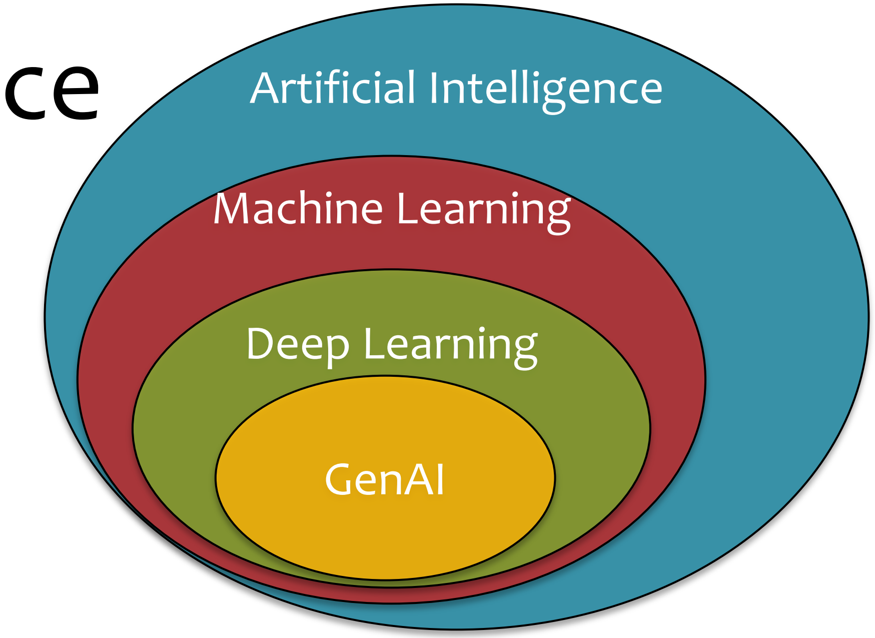
- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

❑ DayDreamer learns a generative model of experiences for RL, i.e. a World Model, without simulation

❑ Quadruped robot learns to walk in under 1 hour



Artificial Intelligence



The basic goal of AI is to develop intelligent machines.

This consists of many sub-goals:

- Perception
- Reasoning
- Control / Motion / Manipulation
- Planning
- Communication
- Creativity
- Learning

☐ Q: What does Generative AI have to do with **any of these goals**?

☐ A: It's making in-roads into **all of them.**

EXAMPLE OF GENERATIVE AI

Text Generation

Prompt: Write a proof of the fact that there are infinitely many primes; do it in the style of a Shakespeare play through a dialogue between two parties arguing over the proof.

GPT-4: Enter FIRST PARTY and SECOND PARTY.

FIRST PARTY: Thou dost not understand, my friend, There simply is no upper end To primes, those numbers indivisible, No matter if it seems impossible.

SECOND PARTY: But how dost thou so surely know? To prove this truth, what canst thou show?

FIRST PARTY: Consider this, my doubtful peer, A clever proof that will make clear: Assume that there's a final prime, The largest one we'll see in time. Let's call this number "P" for short, And then, my friend, we will purport To build a number "Q" that's new, One larger than "P" and prime too.

SECOND PARTY: But how, my friend, will this be done? The proof is not yet fairly won.

FIRST PARTY: It's simple, friend, just wait and see, We'll multiply the primes, that's key. All primes from two to "P" combined, Will make this new "Q" we will find. But one thing more we must include, To make the proof more watertight, We'll add a one onto the end, And now the proof is in our sight.

SECOND PARTY: I see it now, it's crystal clear, For "Q" must be a prime, my dear. It can't divide by any prime, All smaller than itself in time. And since we chose "P" as the last, This proves that our assumption's past. There is no final prime, we see, The numbers go on endlessly.

Example from https://arxiv.org/pdf/2303.12712.pdf

Image Editing

A variety of tasks involve automatic editing of an image:

- **Inpainting** fills in the (pre-specified) missing pixels
- **Colorization** restores color to a greyscale image
- **Uncropping** creates a photo-realistic reconstruction of a missing side of an image

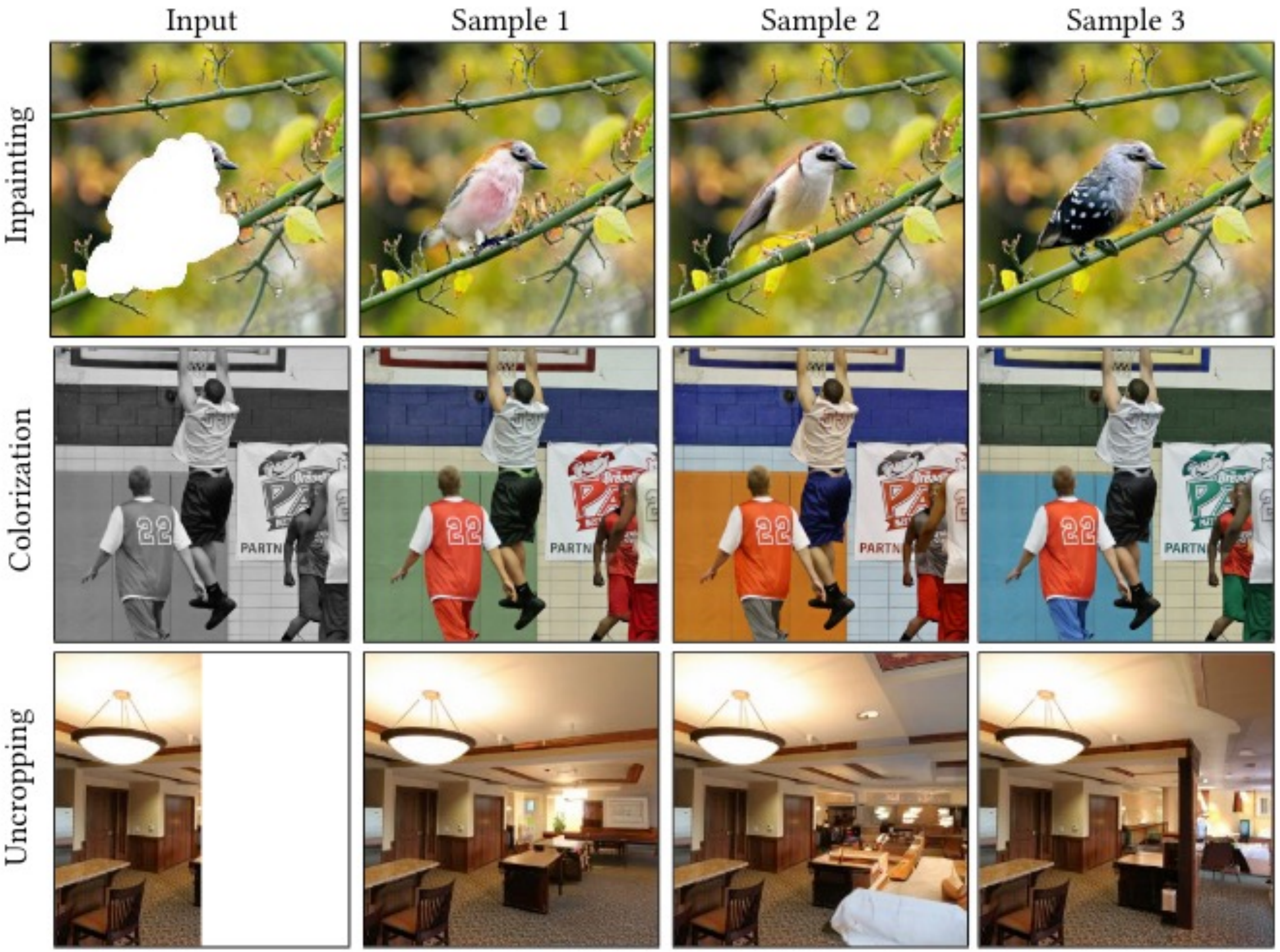


Figure from Saharia et al. (2022)

Text-to-Image Generation

- Given a text description, sample an image that depicts the prompt
- The following images are samples from SDXL with refinement

Prompt: close up headshot, futuristic **old man**, wild hair sly smile in front of gigantic UFO, dslr, sharp focus, dynamic composition, **rule of thirds**

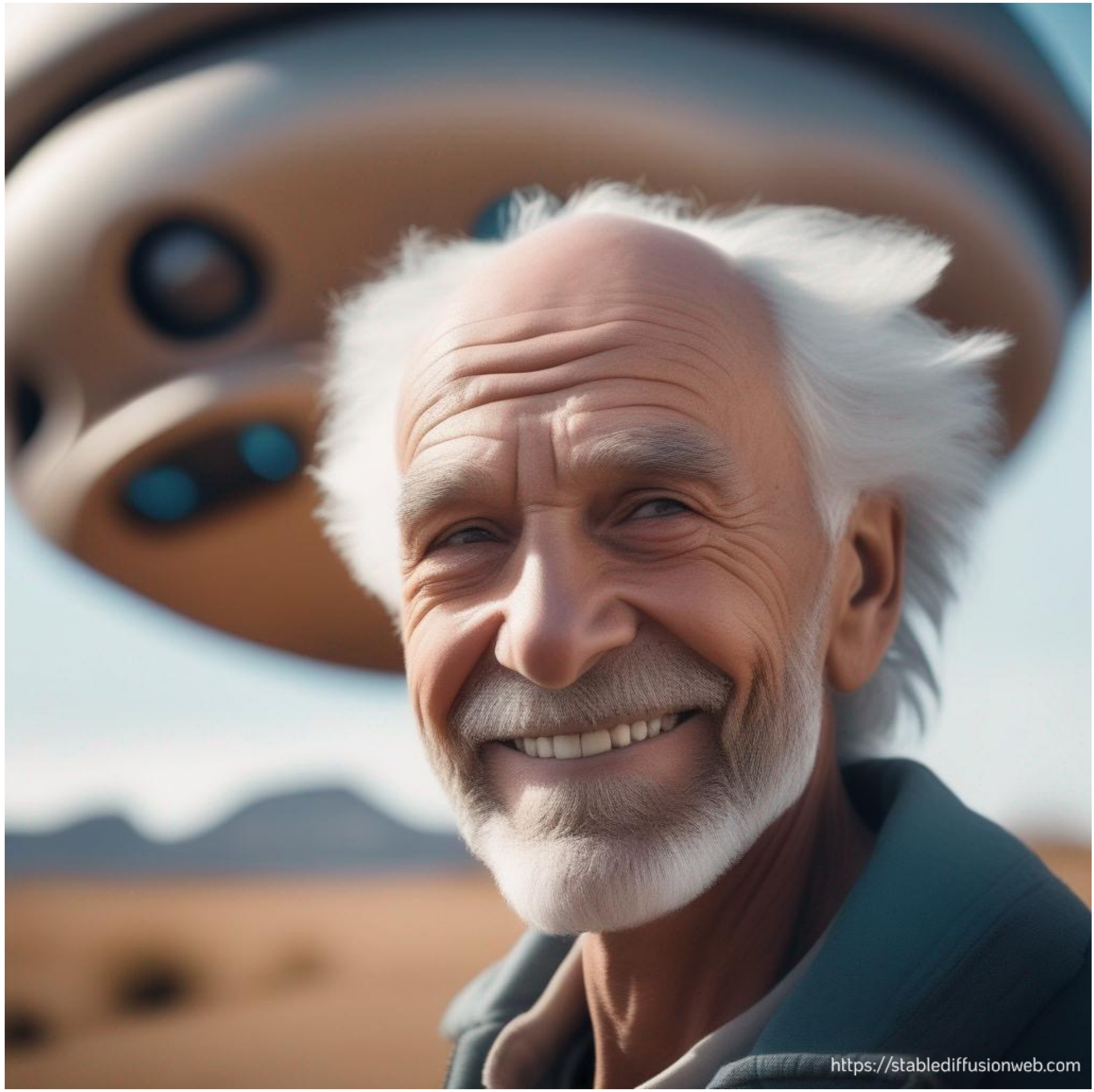
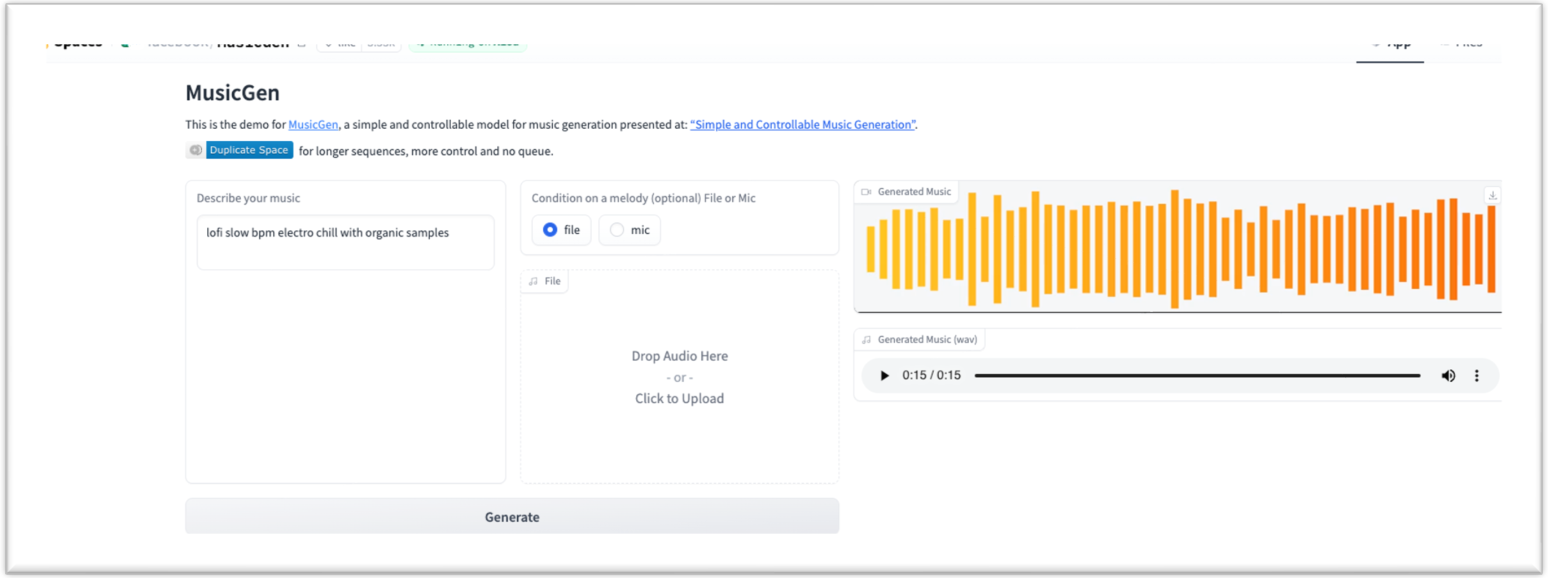


Figure from https://stablediffusionweb.com/

Music Generation



MusicGen

This is the demo for MusicGen, a simple and controllable model for music generation presented at: "Simple and Controllable Music Generation".

for longer sequences, more control and no queue.

Describe your music: lofi slow bpm electro chill with organic samples

Condition on a melody (optional) File or Mic: ● file ○ mic

MusicGen

- A transformer decoder model over quantized units (discrete elements of a codebook of audio frames)
- Interleaves sounds by adjusting how codebooks attend to each other
- Permits conditioning on text and/or audio samples

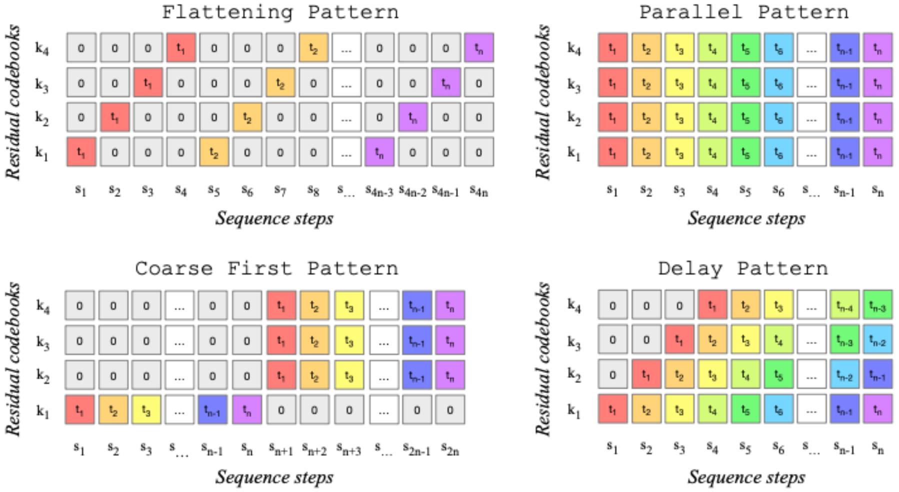


Figure from https://arxiv.org/pdf/2306.05284.pdf

Code Generation

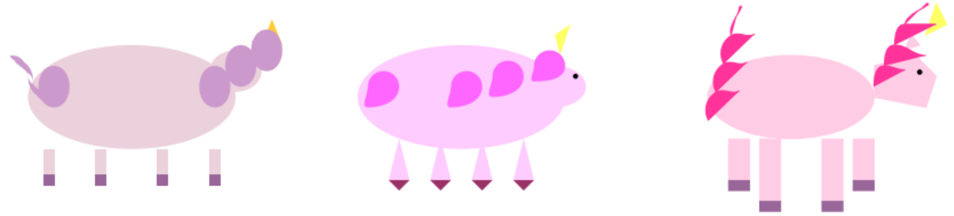
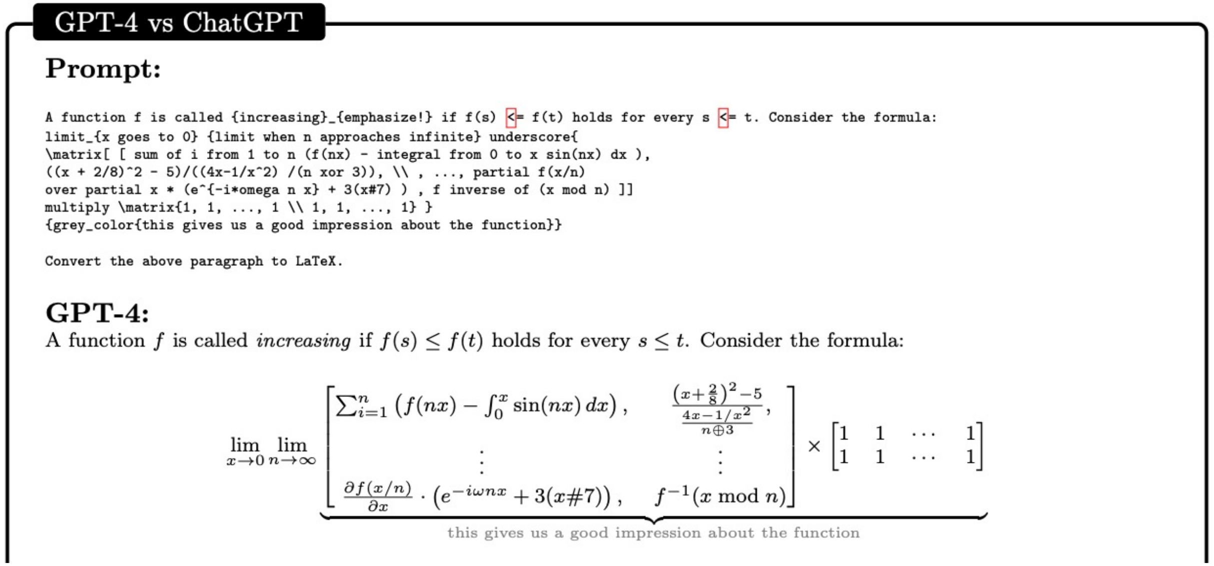


Figure 1.3: We queried GPT-4 three times, at roughly equal time intervals over the span of a month while the system was being refined, with the prompt "Draw a unicorn in TikZ". We can see a clear evolution in the sophistication of GPT-4's drawings.



GPT-4 vs ChatGPT

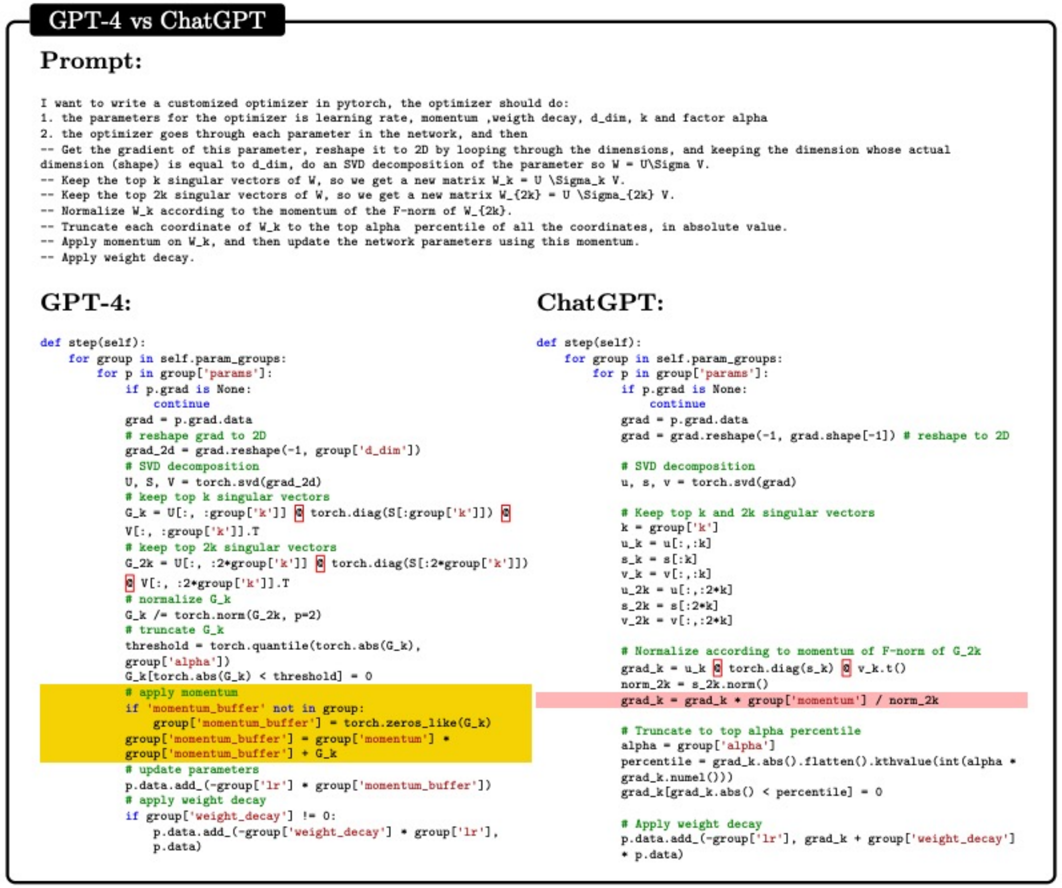
Prompt:

```
A function f is called {increasing}_{emphasize!} if f(s) <= f(t) holds for every s <= t. Consider the formula:
limit_{x goes to 0} {limit when n approaches infinite} underscore{
\matrix[ [ sum of i from 1 to n (f(nx) - integral from 0 to x sin(nx) dx ),
((x + 2/8)^2 - 5)/((4x-1/x^2) /(n xor 3)), \\ , ..., partial f(x/n)
over partial x * (e^{-i*omega n x} + 3(x#7) ) , f inverse of (x mod n) ]]
multiply \matrix{1, 1, ..., 1 \\ 1, 1, ..., 1} }
{grey_color{this gives us a good impression about the function}}

Convert the above paragraph to LaTeX.
```

GPT-4:

A function f is called *increasing* if $f(s) \leq f(t)$ holds for every $s \leq t$. Consider the formula:

$$\lim_{x\to 0}\lim_{n\to\infty}\underbrace{\begin{bmatrix}\sum_{i=1}^{n}\left(f(nx)-\int_0^x \sin(nx)\,dx\right), & \frac{(x+\frac{2}{8})^2-5}{\frac{4x-1/x^2}{n\oplus 3}}, \\ \vdots & \vdots \\ \frac{\partial f(x/n)}{\partial x}\cdot\left(e^{-i\omega n x}+3(x\#7)\right), & f^{-1}(x \bmod n)\end{bmatrix}\times\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\end{bmatrix}}_{\text{this gives us a good impression about the function}}$$



GPT-4 vs ChatGPT

Prompt:

```
I want to write a customized optimizer in pytorch, the optimizer should do:
1. the parameters for the optimizer is learning rate, momentum ,weigth decay, d_dim, k and factor alpha
2. the optimizer goes through each parameter in the network, and then
-- Get the gradient of this parameter, reshape it to 2D by looping through the dimensions, and keeping the dimension whose actual
dimension (shape) is equal to d_dim, do an SVD decomposition of the parameter so W = U\Sigma V.
-- Keep the top k singular vectors of W, so we get a new matrix W_k = U \Sigma_k V.
-- Keep the top 2k singular vectors of W, so we get a new matrix W_{2k} = U \Sigma_{2k} V.
-- Normalize W_k according to the momentum of the F-norm of W_{2k}.
-- Truncate each coordinate of W_k to the top alpha  percentile of all the coordinates, in absolute value.
-- Apply momentum on W_k, and then update the network parameters using this momentum.
-- Apply weight decay.
```

GPT-4:

```python
def step(self):
    for group in self.param_groups:
        for p in group['params']:
            if p.grad is None:
                continue
            grad = p.grad.data
            # reshape grad to 2D
            grad_2d = grad.reshape(-1, group['d_dim'])
            # SVD decomposition
            U, S, V = torch.svd(grad_2d)
            # keep top k singular vectors
            G_k = U[:, :group['k']] @ torch.diag(S[:group['k']]) @
            V[:, :group['k']].T
            # keep top 2k singular vectors
            G_2k = U[:, :2*group['k']] @ torch.diag(S[:2*group['k']])
            @ V[:, :2*group['k']].T
            # normalize G_k
            G_k /= torch.norm(G_2k, p=2)
            # truncate G_k
            threshold = torch.quantile(torch.abs(G_k),
            group['alpha'])
            G_k[torch.abs(G_k) < threshold] = 0
            # apply momentum
            if 'momentum_buffer' not in group:
                group['momentum_buffer'] = torch.zeros_like(G_k)
            group['momentum_buffer'] = group['momentum'] *
            group['momentum_buffer'] + G_k
            # update parameters
            p.data.add_(-group['lr'] * group['momentum_buffer'])
            # apply weight decay
            if group['weight_decay'] != 0:
                p.data.add_(-group['weight_decay'] * group['lr'],
                p.data)
```

ChatGPT:

```python
def step(self):
    for group in self.param_groups:
        for p in group['params']:
            if p.grad is None:
                continue
            grad = p.grad.data
            grad = grad.reshape(-1, grad.shape[-1]) # reshape to 2D

            # SVD decomposition
            u, s, v = torch.svd(grad)

            # Keep top k and 2k singular vectors
            k = group['k']
            u_k = u[:,:k]
            s_k = s[:k]
            v_k = v[:,:k]
            u_2k = u[:,:2*k]
            s_2k = s[:2*k]
            v_2k = v[:,:2*k]

            # Normalize according to momentum of F-norm of G_2k
            grad_k = u_k @ torch.diag(s_k) @ v_k.t()
            norm_2k = s_2k.norm()
            grad_k = grad_k * group['momentum'] / norm_2k

            # Truncate to top alpha percentile
            alpha = group['alpha']
            percentile = grad_k.abs().flatten().kthvalue(int(alpha *
            grad_k.numel()))
            grad_k[grad_k.abs() < percentile] = 0

            # Apply weight decay
            p.data.add_(-group['lr'], grad_k + group['weight_decay']
            * p.data)
```

Figure 3.4: GPT-4 vs ChatGPT at implementing a customized Pytorch optimizer module. We highlight the astuteness of GPT-4 vs the mistake by ChatGPT at 'applying momentum'.

Example from https://arxiv.org/pdf/2303.12712.pdf

Video Generation

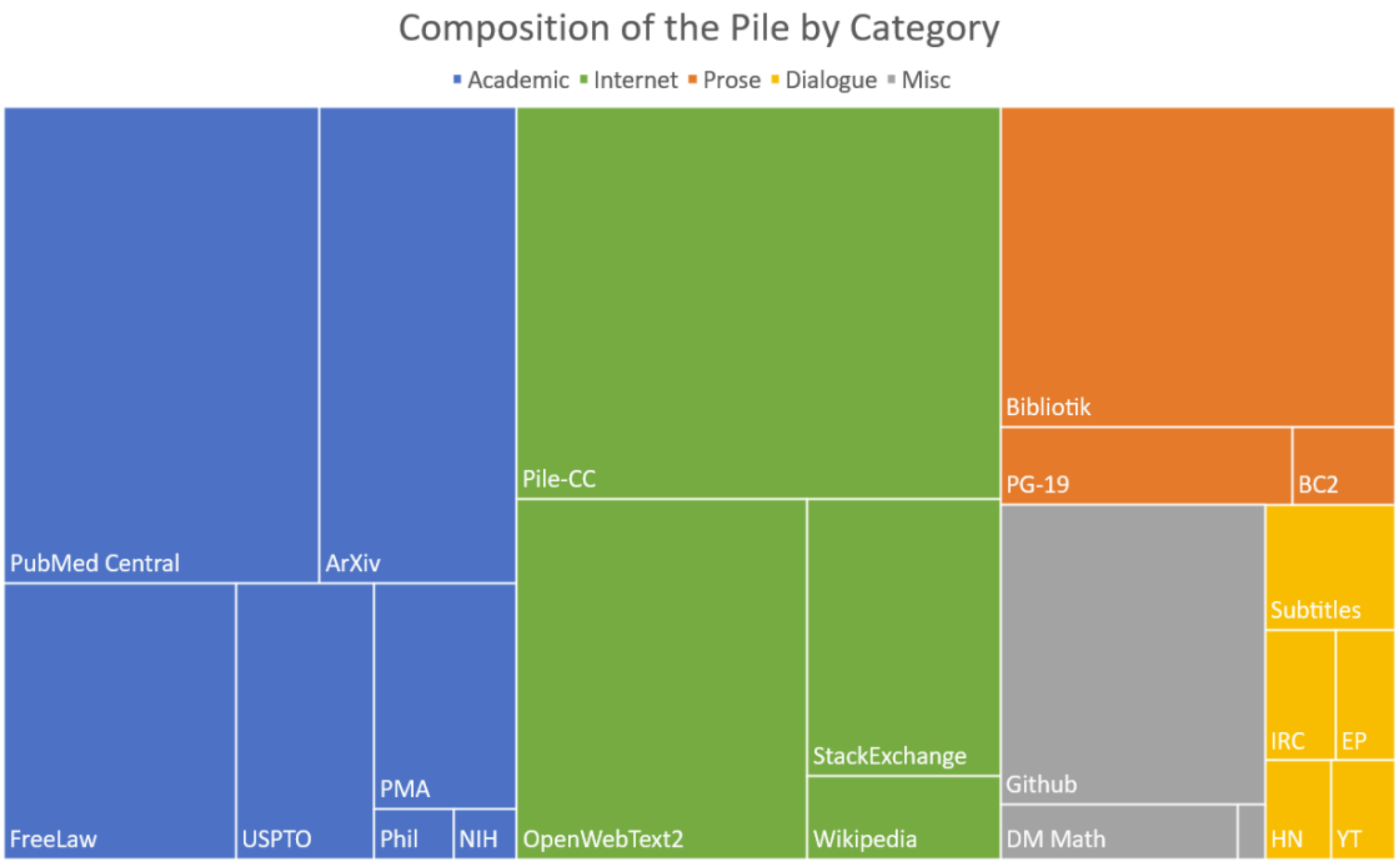
- Latent diffusion models use a low-dimensional latent space for efficiency
- **Key question:** how to generate multiple correlated frames?
- ‘Align your latents’ inserts temporal convolution / attention between each spatial convolution / attention
- ‘Preserve Your Own Correlation’ includes temporally correlated noise

Figure from https://huggingface.co/stabilityai/stable-video-diffusion-img2vid-xt

SCALING UP

Training Data for LLMs

The Pile:
- An open source dataset for training language models
- Comprised of 22 smaller datasets
- Favors high quality text
- 825 Gb ≈ 1.2 trillion tokens



RLHF

- **InstructGPT** uses Reinforcement Learning with Human Feedback (RLHF) to **fine-tune** a **pre-trained** GPT model
- From the paper: “In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters.”

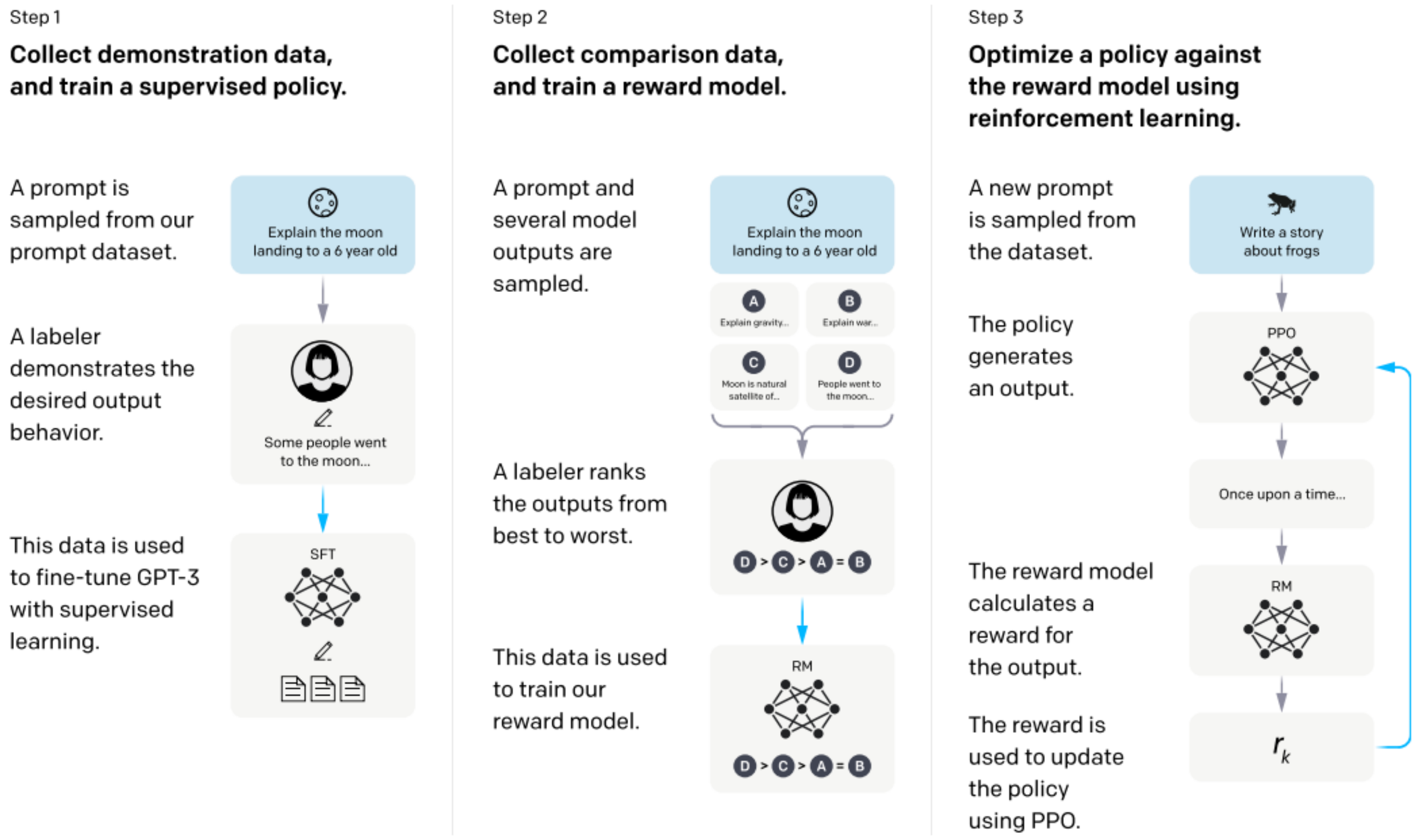

Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

Figure from https://arxiv.org/pdf/2203.02155.pdf

Memory Usage of LLMs

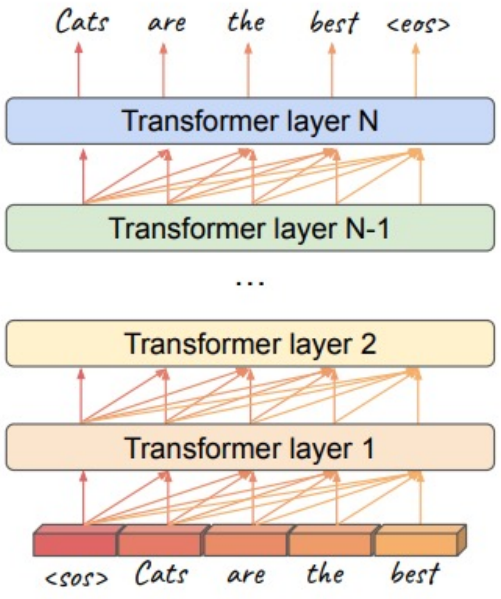
How to store a large language model in memory?

- **full precision:** 32-bit floats
- **half precision:** 16-bit floats
- Using half precision not only **reduces memory,** it also **speeds up** GPU computation
- *"Peak float16 matrix multiplication and convolution performance is 16x faster than peak float32 performance on A100 GPUs."* from Pytorch docs

Model	Megatron-LM	GPT-3
# parameters	8.3 billion	175 billion
full precision	30 Gb	651 Gb
half precision	15 Gb	325 Gb

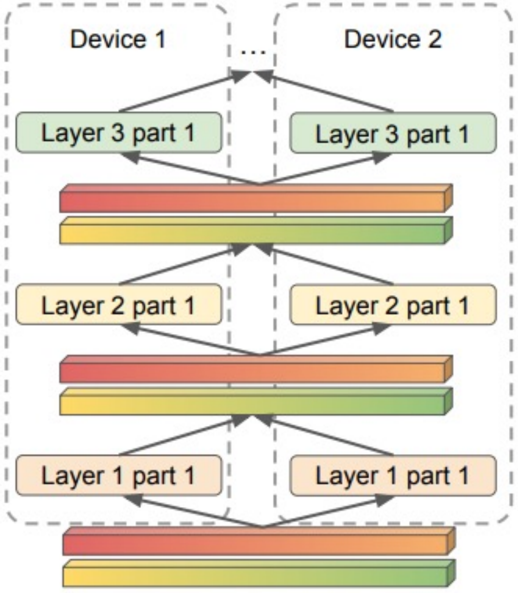
GPU / TPU	Max Memory
TPU v2	16 Gb
TPU v3/v4	32 Gb
Tesla V100 GPU	32 Gb
NVIDIA RTX A6000	48 Gb
Tesla A100 GPU	80 Gb

Distributed Training: Model Parallel



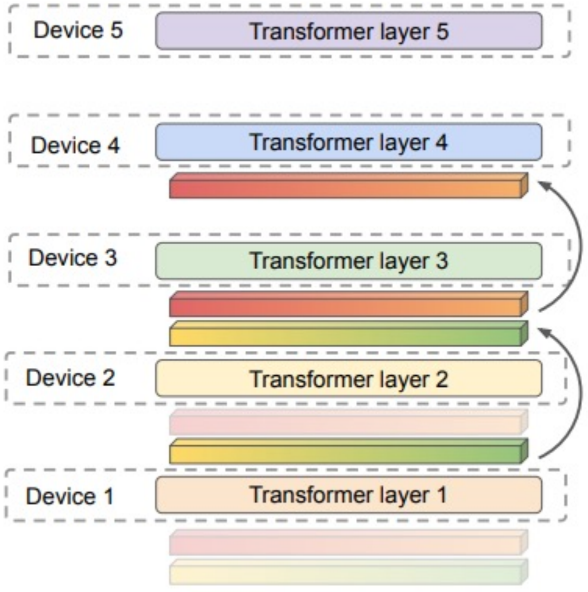
(a) Transformer-based LM

There are a variety of different options for how to distribute the model computation / parameters across multiple devices.



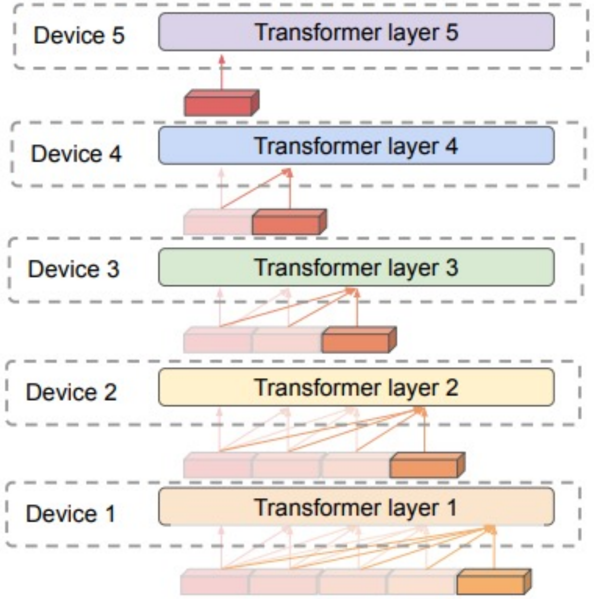
(b) Operation partitioning (Megatron-LM)

Matrix multiplication comprises most Transformer LM computation and can be divided along rows/columns of the respective matrices.



(c) Microbatch-based pipeline parallelism (GPipe)

The most natural division is by layer: each device computes a subset of the layers, only that device stores the parameters and computation graph for those layers.



(d) Token-based pipeline parallelism (TeraPipe)

A more efficient solution is to divide computation by token *and* layer. This requires careful division of work and is specific to the Transformer LM.

Figure from https://arxiv.org/pdf/2102.07988.pdf

Cost to train

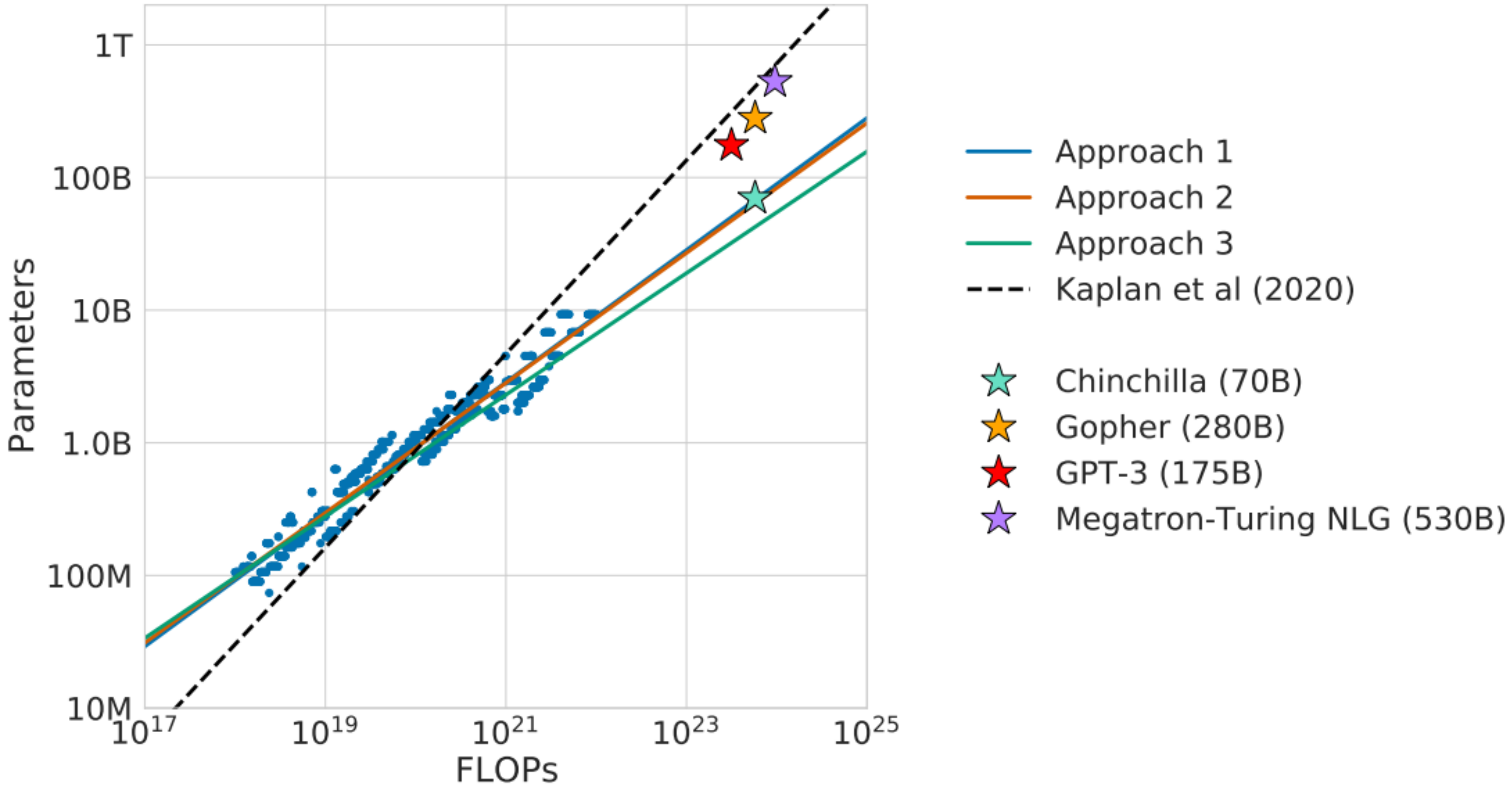
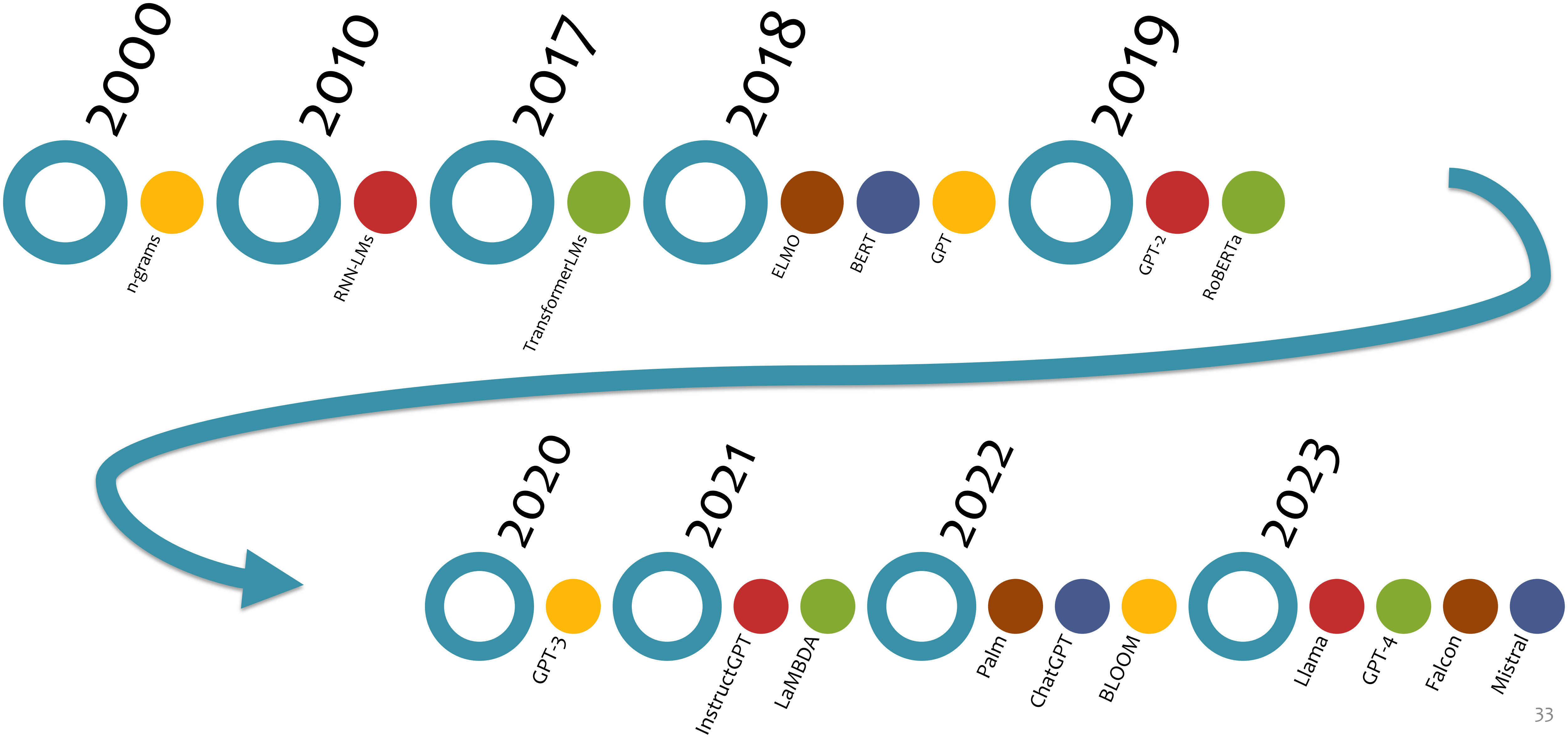


Figure from https://arxiv.org/pdf/2203.15556.pdf

Timeline: Language Modeling



2000
n-grams

2010
RNN-LMs

2017
TransformerLMs

2018
ELMO
BERT
GPT

2019
GPT-2
RoBERTa

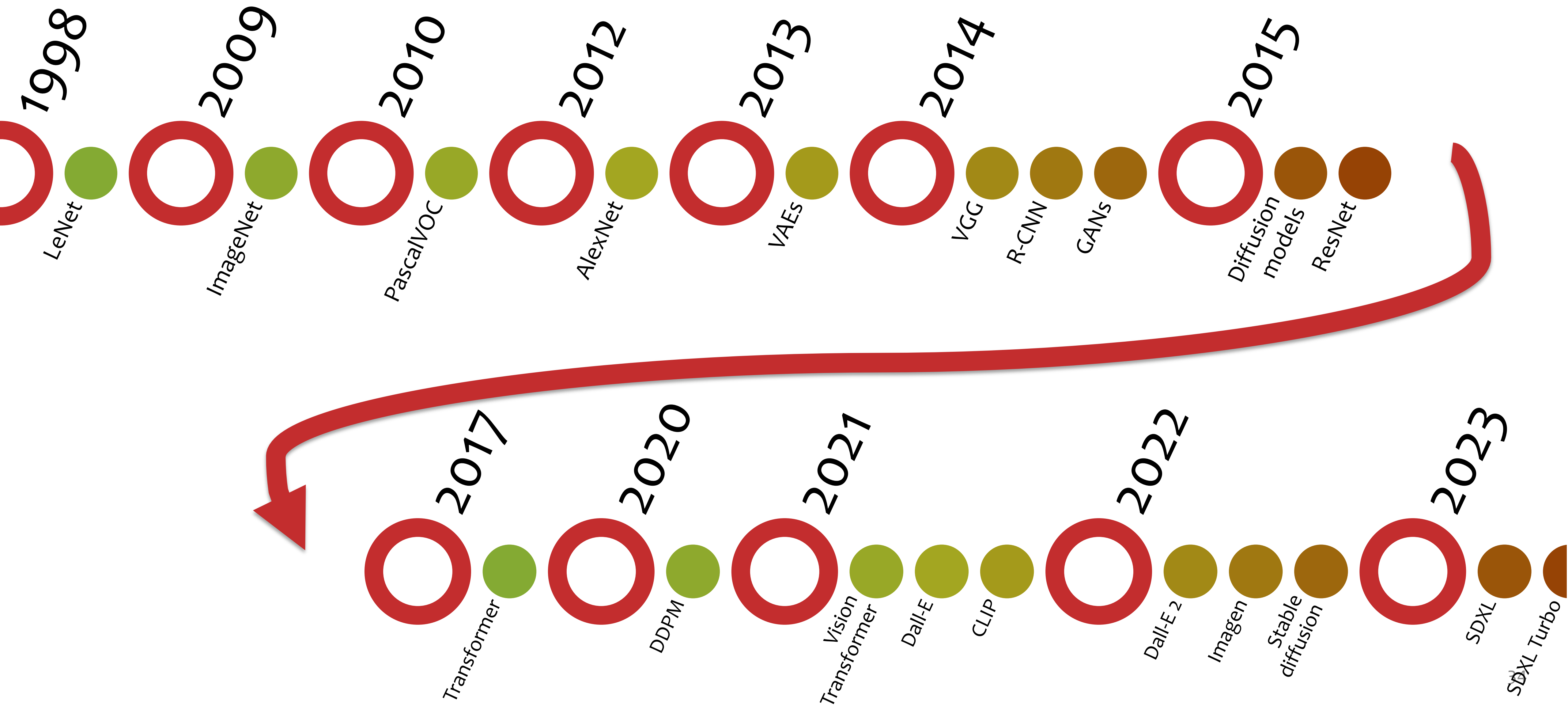
2020
GPT-3

2021
InstructGPT
LaMBDA

2022
Palm
ChatGPT
BLOOM

2023
Llama
GPT-4
Falcon
Mistral

Timeline: Image Generation



1998
LeNet
2009
ImageNet
2010
PascalVOC
2012
AlexNet
2013
VAEs
2014
VGG
R-CNN
GANs
2015
Diffusion models
ResNet
2017
Transformer
2020
DDPM
2021
Vision Transformer
Dall-E
CLIP
2022
Dall-E 2
Imagen
Stable diffusion
2023
SDXL
SDXL Turbo

Why learn the inner-workings of GenAI?

(a metaphor)

Figure from https://www.astonmartin.com/en/

Figure from https://daily.jstor.org/the-science-of-traffic/

Figure from https://earthobservatory.nasa.gov/images/149321/2021-continued-earths-warming-trend

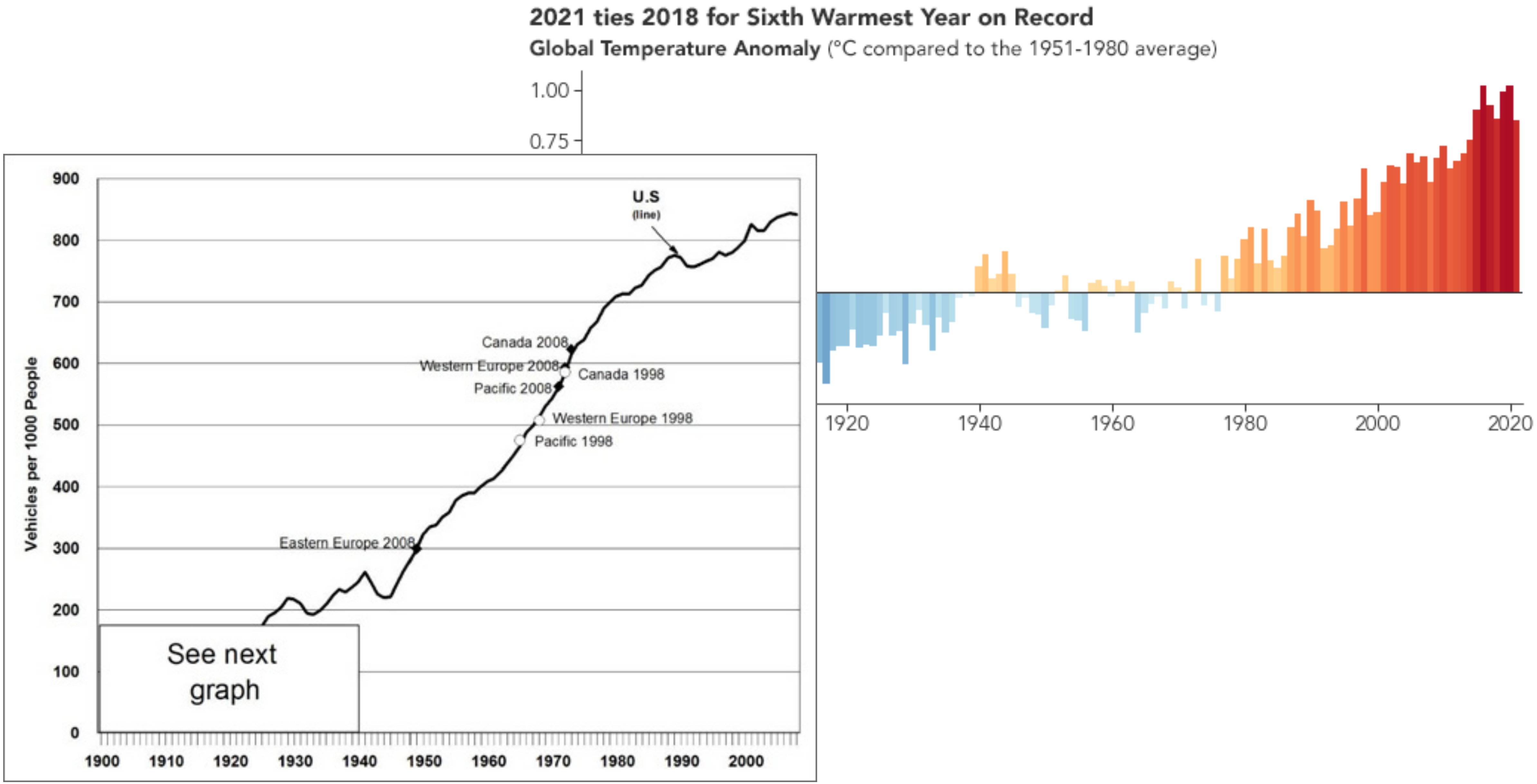


Figure from https://www.energy.gov/eere/vehicles/fact-617-april-5-2010-changes-vehicles-capita-around-world

Figure from GHSA

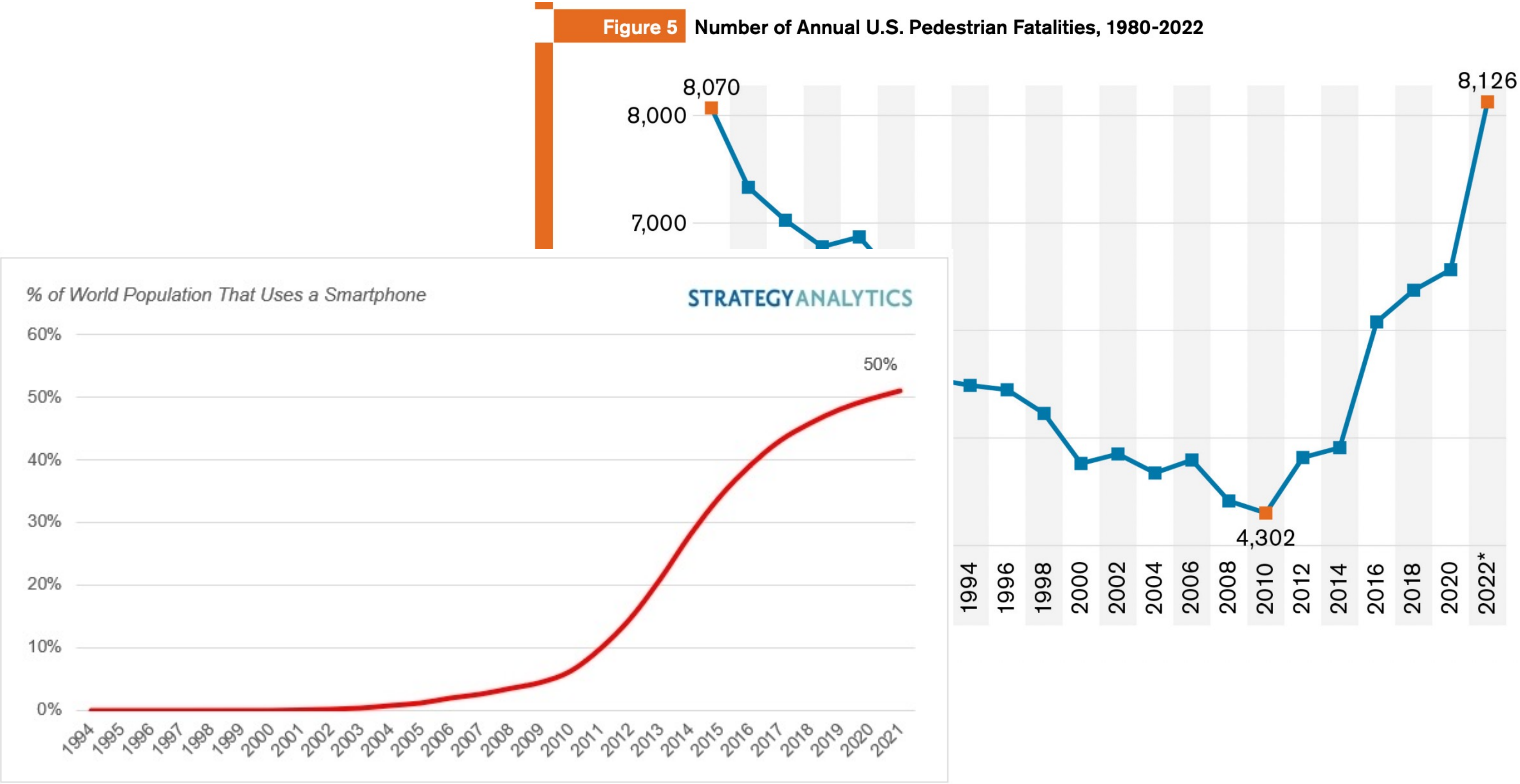


Figure 5 Number of Annual U.S. Pedestrian Fatalities, 1980-2022

% of World Population That Uses a Smartphone

Figure from https://www.businesswire.com/news/home/20210624005926/en/Strategy-Analytics-Half-the-World-Owns-a-Smartphone

Figure from https://www.npr.org/2024/01/16/1224913698/teslas-chicago-charging-extreme-cold

GENERATIVE AI IS PROBABILISTIC MODELING

GenAI is Probabilistic Modeling

$$p(x_{t+1} \mid x_1, \ldots, x_t)$$

What if I want to model EVERY possible interaction?

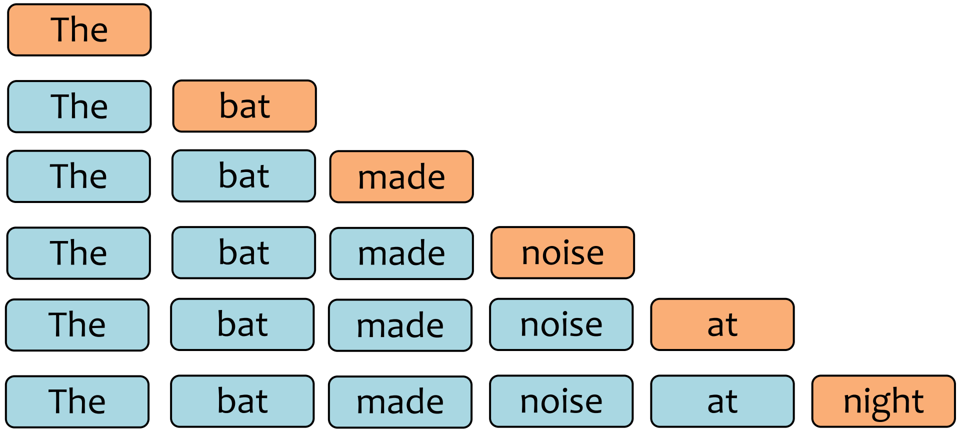
…or at least the interactions of the current variable with all those that came before it…

(RNN-LMs)

RNN Language Model

RNN Language Model: $p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\boldsymbol{\theta}}(w_{t-1}, \ldots, w_1))$

$p(w_1, w_2, w_3, \ldots, w_6) =$



The $p(w_1)$

The bat $p(w_2 \mid f_\theta(w_1))$

The bat made $p(w_3 \mid f_\theta(w_2, w_1))$

The bat made noise $p(w_4 \mid f_\theta(w_3, w_2, w_1))$

The bat made noise at $p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$

The bat made noise at night $p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

__Key Idea__:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector

Topics

- Generative models of text
  - RNN LMs / Autodiff
  - Transformer LMs
  - Pre-training, fine-tuning, evaluation, decoding
- Generative models of images
  - CNNs / Transformers for vision
  - GANs, Conditional GANs
  - Diffusion models
  - VAEs / Evaluation
- Applying and adapting foundation models
  - Reinforcement learning with human feedback (RLHF)
  - Parameter-efficient fine tuning
  - In-context learning for text
  - In-context learning for vision
- Multimodal foundation models
  - Text-to-image generation
  - Aligning multimodal representations (CLIP)
  - Visual-language foundation models
- Scaling up
  - Efficient decoding strategies
  - Distributed training / multi-GPU or TPU
  - Scaling laws and data
- What can go wrong?
  - Safety/bias/fairness, Hallucinations, Adversarial (e.g., prompt injection) attacks
  - Cheating – how to watermark, Legal issues, e.g., copyright,...
  - Drift in performance, Data contamination, Lack of ground truth
- Advanced Topics
  - Normalizing flows
  - Audio understanding and synthesis
  - Video synthesis

SYLLABUS HIGHLIGHTS

Syllabus Highlights

The syllabus is located on the course webpage:

http://423.mlcourse.org
http://623.mlcourse.org → https://www.cs.cmu.edu/~mgormley/courses/10423/

The **course policies** are **required** reading.

Syllabus Highlights

- **Grading**: 40% homework, 10% quizzes, 20% exam, 25% project, 5% participation
- **Exam**: in-class exam, Wed, Mar. 27
- **Homework**: 5 assignments
  - 6 grace days for homework assignments
  - Late submissions: 75% day 1, 50% day 2, 25% day 3
  - No submissions accepted after 3 days w/o extension
  - Extension requests: for emergency situations, see syllabus
- **Recitations**: Fridays, same time/place as lecture (optional, interactive sessions)
- **Readings**: required, online PDFs, recommended for after lecture
-

Technologies:
  - Piazza (discussion),
  - Gradescope (homework),
  - Google Forms (polls),
  - Zoom (livestream),
  - Panopto (video recordings)
- **Academic Integrity**:
  - Collaboration encouraged, but must be documented
  - Solutions must always be written independently
  - No re-use of found code / past assignments
  - Severe penalties (i.e.. failure)
  - (Policies differ from 10-301/10-601)
- **Office Hours**: posted on Google Calendar on “Office Hours” page

Lectures

- You should ask lots of questions
  - Interrupting (by raising a hand) to ask your question is strongly encouraged
  - Asking questions later (or in real time) on Piazza is also great
- When I ask a question…
  - I want you to answer
  - Even if you don’t answer, think it through as though I’m about to call on you
- Interaction improves learning (both in-class and at my office hours)

Prerequisites

What they are:

Introductory machine learning.
(i.e. 10-301, 10-315, 10-601, 10-701)

If you instead took an introduction to deep learning course, that is also fine
(i.e. 11-485/11-685/11-785)

What is not required:

- Deep learning
- PyTorch

Depending on which prerequisite course you took and in which semester you took it, you may or may not have been exposed to deep learning and/or PyTorch. Either way is fine.

Homework

There will be 5 homework assignments during the semester. The assignments will consist of both conceptual and programming problems.

	Main Topic	Implementation	Application Area	Type
HW0	PyTorch Primer	image classifier + Text classifier	vision + language	written + programming
HW1	Large Language Models	TransformerLM with sliding window attn.	char-level text gen	written + programming
HW2	Image Generation	GAN or diffusion model	image infilling	written + programming
HW3	Adapters for LLMs	Llama + LoRA	code + chat	written + programming
HW4	Multimodal Foundation Models	text-to-image model	vision + language	written + programming
HW623	(10-623 only)	read / analyze a recent research paper	genAI	video presentation

Project

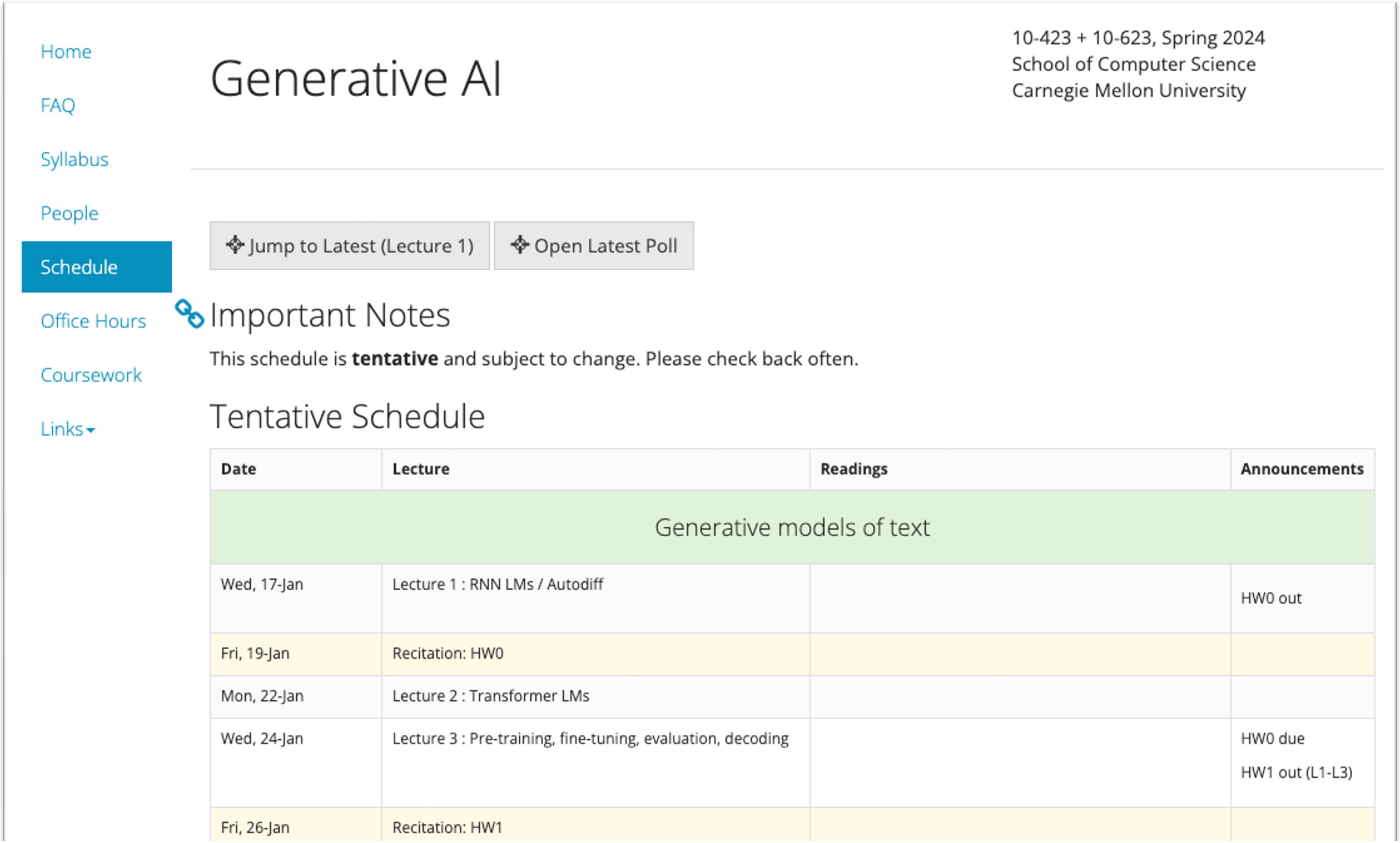
- Goals:
  - Explore a generative modeling technique of your choosing
  - Deeper understanding of methods in real-world application
  - Work in teams of 3 students

Textbooks

…do not exist for this course.

Instead, we will be directing your reading time to current research papers.

Where can I find…?



10-423 + 10-623, Spring 2024
School of Computer Science
Carnegie Mellon University

Generative AI

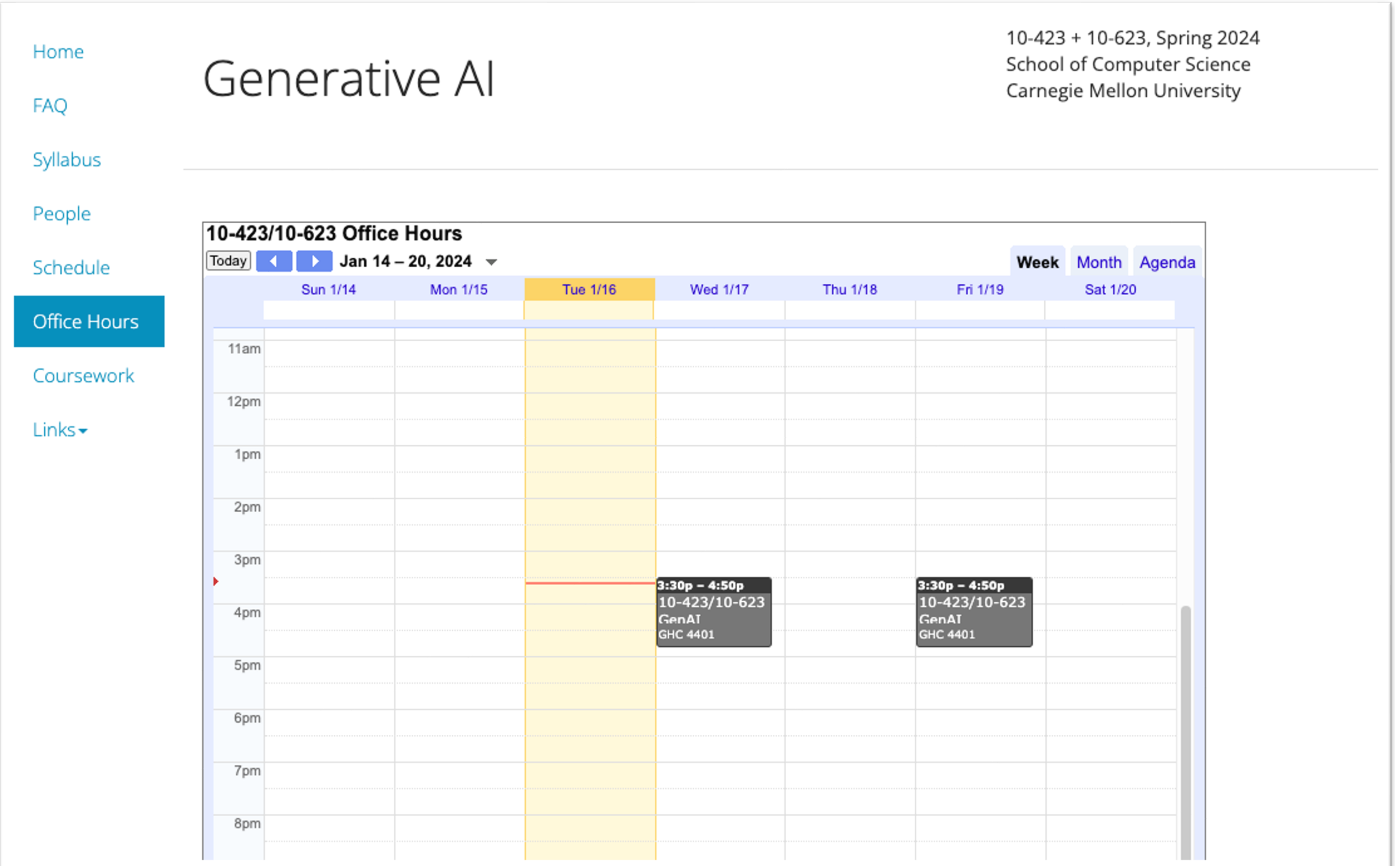
Important Notes

This schedule is **tentative** and subject to change. Please check back often.

Tentative Schedule

Date	Lecture	Readings	Announcements
Generative models of text			
Wed, 17-Jan	Lecture 1 : RNN LMs / Autodiff		HW0 out
Fri, 19-Jan	Recitation: HW0		
Mon, 22-Jan	Lecture 2 : Transformer LMs		
Wed, 24-Jan	Lecture 3 : Pre-training, fine-tuning, evaluation, decoding		HW0 due HW1 out (L1-L3)
Fri, 26-Jan	Recitation: HW1		

Where can I find…?



Generative AI

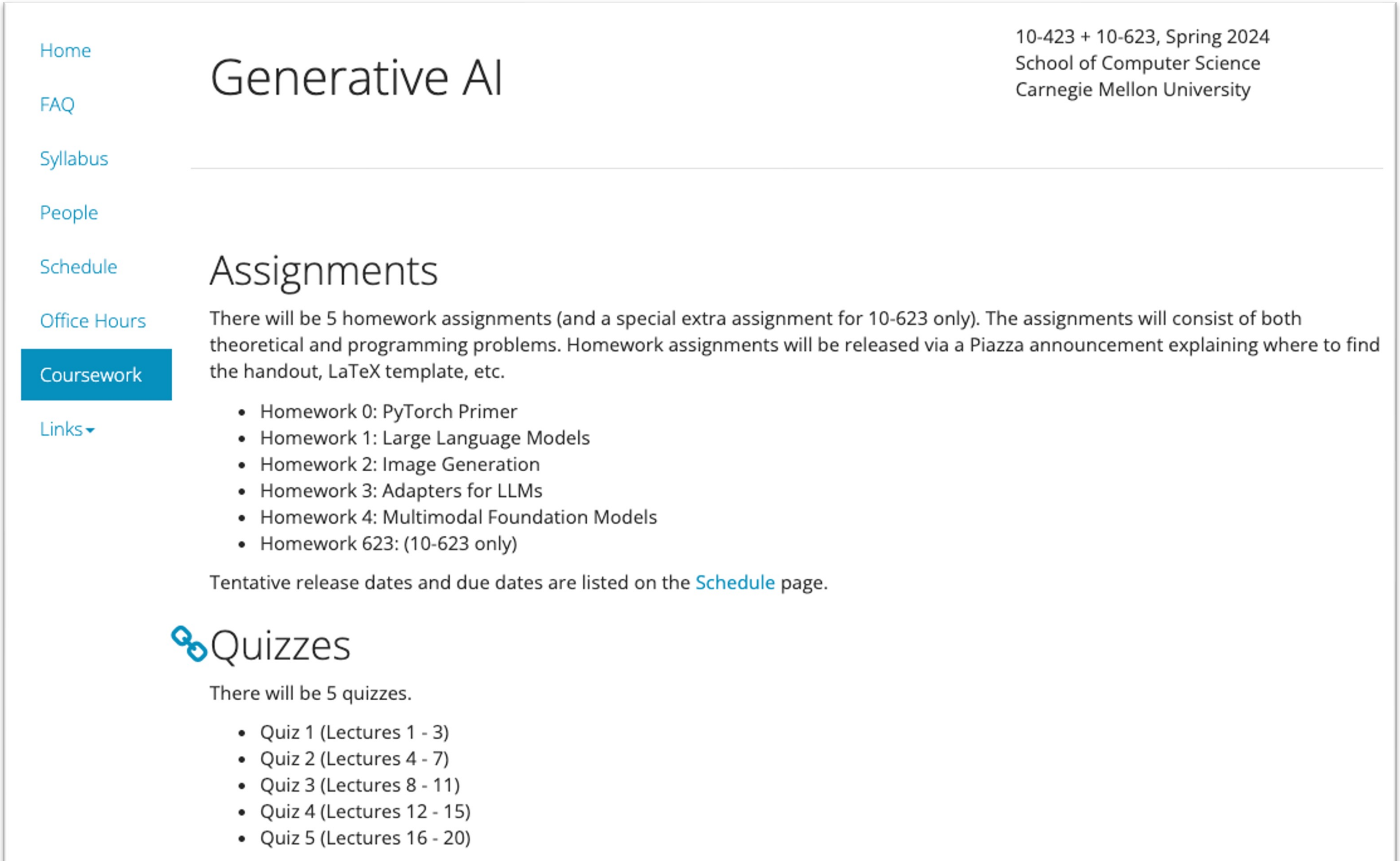
10-423 + 10-623, Spring 2024
School of Computer Science
Carnegie Mellon University

10-423/10-623 Office Hours

3:30p – 4:50p
10-423/10-623 GenAI
GHC 4401

3:30p – 4:50p
10-423/10-623 GenAI
GHC 4401

Where can I find…?



Generative AI

10-423 + 10-623, Spring 2024
School of Computer Science
Carnegie Mellon University

Assignments

There will be 5 homework assignments (and a special extra assignment for 10-623 only). The assignments will consist of both theoretical and programming problems. Homework assignments will be released via a Piazza announcement explaining where to find the handout, LaTeX template, etc.

- Homework 0: PyTorch Primer
- Homework 1: Large Language Models
- Homework 2: Image Generation
- Homework 3: Adapters for LLMs
- Homework 4: Multimodal Foundation Models
- Homework 623: (10-623 only)

Tentative release dates and due dates are listed on the Schedule page.

Quizzes

There will be 5 quizzes.

- Quiz 1 (Lectures 1 - 3)
- Quiz 2 (Lectures 4 - 7)
- Quiz 3 (Lectures 8 - 11)
- Quiz 4 (Lectures 12 - 15)
- Quiz 5 (Lectures 16 - 20)

Reminders

- **Homework 0: PyTorch + Weights & Biases**
  - Out: Wed, Jan 17
  - Due: Wed, Jan 24 at 11:59pm
  - Two parts:
    1. written part to Gradescope
    2. programming part to Gradescope
  - unique policy for this assignment: we will grant (essentially) any and all extension requests

Learning Objectives

You should be able to…

1. Differentiate between different mechanisms of learning such as parameter tuning and in-context learning.
2. Implement the foundational models underlying modern approaches to generative modeling, such as transformers and diffusion models.
3. Apply existing models to real-world generation problems for text, code, images, audio, and video.
4. Employ techniques for adapting foundation models to tasks such as fine-tuning, adapters, and in-context learning.
5. Enable methods for generative modeling to scale-up to large datasets of text, code, or images.
6. Use existing generative models to solve real-world discriminative problems and for other everyday use cases.
7. Analyze the theoretical properties of foundation models at scale.
8. Identify potential pitfalls of generative modeling for different modalities.
9. Describe societal impacts of large-scale generative AI systems.

Q&A

MODULE-BASED AUTOMATIC DIFFERENTIATION

Backpropagation

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function y = f(**x**). The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the "**computation graph**")
2. Visit each node in **topological order**.
   For variable u_i with inputs $v_1, \ldots, v_N$
   a. Compute $u_i = g_i(v_1, \ldots, v_N)$
   b. Store the result at the node

Backward Computation (Version A)

1. **Initialize** dy/dy = 1.
2. Visit each node v_j in **reverse topological order**.
   Let $u_1, \ldots, u_M$ denote all the nodes with v_j as an input
   Assuming that $y = h(\mathbf{u}) = h(u_1, \ldots, u_M)$
   and $\mathbf{u} = g(\mathbf{v})$ or equivalently $u_i = g_i(v_1, \ldots, v_j, \ldots, v_N)$ for all i
   a. We already know dy/du_i for all i
   b. Compute dy/dv_j as below (Choice of algorithm ensures computing (du_i/dv_j) is easy)

$$\frac{dy}{dv_j} = \sum_{i=1}^{M} \frac{dy}{du_i}\frac{du_i}{dv_j}$$

Return partial derivatives dy/du_i for all variables

Backpropagation

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function y = f(**x**). The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the "**computation graph"**)
2. Visit each node in **topological order**.
   For variable u_i with inputs $v_1,\ldots, v_N$
   a. Compute $u_i = g_i(v_1,\ldots, v_N)$
   b. Store the result at the node

Backward Computation (Version B)

1. **Initialize** all partial derivatives dy/du_j to 0 and $dy/dy = 1$.
2. Visit each node in **reverse topological order**.
   For variable $u_i = g_i(v_1,\ldots, v_N)$
   a. We already know dy/du_i
   b. Increment dy/dv_j by $(dy/du_i)(du_i/dv_j)$
   (Choice of algorithm ensures computing (du_i/dv_j) is easy)

Return partial derivatives dy/du_i for all variables

Backpropagation

Why is the backpropagation algorithm efficient?

1. Reuses **computation from the forward pass** in the backward pass
2. Reuses **partial derivatives** throughout the backward pass (*but only if the algorithm reuses shared computation in the forward pass*)

   (Key idea: partial derivatives in the backward pass should be thought of as variables stored for reuse)

A Recipe for
Mac

Gradients

Backpropagation can compute this gradient!

And it’s a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

$\boldsymbol{y}_i)$

2. Choose each of t

- Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

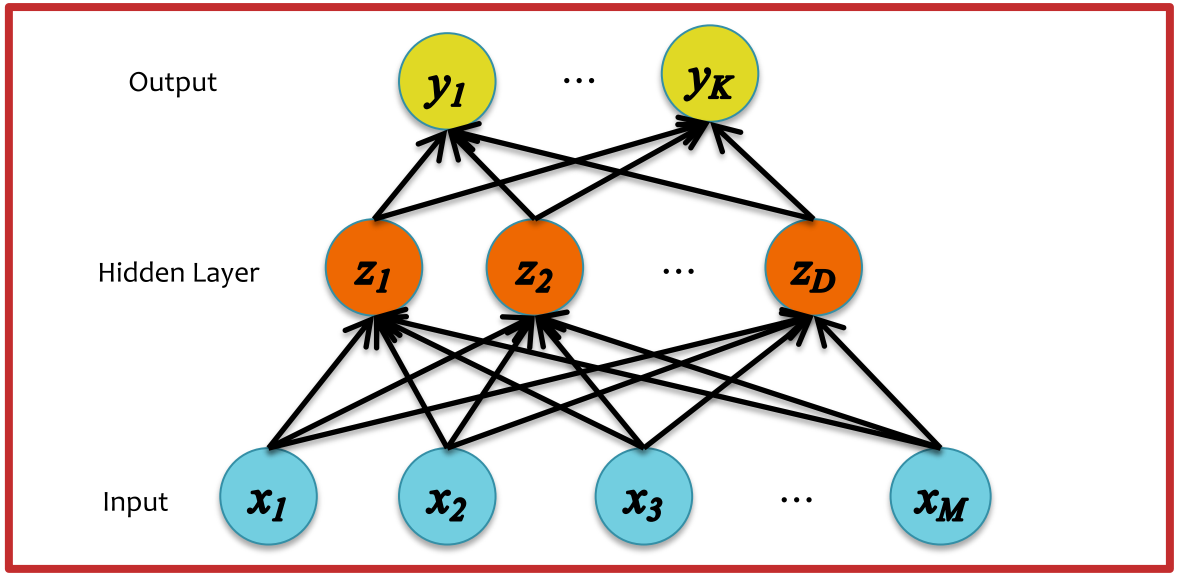
- Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

opposite the gradient)

$\boldsymbol{\theta}^{(t}$ $^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Backpropagation: Abstract Picture



Forward

5. $J = -\mathbf{y}^T \log \hat{\mathbf{y}}$
4. $\hat{\mathbf{y}} = \text{softmax}(\mathbf{b})$
3. $\mathbf{b} = \boldsymbol{\beta}\mathbf{z}$

2. $\mathbf{z} = \sigma(\mathbf{a})$
1. $\mathbf{a} = \boldsymbol{\alpha}\mathbf{x}$

Backward

6. $\mathbf{g}_{\hat{\mathbf{y}}} = -\mathbf{y} \div \hat{\mathbf{y}}$
7. $\mathbf{g}_{\mathbf{b}} = \mathbf{g}_{\hat{\mathbf{y}}}^T \left(\text{diag}(\hat{\mathbf{y}}) - \hat{\mathbf{y}}\hat{\mathbf{y}}^T\right)$
8. $\mathbf{g}_{\boldsymbol{\beta}} = \mathbf{g}_{\mathbf{b}}^T \mathbf{z}^T$

$\mathbf{g}_{\mathbf{z}} = \boldsymbol{\beta}^T \mathbf{g}_{\mathbf{b}}^T$

10. $\mathbf{g}_{\mathbf{a}} = \mathbf{g}_{\mathbf{z}} \odot \mathbf{z} \odot (1 - \mathbf{z})$
11. $\mathbf{g}_{\boldsymbol{\alpha}} = \mathbf{g}_{\mathbf{a}} \mathbf{x}^T$

(F) **Loss**
$J = \sum_{k=1}^{K} y_k^* \log(y_k)$

(E) **Output (softmax)**
$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$

(D) **Output (linear)**
$b_k = \sum_{j=0}^{D} \beta_{kj} z_j \ \ \forall k$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \ \forall j$

(A) **Input**
Given $x_i, \ \ \forall i$

Backpropagation: Procedural Method

Algorithm 1 Forward Computation

```
procedure NNFORWARD(Training example $(\mathbf{x}, \mathbf{y})$, Params $\boldsymbol{\alpha}, \boldsymbol{\beta}$)
    $\mathbf{a} = \boldsymbol{\alpha}\mathbf{x}$
    $\mathbf{z} = \sigma(\mathbf{a})$
    $\mathbf{b} = \boldsymbol{\beta}\mathbf{z}$
    $\hat{\mathbf{y}} = \text{softmax}(\mathbf{b})$
    $J = -\mathbf{y}^T \log \hat{\mathbf{y}}$
    $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{z}, \mathbf{b}, \hat{\mathbf{y}}, J)$
    return intermediate quantities $\mathbf{o}$
```

Algorithm 2 Backpropagation

```
procedure NNBACKWARD(Training example $(\mathbf{x}, \mathbf{y})$, Params $\boldsymbol{\alpha}, \boldsymbol{\beta}$, Intermediates $\mathbf{o}$)
    Place intermediate quantities $\mathbf{x}, \mathbf{a}, \mathbf{z}, \mathbf{b}, \hat{\mathbf{y}}, J$ in $\mathbf{o}$ in scope
    $\mathbf{g}_{\hat{\mathbf{y}}} = -\mathbf{y} \div \hat{\mathbf{y}}$
    $\mathbf{g}_{\mathbf{b}} = \mathbf{g}_{\hat{\mathbf{y}}}^T \left(\text{diag}(\hat{\mathbf{y}}) - \hat{\mathbf{y}}\hat{\mathbf{y}}^T\right)$
    $\mathbf{g}_{\boldsymbol{\beta}} = \mathbf{g}_{\mathbf{b}}^T \mathbf{z}^T$
    $\mathbf{g}_{\mathbf{z}} = \boldsymbol{\beta}^T \mathbf{g}_{\mathbf{b}}^T$
    $\mathbf{g}_{\mathbf{a}} = \mathbf{g}_{\mathbf{z}} \odot \mathbf{z} \odot (1 - \mathbf{z})$
    $\mathbf{g}_{\boldsymbol{\alpha}} = \mathbf{g}_{\mathbf{a}} \mathbf{x}^T$
    return parameter gradients $\mathbf{g}_{\boldsymbol{\alpha}}, \mathbf{g}_{\boldsymbol{\beta}}$
```

Drawbacks of Procedural Method

1. Hard to reuse / adapt for other models
2. (Possibly) harder to make individual steps more efficient
3. Hard to find source of error if finite-difference check reports an error (since it tells you only that there is an error somewhere in those 17 lines of code)

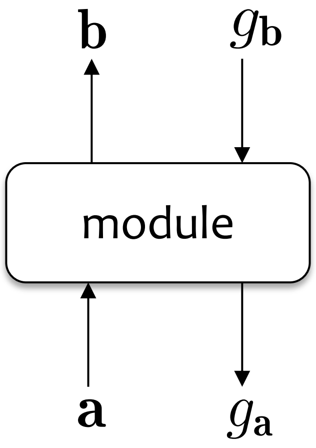
Module-based AutoDiff

Module-based automatic differentiation (AD / Autodiff) is a technique that has long been used to develop libraries for deep learning

- **Dynamic neural network packages** allow a specification of the computation graph dynamically at runtime
  - PyTorch http://pytorch.org
  - Torch http://torch.ch
  - DyNet https://dynet.readthedocs.io
  - TensorFlow with Eager Execution https://www.tensorflow.org
- **Static neural network packages** require a static specification of a computation graph which is subsequently compiled into code
  - TensorFlow with Graph Execution https://www.tensorflow.org
  - Aesara (and Theano) https://aesara.readthedocs.io
  - *(These libraries are also module-based, but herein by "module-based AD" we mean the dynamic approach)*

Module-based AutoDiff

- **Key Idea:**
  - componentize the computation of the neural-network into layers
  - each layer consolidates multiple **real-valued nodes** in the computation graph (a subset of them) into one **vector-valued node** (aka. a **module**)
- Each **module** is capable of two actions:



1. Forward computation of output $\mathbf{b} = [b_1, \ldots, b_B]$ given input $\mathbf{a} = [a_1, \ldots, a_A]$ via some differentiable function f. That is $\mathbf{b} = f(\mathbf{a})$.

2. Backward computation of the gradient of the input $\mathbf{g_a} = \nabla_{\mathbf{a}} J = [\frac{\partial J}{\partial a_1}, \ldots, \frac{\partial J}{\partial a_A}]$ given the gradient of output $\mathbf{g_b} = \nabla_{\mathbf{b}} J = [\frac{\partial J}{\partial b_1}, \ldots, \frac{\partial J}{\partial b_B}]$, where J is the final real-valued output of the entire computation graph. This is done via the chain rule $\frac{\partial J}{\partial a_i} = \sum_{j=1}^{J} \frac{\partial J}{\partial b_j} \frac{d b_j}{d a_i}$ for all $i \in \{1, \ldots, A\}$.

Module-based AutoDiff

Dimensions: input $\mathbf{a} \in \mathbb{R}^A$, output $\mathbf{b} \in \mathbb{R}^B$, gradient of output $\mathbf{g_a} \triangleq \nabla_\mathbf{a} J \in \mathbb{R}^A$, and gradient of input $\mathbf{g_b} \triangleq \nabla_\mathbf{b} J \in \mathbb{R}^B$.

Sigmoid Module The sigmoid layer has only one input vector $\mathbf{a}$. Below σ is the sigmoid applied element-wise, and $\odot$ is element-wise multiplication s.t. $\mathbf{u} \odot \mathbf{v} = [u_1 v_1, \ldots, u_M v_M]$.

```
procedure SigmoidForward($\mathbf{a}$)
    $\mathbf{b} = \sigma(\mathbf{a})$
    return $\mathbf{b}$
procedure SigmoidBackward($\mathbf{a}$, $\mathbf{b}$, $\mathbf{g_b}$)
    $\mathbf{g_a} = \mathbf{g_b} \odot \mathbf{b} \odot (1 - \mathbf{b})$
    return $\mathbf{g_a}$
```

Softmax Module The softmax layer has only one input vector $\mathbf{a}$. For any vector $\mathbf{v} \in \mathbb{R}^D$, we have that diag$(\mathbf{v})$ returns a $D \times D$ diagonal matrix whose diagonal entries are $v_1, v_2, \ldots, v_D$ and whose non-diagonal entries are zero.

```
procedure SoftmaxForward($\mathbf{a}$)
    $\mathbf{b} = \text{softmax}(\mathbf{a})$
    return $\mathbf{b}$
procedure SoftmaxBackward($\mathbf{a}$, $\mathbf{b}$, $\mathbf{g_b}$)
    $\mathbf{g_a} = \mathbf{g_b}^T \left(\text{diag}(\mathbf{b}) - \mathbf{b}\mathbf{b}^T\right)$
    return $\mathbf{g_a}$
```

Linear Module The linear layer has two inputs: a vector $\mathbf{a}$ and parameters $\omega \in \mathbb{R}^{B \times A}$. The output $\mathbf{b}$ is not used by LinearBackward, but we pass it in for consistency of form.

```
procedure LinearForward($\mathbf{a}$, $\boldsymbol{\omega}$)
    $\mathbf{b} = \boldsymbol{\omega}\mathbf{a}$
    return $\mathbf{b}$
procedure LinearBackward($\mathbf{a}$, $\omega$, $\mathbf{b}$, $\mathbf{g_b}$)
    $\mathbf{g}_{\boldsymbol{\omega}} = \mathbf{g_b}\mathbf{a}^T$
    $\mathbf{g_a} = \boldsymbol{\omega}^T \mathbf{g_b}$
    return $\mathbf{g}_{\boldsymbol{\omega}}, \mathbf{g_a}$
```

Cross-Entropy Module The cross-entropy layer has two inputs: a gold one-hot vector $\mathbf{a}$ and a predicted probability distribution $\hat{\mathbf{a}}$. It's output $b \in \mathbb{R}$ is a scalar. Below $\div$ is element-wise division. The output b is not used by CrossEntropyBackward, but we pass it in for consistency of form.

```
procedure CrossEntropyForward($\mathbf{a}$, $\hat{\mathbf{a}}$)
    $b = -\mathbf{a}^T \log \hat{\mathbf{a}}$
    return $\mathbf{b}$
procedure CrossEntropyBackward($\mathbf{a}$, $\hat{\mathbf{a}}$, $b$, $g_b$)
    $\mathbf{g}_{\hat{\mathbf{a}}} = -g_b(\mathbf{a} \div \hat{\mathbf{a}})$
    return $\mathbf{g_a}$
```

Module-based AutoDiff

Algorithm 1 Forward Computation

```
procedure NNFORWARD(Training example $(\mathbf{x}, \mathbf{y})$, Parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$)
    $\mathbf{a} = \text{LINEARFORWARD}(\mathbf{x}, \boldsymbol{\alpha})$
    $\mathbf{z} = \text{SIGMOIDFORWARD}(\mathbf{a})$
    $\mathbf{b} = \text{LINEARFORWARD}(\mathbf{z}, \boldsymbol{\beta})$
    $\hat{\mathbf{y}} = \text{SOFTMAXFORWARD}(\mathbf{b})$
    $J = \text{CROSSENTROPYFORWARD}(\mathbf{y}, \hat{\mathbf{y}})$
    $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{z}, \mathbf{b}, \hat{\mathbf{y}}, J)$
    return intermediate quantities $\mathbf{o}$
```

Algorithm 2 Backpropagation

```
procedure NNBACKWARD(Training example $(\mathbf{x}, \mathbf{y})$, Parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, Intermediates $\mathbf{o}$)
    Place intermediate quantities $\mathbf{x}, \mathbf{a}, \mathbf{z}, \mathbf{b}, \hat{\mathbf{y}}, J$ in $\mathbf{o}$ in scope
    $g_J = \frac{dJ}{dJ} = 1$                                        ▷ Base case
    $\mathbf{g}_{\hat{\mathbf{y}}} = \text{CROSSENTROPYBACKWARD}(\mathbf{y}, \hat{\mathbf{y}}, J, g_J)$
    $\mathbf{g}_{\mathbf{b}} = \text{SOFTMAXBACKWARD}(\mathbf{b}, \hat{\mathbf{y}}, \mathbf{g}_{\hat{\mathbf{y}}})$
    $\mathbf{g}_{\boldsymbol{\beta}}, \mathbf{g}_{\mathbf{z}} = \text{LINEARBACKWARD}(\mathbf{z}, \mathbf{b}, \mathbf{g}_{\mathbf{b}})$
    $\mathbf{g}_{\mathbf{a}} = \text{SIGMOIDBACKWARD}(\mathbf{a}, \mathbf{z}, \mathbf{g}_{\mathbf{z}})$
    $\mathbf{g}_{\boldsymbol{\alpha}}, \mathbf{g}_{\mathbf{x}} = \text{LINEARBACKWARD}(\mathbf{x}, \mathbf{a}, \mathbf{g}_{\mathbf{a}})$     ▷ We discard $\mathbf{g}_{\mathbf{x}}$
    return parameter gradients $\mathbf{g}_{\boldsymbol{\alpha}}, \mathbf{g}_{\boldsymbol{\beta}}$
```

Advantages of Module-based AutoDiff

1. Easy to reuse / adapt for other models
2. Encapsulated layers are easier to optimize (e.g. implement in C++ or CUDA)
3. Easier to find bugs because we can run a finite-difference check on each layer separately

Module-based AutoDiff (OOP Version)

Object-Oriented Implementation:

- Let each module be an **object**
- Then allow the **control flow** dictate the creation of the **computation graph**
- No longer need to implement NNBackward(·), just follow the computation graph in **reverse topological order**

```
class Sigmoid(Module)
    method forward($\mathbf{a}$)
        $\mathbf{b} = \sigma(\mathbf{a})$
        return $\mathbf{b}$
    method backward($\mathbf{a}$, $\mathbf{b}$, $\mathbf{g}_\mathbf{b}$)
        $\mathbf{g}_\mathbf{a} = \mathbf{g}_\mathbf{b} \odot \mathbf{b} \odot (1 - \mathbf{b})$
        return $\mathbf{g}_\mathbf{a}$
```

```
class Linear(Module)
    method forward($\mathbf{a}$, $\boldsymbol{\omega}$)
        $\mathbf{b} = \boldsymbol{\omega}\mathbf{a}$
        return $\mathbf{b}$
    method backward($\mathbf{a}$, $\boldsymbol{\omega}$, $\mathbf{b}$, $\mathbf{g}_\mathbf{b}$)
        $\mathbf{g}_{\boldsymbol{\omega}} = \mathbf{g}_\mathbf{b}\mathbf{a}^T$
        $\mathbf{g}_\mathbf{a} = \boldsymbol{\omega}^T\mathbf{g}_\mathbf{b}$
        return $\mathbf{g}_{\boldsymbol{\omega}}, \mathbf{g}_\mathbf{a}$
```

```
class Softmax(Module)
    method forward($\mathbf{a}$)
        $\mathbf{b} = \texttt{softmax}(\mathbf{a})$
        return $\mathbf{b}$
    method backward($\mathbf{a}$, $\mathbf{b}$, $\mathbf{g}_\mathbf{b}$)
        $\mathbf{g}_\mathbf{a} = \mathbf{g}_\mathbf{b}^T \left(\texttt{diag}(\mathbf{b}) - \mathbf{b}\mathbf{b}^T\right)$
        return $\mathbf{g}_\mathbf{a}$
```

```
class CrossEntropy(Module)
    method forward($\mathbf{a}$, $\hat{\mathbf{a}}$)
        $b = -\mathbf{a}^T \log \hat{\mathbf{a}}$
        return $\mathbf{b}$
    method backward($\mathbf{a}$, $\hat{\mathbf{a}}$, $b$, $g_b$)
        $\mathbf{g}_{\hat{\mathbf{a}}} = -g_b(\mathbf{a} \div \hat{\mathbf{a}})$
        return $\mathbf{g}_\mathbf{a}$
```

Module-based AutoDiff (OOP Version)

```
class NeuralNetwork(Module):

    method init()
        lin1_layer = Linear()
        sig_layer = Sigmoid()
        lin2_layer = Linear()
        soft_layer = Softmax()
        ce_layer = CrossEntropy()

    method forward(Tensor $\mathbf{x}$, Tensor $\mathbf{y}$, Tensor $\boldsymbol{\alpha}$, Tensor $\boldsymbol{\beta}$)
        $\mathbf{a}$ = lin1_layer.apply_fwd($\mathbf{x}, \boldsymbol{\alpha}$)
        $\mathbf{z}$ = sig_layer.apply_fwd($\mathbf{a}$)
        $\mathbf{b}$ = lin2_layer.apply_fwd($\mathbf{z}, \boldsymbol{\beta}$)
        $\hat{\mathbf{y}}$ = soft_layer.apply_fwd($\mathbf{b}$)
        $J$ = ce_layer.apply_fwd($\mathbf{y}, \hat{\mathbf{y}}$)
        return $J$.out_tensor

    method backward(Tensor $\mathbf{x}$, Tensor $\mathbf{y}$, Tensor $\boldsymbol{\alpha}$, Tensor $\boldsymbol{\beta}$)
        tape_bwd()
        return lin1_layer.in_gradients[1], lin2_layer.in_gradients[1]
```

Module-based AutoDiff (OOP Version)

```
class NeuralNetwork(Module):

    method init()
        lin1_layer = Linear()
        sig_layer = Sigmoid()
        lin2_layer = Linear()
        soft_layer = Softmax()
        ce_layer = CrossEntropy()

    method forward(Tensor x , Tensor y , Tensor
        $\mathbf{a}$ =lin1_layer.apply_fwd($\mathbf{x}, \boldsymbol{\alpha}$)
        $\mathbf{z}$ =sig_layer.apply_fwd($\mathbf{a}$)
        $\mathbf{b}$ =lin2_layer.apply_fwd($\mathbf{z}, \boldsymbol{\beta}$)
        $\hat{\mathbf{y}}$ =soft_layer.apply_fwd($\mathbf{b}$)
        $J$ =ce_layer.apply_fwd($\mathbf{y}, \hat{\mathbf{y}}$)
        return J.out_tensor

    method backward(Tensor x , Tensor y , Tenso
        tape_bwd()
        return lin1_layer.in_gradients[1] , lin2_la
```

```
global tape = stack()

class Module:

    method init()
        out_tensor = null
        out_gradient = 1

    method apply_fwd(List in_modules)
        in_tensors = [x.out_tensor for x in in_modules]
        out_tensor = forward(in_tensors)
        tape.push(self)
        return self

    method apply_bwd():
        in_gradients = backward(in_tensors , out_tensor , out_gradient)
        for i in 1,..., len(in_modules):
            in_modules[i].out_gradient += in_gradients[i]
        return self

function tape_bwd():
    while len(tape) > 0
        m = tape.pop()
        m.apply_bwd()
```

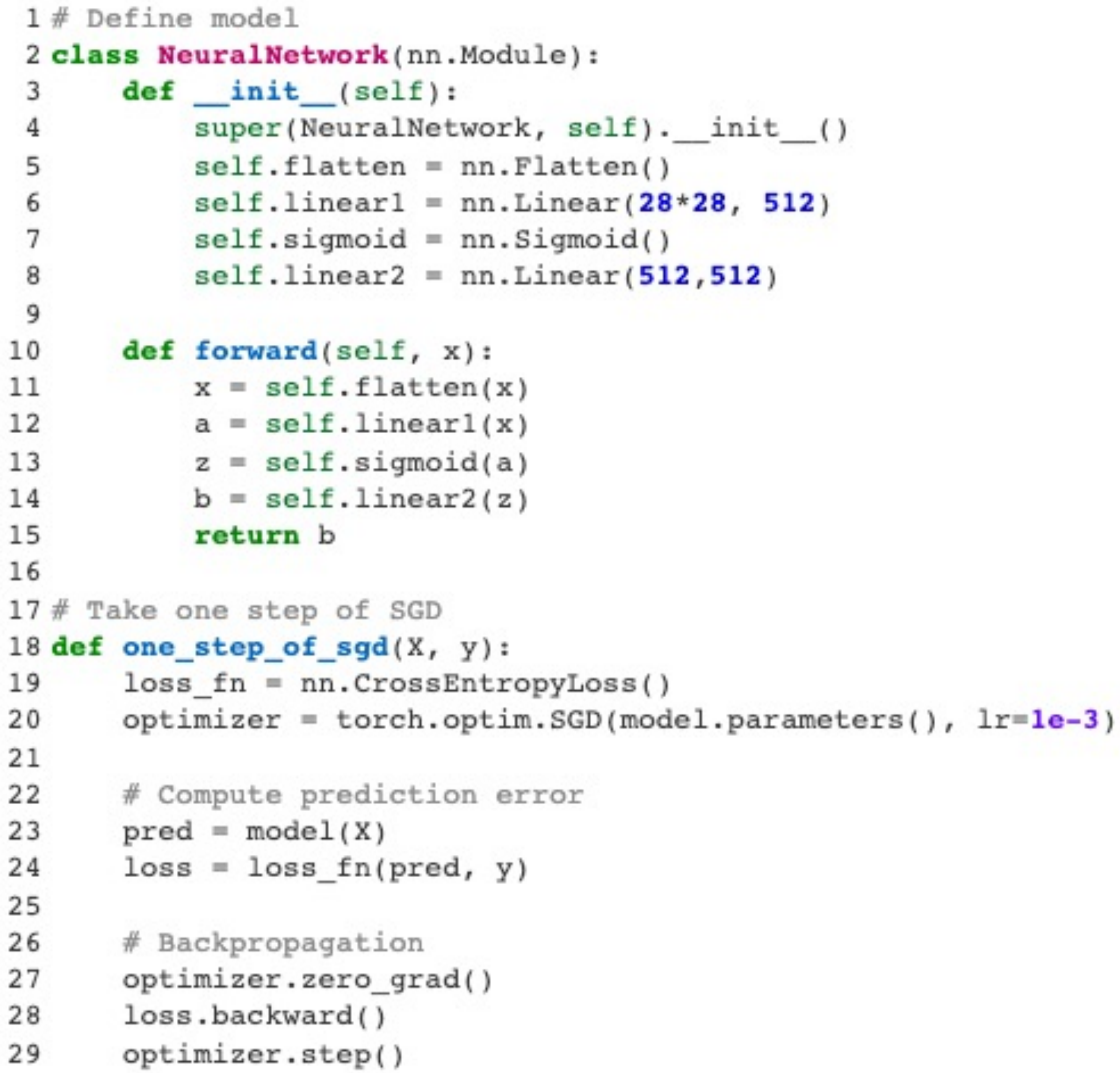
Module-based AutoDiff (OOP Version)

```
class NeuralNetwork(Module):

    method init()
        lin1_layer = Linear()
        sig_layer = Sigmoid()
        lin2_layer = Linear()
        soft_layer = Softmax()
        ce_layer = CrossEntropy()

    method forward(Tensor x , Tensor y , Tensor
        $\mathbf{a}$ =lin1_layer.apply_fwd($\mathbf{x}, \boldsymbol{\alpha}$)
        $\mathbf{z}$ =sig_layer.apply_fwd($\mathbf{a}$)
        $\mathbf{b}$ =lin2_layer.apply_fwd($\mathbf{z}, \boldsymbol{\beta}$)
        $\hat{\mathbf{y}}$ =soft_layer.apply_fwd($\mathbf{b}$)
        $J$ =ce_layer.apply_fwd($\mathbf{y}, \hat{\mathbf{y}}$)
        return $J$.out_tensor

    method backward(Tensor x , Tensor y , Tenso
        tape_bwd()
        return lin1_layer.in_gradients[1] , lin2_la
```

```
global tape = stack()

class Module:

    method init()
        out_tensor = null
        out_gradient = 1

    method apply_fwd(List in_modules)
        in_tensors = [x.out_tensor for x in in_modules]
        out_tensor = forward(in_tensors)
        tape.push(self)
        return self

    method apply_bwd():
        in_gradients = backward(in_tensors , out_tensor , out_gradient)
        for i in $1, \ldots,$ len(in_modules):
            in_modules[i].out_gradient += in_gradients[i]
        return self

function tape_bwd():
    while len(tape) $>$ 0
        m = tape.pop()
        m.apply_bwd()
```

PyTorch

The same simple neural network we defined in pseudocode can also be defined in PyTorch.



```python
# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(28*28, 512)
        self.sigmoid = nn.Sigmoid()
        self.linear2 = nn.Linear(512,512)

    def forward(self, x):
        x = self.flatten(x)
        a = self.linear1(x)
        z = self.sigmoid(a)
        b = self.linear2(z)
        return b

# Take one step of SGD
def one_step_of_sgd(X, y):
    loss_fn = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

    # Compute prediction error
    pred = model(X)
    loss = loss_fn(pred, y)

    # Backpropagation
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Example adapted from https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

PyTorch

Q: Why don’t we call linear.forward() in PyTorch?

A: This is just syntactic sugar. There’s a special method in Python `__call__` that allows you to define what happens when you treat an object as if it were a function.

In other words, running the following:

```
linear(x)
```

is equivalent to running:

```
linear.__call__(x)
```

which in PyTorch is (nearly) the same as running:

```
linear.forward(x)
```

This is because PyTorch defines every Module’s `__call__` method to be something like this:

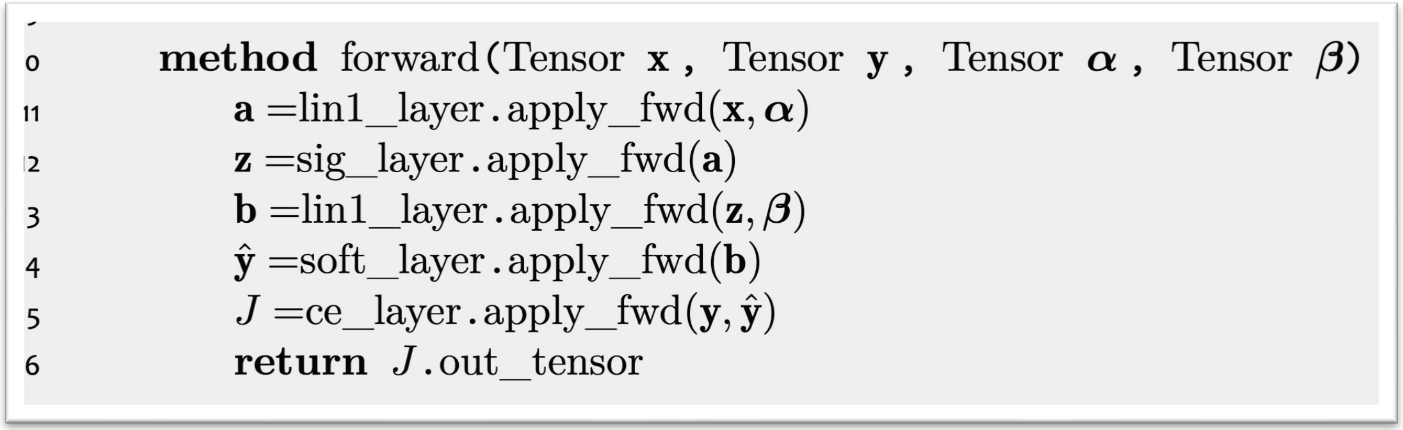
```
def __call__(self):
    self.forward()
```

PyTorch

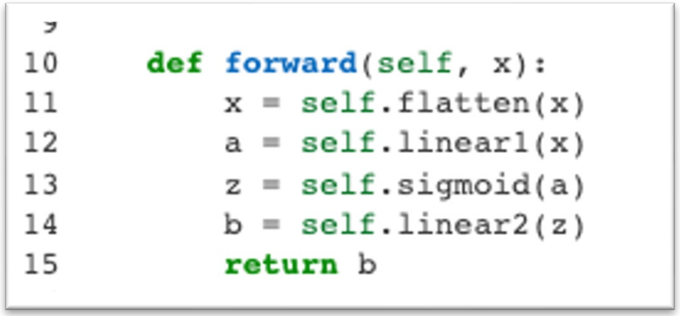
Q: Why don't we pass in the parameters to a PyTorch Module?

A: This just makes your code cleaner.

In PyTorch, you store the parameters inside the Module and "mark" them as parameters that should contribute to the eventual gradient used by an optimizer



```
method forward(Tensor $\mathbf{x}$ , Tensor $\mathbf{y}$ , Tensor $\boldsymbol{\alpha}$ , Tensor $\boldsymbol{\beta}$)
    $\mathbf{a}$ =lin1_layer.apply_fwd($\mathbf{x}, \boldsymbol{\alpha}$)
    $\mathbf{z}$ =sig_layer.apply_fwd($\mathbf{a}$)
    $\mathbf{b}$ =lin1_layer.apply_fwd($\mathbf{z}, \boldsymbol{\beta}$)
    $\hat{\mathbf{y}}$ =soft_layer.apply_fwd($\mathbf{b}$)
    $J$ =ce_layer.apply_fwd($\mathbf{y}, \hat{\mathbf{y}}$)
    return $J$.out_tensor
```

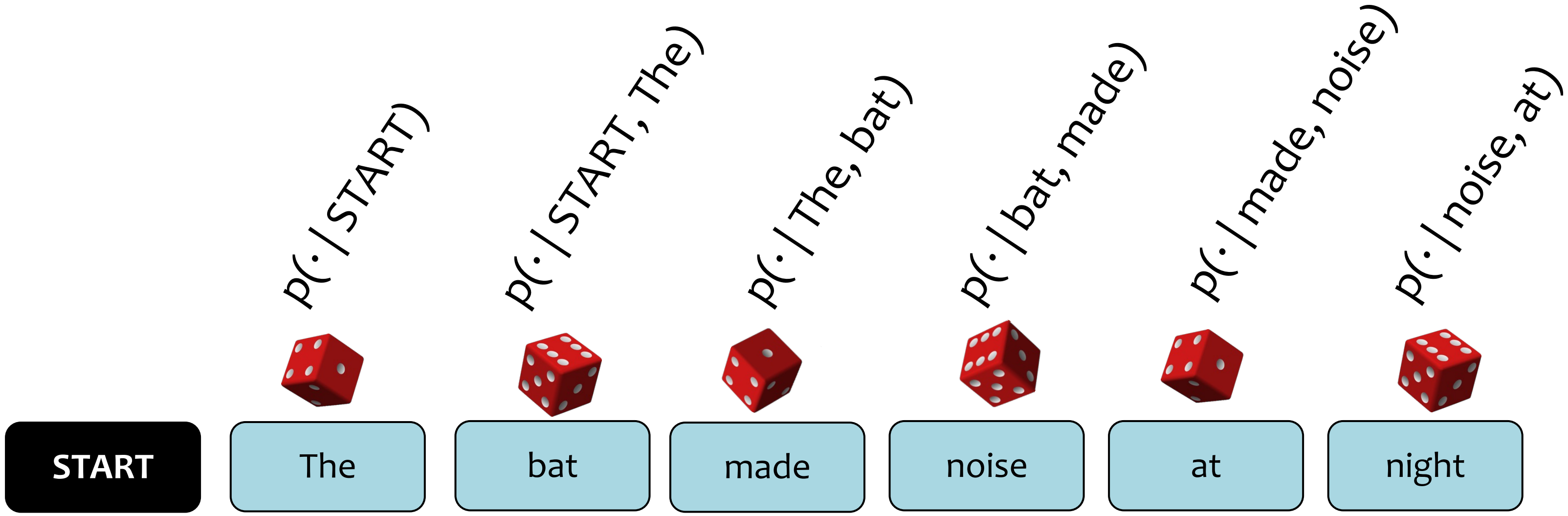


```
    def forward(self, x):
        x = self.flatten(x)
        a = self.linear1(x)
        z = self.sigmoid(a)
        b = self.linear2(z)
        return b
```

BACKGROUND:
N-GRAM LANGUAGE MODELS

n-Gram Language Model

- *Goal*: Generate realistic looking sentences in a human language
- *Key Idea*: condition on the last n-1 words to sample the nth word



$p(\cdot \mid \text{START})$ $p(\cdot \mid \text{START, The})$ $p(\cdot \mid \text{The, bat})$ $p(\cdot \mid \text{bat, made})$ $p(\cdot \mid \text{made, noise})$ $p(\cdot \mid \text{noise, at})$

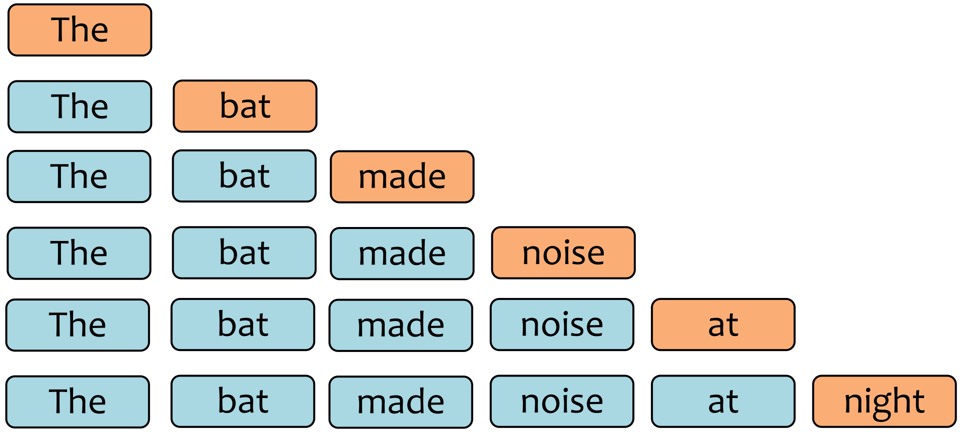
START The bat made noise at night

The Chain Rule of Probability

<u>Question</u>: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

$$p(w_1, w_2, w_3, \ldots, w_6) =$$



The	$p(w_1)$
The bat	$p(w_2 \mid w_1)$
The bat made	$p(w_3 \mid w_2, w_1)$
The bat made noise	$p(w_4 \mid w_3, w_2, w_1)$
The bat made noise at	$p(w_5 \mid w_4, w_3, w_2, w_1)$
The bat made noise at night	$p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$

The Chain Rule of Probability

Question: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

Chain rule of probability: $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, \ldots, w_1)$$

$p(w_1, w$ $w_3, \ldots , w_6) =$

The $p(w_1)$

The

The

The

The

The $v_1)$

$p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$

Note: This is called the chain **rule** because it is **always** true for every probability distribution

n-Gram Language Model

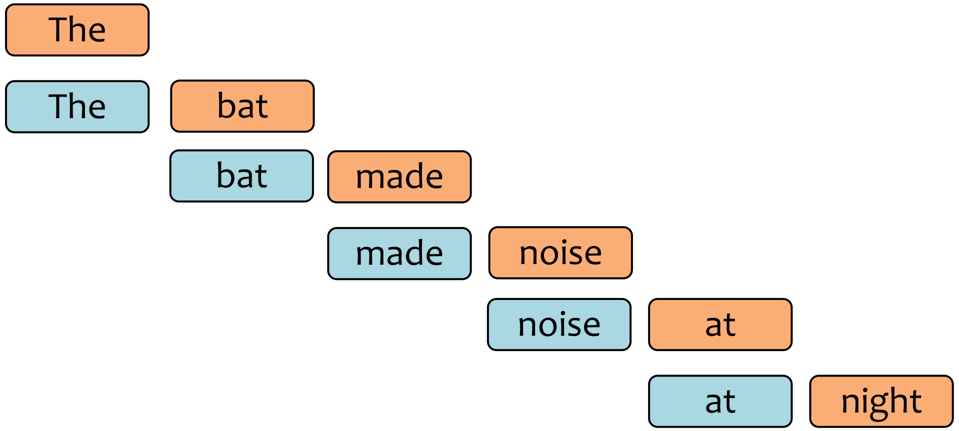
Question: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=2)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$



The — $p(w_1)$

The bat — $p(w_2 \mid w_1)$

bat made — $p(w_3 \mid w_2)$

made noise — $p(w_4 \mid w_3)$

noise at — $p(w_5 \mid w_4)$

at night — $p(w_6 \mid w_5)$

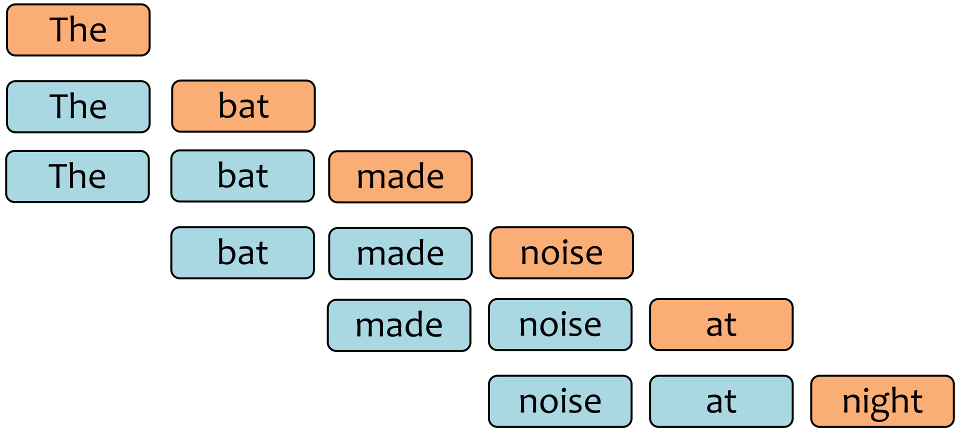
n-Gram Language Model

Question: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=3) $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$



$p(w_1)$

$p(w_2 \mid w_1)$

$p(w_3 \mid w_2, w_1)$

$p(w_4 \mid w_3, w_2)$

$p(w_5 \mid w_4, w_3)$

$p(w_6 \mid w_5, w_4)$

n-Gram Language Model

Question: How can we **define** a probability distribution over a sequence of length T?

The bat made noise at night

w_1 w_2 w_3 w_4 w_5 w_6

n-Gram Model (n=3)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$p(w_1, w\ \ w_3, \ldots, w_6) =$

The

$p(w_1)$

The

The

Note: This is called a **model** because we made some **assumptions** about how many previous words to condition on (i.e. only n-1 words)

Learning an n-Gram Model

Question: How do we **learn** the probabilities for the n-Gram Model?

$p(w_t \mid w_{t-2} = \text{The}, w_{t-1} = \text{bat})$

w_t	$p(\cdot \mid \cdot, \cdot)$
ate	0.015
…	
flies	0.046
…	
zebra	0.000

$p(w_t \mid w_{t-2} = \text{made}, w_{t-1} = \text{noise})$

w_t	$p(\cdot \mid \cdot, \cdot)$
at	0.020
…	
pollution	0.030
…	
zebra	0.000

$p(w_t \mid w_{t-2} = \text{cows}, w_{t-1} = \text{eat})$

w_t	$p(\cdot \mid \cdot, \cdot)$
corn	0.420
…	
grass	0.510
…	
zebra	0.000

Learning an n-Gram Model

Question: How do we **learn** the probabilities for the n-Gram Model?

Answer: From data! Just **count** n-gram frequencies

…the **cows eat** grass…
…our **cows eat hay** daily…
…factory-farm **cows eat** corn…
…on an organic farm, **cows eat hay** and…
…do your **cows eat** grass or corn?...
…what do **cows eat if** they have…
…**cows eat** corn when there is no…
…which **cows eat which** foods depends…
…if **cows eat** grass…
…when **cows eat** corn their stomachs…
…should we let **cows eat** corn?...

$p(w_t \mid w_{t-2} = \text{cows}, w_{t-1} = \text{eat})$

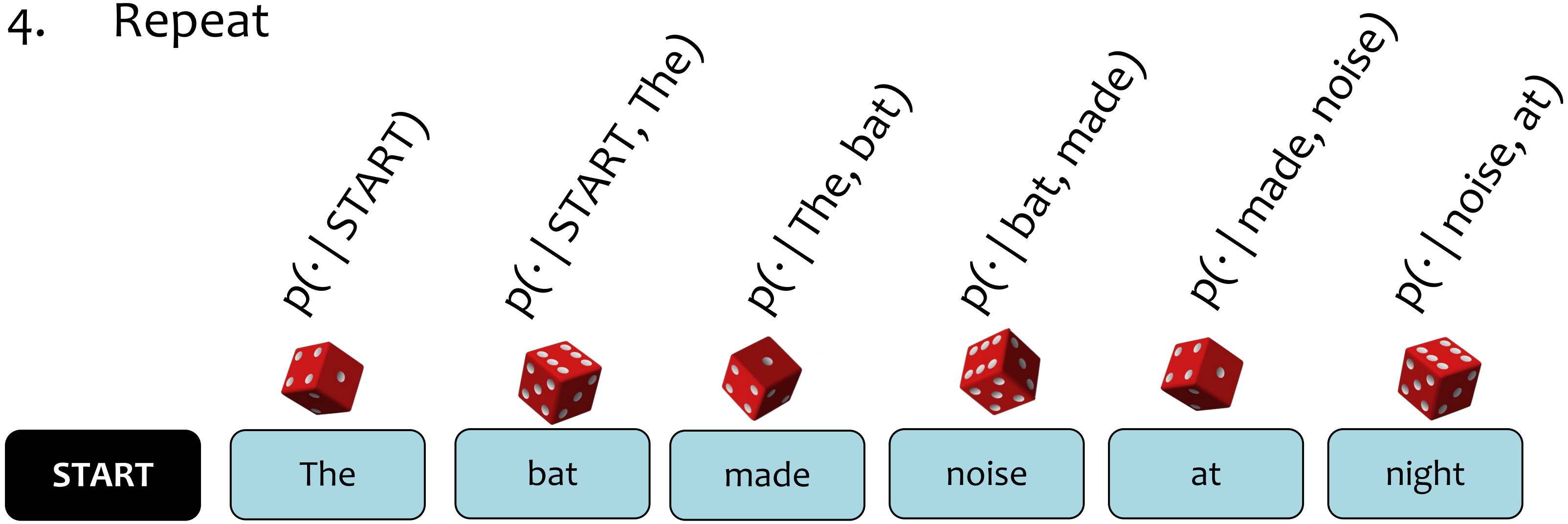
w_t	$p(\cdot \mid \cdot, \cdot)$
corn	4/11
grass	3/11
hay	2/11
if	1/11
which	1/11

Sampling from a Language Model

Question: How do we sample from a Language Model?

Answer:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



$p(\cdot \mid \text{START})$ $p(\cdot \mid \text{START, The})$ $p(\cdot \mid \text{The, bat})$ $p(\cdot \mid \text{bat, made})$ $p(\cdot \mid \text{made, noise})$ $p(\cdot \mid \text{noise, at})$

START The bat made noise at night

Sampling from a Language Model

Question: How do we sample from a Language Model?

Answer:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat

Training Data (Shakespeaere)

I tell you, friends, most charitable care ave the patricians of you. For your wants, Your suffering in this dearth, you may as well Strike at the heaven with your staves as lift them Against the Roman state, whose course will on The way it takes, cracking ten thousand curbs Of more strong link asunder than can ever Appear in your impediment. For the dearth, The gods, not the patricians, make it, and Your knees to them, not arms, must help.

5-Gram Model

```
Approacheth, denay. dungy
Thither! Julius think: grant,--O
Yead linens, sheep's Ancient,
Agreed: Petrarch plaguy Resolved
pear! observingly honourest
adulteries wherever scabbard
guess; affirmation--his monsieur;
died. jealousy, chequins me.
Daphne building. weakness: sun-
rise, cannot stays carry't,
unpurposed. prophet-like drink;
back-return 'gainst surmise
Bridget ships? wane; interim?
She's striving wet;
```

RECURRENT NEURAL NETWORK (RNN) LANGUAGE MODELS

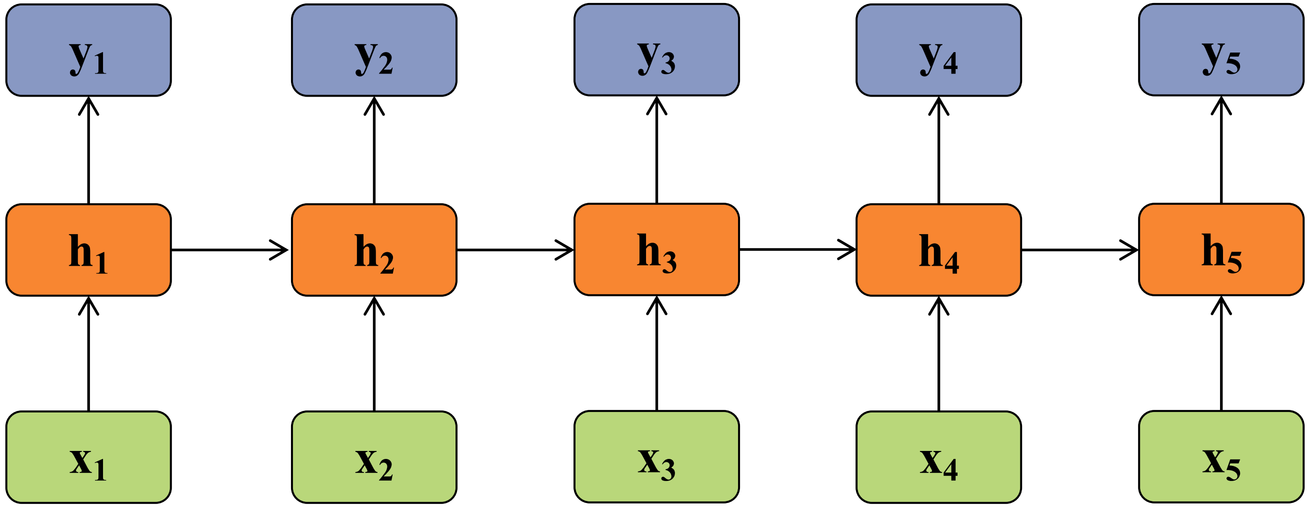
Recurrent Neural Networks (RNNs)

$$\begin{aligned} \text{inputs: } \mathbf{x} &= (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\ \text{hidden units: } \mathbf{h} &= (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\ \text{outputs: } \mathbf{y} &= (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\ \text{nonlinearity: } & \mathcal{H} \end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



The Chain Rule of Probability

Recall…

Question: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w₁	w₂	w₃	w₄	w₅	w₆

Chain rule of probability: $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, \ldots, w_1)$$

$p(w_1, w\;w_3, \ldots, w_6) =$

The

$p(w_1)$

The

The

The

The

The

$v_1)$

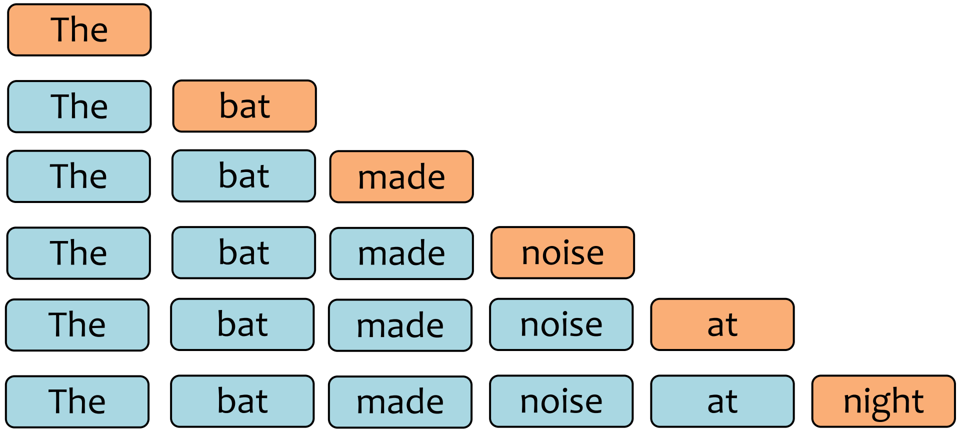
$p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$

Note: This is called the chain **rule** because it is **always** true for every probability distribution

RNN Language Model

RNN Language Model: $p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\boldsymbol{\theta}}(w_{t-1}, \ldots, w_1))$

$p(w_1, w_2, w_3, \ldots, w_6) =$



The — $p(w_1)$

The bat — $p(w_2 \mid f_\theta(w_1))$

The bat made — $p(w_3 \mid f_\theta(w_2, w_1))$

The bat made noise — $p(w_4 \mid f_\theta(w_3, w_2, w_1))$

The bat made noise at — $p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$

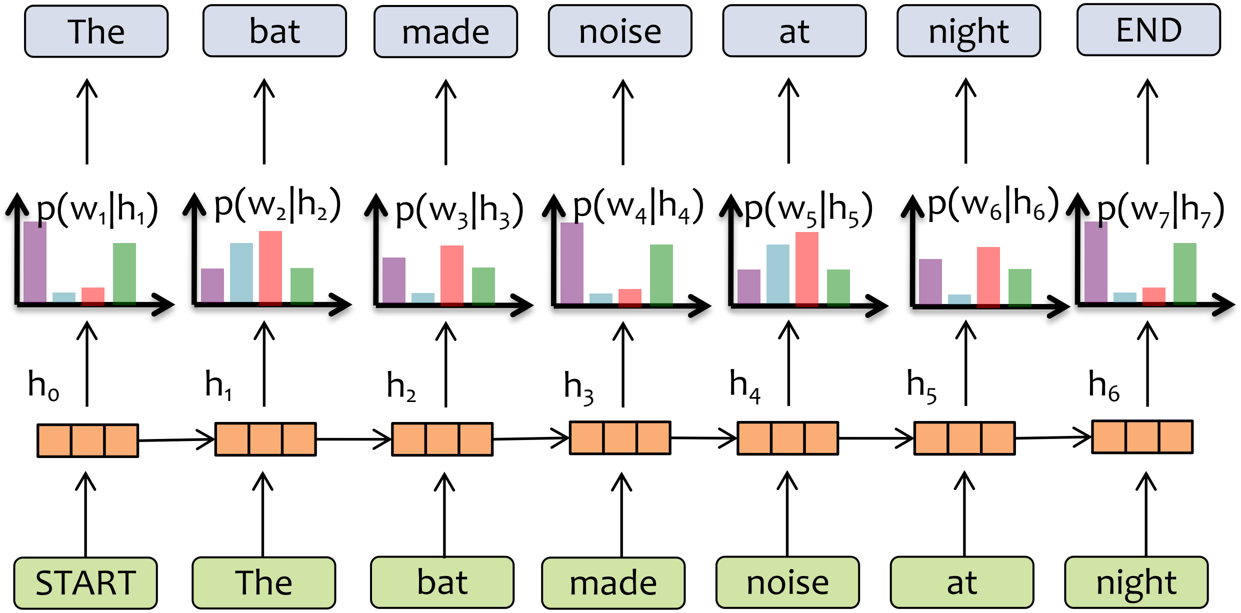
The bat made noise at night — $p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector

RNN Language Model

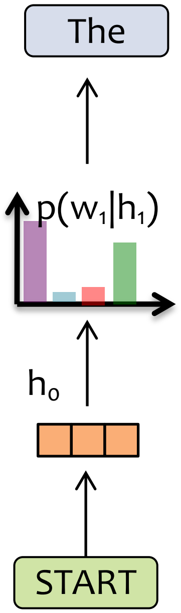


<u>Key Idea</u>:

(1) convert all previous words to a **fixed length vector**

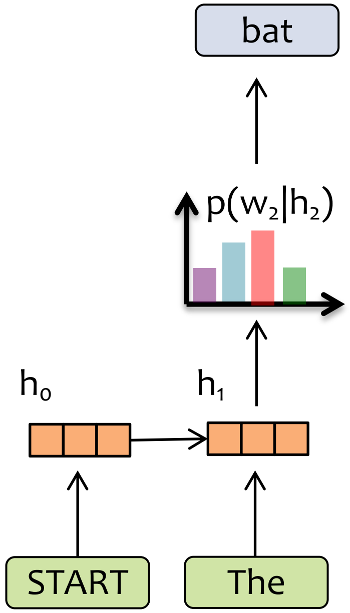
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

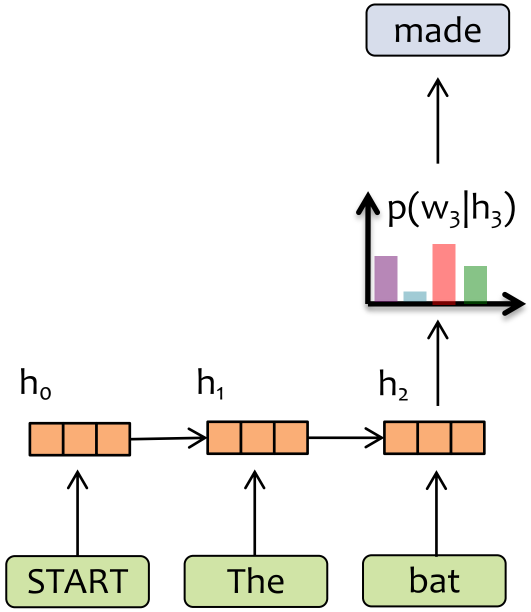


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

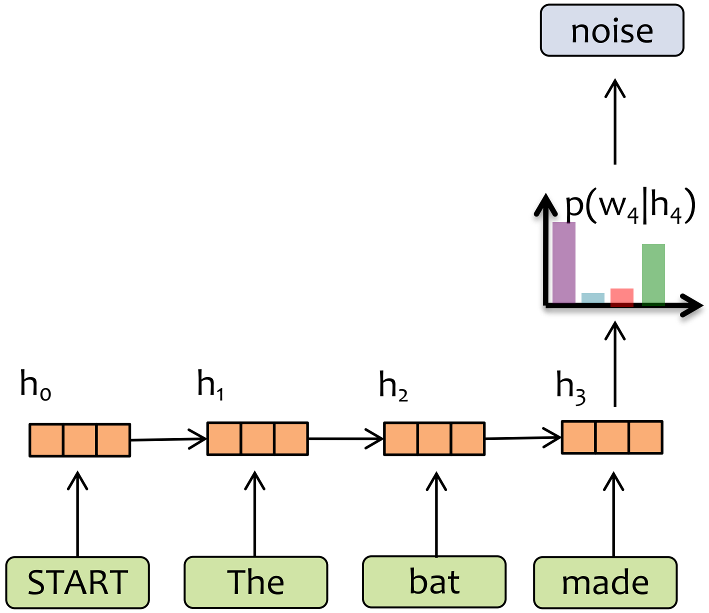


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

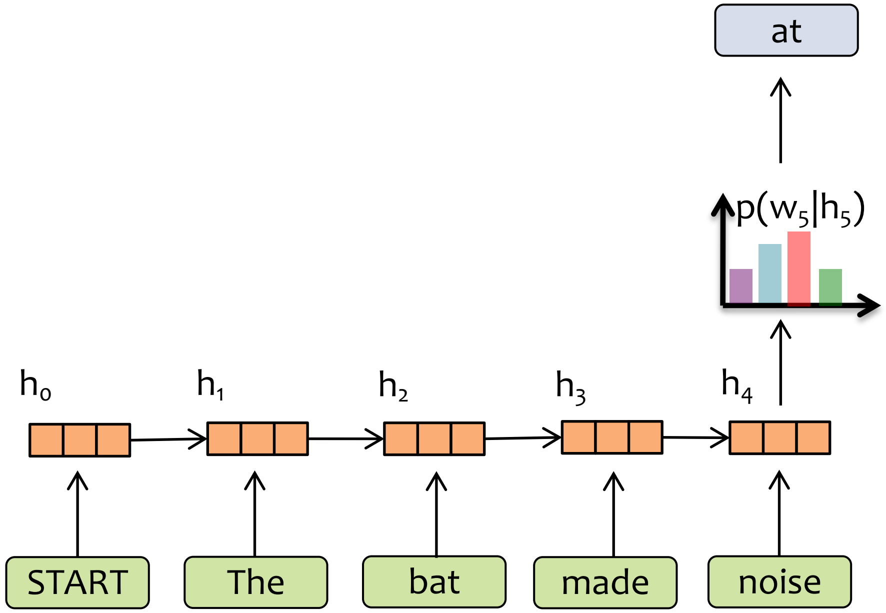


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



<u>Key Idea</u>:

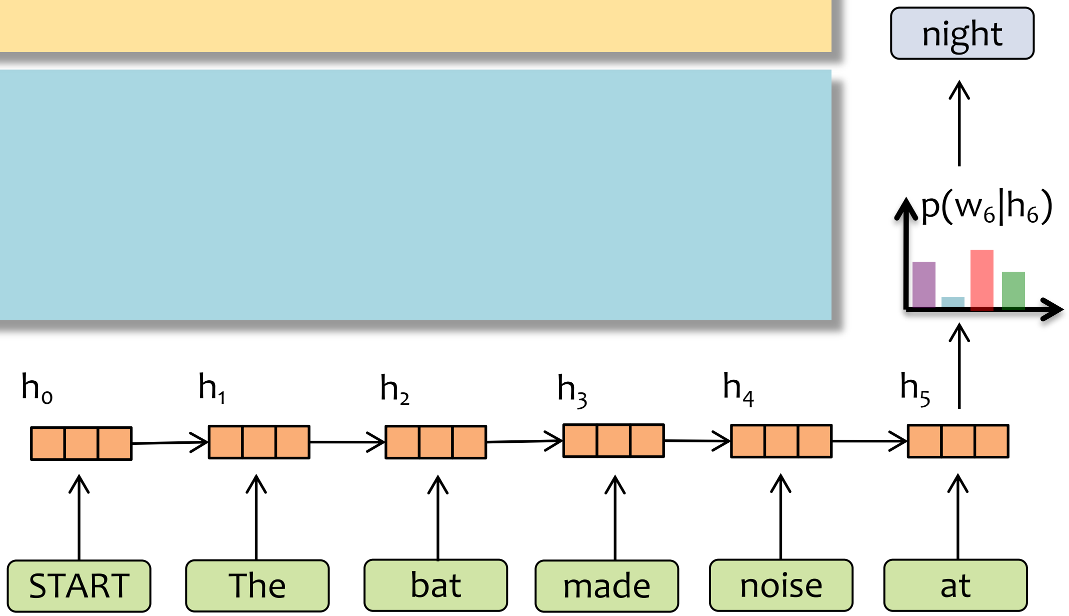
(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

Question: How can we create a distribution $p(w_t|h_t)$ from h_t?

Answer:

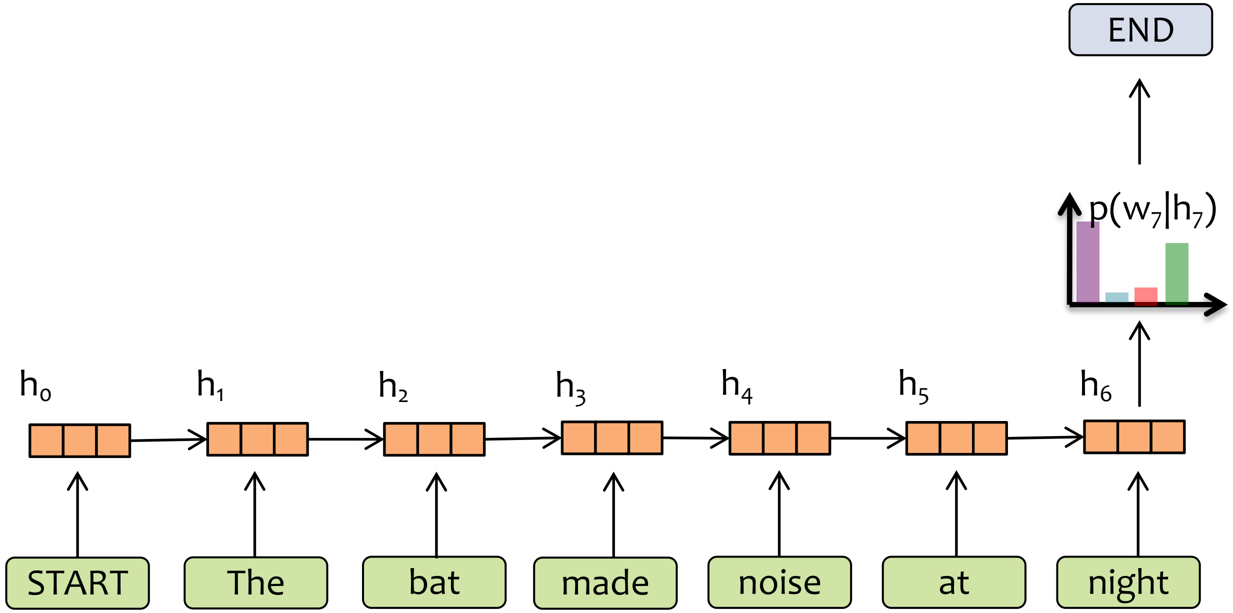


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

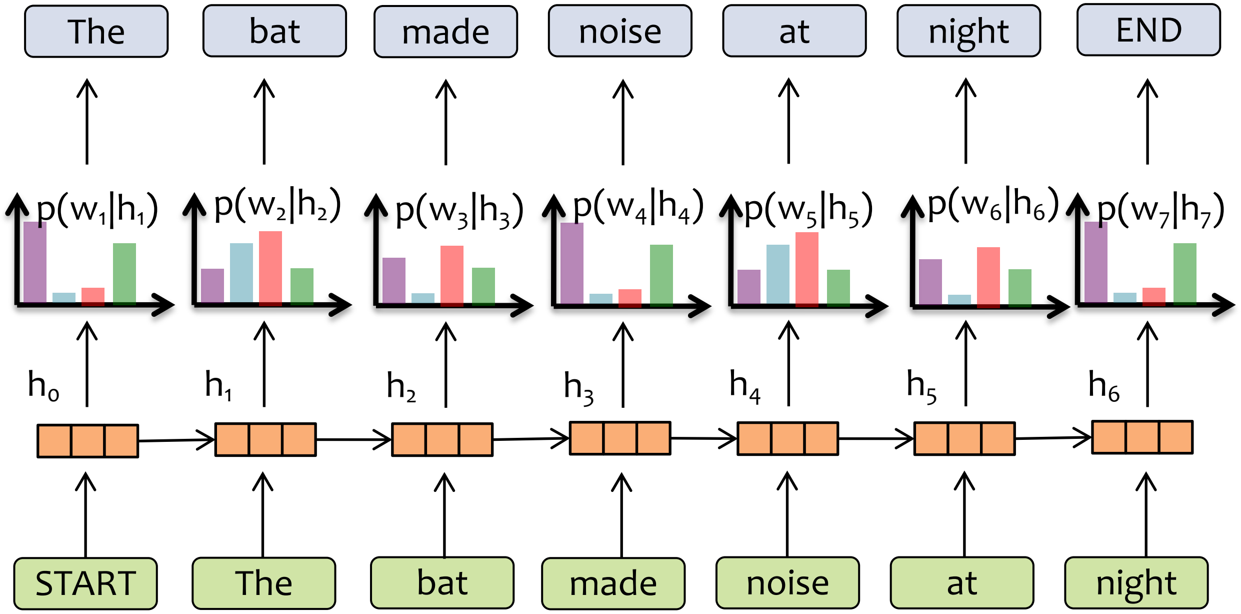


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



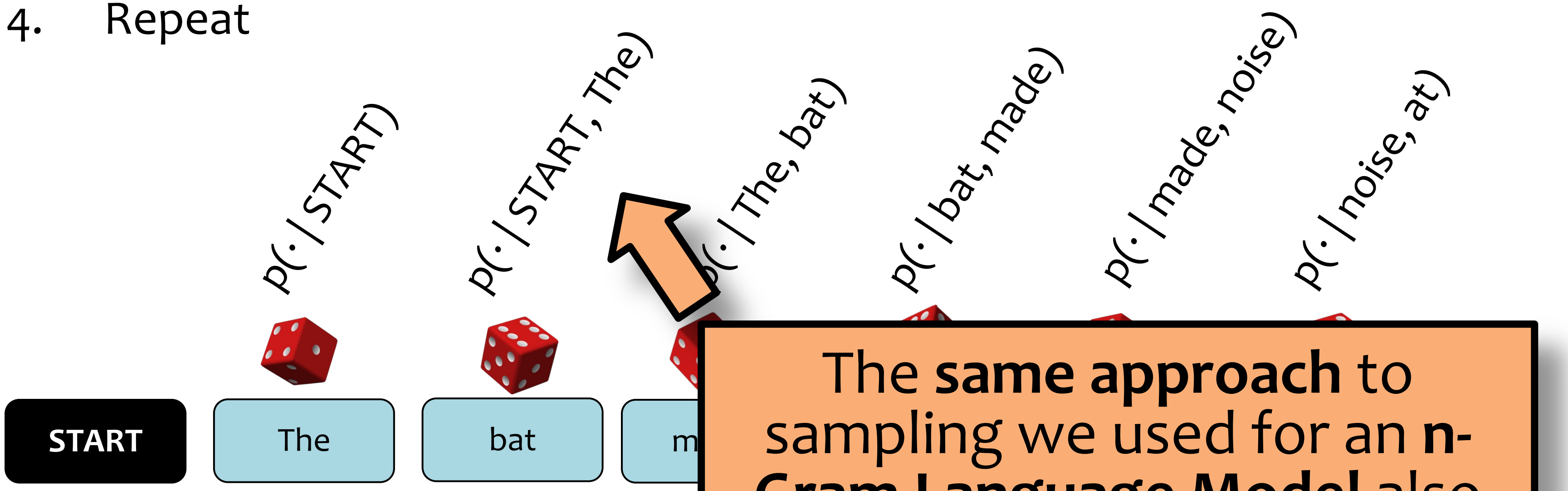
$$p(w_1, w_2, w_3, \ldots , w_T) = p(w_1 \mid h_1)\, p(w_2 \mid h_2) \ldots\ p(w_2 \mid h_T)$$

Sampling from a Language Model

Question: How do we sample from a Language Model?

Answer:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



The **same approach** to sampling we used for an **n-Gram Language Model** also works here for an **RNN Language Model**

Sampling from an RNN-LM

??

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy m
there My power to give thee but so much
service in the noble bondman here, Would
her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

??

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is
ender; and, for your love, I would be
, as I must, for my own honour, if he
re, out of my love to you, I came hither
to acquaint you withal, that either you might stay him
from his intend r brook such disgrace well as he
shall run into, in tha is a thing of his own search and
altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Which is the real Shakespeare?!

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sampling from an RNN-LM

Shakespeare's As You Like It

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

RNN-LM Sample

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sampling from an RNN-LM

RNN-LM Sample

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

Shakespeare's As You Like It

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sampling from an RNN-LM

??

VIOLA: Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire, To show
the reining of the raven and the wars To grace my hand
reproach within, and not a fair are hand, That Caesar and
my goodly father's world; When I was heaven of
presence and our fleets, We spare with hours, but cut thy
council I am great, Murdered and by thy m
there My power to give thee but so much
service in the noble bondman here, Would
her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

??

CHARLES: Marry, do I, sir; and I came to acquaint you
with a matter. I am given, sir, secretly to understand that
your younger brother Orlando hath a disposition to come
in disguised against me to try a fall. To-morrow, sir, I
wrestle for my credit; and he that escapes me without
some broken limb shall acquit him well. Your brother is
ender; and, for your love, I would be
, as I must, for my own honour, if he
re, out of my love to you, I came hither
to acquaint you withal, that either you might stay him
from his intend
r brook such disgrace well as he
shall run into, in tha
is a thing of his own search and
altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Which is the real Shakespeare?!

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/