CS 760: Machine Learning
Transfer learning

Misha Khodak

University of Wisconsin-Madison

8 December 2025

Announcements

- **Exam:** 5:05 – 7:05 PM on Dec 16th in Sewell 5208
  - covers all course topics, with emphasis on second half
  - otherwise same policies as midterm

- What should we review on Wednesday?
  1. neural network basics
  2. CNNs
  3. RNNs, Transformers, LMs
  4. generative models
  5. learning theory
  6. kernel methods
  7. reinforcement learning
  8. data-efficient learning

Outline

- **Transfer learning**
  - motivations, basic approaches, self-supervised learning
- **Learning across multiple tasks**
  - setups, multi-task methods, meta-learning methods
- **Foundation models**
  - overview, fine-tuning, in-context learning

Outline

- **Transfer learning**
  - motivations, basic approaches, self-supervised learning
- **Learning across multiple tasks**
  - setups, multi-task methods, meta-learning methods
- **Foundation models**
  - overview, fine-tuning, in-context learning

Dealing with low-data scenarios

Numerous approaches (too many to cover in detail)

- which one to take is highly application-dependent
- can construct a basic taxonomy:

less-than-full supervision

- do more with less (labeled data)
- last week's lecture

transfer learning

- do more with more (o.o.d. data)
- today's lecture

semi-supervised learning

active learning

weakly-supervised learning

multi-task learning

meta learning

foundation models

Transfer learning

We typically assume labeled points $(x_1, y_1), \ldots, (x_n, y_n) \sim D$ drawn i.i.d. from the **target distribution** D

What if:

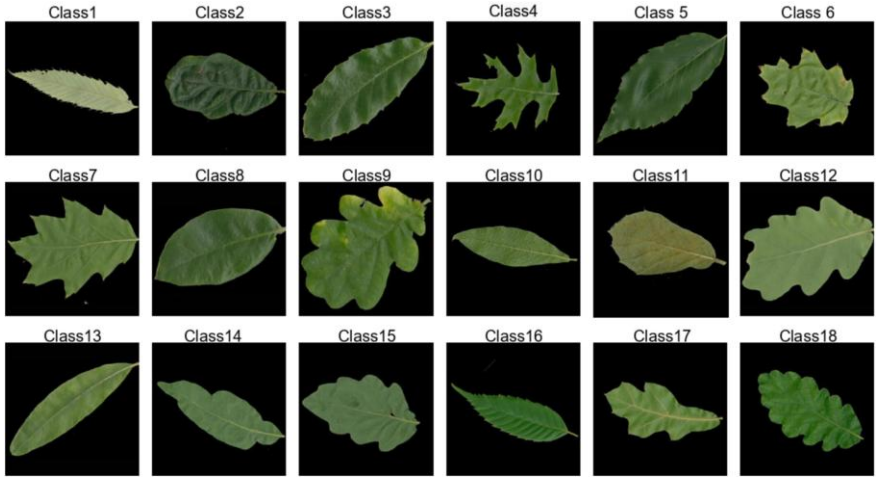
- n is too small to learn a sufficiently expressive model
- but we have access to more data $(x'_1, y'_1), \ldots, (x'_N, y'_N) \sim D'$ from a **related distribution** D'?

Using data from a related distribution to improve performance on the target distribution is **transfer learning**

Canonical example: ImageNet

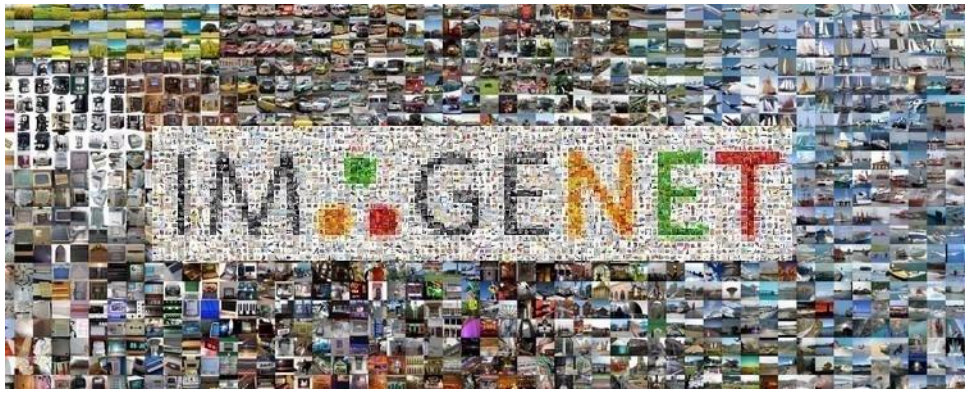
standard vision pipeline:

1. collect a bunch of data for your target task
2. download a large CNN (e.g. a big ResNet) trained on ImageNet and **replace its classification layer**
3. then
   I. either pass its **features** to a simpler model
   II. or **fine-tune** it directly on the task



Arun et al. *J. Phytopathology*.

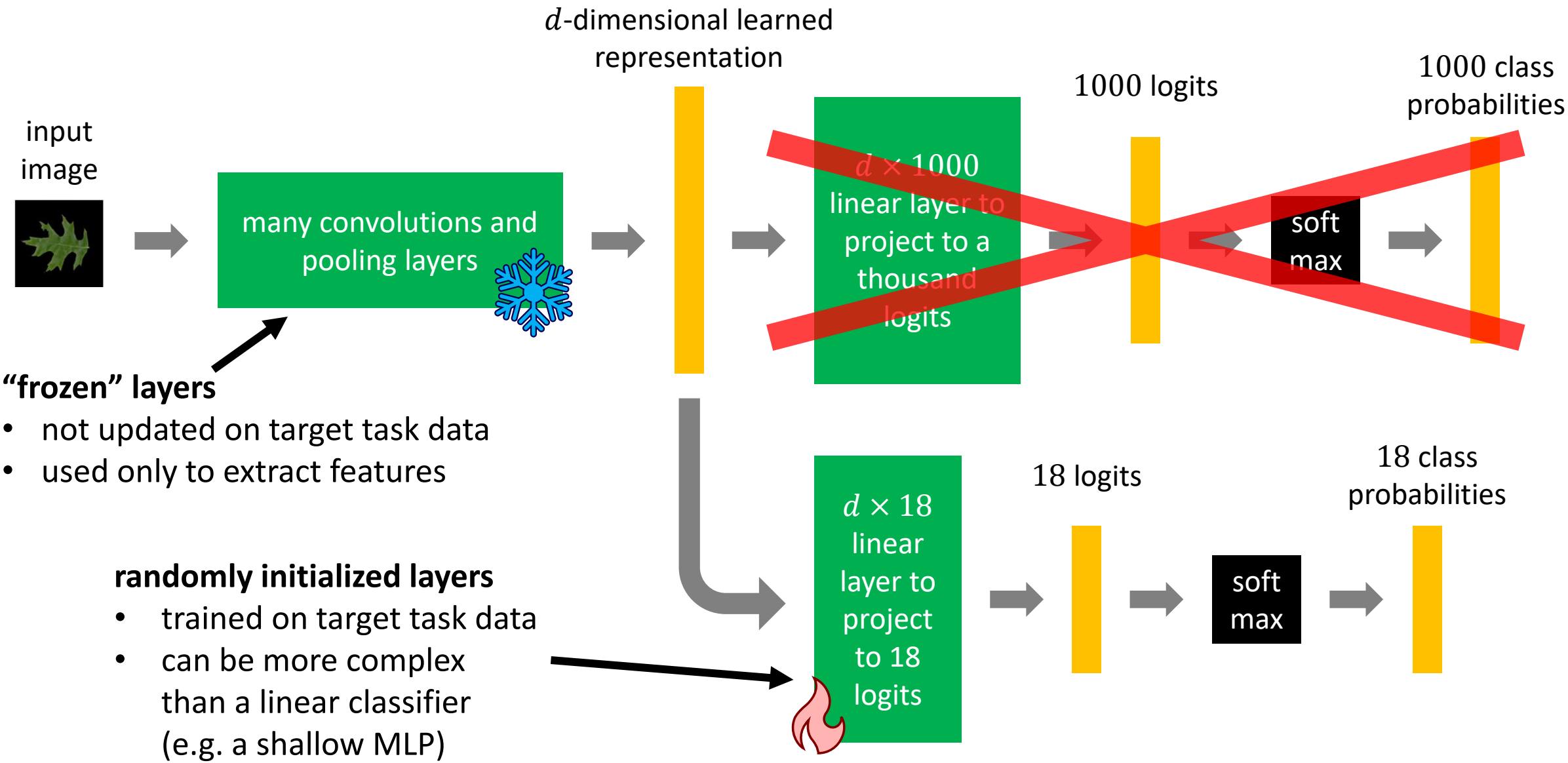
a few datapoints for a few classes



Kaggle

thousands of datapoints for each of a thousand classes

Approach I: feature extraction



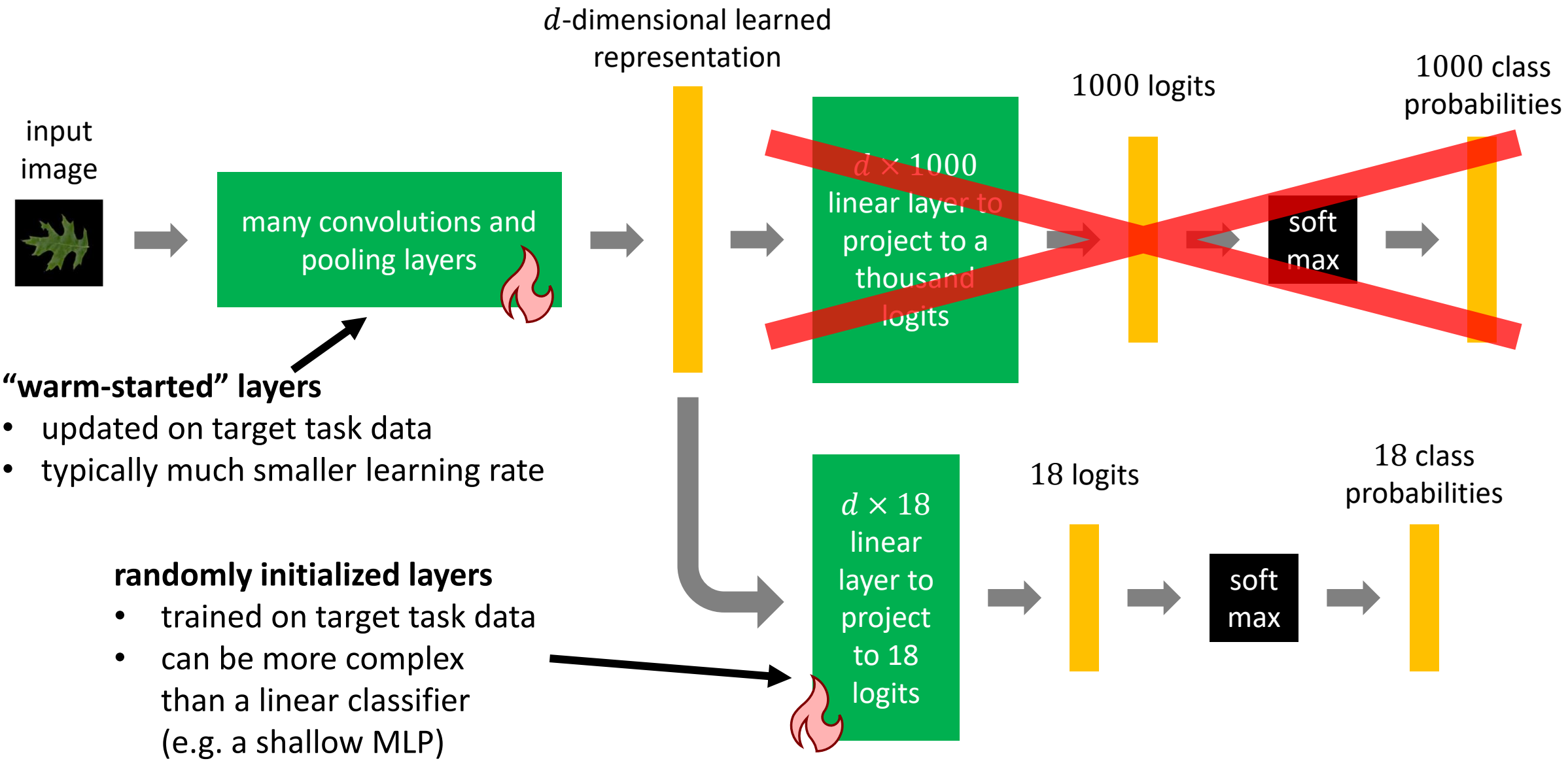
"frozen" layers

- not updated on target task data
- used only to extract features

randomly initialized layers

- trained on target task data
- can be more complex than a linear classifier (e.g. a shallow MLP)

Approach II: fine-tuning



"warm-started" layers

- updated on target task data
- typically much smaller learning rate

randomly initialized layers

- trained on target task data
- can be more complex than a linear classifier (e.g. a shallow MLP)

Transfer learning

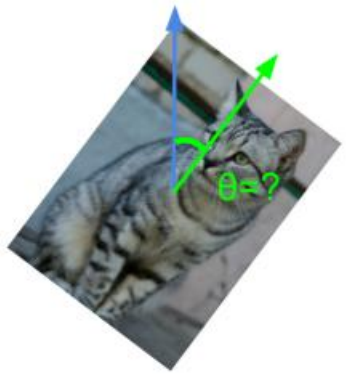
- Transfer learning has been hugely successful
- Numerous other potential approaches
- Big remaining question: **what if the related data lacks labels?**
  - we chop off the classification layers anyway, so we just need to extract some **representation** of the data
  - can do so using classical unsupervised learning (PCA, etc.)
  - or we can do it with **self-supervised learning (SSL)**

Self Supervision: Basic Idea

- Use domain-specific properties of the inputs (x) to create pseudo-labels (y) corresponding to **"pretext tasks"**

- Ex: predict stuff you already know

image completion



rotation prediction

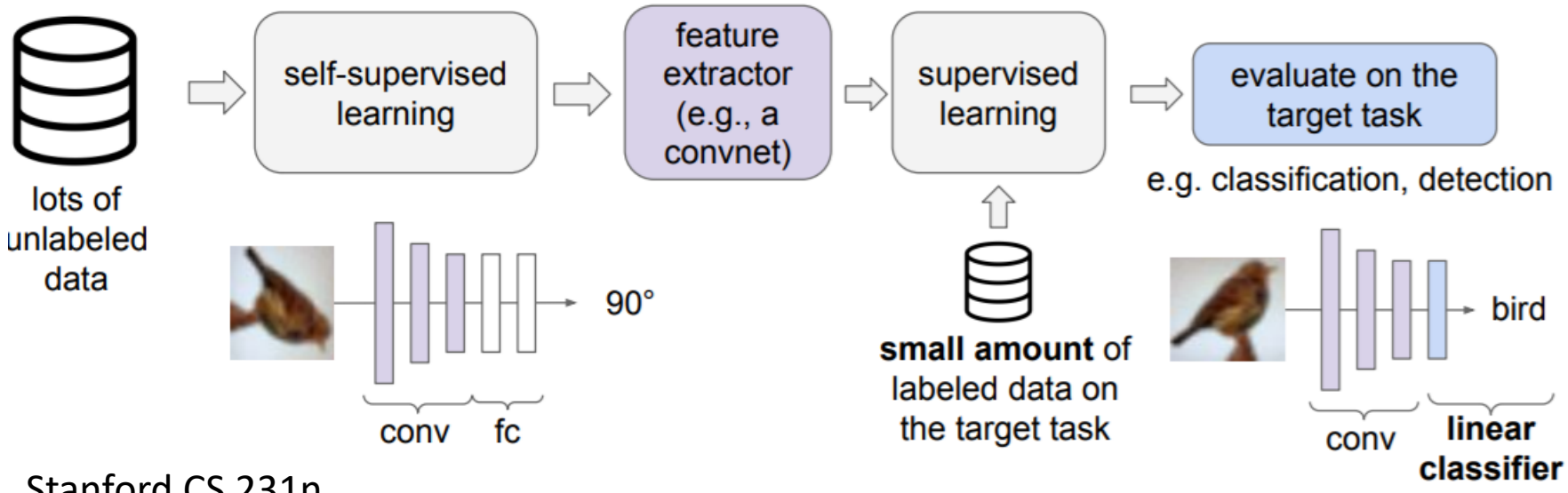
"jigsaw puzzle"

colorization

Stanford CS 231n

Self Supervision: Using the Representations

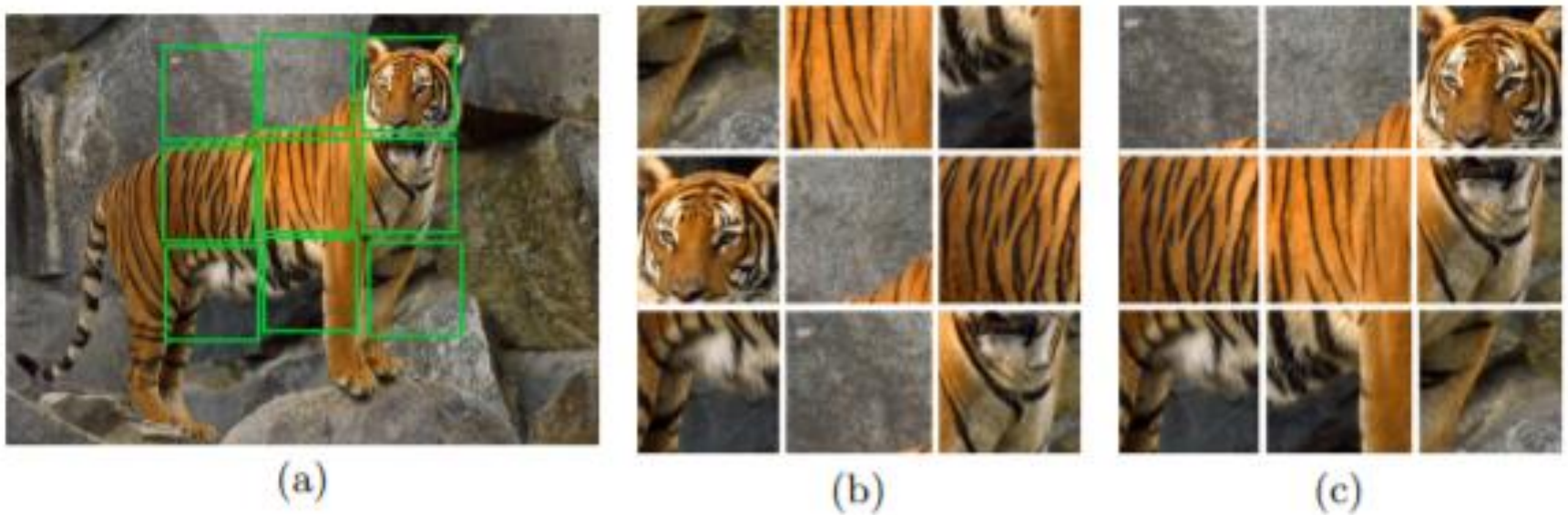
- Don't care specifically about our performance on pretext task
- Use the learned network as a feature extractor
- Once we have labels for a particular task, train on a small amount of data



Stanford CS 231n

Self Supervision: Pretext Tasks

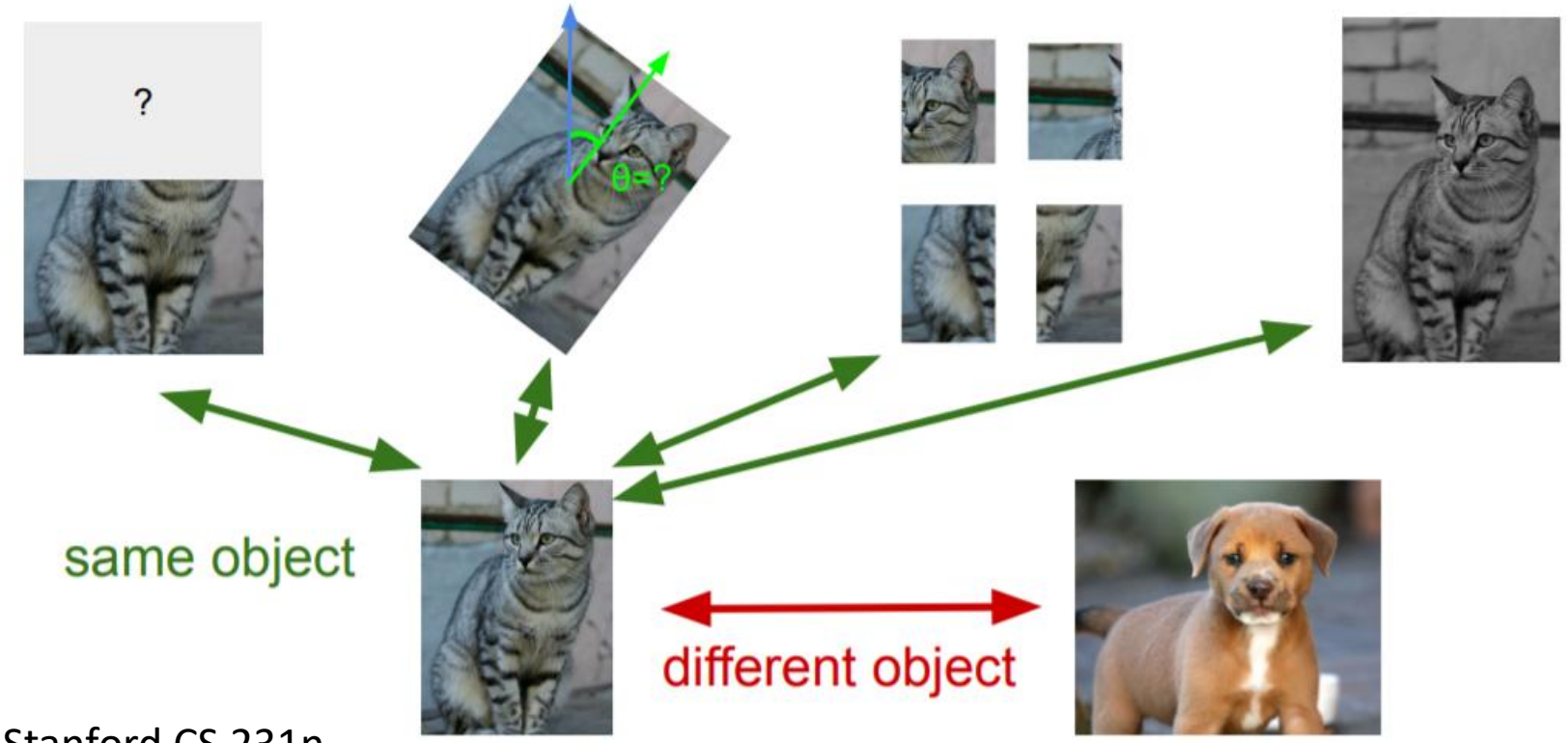
- Lots of options for pretext tasks
  - Predict rotations
  - Coloring
  - Fill in missing portions of the image
  - Solve puzzles



Noroozi and Favaro

Contrastive Learning: Basics

- Type of SSL where we learn representations such that:
  - transformed versions of single sample are similar
  - different samples are different



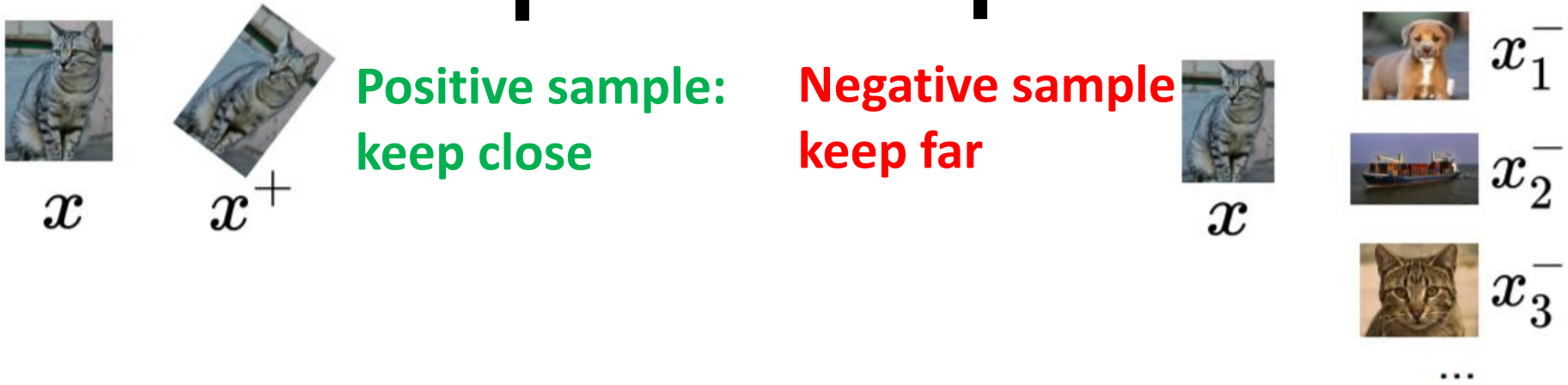
Stanford CS 231n

Contrastive Learning: Motivation

- Goal:
  - Keep together related representations, push unrelated apart.
  - The InfoNCE loss function:

Van den Oord et al., 2018

$$L = -E_X \left[\log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{k-1} \exp(s(f(x), f(x_j^-))} \right]$$



Self-supervised learning: Summary

Procedure:

- **pretrain** a network to do well on a pretext task
- **transfer** the network to your target task

Most well-known example: predict-the-next-word

Difference with regular (un)supervised training not obvious:

- sometimes pretext tasks are useful (e.g. autocomplete)
- sometimes unsupervised methods are implicitly SSL (e.g. GloVe)

Outline

- **Transfer learning**
  - motivations, basic approaches, self-supervised learning
- **Learning across multiple tasks**
  - setups, multi-task methods, meta-learning methods
- **Foundation models**
  - overview, fine-tuning, in-context learning

Transfer learning from **multiple tasks**

What if instead of one related task with lots of data we have **many related tasks with similar amounts of data?**

Many setups:

- multi-task learning
- meta-learning
- continual learning
- lifelong learning
- …

$$(x_{1,1}, y_{1,1}), \dots, (x_{1,n_1}, y_{1,n_1}) \sim D_1$$

$$\vdots$$

$$(x_{t,1}, y_{t,1}), \dots, (x_{t,n_t}, y_{t,n_t}) \sim D_t$$

$$\vdots$$

We'll cover two of them: **multi-task** and **meta**-learning

Multi-task learning

Setup: **fixed number of related tasks**

Examples:

- predict the weather in nearby cities
- diagnose patients in different hospitals

Key challenges:

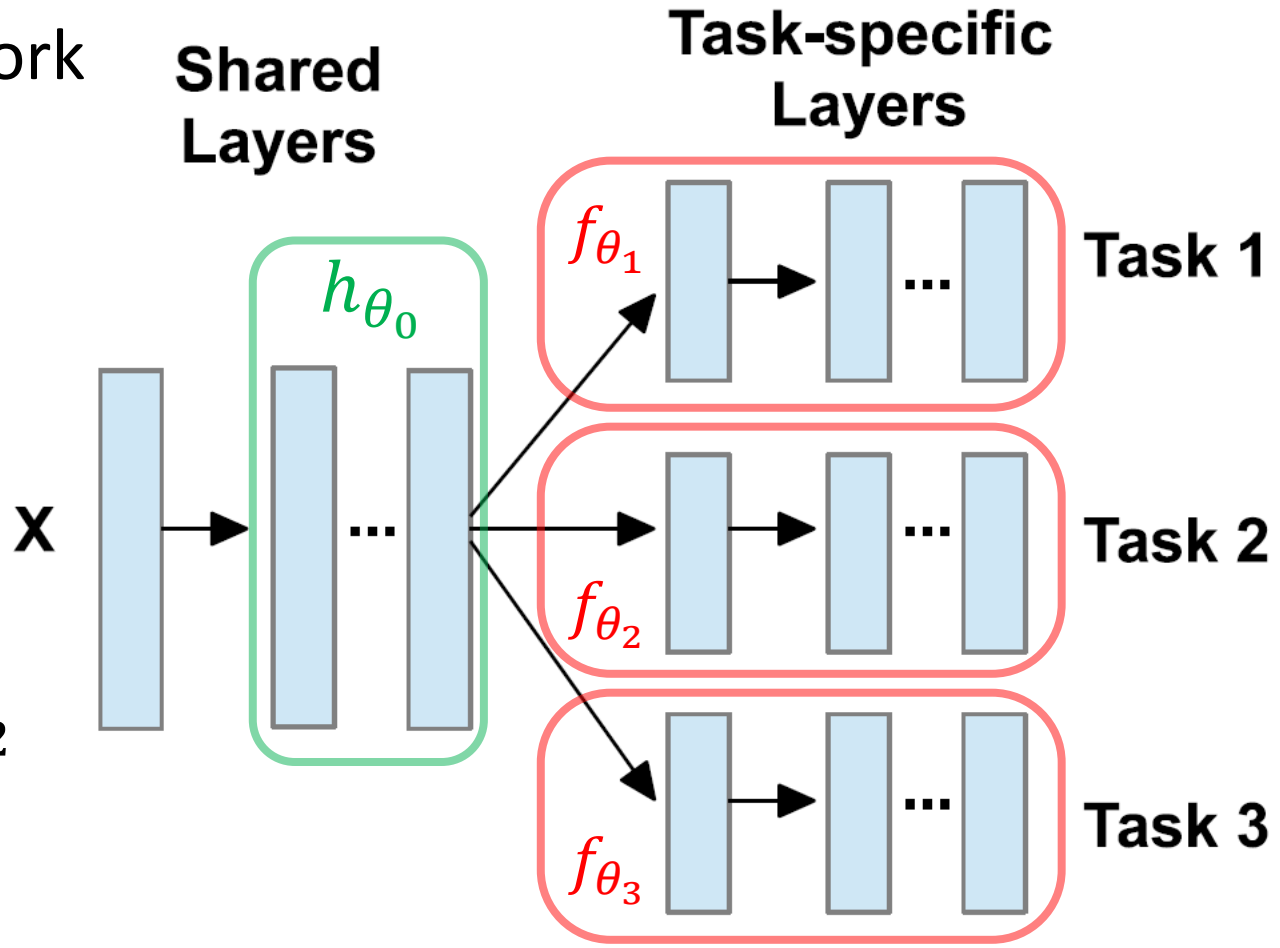
- how to encode task-relationships?
- how to avoid conflicting tasks?

Fox6 (don't worry, this isn't the actual forecast…yet)

One common approach: **Layer-sharing**

- jointly train a multi-output network
- assumes existence of a good **shared representation** h_{θ_0}
- example objective:

$$\sum_{t=1}^{T}\sum_{i=1}^{n_t}\left(y_{t,i} - f_{\theta_t}\left(h_{\theta_0}(x_{t,i})\right)\right)^2$$



Thung & Wee. *Multimedia Tools & Applications*

Another common approach: **Regularization**

- jointly train separate networks
- regularize parameters to be closer together
- example objective:

$$\sum_{t=1}^{T}\sum_{i=1}^{n_t}\left(y_{t,i} - f_{\theta_t}(x_{t,i})\right)^2 + \sum_{t=1}^{T}\sum_{u=t+1}^{T}\lambda_{t,u}\|\theta_t - \theta_u\|^2$$

- allows hand-encoding of task-relationships via the regularization strengths $\lambda_{t,u}$

Meta-learning

Setup:

- **meta-training** dataset of related tasks
- **at meta-test time** we get a new dataset $(x_1, y_1), \ldots, (x_n, y_n) \sim D$
- **our goal:** low expected error on unseen examples $(x, y) \sim D$

$$(x_{1,1}, y_{1,1}), \ldots, (x_{1,n_1}, y_{1,n_1}) \sim D_1$$

$$\vdots$$

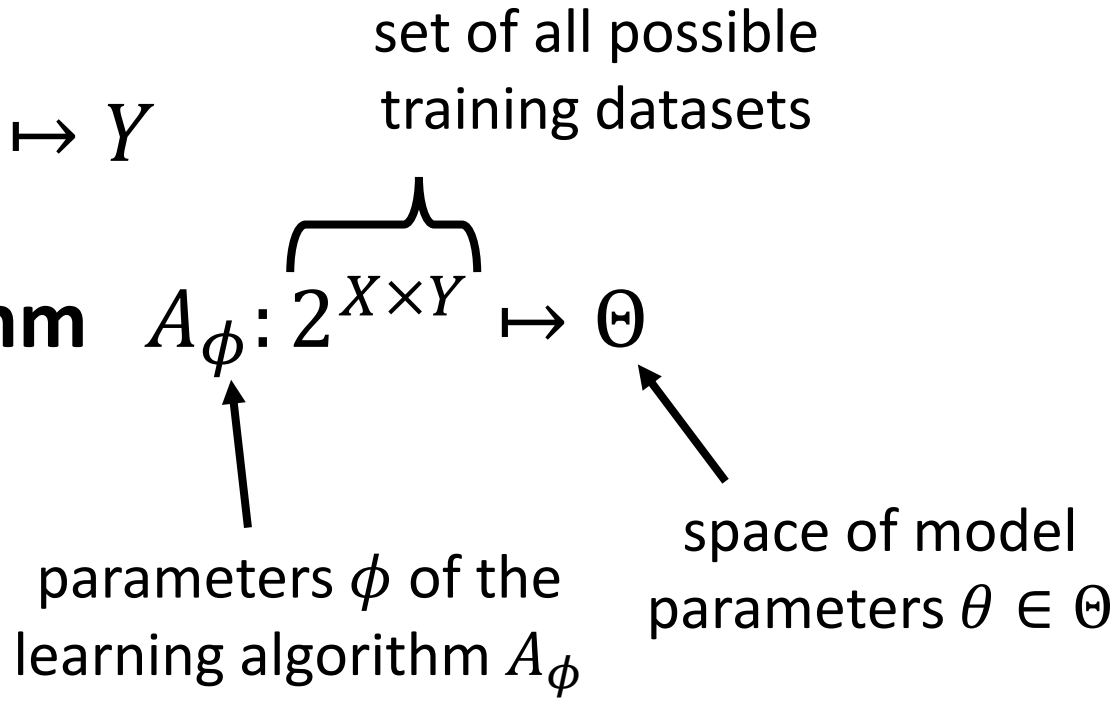
$$(x_{T,1}, y_{T,1}), \ldots, (x_{T,n_T}, y_{T,n_T}) \sim D_T$$

Applications:

- auto-complete for new cellphone users (federated learning)
- image classification with limited labels (few-shot learning)
- robots in related environments (meta-RL)

Why is it called meta-learning?

- no longer learning a model $f_\theta : X \mapsto Y$
- we are learning a **learning algorithm** $A_\phi : 2^{X \times Y} \mapsto \Theta$
- thus also called **learning-to-learn**



Example: meta-learn an initialization ϕ for gradient descent

Meta-learning (one step) gradient descent

MAML approach: $A_\phi(\{(x_i, y_i)\}_{i=1}^n) = \phi - \alpha \nabla_\phi \sum_{i=1}^n \ell(y_i, f_\phi(x_i))$

- meta-training objective:

$$\underset{\phi}{\operatorname{argmin}} \sum_{t=1}^{T} \sum_{i=n_t/2+1}^{n_t} \ell\left(y_{t,i}, f_{A_\phi\left(\{(x_{t,i}, y_{t,i}\}_{i=1}^{n_t/2}\right)}(x_{t,i})\right)$$

- at meta-test time:
  - $\theta \leftarrow \phi - \alpha \nabla_\phi \sum_{i=1}^n \ell_i(y_i, f_\phi(x_i))$
  - make predictions using $f_\theta(x)$

Finn et al.
ICML 2018

Outline

- **Transfer learning**
  - motivations, basic approaches, self-supervised learning
- **Learning across multiple tasks**
  - setups, multi-task methods, meta-learning methods
- **Foundation models**
  - overview, fine-tuning, in-context learning

What is a foundation model?

1. take a **massive neural network**
   - older / specialized models had 100M+ params
   - latest models have 1-100 billion or more

2. **pretrain** it on Internet-scale data

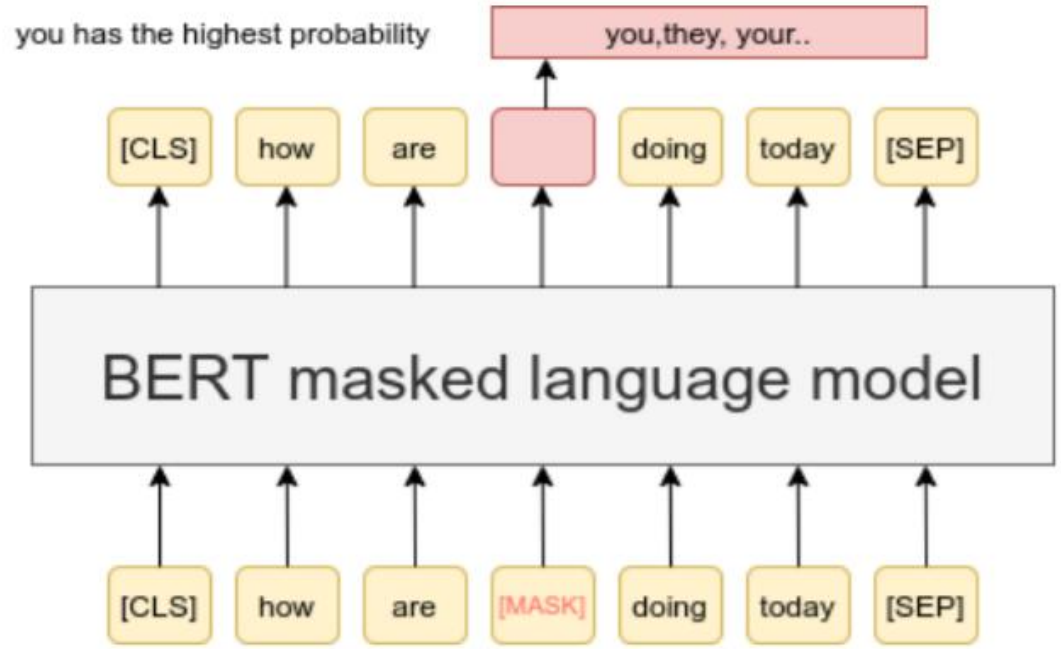
3. (optionally) **post-train** on large-scaled supervised data

4. use it for transfer learning for many different tasks

Early history

2017: BERT model (340M)

- Transformer trained on masked language modeling (pretext task)
- “solved” transfer learning for language



2017-present: GPT series

- Transformer trained on next-word prediction
- first observation of **in-context** learning capabilities in GPT-3
- ChatGPT post-trained on GPT-3.5

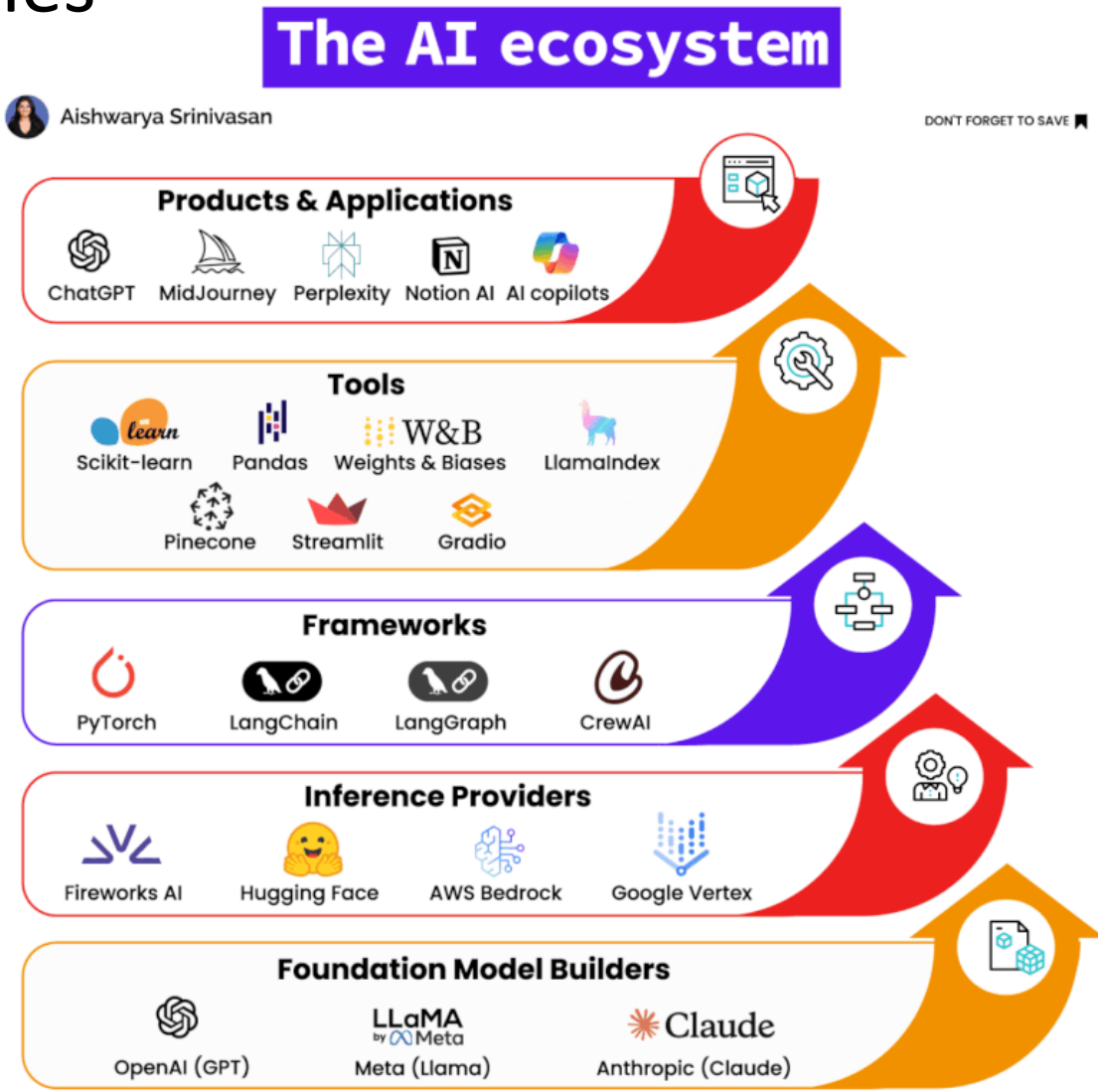
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:          ← task description
2  sea otter => loutre de mer            ← examples
3  peppermint => menthe poivrée          ←
4  plush girafe => girafe peluche        ←
5  cheese =>                             ← prompt
```

Post-ChatGPT

- many models with varying capabilities
- closed-source models typically outperform open-source models
- new challenges:
  - **massive compute costs**
  - privacy, security, safety
- new opportunities:
  - **in-context learning**
  - reasoning



Challenge: **Compute costs**

pretraining FMs limited to large orgs

- one training run requires 100s of GPUs
- need many training runs (to tune) and engineers (to manage training)

even fine-tuning is hard:

- SGD on GPT-3 (175B) uses 1.2TB VRAM
- NVIDIA GPUs max out below 200GB
- what can we do?

Model	Microarchitecture	Launch	Core	Core clock (MHz)	Shaders			Memory				
					Core config [c]	Base clock (MHz)	Max boost clock (MHz)[d]	Bus type	Bus width (bit)	Size (GB)	Clock (MT/s)	Bandwidth (GB/s)
A100 GPU accelerator (PCIe card)[442][443]	Hopper	May 14, 2020[444]	1× GA100-883AA-A1	—	6912:432 :160:432:0 (108)	765	1410	HBM2	5120	40 or 80	1215	1555
H100 GPU accelerator (PCIe card)[445]		March 22, 2022[446]	1× GH100[447]	—	14592:456 :24:456:0 (114)	1065	1755 CUDA 1620 TC	HBM2E	5120	80	1000	2039
H100 GPU accelerator (SXM card)				—	16896:528 :24:528:0 (132)	1065	1980 CUDA 1830 TC	HBM3	5120	64 or 80 or 96	1500	3352
H200 GPU accelerator (PCIe card)[448]		November 18, 2024[449]	1× GH100	—		1365	1785	HBM3E	5120	141	1313	3360
H200 GPU accelerator (SXM card)				—		1590	1980	HBM3E	5120	141	1313	3360
H800 GPU accelerator (SXM card)		March 21, 2023[450]	1× GH100	—		1095	1755	HBM3	5120	80	1313	3360
L40 GPU accelerator[451]	Ada Lovelace	October 13, 2022	1× AD102[452]	—	18176:568 :192:568:142 (142)	735	2490	GDDR6	384	48	2250	864
L4 GPU accelerator[453][454]		March 21, 2023[455]	1x AD104[456]	—	7424:240 :80:240:0 (60)	795	2040	GDDR6	192	24	1563	300
B100 GPU accelerator[457]	Blackwell	November 2024	2× GB102	—	2× 16896:528 :24:528:0 (132)	1665	1837	HBM3E	2× 4096	2× 96	2000	2× 4100
B200 GPU accelerator[459]		2024	2× GB100	—		1665	1837	HBM3E	2× 4096	2× 96	2000	2× 4100

Parameter-efficient fine-tuning (PEFT)

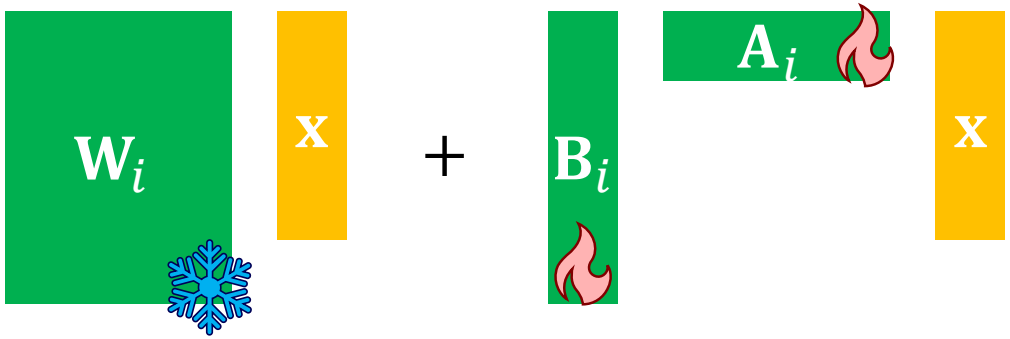
Most popular approach: **LoRA**

1. take an FM with **pretrained weight** matrices $\mathbf{W}_1, \ldots, \mathbf{W}_N$
2. for each matrix $\mathbf{W}_i \in \mathbb{R}^{d \times k}$:
   - set $r \ll \min\{d, k\}$ and initialize **fine-tuning weights**:
     - $\mathbf{B}_i \in \mathbb{R}^{d \times r}$ to $\mathbf{B}_i = 0$
     - $\mathbf{A}_i \in \mathbb{R}^{r \times k}$ to $\mathbf{A}_i \sim$ Gaussian
   - replace $\mathbf{W}_i$ by $\mathbf{W}_i + \mathbf{B}_i\mathbf{A}_i$
3. fine-tune on target task but
   - freeze $\mathbf{W}_i$
   - update $\mathbf{B}_i$ and $\mathbf{A}_i$

$$f_i(\mathbf{x}) = \mathbf{W}_i\mathbf{x} =$$

$$= \mathbf{W}_i\mathbf{x} + \mathbf{B}_i\mathbf{A}_i\mathbf{x}$$

$=$



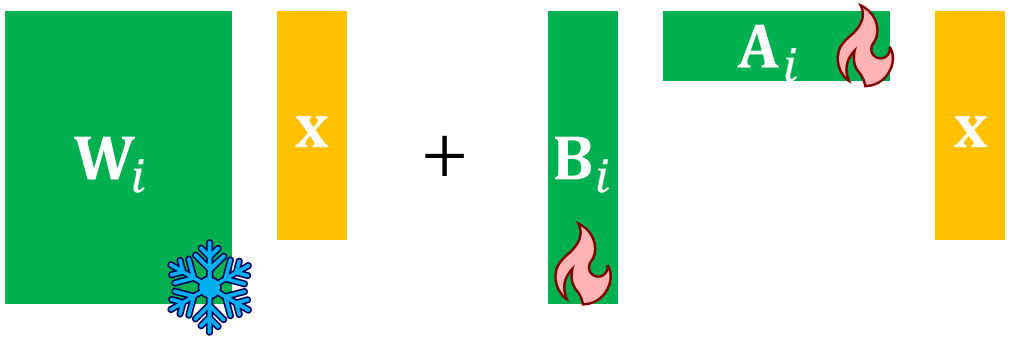
How does LoRA save memory?

- original weights $\mathbf{W}_i \in \mathbb{R}^{d \times k}$ have dk trainable params
- new weights $\mathbf{B}_i \in \mathbb{R}^{d \times r}$ and $\mathbf{A}_i \in \mathbb{R}^{r \times k}$ have $(\mathrm{d} + \mathrm{k})r$
- typical values in GPT-3 175B:
  - $d \approx k \approx 10^4$
  - $r \leq 10$
- $\geq 10^4$x fewer trainable params!
- 3x less fine-tuning VRAM

$$f_i(\mathbf{x}) = \mathbf{W}_i\mathbf{x} =$$

$$= \mathbf{W}_i\mathbf{x} + \mathbf{B}_i\mathbf{A}_i\mathbf{x}$$

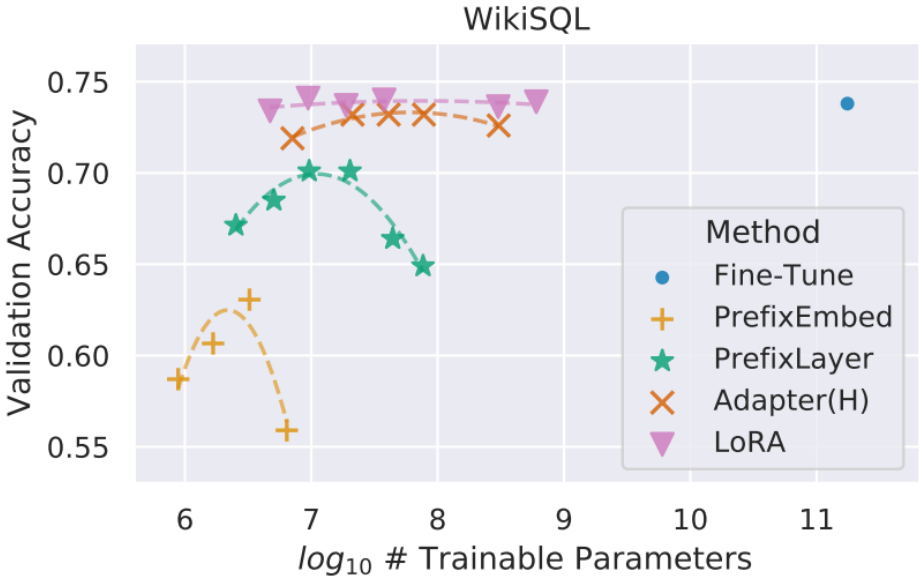
$=$

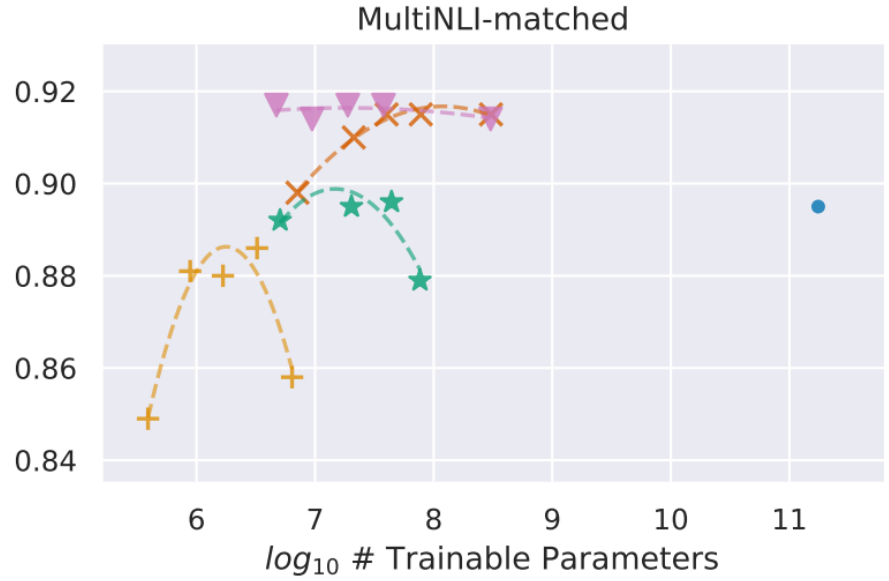


Does LoRA affect accuracy?

Yes, it constrains weights of the fine-tuned model:

- fine-tuned matrices $\mathbf{W}_i + \mathbf{B}_i\mathbf{A}_i$ at most a rank $r \ll \min\{d, k\}$ update away from pretrained matrices $\mathbf{W}_i$
- LoRA = **L**ow-**R**ank **A**daptation
- in practice do not need large r for good performance
- learning theory intuition?





Hu et al.

Opportunity: **In-context learning**

Observation: the perfect next-word predictor can be **prompted** to answer any question correctly

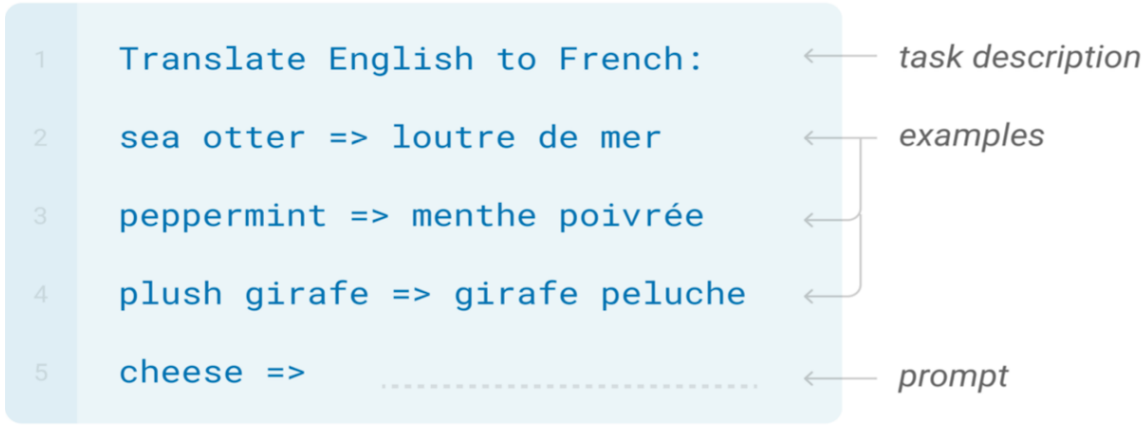
Idea: **in-context learning**

1. encode task instructions and data as a **context** sequence
2. make the FM generate the remainder of the sequence

Enables learning with target data **without updating the weights at all!**

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Brown et al.

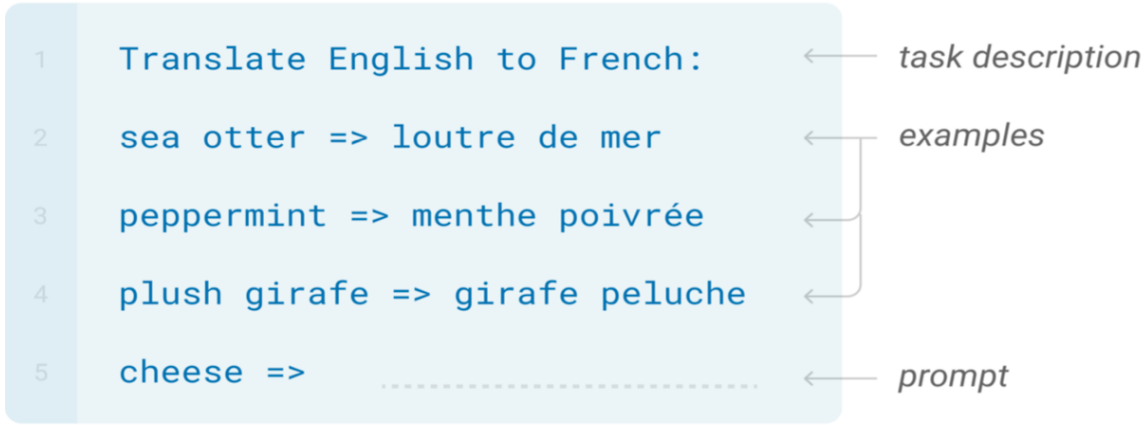
Opportunity: **In-context learning**

Usefulness:

- handles tasks with diverse input and output structures
- directly incorporates pretraining knowledge
- enables multi-step reasoning

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Brown et al.

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Fred Sala