Unsupervised pre-training for few-shot learning, vol. 2: reconstruction-based methods
Logistics

Project proposal due TODAY!

Homework 2 due Wednesday

Make sure you have set-up Azure!
(well before the HW deadline)
Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
- Autoencoders
- Masked autoencoders: BERT, MAE
- Autoregressive models: GPT, Flamingo

Goals for by the end of lecture:
- Familiarize you with widely-used methods for unsupervised pre-training
- Introduce methods for efficient fine-tuning of pre-trained models
- Prepare you for HW3
Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
- Autoencoders
- Masked autoencoders: BERT, MAE
- Autoregressive models: GPT, Flamingo
Unsupervised Pre-Training Set-Up

Goal: Get predictor for task $\mathcal{T}_j$

- Diverse unlabeled dataset $\{x_i\}$
- Unsupervised pre-training
- Pre-trained model
- Fine-tuning Labeled $\mathcal{D}_j^{tr}$
Plan for Today

Recap
- Problem formulation
- **Contrastive learning**

*Reconstruction-based unsupervised pre-training*
- Why reconstruction?
- Autoencoders
- *Masked* autoencoders: BERT, MAE
- Autoregressive models: GPT, Flamingo
Key Idea of Contrastive Learning

**Similar examples should have similar representations**

Examples with the same class label

Augmented versions of the example

Nearby image patches

Nearby video frames

(Requires labels, related to Siamese nets, ProtoNets)

(flip & crop)

Dog credit to Maggie & Luke

van den Oord, Li, Vinyals. CPC. 2018; Chen, Kornblith, Norouzi, Hinton. SimCLR. ICML 2020
Contrastive Learning Implementation

**Similar examples** should have **similar representations**

Need to both **compare** & **contrast**!

V1. Triplet loss:

$$\min_\theta \sum_{(x,x^+,x^-)} \max (0, \|f_\theta(x) - f_\theta(x^+)^2 - \|f_\theta(x) - f_\theta(x^-)\|^2 + \epsilon)$$

Embedding space $f_\theta(x)$
Contrastive Learning Implementation

**Similar examples** should have **similar representations**

Need to both **compare & contrast**!

V1. Triplet loss:

\[
\min_{\theta} \sum_{(x, x^+, x^-)} \max(0, \|f_\theta(x) - f_\theta(x^+)\|^2 - \|f_\theta(x) - f_\theta(x^-)\|^2 + \epsilon)
\]

V2. From binary to N-way classification (aka **SimCLR**):

\[
\mathcal{L}_{N\text{-way}}(\theta) = - \sum_z \log \frac{\exp(-d(z, z^+))}{\exp(-d(z, z^+)) + \sum_i \exp(-d(z, z^-))}
\]

Loss read as “**classification loss when discriminating positive pair from negatives**”

*also known as the **NT-Xent** loss, when \(d(\cdot, \cdot)\) is scaled cosine similarity*

---

Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
  - Autoencoders
  - Masked autoencoders: BERT, MAE
  - Autoregressive models: GPT, Flamingo
  - Emergent behaviors in large models
Why reconstruction?

**Simple intuition:** a good representation of an input should be sufficient to **reconstruct** it

\[ \hat{x} \]

**Bonus:** no need to worry about pesky things like **sampling negatives** or large batch sizes!

If the encoder is producing a "good" representation, a reasonably-sized decoder should be able to produce **reconstruction** \( \hat{x} \) very close to **input** \( x \) from **representation** \( r \).
Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
- **Autoencoders**
  - Masked autoencoders: BERT, MAE
  - Autoregressive models: GPT, Flamingo
  - Emergent behaviors in large models
**Autoencoders: a first attempt**

Simple intuition: a good representation lets us **reconstruct** the input

\[ x \]

Input image, sentence, audio signal, etc.

Encoder (CNN) → \( r \) → Decoder (CNN) → \( \hat{x} \)

Loss function is reconstruction error, e.g. L2 distance:

\[ \mathcal{L} = d(x, \hat{x}) \]

\[ d(x, \hat{x}) = \| x - \hat{x} \|^2 \]

**What can go wrong here?**

Is the **identity function** a good encoder/decoder?
Autoencoders: adding a bottleneck

Key idea: latent representation is **bottlenecked**, e.g., **lower-dimensional** than the input

Hope: latent dimensions are forced to represent **high-level** concepts that **generalize** to other tasks
Autoencoders: few-shot learning

Few-shot learning recipe: freeze encoder, fine-tune prediction head using our few-shot data (e.g., a linear layer)
Autoencoders

**Pros:**
- Simple, general
- Just need to pick $d(x, \hat{x})$
- No need to select positive/negative pairs

**Cons:**
- Need to design a bottlenecking mechanism
- Relatively poor few-shot performance

Why?

$r$ is just **memorizing** details of $x$ needed to minimize pixel-level reconstruction loss

$r$ is more like a **hash** of $x$ than a **conceptual summary**

How do we encourage the encoder to extract high-level features?

One strategy is **other types of bottlenecks**:
- **information** bottlenecks (adding noise)
- **sparsity** bottlenecks (zero most dimensions)
- **capacity** bottlenecks (weak decoder)

*In practice,* we’ll stop worrying about designing bottlenecks and just make the task a little **harder**
Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
- Autoencoders
  - **Masked autoencoders**: BERT, MAE
  - Autoregressive models: GPT, Flamingo
Ultimately, regular autoencoders are trying to predict $x$ from… $x$ (through $r$)

We bottleneck $z$ to avoid totally degenerate solutions, but what if the task is just “too easy”, admitting unhelpful solutions?

Masked autoencoders use a more difficult learning task to encourage the encoder to extract more meaningful features.

Beyond the bottleneck: masked autoencoders
Ultimately, regular autoencoders are trying to predict $x$ from $x$ (through $z$)

We bottleneck $z$ to avoid totally degenerate solutions, but what if the task is just “too easy”, admitting unhelpful solutions?

Masked autoencoders use a more difficult learning task to encourage the encoder to extract more meaningful features.
Beyond the bottleneck: **masked autoencoders**

**General recipe** for pre-training masked autoencoder $f_{\theta}$:

1. Choose **distance function** $d(\cdot,\cdot) \rightarrow \mathbb{R}$
2. For **train batch** examples $x_i$:

   A. Sample $\tilde{x}_i, y_i \sim \text{mask}(x_i)$
   
   B. Make prediction $\hat{y}_i = f_{\theta}(\tilde{x}_i)$
   
   C. Compute loss $L_i = d(y_i, \hat{y}_i)$

   $\tilde{x}_i, y_i$ are typically two **disjoint** sub-regions of $x_i$

   in some cases, the target $y_i$ may be all of $x_i$

---

$\text{mask( Joe Biden is the US president )} =$

$\tilde{x}_i, y_i$

$\text{mask( Joe <mask> is the US <mask>, } \{ \text{ Biden; president } \}$

---

$f_{\theta}$ : **CNN or Transformer** (stay tuned)

$d(y, \hat{y}) = \| y - \hat{y} \|^2$

$\text{mask( } e.g., \text{ BERT; stay tuned) }$

$d(y, \hat{y}) = \text{KL}(y || \hat{y})$
Masked autoencoders for language:

**BERT** (Devlin et al, 2017)
Case study: BERT as a masked autoencoder

\[ x : [CLS] \text{Joe Biden is the US president.} [SEP] \text{He was inaugurated on January...} \]

\[ \tilde{x} : [CLS] \text{Joe <mask> is the US <mask>.} [SEP] \text{He <mask> inaugurated on January...} \]

\[ y_2 = \text{Biden} \]
\[ y_6 = \text{president} \]
\[ y_9 = \text{was} \]

Target word for each masked index

\[ \mathbb{KL}(y_j \parallel \hat{y}_j) = -\log p^2_\theta(\text{Biden} \mid \tilde{x}) - \log p^6_\theta(\text{president} \mid \tilde{x}) - \log p^9_\theta(\text{was} \mid \tilde{x}) \]

Probability distribution over possible words at each masked index

Details of BERT masking:

1. Choose random 15%* of input timesteps
2. Of these, 80% are replaced with <mask> token
3. Replace other 20% with a random token

*It's possible we can do better than just picking random timesteps:
- Mask longer spans of text
- Selecting for information-dense spans
Masked autoencoders for language: **BERT** (Devlin et al, 2017)

**For images:** **MAE** (He et al, 2021)

Instead of words, we have a sequence of **image patches**

1. Mask ~75% of image patches
2. Compute representations of **only** unmasked patches
3. Insert **placeholder** patches at masked locations
4. Decode back into original image

*Fine-tune on top of the output of step 2*
Masked AEs give state-of-the-art **few-shot image classification** performance (with unsup. pre-training)

The unsupervised masked autoencoding recipe works better than pre-training with labels on the same data!

When **fine-tuning** (not just linear probing on frozen pre-trained model), better than **contrastive learning**!

![Graph](image1.png)

**Figure 8.** MAE pre-training vs. supervised pre-training, evaluated by fine-tuning in ImageNet-1K (224 size). We compare with the original ViT results [16] trained in IN1K or JFT300M.

![Graph](image2.png)

**Figure 9.** Partial fine-tuning results of ViT-L w.r.t. the number of fine-tuned Transformer blocks under the default settings from Table 1. Tuning 0 blocks is linear probing; 24 is full fine-tuning. Our MAE representations are less linearly separable, but are consistently better than MoCo v3 if one or more blocks are tuned.

He et al, 2021
A (very quick) overview of Transformers
A (very quick) overview of Transformers

ViT; Dosovitskiy, Beyer, Kolesnikov, et al. (2021)

The ~only difference~ between Transformers for vision/language/RL/molecules/etc. is what we do for this initial embedding step.
Transformers in a bit more detail

Three separate linear projections of inputs $X$:

- "queries": $X_Q = XW_Q$
- "keys": $X_K = XW_K$
- "values": $X_V = XW_V$

Self-attention output:

$A = \text{softmax} \left( X_Q X_K^T \right) X_V$

Length-T input sequence

$I \in \{\text{vocab}\}^T$

Input tokens

$E \in \mathbb{R}^{T \times d}$

Input embeddings

$X \in \mathbb{R}^{T \times d}$

Self-attention output

$A$ is $X_Q X_K^T X_V$

Self-attention matrix

$A_{ij} = \text{softmax} \left( X_i X_j^T \right)$

Value matrix

$V_{ij} = X_i$

Outputs of block

$O \in \mathbb{R}^{T \times d}$

Project to vocab. size dimensions

$p_0^i(\cdot) \in \mathbb{R}^{\text{vocab}}$

Residual connection

$x + \text{residual connection}$

Length-T input sequence

One transformer block; repeat typically 6-96 times

Joe Biden is the US President

Length-T input sequence

$I \in \{\text{vocab}\}^T$

Input tokens

$E \in \mathbb{R}^{T \times d}$

Input embeddings

$X \in \mathbb{R}^{T \times d}$

Self-attention output

$A = \text{softmax} \left( X_Q X_K^T \right) X_V$

Value matrix

$V_{ij} = X_i$

Outputs of block

$O \in \mathbb{R}^{T \times d}$

Project to vocab. size dimensions

$p_0^i(\cdot) \in \mathbb{R}^{\text{vocab}}$

Residual connection

$x + \text{residual connection}$

Length-T input sequence

One transformer block; repeat typically 6-96 times
So... how do we pre-train fine-tune Transformers?

Extra token prepended to beginning of sequence

[CLS]

Joe Biden is the US President

What should we do with the parameters of this guy during fine-tuning?

Options:
1. **Freeze** them
2. **Fine-tune** them
3. **Something** else???
   a. Fine-tune **some** of them?
   b. Freeze and inject **new** parameters?
LoRA: Low-rank adaptation of language models (Hu et al., 2021)

What if we just want to fine-tune our model... “a little bit”?

What does “a little bit” even mean? <discuss>

1. Preserve the knowledge in the pre-trained model (to avoid overfitting)
2. Avoid needing to store a new version of every single parameter in the model (to save space)
LoRA: Low-rank adaptation of language models (Hu et al., 2021)

What if we just want to fine-tune our model... “a little bit”?

What does “a little bit” even mean? <discuss>

1. Preserve the knowledge in the pre-trained model (to avoid overfitting)
2. Avoid needing to store a new version of every single parameter in the model (to save space)

Associative [key-value] memory view of linear transform (Kohonen, 1972)

Consider the linear transform, the building block of NNs & Transformers

\[ W = \sum_r v_r u_r^\top \]

For rank-\( r \) matrix, we have this decomposition (with orthogonal \( u_r \) by SVD)

Therefore, \( Wx = \left( \sum_r v_r u_r^\top \right) x = \sum_r v_r (u_r^\top x) \rightarrow Wx \) produces a sum over the ‘memories’ \( v_r \) weighted by the relevance \( u_r^\top x \) (each \( u_r \) is a ‘key’)

“A little bit” means only add a few memories \( \rightarrow \) only make a low-rank change to \( W \)

**LoRA:** \[ W_{ft} = W_0 + AB^\top, \quad A, B \in \mathbb{R}^{d \times p}, \quad p < d \]

pre-trained \( d \times d \) weights (frozen)  \( \rightarrow \) new low-rank residual (fine-tuned) \( AB^\top \) should be zero-initialized (how?)
(Many) other approaches to “lightweight” fine-tuning

When “few-shot” means ~20-70, lightweight fine-tuning (T-Few) can outperform in-context learning in much larger models!

T-Few; Lu, Tam, Muqeeth, et al. (2022)  
You will compare fine-tuning and in-context learning in HW3!
Plan for Today

Recap
- Problem formulation
- Contrastive learning

Reconstruction-based unsupervised pre-training
- Why reconstruction?
- Autoencoders
- Masked autoencoders: BERT, MAE
- Autoregressive models: GPT, Flamingo
Striving for simplicity: **autoregressive models**

(recall GPT-3 from the black-box meta-learning lecture!)

What are some **downsides** of masked autoencoders?

1. Need to pick **mask**
2. Only using \( \sim 15\% \) of the example for training
3. Difficult to sample from

Instead of masking a **random subset**, what if we just **predict the next word/pixel/token**?

\[ p_{\theta}(x_t | x_{<t}) \]

No need to pick a masking strategy; **mask every token!**

Simply learn \( p_{\theta}(x_t | x_{<t}) \), **probability of the next token given the previous tokens**

**Autoregressive Transformers** let us compute each \( p_{\theta}(x_t | x_{<t}) \) efficiently:

we can **re-use** representations from the previous step
Autoregressive Transformers are everywhere these days

...for vision too!
...and RL/decision-making!
...and vision + language!

Improving Language Understanding by Generative Pre-Training

Language Models are Unsupervised Multitask Learners

Language Models are Few-Shot Learners

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

OPT: Open Pre-trained Transformer Language Models

Announcing GPT-NeoX-20B

Announcing GPT-NeoX-20B, a 20 billion parameter model trained in collaboration with CoreWeave.

February 2, 2022 - Connor Leahy

As of February 9, 2022, GPT-NeoX-20B checkpoints are available for download from The Eye under Apache 2.0. More in-depth information on GPT-NeoX-20B can be found in the associated technical report on arXiv.

Looking for a demo? Try GPT-NeoX-20B via CoreWeave and Anlatan’s inference service, GooseAI!

WebGPT: Browser-assisted question-answering with human feedback

DeepMind

Flamingo: a Visual Language Model for Few-Shot Learning

Jean-Baptiste Alayrac1, Jeff Donahue1, Pauline Lu1, Antoine Miech1, Iain Barr1, Yana Hasson1, Karel Lenc1, Arthur Menisch1, Katie Millican1, Malcolm Reynolds1, Roman Ring1, Eliza Rutherford1, Serhan Cubi, Tingda Han, Zhizao Gong, Sina Samangooee, Marianne Monteiro, Jacob Menick1, Sebastian Borghard2, Andrew Brock2, Aidan N. Gomez2, Sharifzadeh2, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, Karen Simonyan12

1Equal contributions, ordered alphabetically. 1Equal contributions, ordered alphabetically. 2Equal senior contributions

RT-1: ROBOTICS TRANSFORMER FOR REAL-WORLD CONTROL AT SCALE

Decision Transformer: Reinforcement Learning via Sequence Modeling

Mark Chu
**Case study: Flamingo**

How would you build a multimodal autoregressive model? From scratch? (NO)

[so far] Fine-tuning to **specialize:**

- **General-purpose LM** + **Few-shot data** = **Task-specific LM**

**Flamingo**

Fine-tuning to **combine models:**

- **General-purpose LM** + **General-purpose Vision Model** + **Multimodal data** = **General-purpose Vision-Language Model**

![Flamingo diagram](image-url)
Case study: **Flamingo**

In-context few-shot learning on sequences that freely mix **text** and **images**! Enables few-shot captioning, visual question-answering, etc.
Case study: Flamingo

Few-shot Flamingo ≈ Non-few-shot state of the art!
Are AR models really **different** from masked autoencoders?

**General recipe** for training masked autoencoder $f_\theta$:

1. Choose **distance function** $d(\cdot, \cdot) \to \mathbb{R}$
2. For **train batch** examples $x_i$:
   A. Sample $\tilde{x}_i, y_i \sim \text{mask}(x_i)$
   B. Make prediction $\hat{y}_i = f_\theta(\tilde{x}_i)$
   C. Compute loss $= d(y_i, \hat{y}_i)$

**AR models are just masked AEs with a special choice of mask**
Summary of today

1. Intuition for autoencoders (AEs): “A good representation lets us reconstruct the input”

2. **Masked** AEs learn to restore a **partially-deleted** input & help avoid degeneracies in unmasked AEs

3. **State of the art** in pre-training for few-shot learning in **language & vision**

4. **Autoregressive** models (e.g., GPT-3) are **special case** of masked AEs; give a generative model for free at some cost to fine-tuning performance
# Contrastive Learning vs AEs vs Masked AEs

**Contrastive learning:**

+ Learns very high-quality representations
+ Don’t need as large a model
  - Need to select negatives carefully*
  - Generally needs larger batch size*
  - Cross-example dependencies can make implementation more difficult

* new methods are addressing these downsides but are more difficult to interpret/analyze

**Bottlenecked Autoencoders:**

+ Simple to implement
+ No need to select pos/neg pairs; just $d(x, \hat{x})$
  - Generally need a larger model

**Masked autoencoders:**

+ Few-shot performance as good or better than contrastive
+ AR special case gives generative model for free
  - Raw representations (without fine-tuning) still can be lower quality than contrastive
Reminders

Project proposal due TODAY!

Homework 2 due Wednesday

Make sure you have set-up Azure!
(well before the HW deadline)