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

CS 330
Logistics

Project proposal due TODAY!

Homework 2 due Monday, October 24

Kyle’s office hours are hybrid going forward (see Ed for details)

Azure invites have been re-sent - you have one week to accept!

You will need Azure for HW3, so do this today!
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 $\mathcal{D}^{tr}_j$
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)

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:

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

Schroff, Kalenichenko, Philbin. CVPR 2015
**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**):
\[
\mathcal{L}_{N\text{-way}}(\theta) = -\sum_z \log \frac{\exp(-d(z,z^+))}{\sum_i \exp(-d(z,z_i^-))}
\]

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

**Contrastive Learning Implementation**

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

Need to both compare & **contrast**!

---

**V1. Triplet loss:**

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

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

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

Positive score in denominator → 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

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 $r$

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

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

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

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

What can go wrong here?

Is the identity function a good encoder/decoder?
Autoencoders: a first attempt

**How to fix??**
**Autoencoders: adding a bottleneck**

- **Input image, sentence, audio signal, etc.**
- **Encoder (CNN)**
- **Compact, latent representation of input image**
- **Decoder (CNN)**
- **Reconstruction of input image**

**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

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

\[
\mathcal{L} = d(x, \hat{x}) = \|x - \hat{x}\|^2
\]
Autoencoders: few-shot learning

\[ \chi \]

Encoder (CNN)

Prediction head mapping \( r \) to output space

\[ \hat{y} \]

**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 $\ldots x \ (\text{through } r)$

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

![Diagram of masked autoencoder](image_url)
Beyond the bottleneck: **masked autoencoders**

**General recipe** for **pre-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 $\mathcal{L}_i = d(y_i, \hat{y}_i)$

$x_i, y_i$ are typically two **disjoint** sub-regions of $x_i$

---

** These pieces are our design choices/control knobs

---

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

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

---

$x_i$

\[
\text{mask}(\text{Joe Biden is the US president}) = \text{mask}(\text{Joe <mask> is the US <mask>}, \{\text{Biden; president}\})
\]

---

$f_\theta : \text{Transformer (e.g., BERT; stay tuned)}$

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

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

\[
\begin{align*}
X : \, & \text{[CLS]} \text{ Joe Biden is the US president. [SEP] He was inaugurated on January...} \\
\tilde{X} : \, & \text{[CLS]} \text{ Joe <mask> is the US <mask>. [SEP] He <mask> inaugurated on January...} \\
\end{align*}
\]

\[
\begin{align*}
y_2 = & \text{ Biden} \\
y_6 = & \text{ president} \\
y_9 = & \text{ was} \\
\end{align*}
\]

Target word for each masked index

\[
\sum_j \text{KL}(y_j \| \hat{y}_j) = - \log p_\theta^2(\text{ Biden} | \tilde{x}) - \log p_\theta^6(\text{ president} | \tilde{x}) - \log p_\theta^9(\text{ was} | \tilde{x})
\]

**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*
More recently: Masked AEs give state-of-the-art **few-shot image classification** performance

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**!

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.

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

The only difference between Transformers for vision/language/RL/molecules/etc. is what we do for this initial embedding step.

ViT; Dosovitskiy, Beyer, Kolesnikov, et al. (2021)
Transformers in a bit more detail

Three separate linear projections of inputs $X$:

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

For each position $t$ in the input sequence:

1. **Norm** $\mu(x_t) = 0$ and $\sigma(x_t) = 1$
2. **Self-attention output**
   - $a_1 = e^{x_1^Q x_2^K}$
   - $a_2 = e^{x_2^Q x_3^K} + \sum_t e^{x_3^Q x_t^K}$
   - $a_3 = e^{x_3^Q x_T^K}$
3. **Residual connection** $+ X$
4. **MLP**
5. **Residual connection** $+ X$

Outputs of block $O = X + \text{MLP}(X)$

Inputs to block $X = E$, where $E \in \mathbb{R}^{T \times d}$.

Project to vocab. size dimensions $p_\theta^t(\cdot)$ for $t = 1, 2, \ldots, 6$.

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

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)

\[ Wx = \left( \sum_r v_r u_r^\top \right) x = \sum_r v_r \left( u_r^\top x \right) \rightarrow Wx \text{ produces a sum over the 'memories' } v_r \text{ weighted by the relevance } u_r^\top x \]  

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

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

pre-trained weights (frozen)  
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 from the black-box meta-learning lecture! /

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

1. Need to pick **mask**
2. Only using ~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**?

→ 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.

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 Anlatai’s inference service, GooseAI!

DeepMind

Flamingo: a Visual Language Model for Few-Shot Learning

Jean-Baptiste Alayrac1,2, Jeff Donahue1, Pauline Luc1, Antoine Miech1, Iain Barr1, Yana Hasson1, Karel Lenc1, Arthur Mensch1, Katie Millican1, Malcolm Reynolds1, Roman Ring1, Eliza Rutherford1, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangoei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, Karen Simonyan1,2

1Equal contributions, ordered alphabetically, 2Equal contributions, ordered alphabetically, 3Equal senior contributions

28-04-2022
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

Fine-tuning to combine models:

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** \( d(\cdot, \cdot) \rightarrow \mathbb{R} \)
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 \( = d(y_i, \hat{y}_i) \)

**Masked autoencoder:**

\[
\tilde{x} : \quad y : \\
\text{Joe} : \quad \text{Joe}
\]

**AR model:**

\[
\tilde{x} : \quad y : \\
\text{Joe} : \quad \text{Joe}
\]

**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
<table>
<thead>
<tr>
<th><strong>Contrastive learning:</strong></th>
<th><strong>(Bottlenecked) Autoencoders:</strong></th>
<th><strong>Masked autoencoders:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Learns very high-quality representations</td>
<td>+ Simple to implement</td>
<td>+ Few-shot performance as good or better than contrastive</td>
</tr>
<tr>
<td>+ Don’t need as large a model</td>
<td>+ No need to select pos/neg pairs; just $d(x, \hat{x})$</td>
<td>+ AR special case gives generative model for free</td>
</tr>
<tr>
<td>- Need to select negatives carefully*</td>
<td>- Need to design a bottleneck</td>
<td>- Raw representations (without fine-tuning) still can be lower quality than contrastive</td>
</tr>
<tr>
<td>- Generally needs larger batch size*</td>
<td>- (Comparatively) poor few-shot performance</td>
<td></td>
</tr>
<tr>
<td>- Cross-example dependencies can make implementation more difficult</td>
<td>- Not generally used in practice</td>
<td></td>
</tr>
</tbody>
</table>

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

Project proposal due TODAY!
Homework 2 due Monday, October 24

Kyle’s office hours are hybrid going forward (see Ed for details)

Azure invites have been re-sent - you have one week to accept!

You will need Azure for HW3, so do this today!