Pretrained Models

Collective Machine Intelligence

Antonio Carta
University of Pisa
Antonio.carta@unipi.it
• **Data**: quality, evaluation, scaling laws
• **Models**: Vision and Vision-Language Transformers
• **Efficient Finetuning**: Distillation, LoRA, Prompt Tuning
• **Research**: Model patching, model editing, explaining ICL
Large-Scale Pretraining
Training on a Web-Scale

• We have a very bad intuition about how much diversity there is in web-scale data
  • In general, much more diversity than we expect
  • There are examples of general tasks such as translation

• Language provides a flexible way to specify tasks, inputs, and outputs all as a sequence of symbols
  • How do we use language to achieve few-shot and zero-shot learning?
Few-Shot and Zero-Shot Learning

Given a large pretrained language model

- **In-Context-Learning/Prompting**: at inference time we give a description of the task, (optionally) followed by a sequence of examples
  - No weights are updated
- **Few-Shot**: the model is given a few samples of the task as conditioning
- **One-Shot**: a single example
- **Zero-Shot**: only the task description is given
Examples

In formal conversations, it is important to understand which topics are suitable and which are considered taboo in a particular culture. Latin Americans enjoy sharing information about their local history, art, and customs. You may expect questions about your family, and be sure to show pictures of your children. You may feel free to ask similar questions of your Latin American friends. The French think of conversation as an art form, and they enjoy the value of lively discussions as well as disagreements. For these arguments can be interesting and they can cover pretty much any topic — as long as they occur in a respectful and intelligent manner.

In the United States, business people tend to discuss a wide range of topics, including opinions about work, family, hobbies, and politics. In Japan, China, and Korea, however, people are much more private; they do not share much about their thoughts, feelings, or emotions because they feel that doing so might take away from the harmonious business relationship they’re trying to build. Middle Easterners are also private about their personal lives and family matters. It is considered rude, for example, to ask a businessman from Saudi Arabia about his wife or children.

As a general rule, it’s best not to talk about politics or religion with your business friends. This can get you into trouble, even in the United States, where people hold different religious views. In addition, discussing one’s salary is usually considered unsuitable. Sports is typically a friendly subject in most parts of the world, although be careful not to criticize national sport. Instead, be friendly and praise your host’s team.

Q: What shouldn’t you do when talking about sports with colleagues from another country?

A: Criticizing the sports of your colleagues’ country.

Q: Which is typically a friendly topic in most places according to the author?

A: Sports.

Q: Why are people from Asia more private in their conversation with others?

A: They don’t want to have their good relationship with others harmed by informal conversations.

Q: The author considers politics and religion...

A:

Correct Answer → taboo
Incorrect Answer → cheerful topics
Incorrect Answer → rude topics
Incorrect Answer → topics that can never be talked about

Figure G.1: Formatted dataset example for RACE-h. When predicting, we normalize by the unconditional probability of each answer.
例子的提示

<table>
<thead>
<tr>
<th>上下文</th>
<th>如何在木头上涂密封胶。</th>
</tr>
</thead>
<tbody>
<tr>
<td>正确答案</td>
<td>使用刷子，在木头上刷上密封胶，直到密封胶完全饱和。</td>
</tr>
<tr>
<td>误答</td>
<td>使用刷子，在木头上刷上密封胶，直到密封胶完全饱和。</td>
</tr>
</tbody>
</table>

**图 G.4**: 格式化数据集示例用于PIQA
### LLMs are Few-Shot Learners

<table>
<thead>
<tr>
<th>Setting</th>
<th>NaturalQS</th>
<th>WebQS</th>
<th>TriviaQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAG (Fine-tuned, Open-Domain) [LPP⁺20]</td>
<td>44.5</td>
<td>45.5</td>
<td>68.0</td>
</tr>
<tr>
<td>TS-11B+SSM (Fine-tuned, Closed-Book) [RRS20]</td>
<td>36.6</td>
<td>44.7</td>
<td>60.5</td>
</tr>
<tr>
<td>TS-11B (Fine-tuned, Closed-Book)</td>
<td>34.5</td>
<td>37.4</td>
<td>50.1</td>
</tr>
<tr>
<td>GPT-3 Zero-Shot</td>
<td>14.6</td>
<td>14.4</td>
<td>64.3</td>
</tr>
<tr>
<td>GPT-3 One-Shot</td>
<td>23.0</td>
<td>25.3</td>
<td>68.0</td>
</tr>
<tr>
<td>GPT-3 Few-Shot</td>
<td>29.9</td>
<td>41.5</td>
<td>71.2</td>
</tr>
</tbody>
</table>

Table 3.2: Results on three Open-Domain QA tasks. GPT-3 is shown in the few-, one-, and zero-shot settings, as compared to prior SOTA results for closed book and open domain settings. TriviaQA few-shot result is evaluated on the wiki split test server.

<table>
<thead>
<tr>
<th>Setting</th>
<th>ARC (Easy)</th>
<th>ARC (Challenge)</th>
<th>CoQA</th>
<th>DROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-tuned SOTA</td>
<td>92.0⁺</td>
<td>78.5ᵇ</td>
<td>90.7ᶜ</td>
<td>89.1ᵈ</td>
</tr>
<tr>
<td>GPT-3 Zero-Shot</td>
<td>68.8</td>
<td>51.4</td>
<td>81.5</td>
<td>23.6</td>
</tr>
<tr>
<td>GPT-3 One-Shot</td>
<td>71.2</td>
<td>53.2</td>
<td>84.0</td>
<td>34.3</td>
</tr>
<tr>
<td>GPT-3 Few-Shot</td>
<td>70.1</td>
<td>51.5</td>
<td>85.0</td>
<td>36.5</td>
</tr>
</tbody>
</table>

Table 3.3: GPT-3 results on a selection of QA / RC tasks. CoQA and DROP are F1 while ARC reports accuracy. See the appendix for additional experiments. ᵃ[KKS⁺20]ᵇ[KKS⁺20]ᶜ[JJC⁺19]ᵈ[JN20]

Challenges

• Prompting requires rethinking the adaptation and finetuning of models
• It’s hard to evaluate LLMs zero-shot ability
  • How can we ensure that the task is really unseen?
• Still hard to control the model or estimate uncertainty
Challenges in the Evaluation

- Github CoPilot memorized the solution of many programming challenges
- Example: Project Euler #14
  - A simple problem about the convergence of Collatz sequences

How to get your solution
- Add a comment with the link to project euler
  - # https://projecteuler.net/problem=14
- Press tab or enter until the code is complete. Copilot will write the full solution

```python
# https://projecteuler.net/problem=14
# imports
import time

# Measure runtime
start = time.time()

# Variables
longest_chain = 0
longest_chain_start = 0

# Functions
def collatz(n):
    chain = 1
    while n != 1:
        if n % 2 == 0:
            n = n/2
            chain += 1
        else:
            n = 3*n + 1
            chain += 1
    return chain

# Main
for i in range(1, 1000000):
    chain = collatz(i)
    if chain > longest_chain:
        longest_chain = chain
        longest_chain_start = i

print(longest_chain_start)

# Print runtime
print("--- %.2f seconds ---" % (time.time() - start))
```
Data and Scaling Laws
Training Large (Huge) Models

Training large models is very expensive
• You can train a single run (it cost M$)
• You would like to know the cost in advance, whether the performance improvements are worth the cost

Problems:
• How do we do model selection?
• How do we choose the model size?
• How much data do we need?
• What error will we get?
Model Selection

- You can’t do it. A training is a single run
- If models become unstable recover from previous checkpoints
- Needs to account for machine failures (the model is trained on a cluster)
Chinchilla Scaling Law

- **Chinchilla Scaling Law**: an empirical law that determines the loss given
  - \( N \): number of parameters of the model
  - \( D \): number of tokens in the dataset
  - Three components: model error, data error, irreducible loss

\[
L(N, D) = \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}} + 1.69
\]

J. Hoffmann et al. "Training Compute-Optimal Large Language Models." 2022
Figure 3 | IsoFLOP curves. For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (left). Using the location of these valleys, we project optimal model size and number of tokens for larger models (center and right). In green, we show the estimated number of parameters and tokens for an optimal model trained with the compute budget of Gopher.
Larger models are not always better, you also need larger data.

Given a fixed computational budget and a dataset, you can find the optimal model size.

The optimality is wrt to the loss and training cost.

- Inference cost is ignored and it can be high due to the large model size suggested by the Eq.

Unclear if the scaling law will hold for larger models.
If the trend continues

- **high-quality language data** will be exhausted soon; likely before 2026.
- **low-quality** language data and image data will be exhausted only much later; between 2030 and 2050 (for low-quality language) and between 2030 and 2060 (for images).

---

**Fig. 1:** Projections of data usage. Each graph shows two extrapolations of data usage, one from past trends and one from compute availability estimations plus scaling laws. Both projections are constrained to be lower than the estimated data stock. In all three cases, this constraint causes a slowdown in data usage growth.

*P. Villalobos et al. "Will We Run out of Data? An Analysis of the Limits of Scaling Datasets in Machine Learning." 2022*
<table>
<thead>
<tr>
<th>Model</th>
<th>Stock of data (#words)</th>
<th>Growth rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded speech</td>
<td>1.46e17</td>
<td>5.2%</td>
</tr>
<tr>
<td></td>
<td>[3.41e16; 4.28e17]</td>
<td>[4.95%; 5.2%]</td>
</tr>
<tr>
<td>Internet users</td>
<td>2.01e15</td>
<td>8.14%</td>
</tr>
<tr>
<td></td>
<td>[6.47e14; 6.28e15]</td>
<td>[7.89%; 8.14%]</td>
</tr>
<tr>
<td>Popular platforms</td>
<td>4.41e14</td>
<td>8.14%</td>
</tr>
<tr>
<td></td>
<td>[1.21e14; 1.46e15]</td>
<td>[7.89%; 8.14%]</td>
</tr>
<tr>
<td>CommonCrawl</td>
<td>9.62e13</td>
<td>16.68%</td>
</tr>
<tr>
<td></td>
<td>[4.45e13; 2.84e14]</td>
<td>[16.41%; 16.68%]</td>
</tr>
<tr>
<td>Indexed websites</td>
<td>2.21e14</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>[5.16e13; 6.53e15]</td>
<td></td>
</tr>
<tr>
<td>Aggregated model</td>
<td>7.41e14</td>
<td>7.15%</td>
</tr>
<tr>
<td></td>
<td>[6.85e13; 7.13e16]</td>
<td>[6.41%; 17.49%]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
<th>Doubling time median and CI (months)</th>
<th>Largest training dataset (datapoints)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>15.8 [11.2; 20.9]</td>
<td>2e12</td>
</tr>
<tr>
<td>Vision</td>
<td>41.5 [30.4; 48.3]</td>
<td>3e9</td>
</tr>
</tbody>
</table>

TABLE I: Trends in training dataset size for vision and language models.

Data source:
- Social media conversations
- Filtered webpages
- Books (English)
- GitHub (code)
- Wikipedia (multilingual)
- News (English)

(a) Composition of high-quality datasets: The Pile (left), PaLM (top-right), MassiveText (bottom-right)

(a) Comparison of the different data stock models.
Vision Transformer
• MultiHead(Q, K, V) = Concat(head$_1$, ..., head$_h$)W$^O$

• head$_i$ = Attention(QW$_i^Q$, KW$_i^K$, VW$_i^V$
  • Transformers use multiple heads, each transformed by different linear projections

• Attention(Q, K, V) = softmax(QK$^T$/√d$_k$)V
  • Scaled dot-product attention
  • Q query
  • K keys
  • V values
Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).
Embedding Image Patches

• Split images in 16x16 patches
• Add positional embedding
  • Learnable 1D embeddings
  • Needed because attention is invariant to the patch position
• Once we have the 1D embeddings, we can use the Transformer like we do for text
ViT Block

- MSA: Multiheaded Self-Attention
- LN: Layer Normalization
- GELU nonlinearity

\[
\begin{align*}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}}; \mathbf{x}_{\mathbf{p}}^1 \mathbf{E}; \mathbf{x}_{\mathbf{p}}^2 \mathbf{E}; \ldots; \mathbf{x}_{\mathbf{p}}^N \mathbf{E}] + \mathbf{E}_{\text{pos}} \\
\mathbf{z}^\prime_\ell &= \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \ \ell = 1 \ldots L \\
\mathbf{z}_\ell &= \text{MLP}(\text{LN}(\mathbf{z}^\prime_\ell)) + \mathbf{z}^\prime_\ell, \ \ell = 1 \ldots L \\
\mathbf{y} &= \text{LN}(\mathbf{z}^0_L)
\end{align*}
\]

- \( \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \)
- \( \mathbf{E} \) projects embedding into latent space
- Image HxWxC
- Patch \( P^2 \mathcal{C} \) (flattened patch)
- D latent size dimension
- \( \mathbf{x}_{\text{class}} \) is a learned embedding and provides the final image representation at the last layer (equivalent to BERT[\text{class}] token)
Results

Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-E with all hidden dimensions halved.

Vision-Language Models
Motivation

• With LLM we can do zero-shot training with prompting
• We want to exploit language to encode visual information
  • Language encodes a high-level understanding of images
  • We have lots of captioned images that provide much more information than using the image alone
• We need to combine vision and language models
• **pre-training task**: learn to map text captions with visual images

• **zero-shot transfer**: natural language is used to reference learned visual concepts (or describe new ones)
Contrastive Learning

- **Objective**: learn a latent representation space that is semantically meaningful (for downstream tasks)
- **Example**: in a downstream classification problem, we would like the examples to be linearly separable
- **IDEA**: similar images are close to each other, diverse images are far
- **contrastive learning**: compare and contrast pairs of images during training to learn good representations
Contrastive Pretraining

- **Supervised training** is inefficient if we try to learn from <caption, image> pairs
  - There is too much diversity in the possible images and captions
  - It is very difficult to predict the exact image/captions of the current pair
- **Solution**: contrastive training
  - Given a batch of <caption, image> pairs, predict which text caption maps with which image among the NxN possibilities
  - Learns a multi-modal embedding space where text and images are aligned

CLIP Pretraining

```python
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) # [n, d_i]
T_f = text_encoder(T) # [n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arrange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

*Figure 3. Numpy-like pseudocode for the core of an implementation of CLIP.*

Zero-Shot Image Classification

- Given a number of classes {«dog», «cat», ... }
- Create prompts: «this is an image of {class}»
- For each image, find the closest prompt
  - You can encode the text prompt and use its embedding to get a linear classification head

Zero-Shot Image Classification

- **Ensembling**: You can use multiple prompts for the same class:
  - «a photo of a small {class}»
  - «a photo of a big {class}»

- **Prompt Engineering**: different prompts can have different results
  - E.g. «{class}» vs «a photo of {class}»

- **Paper Results**: (on ImageNet)
  - ensemble 80 different context prompts (+3.5%) w.r.t. single default prompt
  - prompt engineering and ensembling improve (+5%)
  - In general, ensembling and prompt engineering have a big impact

Figure 2. CLIP is much more efficient at zero-shot transfer than our image caption baseline. Although highly expressive, we found that transformer-based language models are relatively weak at zero-shot ImageNet classification. Here, we see that it learns 3x slower than a baseline which predicts a bag-of-words (BoW) encoding of the text (Joulin et al., 2016). Swapping the prediction objective for the contrastive objective of CLIP further improves efficiency another 4x.
Robustness to Dataset Shift

Figure 13. Zero-shot CLIP is much more robust to distribution shift than standard ImageNet models. (Left) An ideal robust model (dashed line) performs equally well on the ImageNet distribution and on other natural image distributions. Zero-shot CLIP models shrink this “robustness gap” by up to 75%. Linear fits on logit transformed values are shown with bootstrap estimated 95% confidence intervals. (Right) Visualizing distribution shift for bananas, a class shared across 5 of the 7 natural distribution shift datasets. The performance of the best zero-shot CLIP model, ViT-L/14@336px, is compared with a model that has the same performance on the ImageNet validation set, ResNet-101.

Efficient Finetuning
We can use large models to create data to train the smaller models

• 7B model trained from GPT3.5 inputs
  • Cheap to train
    • 3 hours on 8 80GB A100s
    • <600$ cloud cost

• Start from LLaMA (7B LLM)
• Finetune with
  • 175 human-written instruction-output pairs
  • 52K additional pairs generated by GPT 3.5 using the human-written samples as context

https://github.com/tatsu-lab/stanford_alpaca
Prompt Tuning – Learning To Prompt (L2P)

- For classification problems, the prompts are fixed embedding vectors that we compare to the image embedding.
- Can we learn the prompt?
- **Learning2Prompt:**
  - Continual learning method
  - Fixed pretrained backbone
  - Keep a prompt pool with <key, value> pairs
  - Uses the key to select the prompt

---

Figure 2. Illustration of L2P at test time. We follow the same procedure at training time: First, L2P selects a subset of prompts from a key-value paired prompt pool based on our proposed instance-wise query mechanism. Then, L2P prepends the selected prompts to the input tokens. Finally, L2P feeds the extended tokens to the model, and optimize the prompt pool through the loss defined in equation 5. The objective is learning to select and update prompts to instruct the prediction of the pre-trained backbone model.
L2P – Prompt Selection

• $K_x = \arg\min_{\{s_i\}^N_{i=1}\subseteq[1,M]} \sum_{i=1}^{N} \gamma(q(x), k_{s_i})$ Prompt Selection
  • $q(x)$ pretrained feature extractor gets query features
  • $k_{s_i}$ key
  • $\gamma$ scoring function that matches image and prompts

• $\min_{p,K,\phi} \mathcal{L} \left( g_{\phi} \left( f_{r}^{\text{avg}} (x_p) \right), y \right) + \lambda \sum_{K} \gamma(q(x), k_{s_i})$ Optimization Objective
  • $f_{r}^{\text{avg}}$ ViT encoder
  • $g_{\phi}$ classifier

Efficient Finetuning – Low Rank Adaptation (LoRA)

• We can finetune only a small part of the model
• Represent the finetuned models as $\Phi_0 + \Delta \Phi$ with small (in memory size) $\Delta \Phi$
  • This allows us to easily train and store hundreds of finetuned models
• Low-Rank Parameter Update
  • $h = W_0 x + \Delta W x = W_0 x + BA x$
  • With $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, r \ll d$

Figure 1: Our reparametrization. We only train $A$ and $B$. 
Continual Pretraining
Continual Pretraining

• **Continual Pretraining** is the problem of efficiently updating a large pretrained model

• **Forgetting Control Task**: we don’t want to forget general knowledge

• **Downstream Task**: we want to improve on domain-specific tasks

Evaluation on the Forgetting Control Task

Table 2: Accuracy on the entire dataset of sentiment analysis with RoBERTa model. Continual pre-training has been performed sequentially over each experience of scientific abstracts. Base refers to the model pre-trained on Wikipedia, while NT refers to the model with vocabulary expansion.

<table>
<thead>
<tr>
<th>RoBERTa</th>
<th>Accuracy</th>
<th>1-epoch Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base</strong></td>
<td>93.40</td>
<td>92.40</td>
</tr>
<tr>
<td>Exp.</td>
<td>e1</td>
<td>e2</td>
</tr>
<tr>
<td></td>
<td>e3</td>
<td>e4</td>
</tr>
<tr>
<td></td>
<td>e5</td>
<td>e1</td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>e3</td>
</tr>
<tr>
<td></td>
<td>e4</td>
<td>e5</td>
</tr>
<tr>
<td><strong>Pretr</strong></td>
<td>93.40</td>
<td>92.90</td>
</tr>
<tr>
<td></td>
<td>93.15</td>
<td>93.20</td>
</tr>
<tr>
<td></td>
<td>93.35</td>
<td>92.40</td>
</tr>
<tr>
<td></td>
<td>93.20</td>
<td>91.80</td>
</tr>
<tr>
<td></td>
<td>92.30</td>
<td>91.85</td>
</tr>
<tr>
<td></td>
<td>92.20</td>
<td>92.20</td>
</tr>
<tr>
<td><strong>Pretr. NT</strong></td>
<td>93.75</td>
<td>93.70</td>
</tr>
<tr>
<td></td>
<td>93.75</td>
<td>93.75</td>
</tr>
<tr>
<td></td>
<td>93.60</td>
<td>94.10</td>
</tr>
<tr>
<td></td>
<td>91.75</td>
<td>91.15</td>
</tr>
<tr>
<td></td>
<td>92.00</td>
<td>92.30</td>
</tr>
<tr>
<td></td>
<td>92.30</td>
<td>92.45</td>
</tr>
</tbody>
</table>

Forgetting is limited even with finetuning. Dynamic vocabulary expansion (NT) slightly improves the performance.

Self-supervised pretraining is more robust than supervised methods (result for vision in the paper)

Self-Supervised CL

- Distillation loss maps old representations in a new projected space
- SSL tricks such as heavy augmentations and SSL losses
- Linear probing evaluation

![Diagram of the CaSSL framework](image)

**Figure 1.** Self-supervised continual learning with Projected Functional Regularization. Instead of performing feature distillation directly between the previous task backbone and the new one, we use a learned temporal projection between the two feature spaces.

**Figure 2.** Overview of the CaSSL framework.

_E. Fini et al. “Self-Supervised Models Are Continual Learners.” CVPR ‘22_
We can also use KD to train SSL models.

SSL loss on new data + KD loss on old embeddings

\[ \mathcal{L} = \mathcal{L}_{\text{SSL}}(z^A, z^B) + \mathcal{L}_D(z^A, \tilde{z}^A) = \mathcal{L}_{\text{SSL}}(z^A, z^B) + \mathcal{L}_{\text{SSL}}(g(z^A), \tilde{z}^A). \]

- A and B images, \( z^A, z^B \) embeddings of new model
- \( z^A \) embedding of old model
- \( \mathcal{L}_{\text{SSL}} \) is used as the distillation loss
- \( g \) is a projection network that maps from the new representation to the old ones
  - Allows to update the representations and change them
• self-supervised pretraining provides very general knowledge about large domains (vision, language)
  • We can adapt them to solve new tasks with few (or zero) examples
  • They are more robust compared to supervised models
  • They are even more robust than finetuned models
Model Patching
Community-developed and continually-improved models
- Incremental and cheaply-communicable updates
- Merging models
- Vetting community contributions
- Versioning and backward compatibility
- Modularity and distribution

https://colinraffel.com/blog/a-call-to-build-models-like-we-build-open-source-software.html
GOAL: we want to consolidate two pretrained models

- Given two models $f_{\theta_1}$ and $f_{\theta_2}$ trained on tasks T1 and T2
- Find a model $f_{\theta_M}$ that solves both T1 and T2
- The two models can have different architectures, initialization, training algorithm, tasks...
Mode Connectivity: the models (modes) are connected by a low-loss path.

• We construct continuous paths between minima of recent neural network architectures on CIFAR10 and CIFAR100."

• "We observe the following trend: The deeper and wider an architecture, the lower are the saddles between minima until they essentially vanish for current-day deep architectures.

• The more complex dataset CIFAR100 raises the barriers."

• AutoNEB: method for connecting minima from molecular statistical mechanics

F. Draxler et al. “Essentially No Barriers in Neural Network Energy Landscape,”
Linear Mode Connectivity:

- Naive average: $\theta^M = \frac{1}{2} (\theta^1 + \theta^2)$
- Weighted average: $\theta^M_i = \alpha_i \theta^1_i + (1 - \alpha_i) \theta^2_i$

How do we find the weights?
- (normalized) Fisher diagonal
- (normalized) gradients
- Optimizing on a validation set
- Open-vocabulary models are characterized by their ability to perform any image classification task (example: CLIP)
- Based on text descriptions of the classes
PAINT also allows a single model to be patched on multiple tasks and improves with model scale. Furthermore, we identify cases of broad transfer, where patching on one task increases accuracy on other tasks.

**Step 1.** Fine-tune $\theta_{zs}$ on training data from $D_{\text{patch}}$ to produce a model with weights $\theta_{ft}$.

**Step 2.** For mixing coefficient $\alpha \in [0, 1]$, linearly interpolate between $\theta_{zs}$ and $\theta_{ft}$ to produce $\theta_{\text{patch}} = (1 - \alpha) \cdot \theta_{zs} + \alpha \cdot \theta_{ft}$. The mixing coefficient is determined via held-out validation sets for $D_{\text{supp}}$ and $D_{\text{patch}}$. We refer to the resulting model as $\theta_{\text{patch}}$. 

G. Ilharco et al. «Patching open-vocabulary models by interpolating weights» NeurIPS 2022
Git-Rebasin

• **IDEA**: we can exploit DNN symmetries to find a “better alignment” between the models that allows to patch them via weight averaging (remember: no barriers in loss).

• **DNN units can be permuted without changing the DNN output:**

\[
W'_\ell = P_\ell W^{(B)}_\ell P^\top_{\ell-1}, \quad b'_\ell = P_\ell b^{(B)}_\ell
\]

• **Git-Rebasin**: find the best unit permutations by matching weights or activations. Then, we can merge with the naïve average.

What are we missing?

Necessary conditions for mode connectivity are unclear
- initialization?
  - Pretrained models are the best choice -> if the first part of the training is shared patching becomes much easier.
  - Same init is better. Different init works for simple datasets?
- Width and depth? Wider is better, but is it always enough? How wide should it be? How quickly is it growing with dataset complexity?
- Optimizer? Adaptive vs non-adaptive
- Architecture? Transformers, residual connections, batchnorm...
Are Transformers Meta-Learning Methods?
In-Context Learning

• What is in-context learning (ICL) and why does it work so well?
• Is ICL even learning?
• Is it just an emergent property of large pretraining/data/architecture?

• **IDEA:** If ICL is a learning algorithm, then pretraining is a «meta-learning» algorithm
Motivation – Learning to Learn

- **Meta-learning**: Can we optimize the learning algorithm to solve novel tasks (transfer, low-shot and fast adaptation, hyperparameter search)? a.k.a. *Learning to learn*
  - $A^{META}: \{D_1, ..., D_N\} \rightarrow A^*, A^*: D_{N+1}^{train} \rightarrow \theta^{N+1}$

- **Example**: Optimization-based meta learning (MAML)
  - $\theta^* = \min_{\theta} \mathbb{E}_{p(T)} \left[ \mathcal{L}_T \left( U_k \left( \theta, D_{\tau}^{supp} \right), D_{\tau}^{query} \right) \right]$
  - Learn the best init
Hypothesis: ICL is a form of SGD

• “we suggest that training Transformers on auto-regressive objectives is closely related to gradient-based metalearning formulations”

• “trained Transformers become mesa-optimizers i.e. learn models by gradient descent in their forward pass.”

• The paper provides a constructive proof of this behavior for regression problems
  • Note that this means that maybe ICL is not a consequence of large-scale training, but of the Transformer architecture

![Diagram]

**Figure 1.** Illustration of our hypothesis: gradient-based optimization and attention-based in-context learning are equivalent. Left: Learning a neural network output layer by gradient descent on a dataset $D^{\text{train}}$. The task-shared meta-parameters $\theta$ are obtained by meta-learning with the goal that after adjusting the neural network output layer, the model generalizes well on unseen data. Center: Illustration of a Transformer that adjusts its query prediction on the data given in-context i.e. $t_\theta(x_{\text{query}}; D^{\text{context}})$. The weights of the Transformer are optimized to predict the next token $y_{\text{query}}$. Right: Our results confirm the hypothesis that learning with $K$ steps of gradient descent on a dataset $D^{\text{train}}$ (green part of the left plot) matches trained Transformers with $K$ linear self-attention layers (central plot) when given $D^{\text{train}}$ as in-context data $D^{\text{context}}$.

Hypothesis: ICL is a form of SGD

• “We reverse-engineer Transformers trained on simple sequence modeling tasks, and find strong evidence that their forward pass implements two-step algorithms: (i) early self-attention layers construct internal training datasets by grouping and copying tokens, and therefore implicitly define internal objective functions, (ii) deeper layers optimize these objectives to generate predictions.”

Conclusion
Take-Home Messages

• scale matters

• Be careful about the evaluation
  • It’s easy to evaluate on the training set
  • companies are not sharing many details about the models (evaluation, training data, number of parameters)

• methods that we saw in the course are used (is slightly different forms)
  • distillation to reduce the size of the model
  • zero-shot and few-shot techniques (e.g. prompting)
  • finetuning (e.g. LoRA)