Neural Networks for Data Science Applications

Master's Degree in Data Science

Lecture 10: Transfer learning

Lecturer: J. Pomponi, Post-Doc

SAPIENZA
UNIVERSITÀ DI ROMA

Transfer Learning

Introduction

Imagine having a small dataset containing images of rare animals, and you want to create a model capable of classifying them. This is hard because NNs require tons of data to train correctly...

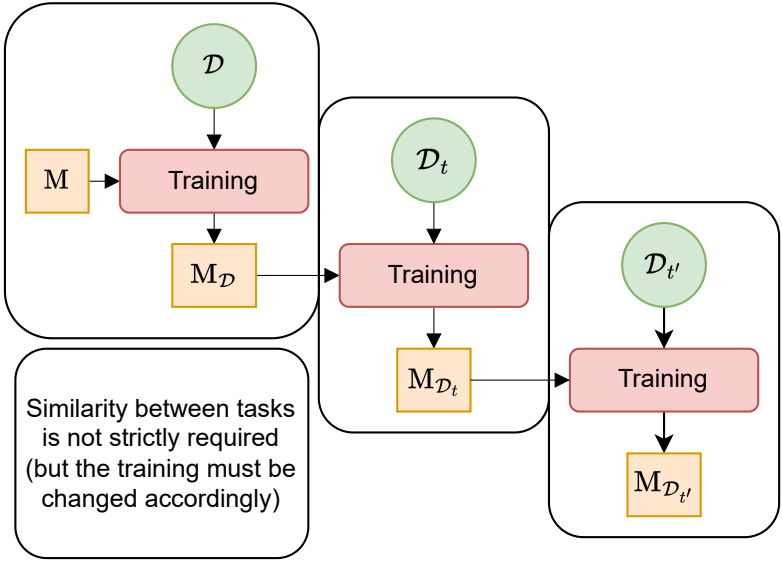
Or to have a big dataset to train a model with, but your computational resources are limited. How to overcome such limitations?

What if you need to satisfy both?

We can use pre-trained models to overcome both issues!

We take an already trained model M (e.g. an image classifier trained on ImageNet), and we take advantage of its weights by adapting them to our task *t*, represented by a dataset $\mathcal{D}_t$.

This process can be repeated multiple times.


$\mathcal{D}$
M
Training
$\mathrm{M}_{\mathcal{D}}$
$\mathcal{D}_t$
Training
$\mathrm{M}_{\mathcal{D}_t}$
$\mathcal{D}_{t'}$
Training
$\mathrm{M}_{\mathcal{D}_{t'}}$
Similarity between tasks is not strictly required (but the training must be changed accordingly)

Transfer Learning

is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

The easiest way to perform Transfer Learning is by *Fine-Tuning* the target model.

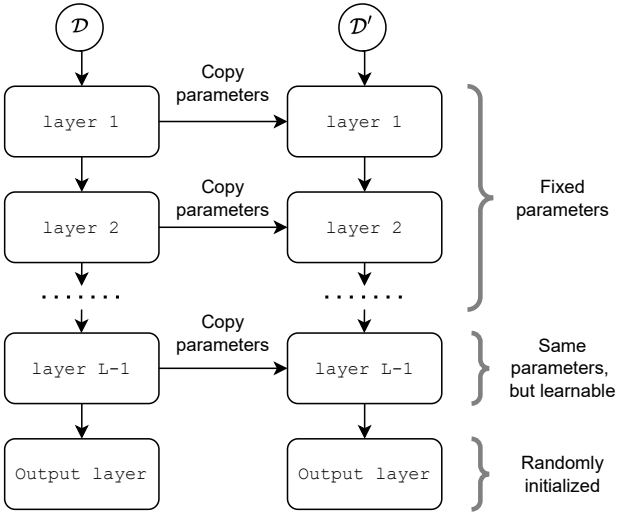
Transfer Learning

Fine-tuning

A model is composed of multiple layers, with the first ones containing low-level features, and the more you go into the model the higher the level of the extracted features.

When performing fine-tuning, we want to take advantage of the extraction ability of a pre-trained model. To do that, we usually substitute the last layer of the model to adapt it to our new task.

Then, only a subset of the layers is trained.


$\mathcal{D}$
$\mathcal{D}'$
Copy parameters
layer 1
layer 1
Copy parameters
layer 2
layer 2
Fixed parameters
Copy parameters
layer L-1
layer L-1
Same parameters, but learnable
Output layer
Output layer
Randomly initialized

1. `model = library.load_model('ResNet50', pretrained=True)`
2. `model.classifier = DenseLayer(in_size, num_classes)`
3. Train the model on $\mathcal{D}'$

Check this link for a complete implementation from Dive into Deep Learning.

This approach works because, usually, by looking at the loss surface of a well-trained model, we see that it relies on a good spot.

To stay as close as possible to this spot, we want to slowly fine-tune the model (e.g., use a low learning rate), possibly on a task similar to the one the model was originally trained on.

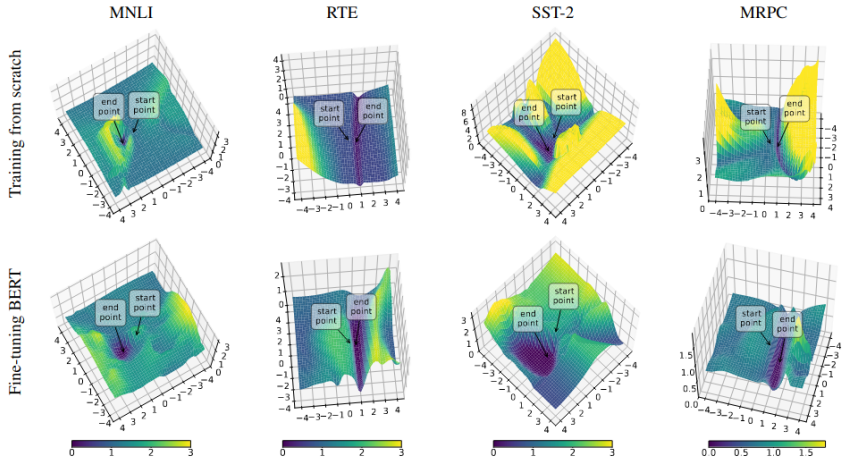


Figure 1: Training loss surfaces of training from scratch (top) and fine-tuning BERT (bottom) on four datasets. Pre-training leads to wider optima and eases optimization compared with random initialization [1].

Usually, to preserve the extraction capabilities of a learned model, we train only a subset of the last layers.

However, also the other layers can be trained, but different techniques must be implemented to avoid wiping out the learned knowledge.

For example, different learning rates can be assigned to each layer, with the first ones having a lower learning rate compared to the last ones to preserve the ability to extract lower-level features.

- Tasks must be similar to achieve the best performances.
- Moreover, a low learning rate must be used, to avoid messing up the model with high gradient steps.
- Fine tuning is easy to implement and manage, but it does not scale well with the model's size...

PEFT

Parameter Efficient Fine Tuning

Image to have a transformer-based Large Language Model, and you want to fine-tune it on a custom dataset.

Fine-tuning such models is indeed possible, even when having limited resources [2], but it is not convenient.

Also, consider deploying one LLM for each task you want to solve. Space and time are crucial in such situations, and we would like to optimize both.

Under these circumstances, fine-tuning is not the best road to take.

In the last years, many approaches designed to overcome such issues on LLM models have been proposed.

A transformer-based model contains many different modules, and, depending on which one we want to modify, we have different fine-tuning techniques.

In this lecture, we will see some of the most ”famous”.

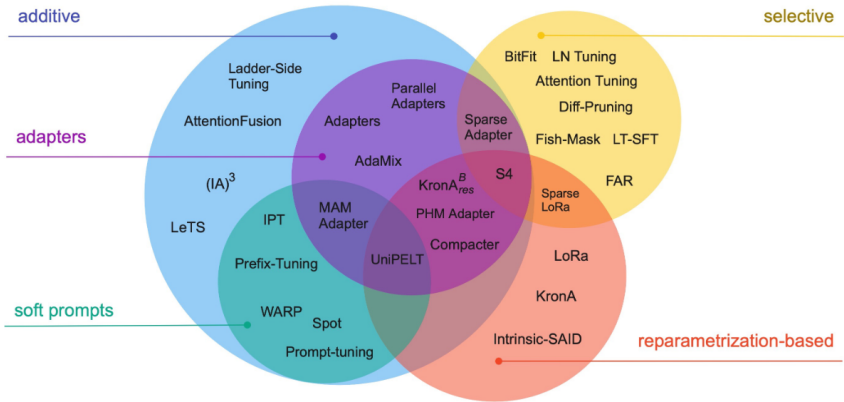


Figure 2: LLMs transfer learning techniques zoo. See [3] for a complete overview of such methods.

PEFT

Soft prompts

Prompting

is the process of prepending series of token P, to the input X, such that the model maximizes the likelihood of the correct Y: $P_\theta(Y|[P, X])$, while keeping the model parameters θ fixed.

We can take advantage of the zero-shot capabilities of an LLMs [4] to create a fine-tuning approach which trains only the input of such models, by creating a fine-tuned prompt stage that solves a given task.

The prompt can be manually tuned, but this is sub-optimal, because:

1. It requires a domain expert.
2. Prompts that are considered reasonable for a human are not necessarily the same for an LLM [5].
3. Pre-trained models are sensitive to the choice of prompts [6].

Prompt	P@1
[X] is located in [Y]. *(original)*	31.29
[X] is located in which country or state? [Y].	19.78
[X] is located in which country? [Y].	31.40
[X] is located in which country? In [Y].	51.08

Figure 3: Case study on LLaMA. A single-word change in the prompts could yield to a drastic difference.

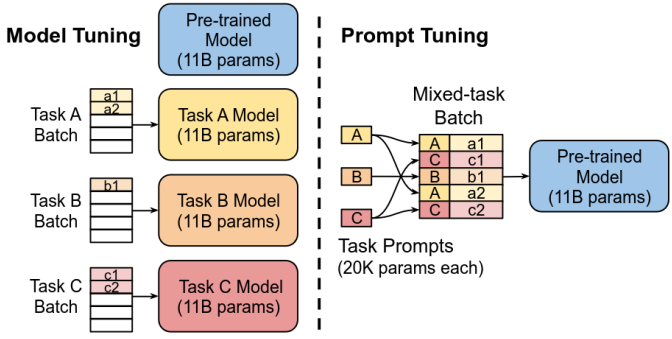
Multiple approaches have been proposed to overcome the limits of discrete and manual tuning approaches. We will see:

1. Prompt Tuning [7]
2. Prefix Tuning [8]

In contrast to prompt design approaches, which aim to select the best input tokes from a fixed set having pre-trained parameters, Prompt Tuning adds a new set of trainable parameters θ_P to the input of the model.

The new likelihood is $P_{\theta,\theta_P}(Y \mid [P, X])$, and it is maximized using standard training approaches, where only the parameters θ_P are trained.

It allows a batch to contain samples from multiple tasks, improving the parallelism.


Model Tuning
Pre-trained Model (11B params)
Task A Batch
a1
a2
Task A Model (11B params)
Task B Batch
b1
Task B Model (11B params)
Task C Batch
c1
c2
Task C Model (11B params)
Prompt Tuning
Mixed-task Batch
A
B
C
A a1
C c1
B b1
A a2
C c2
Pre-trained Model (11B params)
Task Prompts (20K params each)

.

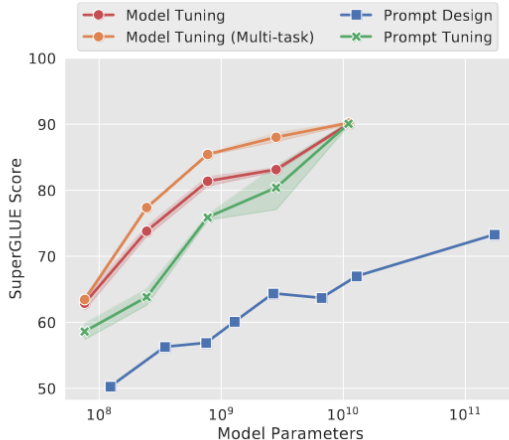
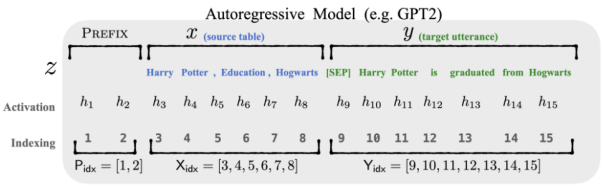


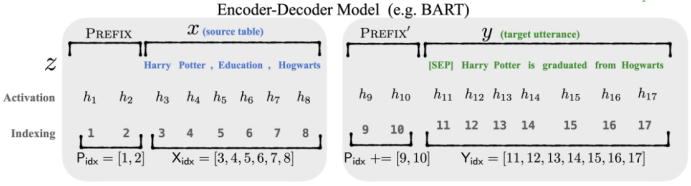
Figure 4: Prompt tuning matches the quality of model tuning as size increases while enabling the reuse of a single frozen model for all tasks.

The set of tokens P can be seen as the context of X that helps the model steer towards the correct output Y. However, there are no guarantees that such a set exists.

Instead of optimising over discrete tokens, the approach optimizes over the instruction as continuous word embeddings, whose effects will be propagated upward to all Transformer activation layers and rightward to subsequent tokens.

These prefixes are added to each transformer layer.





Summarization Example

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are.They say the confusion may lie in the way the brain receives information from different parts of the body.Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

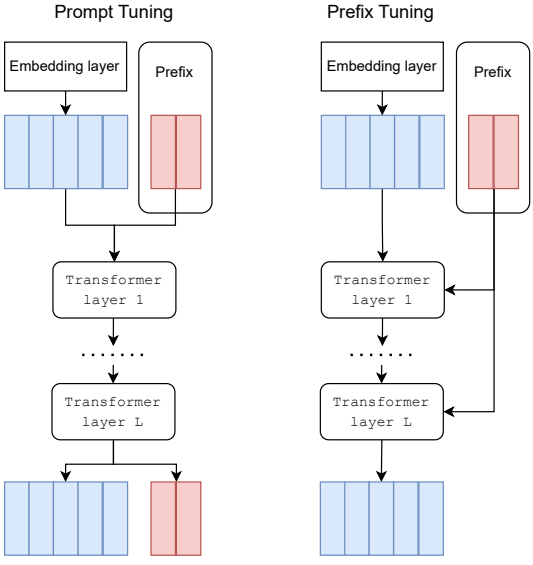
Summary: The brain naturally distorts body image - a finding which could explain eating disorders like anorexia, say experts.

Table-to-text Example

Table: name[Clowns] customer-rating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

Soft prompts: Prompt Tuning vs Prefix Tuning



- We have seen that achieving a fine-tuned model by working on the input level is possible.
- These approaches are fast and easy to implement, even if limited since the overall model is unchanged.
- The intrinsic limitation of such approaches does not guarantee that an optimal fine-tuned model can be achieved.

PEFT

Adapters-based tuning

Adapters-based methods inject small-scale neural modules (adapters) into transformer layers, and only the parameters of such models are trained.

Although such a strategy leaves an open choice of adapter structures, a simple instantiation achieves impressive performance and has become the most widely used baseline in recent research.

One adapter module contains, usually, a down-up projection structure which further reduces the number of trainable parameters.

Originally introduced in [9], it is structured as following:

- An **Adapter** is a module added to a pre-trained model.
- The original model's weights are fixed, while the adapters are tuned.
- Usually, the adapters are initialized so that, before training, the output of the adapted model resembles the one from the pre-trained one.

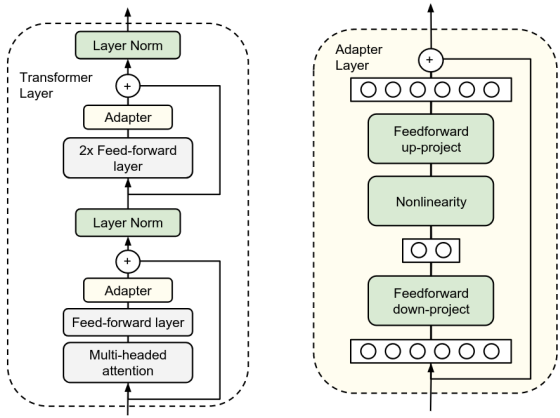


Figure 5: The Adapter module and where it is placed within the model.

The pros of this approach are:

- it is very effective in multi-task settings.
- it is faster than fine-tuning the whole model.
- multiple adapters can be easily combined [10].
- it can be used to build models which are more robust if compared to the full-tuning approach [11].

But it has also some drawbacks:

- adding new layers could make the inference slower.
- it makes the model larger, increasing the necessity for bigger GPUs.
- multiple Adapters must be processed sequentially, breaking possible parallelism.

Approaches that further reduce the training parameters have also been proposed. For example, a Compacter [12] is an adapter that aims to make the technique more efficient by reducing the number of parameters.

Adapters are almost fully connected linear layers, while Compacters replace such layers with a parametrized hypercomplex multiplication layer.

Reducing the number of parameters is not easy, and multiple ”tricks” are involved, making the approaches less stable.

- We have seen that components can be attached to the trained model.
- With adapters, such components are placed inside transformer layers and could make the inference slower, as well as break parallelism.
- Adapters can be easily combined, but it significantly increases the number of parameters.

PEFT

Re-Parameterizing a Model

To fine-tune a model is not necessary to change all the parameters, but only a subset of those. This subset, once fine-tuned, leads to a satisfactory performance, while keeping the rest of the model as it was.

This means we can reparameterize a subset of the original model parameters with low-dimensional proxy parameters, and just optimize the proxy.

Suppose to have a model with a set of parameters θ with cardinality $D = |\theta|$.

Instead of optimizing all of them, we want to optimize only a subset θ_s with $|\theta| \ll D$.

This is done by, usually, applying a factorization on the parameters of the model [13].

Many approaches are based on this idea. Here, we will see:

- LoRa [14]
- IA3 [15]

- The idea is based on empirical observations that LLM weights are of low “intrinsic rank.”
- While the tensors of LLMs are high-dimensional, the information encoded in them tends to be well-approximated in a much lower dimension.
- This manifests itself, for example, in a precipitous drop-off of the singular values calculated using the singular-value decomposition of the model weights.

Consider a weight matrix $W_0 \in \mathrm{R}^{d \times k}$. The hypothesis tells us this matrix is low-rank, and so fine-tuning it might also be.

During training, LoRA freezes the original weights, and calculates low-rank matrices so that:

$$W_0 + \Delta W = W_0 + BA$$

where $B \in \mathrm{R}^{d \times r}$ and $A \in \mathrm{R}^{r \times k}$, with $r > 0$ and also $r \ll \min(d, k)$. If training updates are approximately low-rank, the weights A and B should closely approximate the true update.

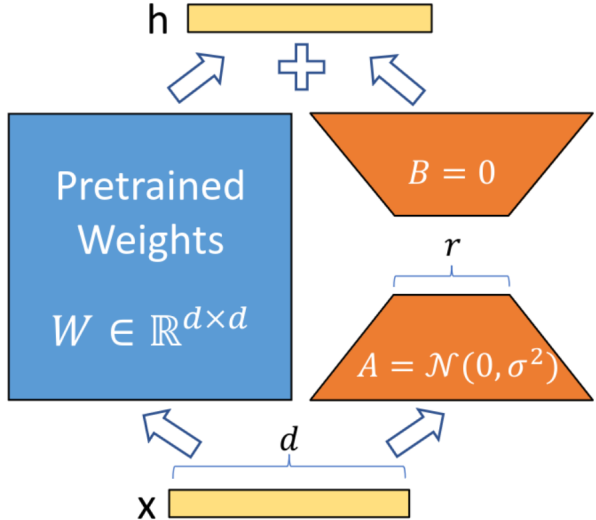


Figure 6: The low-rank parametrization proposed by LoRA.

The update matrix ΔW can be easily added to a deployed model. It can be just added to the model before deploying it, resulting in a fine-tuned model with no additional parameters.

Moreover, multiple ΔW_t, one for each task, can be saved with a small memory overhead, and attached (addition) or detached (subtraction) when needed.

But where to apply such matrices? And what about the size of r?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Figure 7: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters

it was proposed as an improvement of LoRA, and the main idea is to rescale inner activations with low-dimensional learned vectors, which are injected into the module.

It has three goals:

- must allow for mixed-task batches.
- should achieve strong accuracy after training on only a few examples of new tasks.
- must add or update as few parameters as possible to avoid incurring storage and memory costs.

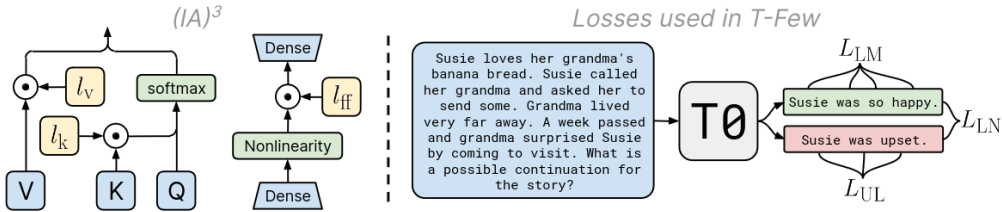


Figure 8: Left) multiple learnable vectors are added and used to scale various portion of the model, right) the new loss preventing similarity between different tasks.

Model arithmetic

Merging models

Merging operations between neural networks can be seen as choosing parameters that approximately maximize the joint likelihood of the posteriors of the models' parameters.

Averaging the parameters of multiple models can be seen as Isotropic merging, where the posterior of each model is approximated with an isotropic Gaussian distribution.

- A model can be fine-tuned even without adding new layers.
- Operating directly on the weights reduces the parameters needed and usually also the training time.
- Depending on the approach, we can have or not parallelism and multi-task batches.

However, this technique does not always lead to good results because there are no guarantees that multiple models can be correctly merged.

In this section we will see two methods:

1. Fisher-Weighted Averaging [16]
2. Git Re-basin [17]

Proposed in [16], this approach aims to find the best parameters by estimating their importance to the overall loss.

For each parameter, a Fisher importance value is estimated, and two models are merged using a weighted sum of such values.

This leads to better results when compared to fine-tuning and ensembling while requiring less memory and training time.

For each parameter, a Fisher importance value is estimated. Given multiple models M, these are merged using a weighted sum of such values:

$$\overline{\theta}^j = \frac{\sum_i^M \lambda_i F_i^j \theta_i^j}{\sum_i^M \lambda_i F_i^j}$$

where $\overline{\theta}^j$ is the new parameter j, and the Fisher value for the parameter i is F^j:

$$F^j = \frac{1}{N} \sum_i^N \mathbb{E}(\nabla_{\theta_i} \log p(y|x_i))^2$$

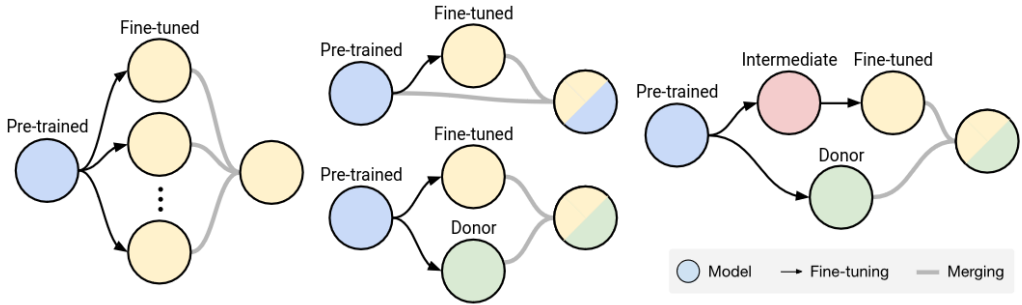


Figure 9: Left: Merging many fine-tuned models as a form of ensembling. Center: "Robust fine-tuning" [18] (top) and donor task merging [19, 20]. Right: Merging an intermediate-task trained model with a donor model

Based on the idea that a loss landscape often contains only one basin after taking into account all the possible symmetries, Git Re-Basin, proposed in [17], combines multiple models linearly after a selected permutation alignment, also proposed in the paper.

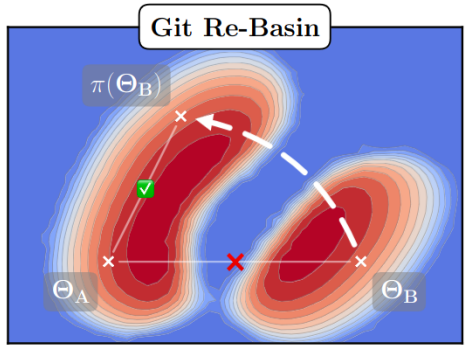


Figure 10: Git Re-Basin merges models by moving solutions into a single basin. Θ_B is permuted into a functionally equivalent $\pi(\Theta_B)$ so that it lies in the same basin of Θ_A.

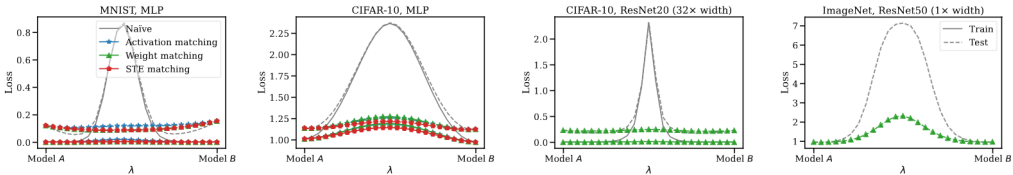


Figure 11: Loss landscapes when interpolating between models trained on MNIST, CIFAR-10, and ImageNet. The image shows that a linear mode connectivity is possible after permuting.

Model arithmetic

Task arithmetic

Proposed in [21], task arithmetic is a new paradigm for adapting neural networks to new tasks.

This paradigm is based on task vectors, defined as a direction in the weight space of a pre-trained model towards a space region in which results on a task are improved.

Such vectors, once obtained, can be added or subtracted, leading to different results.

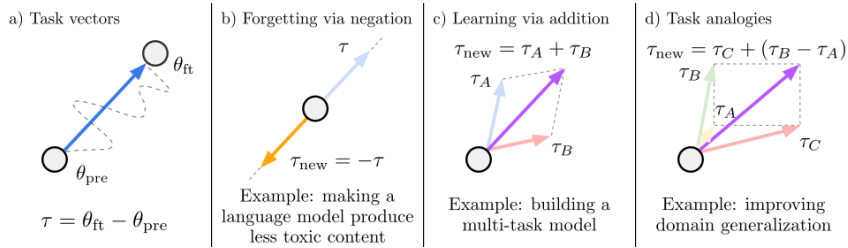


Figure 12: First, we calculate the vector as the distance between the pre-trained model θ_{pre} and the fine-tuned version θ_{ft}, then we can use such vector to modify a model.

However, there are no guarantees that two task vectors will push the model in a direction in which both are satisfied.

In fact, could happen that the same parameter tries to go in two different directions for different tasks.

Which one should be taken as the correct direction?

Task vectors work well and are easy to implement and manage, but modify all the parameters.

Moreover, it requires fine-tuning the whole model on the current task.

More sophisticated ways of merging the models have been proposed, such as in [22, 23].

Conclusion

In this lecture, we have seen how pre-trained models can be adapted to our tasks.

Most of the approaches are general and can be adapted to work for CNNs, even if these approaches make more sense when using very big models (such as LLMs).

Some resources that help us in fine-tuning LLMs are:

1. OpenAI fine-tuning API (probably based on one of the approaches you have seen in this lecture)
2. HuggingFace PEFT Library
3. The just released "Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning" [24].

THE END

Questions?

[1] Yaru Hao et al. **“Visualizing and understanding the effectiveness of BERT”.** In: *arXiv preprint arXiv:1908.05620* (2019).

[2] Kai Lv et al. **“Full Parameter Fine-tuning for Large Language Models with Limited Resources”.** In: *arXiv preprint arXiv:2306.09782* (2023).

[3] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. ***Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning.*** 2023. arXiv: 2303.15647 [cs.CL].

[4] Tom Brown et al. **“Language Models are Few-Shot Learners”.** In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[5] Shaohua Wu et al. **“Yuan 1.0: Large-scale pre-trained language model in zero-shot and few-shot learning”.** In: *arXiv preprint arXiv:2110.04725* (2021).

[6] Xiao Liu et al. **“GPT understands, too”.** In: *AI Open* (2023).

[7] Brian Lester, Rami Al-Rfou, and Noah Constant. **“The power of scale for parameter-efficient prompt tuning”.** In: *arXiv preprint arXiv:2104.08691* (2021).

[8] Xiang Lisa Li and Percy Liang. **“Prefix-tuning: Optimizing continuous prompts for generation”.** In: *arXiv preprint arXiv:2101.00190* (2021).

[9] Neil Houlsby et al. **“Parameter-efficient transfer learning for NLP”.** In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799.

[10] Yaqing Wang et al. **“AdaMix: Mixture-of-adaptations for parameter-efficient model tuning”.** In: *arXiv preprint arXiv:2210.17451* (2022).

[11] Wenjuan Han, Bo Pang, and Yingnian Wu. **“Robust transfer learning with pretrained language models through adapters”.** In: *arXiv preprint arXiv:2108.02340* (2021).

[12] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. **“Compacter: Efficient low-rank hypercomplex adapter layers”.** In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 1022–1035.

[13] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. **"Intrinsic dimensionality explains the effectiveness of language model fine-tuning".** In: *arXiv preprint arXiv:2012.13255* (2020).

[14] Edward J Hu et al. **"Lora: Low-rank adaptation of large language models".** In: *arXiv preprint arXiv:2106.09685* (2021).

[15] Chunyuan Li et al. **"Elevater: A benchmark and toolkit for evaluating language-augmented visual models".** In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 9287–9301.

[16] Michael S Matena and Colin A Raffel. **“Merging models with fisher-weighted averaging”.** In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17703–17716.

[17] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. **“Git re-basin: Merging models modulo permutation symmetries”.** In: *arXiv preprint arXiv:2209.04836* (2022).

[18] Mitchell Wortsman et al. **“Robust fine-tuning of zero-shot models”.** In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 7959–7971.

[19] Jason Phang, Thibault Févry, and Samuel R Bowman. **“Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks”.** In: *arXiv preprint arXiv:1811.01088* (2018).

[20] Yada Pruksachatkun et al. **“Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work?”** In: *arXiv preprint arXiv:2005.00628* (2020).

[21] Gabriel Ilharco et al. **“Editing models with task arithmetic”.** In: *arXiv preprint arXiv:2212.04089* (2022).

[22] Enneng Yang et al. **“AdaMerging: Adaptive Model Merging for Multi-Task Learning”.** In: *ArXiv* abs/2310.02575 (2023). URL: https://api.semanticscholar.org/CorpusID:263620126.

[23] Prateek Yadav et al. **“Resolving Interference When Merging Models”.** In: *arXiv preprint arXiv:2306.01708* (2023).

[24] Clifton Poth et al. **“Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning”.** In: *arXiv preprint arXiv:2311.11077* (2023).