Infrastructure and Progress
Towards the First Community-Built and Continuously-Improved Model

Colin Raffel
Collaborative, Communal, Continual {CCC} ML

- Where are we in terms of making CCCML possible?
- We can build a useful VCS for CCCML now.
- We’re building a git-based VCS for CCCML.
- We’re building a CCC model to test it out.
from https://dvc.org/
How can we enable collaborative and continual development of machine learning models?

We need to be able to cheaply communicate patches and merge updates from different contributors.
How can we enable collaborative and continual development of machine learning models?

We need to be able to cheaply communicate patches and merge updates from different contributors.
From Training Neural Networks with Fixed Sparse Masks by Sung et al.
From Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning by Liu et al.
\[ h_2^l = \text{Dropout}\left( W_{m_1}^l \cdot h_1^l + b_{m_1}^l \right) \] (1)

\[ h_3^l = g_{LN_1}^l \odot \frac{(h_2^l + x) - \mu}{\sigma} + b_{LN_1}^l \] (2)

\[ h_4^l = \text{GELU}\left( W_{m_2}^l \cdot h_3^l + b_{m_2}^l \right) \] (3)

\[ h_5^l = \text{Dropout}\left( W_{m_3}^l \cdot h_4^l + b_{m_3}^l \right) \] (4)

\[ \text{out}^l = g_{LN_2}^l \odot \frac{(h_5^l + h_3^l) - \mu}{\sigma} + b_{LN_2}^l \] (5)

From BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models by Ben Zaken et al.
From LoRA: Low-Rank Adaptation of Large Language Models by Hu et al.
From Measuring the Intrinsic Dimension of Objective Landscapes by Li et al.
Algorithm 1: Sparse Ternary Compression (STC)

1. **input:** flattened tensor $T \in \mathbb{R}^n$, sparsity $p$
2. **output:** sparse ternary tensor $T^* \in \{-\mu, 0, \mu\}^n$
3. $k \leftarrow \max(np, 1)$
4. $v \leftarrow \text{top}_k(|T|)$
5. mask $\leftarrow (|T| \geq v) \in \{0, 1\}^n$
6. $T^{\text{masked}} \leftarrow \text{mask} \odot T$
7. $\mu \leftarrow \frac{1}{k} \sum_{i=1}^{n} |T_i^{\text{masked}}|$
8. return $T^* \leftarrow \mu \times \text{sign}(T^{\text{masked}})$

From Robust and Communication-Efficient Federated Learning from Non-IID Data by Sattler et al.
How can we enable collaborative and continual development of machine learning models?

We need to be able to cheaply communicate patches and merge updates from different contributors.
From *Merging Models with Fisher-Weighted Averaging* by Matena and Raffel
From Robust and Communication-Efficient Federated Learning from Non-IID Data by Sattler et al.
Step 1: **branch** from existing experts, or seed LM

Step 2: branched **training** on $k$ domains in parallel

Step 3: **merge** $k$ domain expert LMs into ELMforest

repeat with another batch of domains

From Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models by Li et al.
From *Patching open-vocabulary models by interpolating weights* by Ilharco et al.
From *Fusing finetuned models for better pretraining* by Choshen et al.
\[ \arg \max_{\theta} \sum_{i=1}^{M} \lambda_i \log \mathcal{N}(\theta | \theta_i, \hat{F}_i^{-1}) \]

\[ \theta^*(j) = \frac{\sum_{i=1}^{M} \lambda_i \hat{F}_i^{(j)} \theta_i^{(j)}}{\sum_{i=1}^{M} \lambda_i \hat{F}_i^{(j)}} \]

From *Merging Models with Fisher-Weighted Averaging* by Matena and Raffel
From *Git Re-Basin: Merging Models modulo Permutation Symmetries* by Ainsworth et al.
a) Task vectors

\[ \tau = \theta_{ft} - \theta_{pre} \]

Example: making a language model produce less toxic content

b) Forgetting via negation

\[ \tau_{new} = -\tau \]

Example: building a multi-task model
d) Task analogies

\[ \tau_{new} = \tau_Q + (\tau_M - \tau_W) \]

Example: learning about kings with data from queens, man and woman

From Editing Models with Task Arithmetic by Ilharco et al.
How can we enable collaborative and continual development of machine learning models?

We need a system to track changes and manage contributions to a model.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.

- **An on-disk format for changes.** This format should only store the information required to reconstruct the change and should support all update types.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.

- **An on-disk format for changes.** This format should only store the information required to reconstruct the change and should support all update types.

- **Restoring the previous state of a checkpoint.** The system should be able to rapidly restore the state of the model at a certain point of history.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.

- **An on-disk format for changes.** This format should only store the information required to reconstruct the change and should support all update types.

- **Restoring the previous state of a checkpoint.** The system should be able to rapidly restore the state of the model at a certain point of history.

- **Communicate/receive history.** There should be a way for me to communicate my local history to a model to someone/someplace else, as well as bring in the changes to the model someone else has proposed.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.

- **An on-disk format for changes.** This format should only store the information required to reconstruct the change and should support all update types.

- **Restoring the previous state of a checkpoint.** The system should be able to rapidly restore the state of the model at a certain point of history.

- **Communicate/receive history.** There should be a way for me to communicate my local history to a model to someone/someplace else, as well as bring in the changes to the model someone else has proposed.

- **Identifying and possibly resolving merge conflicts.** When there are conflicting changes created in parallel, the system should fail or (ideally, eventually) try to resolve them.
What we need(ed)

- **Track changes to a checkpoint.** There are many ways to modify a checkpoint, e.g. updating all parameters, updating a subset of parameters, adding/removing parameters, etc. The system should support all of these operations.

- **An on-disk format for changes.** This format should only store the information required to reconstruct the change and should support all update types.

- **Restoring the previous state of a checkpoint.** The system should be able to rapidly restore the state of the model at a certain point of history.

- **Communicate/receive history.** There should be a way for me to communicate my local history to a model to someone/someplace else, as well as bring in the changes to the model someone else has proposed.

- **Identifying and possibly resolving merge conflicts.** When there are conflicting changes created in parallel, the system should fail or (ideally, eventually) try to resolve them.

- **Easy to go from training → version control system.** We don’t expect contributors to directly create patches; we expect them to use version control-aware training code that needs to interact with the system to create patches, etc.
How can we enable collaborative and continual development of machine learning models?

Let’s try it by focusing on a specific architecture.
Earlier layers...

- Machine translation
- Question answering
- Document summarization
- Paraphrase detection
- Sentiment analysis
- Topic classification

Later layers...
Earlier layers... → Later layers...

Machine translation → Question answering → Document summarization → Paraphrase detection → Sentiment analysis → Topic classification

Earlier layers...
Machine translation

Question answering

Document summarization

Paraphrase detection

Sentiment analysis

Topic classification

Router

Earlier layers...

Later layers...
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.

- **Most version control operations are trivial.** People can add or update specific adapters, which results in cheap-to-communicate updates that are trivial to merge. Testing would be easy too - just evaluate performance on the task for the updated adapter. Updating/adding adapters is probably how most people would contribute.

- **Could occasionally update the router or backbone.** These can be seen as minor/major version updates…

- **Could consider adapters beyond task-level.** For example, we could say each task has some particular domain and language, and have separate domain and language adapter sets.

- **Some precedent in AdapterHub.** This model could be seen as a next-generation version; here, adapters ship with the model, routing can be learned, and there is a more principled way of tracking changes.
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.

- **Most version control operations are trivial.** People can add or update specific adapters, which results in cheap-to-communicate updates that are trivial to merge. Testing would be easy too - just evaluate performance on the task for the updated adapter. Updating/adding adapters is probably how most people would contribute.
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.

- **Most version control operations are trivial.** People can add or update specific adapters, which results in cheap-to-communicate updates that are trivial to merge. Testing would be easy too - just evaluate performance on the task for the updated adapter. Updating/adding adapters is probably how most people would contribute.

- **Could occasionally update the router or backbone.** These can be seen as minor/major version updates...
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.

- **Most version control operations are trivial.** People can add or update specific adapters, which results in cheap-to-communicate updates that are trivial to merge. Testing would be easy too - just evaluate performance on the task for the updated adapter. Updating/adding adapters is probably how most people would contribute.

- **Could occasionally update the router or backbone.** These can be seen as minor/major version updates...

- **Could consider adapters beyond task-level.** For example, we could say each task has some particular domain and language, and have separate domain and language adapter sets.
Why I think this would be a good model

- **It would probably work.** Parameter-efficient fine-tuning methods work well. We can probably get the learned routing to work reasonably well.

- **It would be useful.** The model would be able to perform tons of tasks out-of-the-box. This would make it convenient and compelling to use.

- **Most version control operations are trivial.** People can add or update specific adapters, which results in cheap-to-communicate updates that are trivial to merge. Testing would be easy too - just evaluate performance on the task for the updated adapter. Updating/adding adapters is probably how most people would contribute.

- **Could occasionally update the router or backbone.** These can be seen as minor/major version updates...

- **Could consider adapters beyond task-level.** For example, we could say each task has some particular domain and language, and have separate domain and language adapter sets.

- **Some precedent in AdapterHub.** This model could be seen as a next-generation version; here, adapters ship with the model, routing can be learned, and there is a more principled way of tracking changes.
Join us!


Give me feedback: