Learning to Learn

automl.org/events -> AutoML Tutorial -> Slides

Frank Hutter
University of Freiburg
fh@cs.uni-freiburg.de

Joaquin Vanschoren
Eindhoven University of Technology
j.vanschoren@tue.nl
@joavanschoren
Learning is a never-ending process

Tasks come and go, but learning is forever

Learn more effectively: less trial-and-error, less data
Learning to learn

**Inductive bias:** all assumptions added to the training data to learn effectively.

If prior tasks are *similar*, we can *transfer* prior knowledge to new tasks

(if not it may actually harm learning)
Meta-learning

Collect meta-data about learning episodes and learn from them

Meta-learner learns a (base-)learning algorithm, end-to-end
Three approaches for increasingly similar tasks

1. Transfer prior knowledge about what generally works well
2. Reason about model performance across tasks
3. Start from models trained earlier on similar tasks
1. Learning from prior evaluations

Configurations: settings that **uniquely define the model**
(algorithm, pipeline, neural architecture, hyper-parameters, …)

![Diagram showing the learning process from prior evaluations](https://via.placeholder.com/150)
Top-K recommendation

- Build a global (multi-objective) ranking, recommend the top-K
- Requires fixed selection of candidate configurations (portfolio)
- Can be used as a warm start for optimization techniques

Leite et al. 2012
Abdulrahman et al. 2018
Warm-starting with plugin estimators

- What if prior configurations are not optimal?
- Per task, fit a differentiable plugin estimator on all evaluated configurations
- Do gradient descent to find optimized configurations, recommend those

\[
\begin{align*}
&\text{Tasks} \\
\downarrow \\
&\text{Learning} \\
\downarrow \\
&\text{Models} \\
\downarrow \\
&\text{performance} \\
\rightarrow & \quad \lambda_i \\
\rightarrow & \quad P_{i,j}
\end{align*}
\]

per task:

- task 1: \( \lambda^*_1 \)
- task 2: \( \lambda^*_2 \)
- task 3: \( \lambda^*_3 \)
- …

New Task

\[
\rightarrow \quad \lambda^*_i \rightarrow \quad \text{meta-learner} \\
\rightarrow \quad \text{Models} \\
\downarrow \\
\text{performance}
\]

Wistuba et al. 2015
Configuration space design

- **Functional ANOVA**: select hyperparameters that cause variance in the evaluations\(^1\)
- **Tunability**: improvement from tuning a hyperparameter vs. using a good default\(^2\)
- **Search space pruning**: exclude regions yielding bad performance on similar tasks\(^3\)

\(^1\) van Rijn & Hutter 2018
\(^2\) Probst et al. 2018
\(^3\) Wistuba et al. 2015
Active testing

- **Tasks are similar** if observed relative performance of configurations is similar
- Tournament-style selection, warm-start with overall best configurations $\lambda_{best}$
- Next candidate $\lambda_c$: the one that beats current $\lambda_{best}$ on similar tasks (from portfolio)

Relative landmark on $\lambda_a, \lambda_b, \text{task } t_j$:

$RL_{a,b,j} = P_{a,j} - P_{b,j}$

Update:

$Sim(t_j, t_{new}) = Corr([RL_{a,b,j}], [RL_{a,b,new}])$

Select $\lambda_c >_{RL} \lambda_{best}$ on similar tasks
Bayesian optimization (refresh)

- Learns how to learn within a single task (short-term memory)
- Surrogate model: *probabilistic* regression model of configuration performance
- Can we transfer what we learned to *new* tasks (long term memory)?
Surrogate model transfer

- If task $j$ is *similar* to the new task, its surrogate model $S_j$ will do well.
- Sum up all $S_j$ predictions, weighted by task similarity (relative landmarks)\(^1\).
- Build combined Gaussian process, weighted by current performance on new task\(^2\).

\[ S = \sum w_j S_j \]

---

1 Wistuba et al. 2018
2 Feurer et al. 2018
Warm-started multi-task learning

- Bayesian linear regression (BLR) surrogate model on every task
- Learn a suitable basis expansion $\phi_z(\lambda)$, joint representation for all tasks
- Scales linearly in # observations, transfers info on configuration space

Tasks

Learning $\rightarrow \lambda_i$ $\rightarrow P_{i,j}$

Models $\rightarrow \lambda_i$ $\rightarrow P_{i,j}$

New Task $\rightarrow \lambda_i$

meta-learner

Models $\rightarrow \lambda_i$ $\rightarrow P_{i,j}$

performance

BLR surrogate $\phi_z(\lambda)_i$

Bayesian optimization

warm-start (pre-train) $(\lambda_i, P_{i,j}) \rightarrow \phi_z(\lambda)$

BLR hyperparameters

Perrone et al. 2018
Multi-task Bayesian optimization

- **Multi-task Gaussian processes**: train surrogate model on \( t \) tasks simultaneously\(^1\)
  - If tasks are similar: transfers useful info
  - Not very scalable

- **Bayesian Neural Networks** as surrogate model\(^2\)
  - Multi-task, more scalable

- **Stacking** Gaussian Process regressors (Google Vizier)\(^3\)
  - Sequential tasks, each similar to the previous one
  - Transfers a prior based on residuals of previous GP

\(^1\) Swersky et al. 2013
\(^2\) Springenberg et al. 2016
\(^3\) Golovin et al. 2017
Other techniques

- Transfer learning with multi-armed bandits
  - View every task as an arm, learn to `pull` observations from the most similar tasks
  - Reward: accuracy of configurations recommended based on these observations
- Transfer learning curves
  - Learn a partial learning curve on a new task, find best matching earlier curves
  - Predict the most promising configurations based on earlier curves

1 Ramachandran et al. 2018
2 Leite et al. 2005
3 van Rijn et al. 2015
2. Reason about model performance across tasks

*Meta-features: measurable properties of the tasks*
(number of instances and features, class imbalance, feature skewness,...)

![Diagram showing the process of reasoning about model performance across tasks.](image)
Meta-features

- **Hand-crafted (interpretable) meta-features**¹
  - **Number of** instances, features, classes, missing values, outliers,…
  - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,…
  - **Information-theoretic:** class entropy, mutual information, noise-signal ratio,…
  - **Model-based:** properties of simple models trained on the task
  - **Landmarkers:** performance of fast algorithms trained on the task
  - **Domain specific task properties**

- **Learning a joint task representation**
  - Deep metric learning: learn a representation $h^{mf}$ using a ground truth distance²
  - With Siamese Network:
    - Similar task, similar representation

¹ Vanschoren 2018  
² Kim et al. 2017
Warm-starting from similar tasks

- Find k most similar tasks, warm-start search with best $\theta_i$
  - Genetic hyperparameter search $^1$
  - Auto-sklearn: Bayesian optimization (SMAC) $^2$
    - Scales well to high-dimensional configuration spaces

2. Feurer et al. 2015
Warm-starting from similar tasks

- Collaborative filtering: configurations $\lambda_i$ are `rated’ by tasks $t_j$
  - Probabilistic matrix factorization
    - Learns a latent representation for tasks and configurations
    - Returns probabilistic predictions for Bayesian optimization
    - Use meta-features to warm-start on new task

\[ P_{ij} \rightarrow \lambda_i \rightarrow m_j \rightarrow \text{New Task} \]

\[ \lambda_{1..k} \rightarrow \text{meta-learner} \rightarrow \text{performance} \]

\[ \text{performance} \rightarrow P_{ij} \rightarrow \text{tasks} \rightarrow \text{learning} \rightarrow \text{models} \]

Fusi et al. 2017

$t_{new}$ warm-started with $\lambda_{1..k}$
Global surrogate models

- Train a task-independent surrogate model with meta-features in inputs
  - SCOT: Predict ranking of $\lambda_i$ with surrogate ranking model + $m_j$. ¹
  - Predict $P_{i,j}$ with multilayer Perceptron surrogates + $m_j$. ²
  - Build joint GP surrogate model on most similar $(\|\mathbf{m}_i - \mathbf{m}_j\|_2)$ tasks. ³
- Scalability is often an issue

¹ Bardenet et al. 2013
² Schilling et al. 2015
³ Yogatama et al. 2014
Meta-models

- \textit{Learn} direct mapping between meta-features and $P_{i,j}$
  - Zero-shot meta-models: predict best $\lambda_i$ given meta-features \footnote{Brazdil et al. 2009, Lemke et al. 2015}
    
    \hspace{1cm} $m_j \xrightarrow{\text{meta-learner}} \lambda_{best}$

- Ranking models: return ranking $\lambda_{1..k}$ \footnote{Sun and Pfahringer 2013, Pinto et al. 2017}

    \hspace{1cm} $m_j \xrightarrow{\text{meta-learner}} \lambda_{1..k}$

- Predict which algorithms / configurations to consider / tune \footnote{Sanders and C. Giraud-Carrier 2017}

    \hspace{1cm} $m_j \xrightarrow{\text{meta-learner}} \Lambda$

- Predict performance / runtime for given $\Theta_i$ and task \footnote{Yang et al. 2018}

    \hspace{1cm} $m_j, \lambda_i \xrightarrow{\text{meta-learner}} P_{ij}$

- Can be integrated in larger AutoML systems: warm start, guide search,…
Learning Pipelines

- **Compositionality**: the learning process can be broken down into smaller tasks
  - Easier to learn, more transferable, more robust
- Pipelines are one way of doing this, but how to control the search space?
  - Select a fixed set of possible pipelines. Often works well (less overfitting) \(^1\)
  - Impose a fixed structure on the pipeline \(^2\)
  - (Hierarchical) Task Planning \(^3\)
    - Break down into smaller tasks
- Meta-learning:
  - Mostly warm-starting

\(^1\) Fusi et al. 2017
\(^2\) Feurer et al. 2015
\(^3\) Mohr et al. 2018
Evolving pipelines

- Start from simple pipelines
- Evolve more complex ones if needed
- Reuse pipelines that do specific things

Mechanisms:
- Cross-over: reuse partial pipelines
- Mutation: change structure, tuning

Approaches:
- TPOT: Tree-based pipelines\(^1\)
- GAMA: asynchronous evolution\(^2\)
- RECIPE: grammar-based\(^3\)

Meta-learning:
- Largely unexplored
- Warm-starting, meta-models
Learning to learn through self-play

- Build pipelines by selecting among actions
  - insert, delete, replace pipeline parts
- Neural network (LSTM) receives task meta-features, pipelines and evaluations
  - Predict pipeline performance and action probabilities
- Monte Carlo Tree Search builds pipelines based on probabilities
  - Runs multiple simulations to search for a better pipeline
3. Learning from trained models

Models trained on similar tasks
(model parameters, features, ...)

Task 1 \[\rightarrow\] Task N \[\rightarrow\] New Task

\[\downarrow\] \[\downarrow\] \[\downarrow\]

Learning \[\rightarrow\] Learning \[\rightarrow\] meta-learner

\[\downarrow\] \[\downarrow\] \[\downarrow\]

Models \[\rightarrow\] Models \[\rightarrow\] Models

performance \[\rightarrow\] performance \[\rightarrow\] performance

\[\downarrow\] \[\downarrow\] \[\downarrow\]

configurations \[\rightarrow\] model parameters \[\rightarrow\] performances

\[\lambda_i\] \[\theta_k\] \[\pi_{i,j}\]

intrinsically (very) similar (e.g. shared representation)
Transfer Learning

- Select source tasks, transfer trained models to similar target task
  
- Use as starting point for tuning, or freeze certain aspects (e.g. structure)
  - Bayesian networks: start structure search from prior model
  - Reinforcement learning: start policy search from prior policy

---

1 Thrun and Pratt 1998
2 Niculescu-Mizil and Caruana 2005
3 Taylor and Stone 2009
Transfer features, initializations

- For neural networks, both structure and weights can be transferred.
- Features and initializations learned from:
  - Large image datasets (e.g. ImageNet) \(^1\)
  - Large text corpora (e.g. Wikipedia) \(^2\)
- Fails if tasks are not similar enough \(^3\)

### Feature extraction:
- remove last layers, use output as features
  - if task is quite different, remove more layers

### End-to-end tuning:
- train from initialized weights

### Fine-tuning:
- unfreeze last layers, tune on new task

---

\(^1\) Razavian et al. 2014
\(^2\) Mikolov et al. 2013
\(^3\) Yosinski et al. 2014
Learning to learn by gradient descent

- Our brains probably don’t do backprop, replace it with:
  - Simple parametric (bio-inspired) rule to update weights \(^1\)
  - Single-layer neural network to learn weight updates \(^2\)
- Learn parameters across tasks, by gradient descent (meta-gradient)

\[
\Delta \theta_i = \epsilon \cdot y_{pre(i)} \cdot k
\]

\[
\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}
\]

\(\Delta \theta_i\) represents the change in parameter \(\theta_i\) for a learner. \(\epsilon\) is the learning rate, \(y_{pre(i)}\) is the presynaptic activity, and \(k\) is a reinforcing signal. The meta-gradient term \(\Delta w_{ij}\) is the change in the weight between units \(i\) and \(j\), with \(\eta\) being the learning rate.

\(^1\) Bengio et al. 1995
\(^2\) Runarsson and Jonsson 2000
Learning to learn gradient descent by gradient descent

- Replace backprop with a recurrent neural net (LSTM)\(^1\), not so scalable
- Use a coordinatewise LSTM \([m]\) for scalability/flexibility (cfr. ADAM, RMSprop) \(^2\)
  - Optimizee: receives weight update \(g_t\) from optimizer
  - Optimizer: receives gradient estimate \(\nabla_t\) from optimizee
  - Learns how to do gradient descent across tasks
**Few-shot learning**

- Learn how to learn from few examples (given similar tasks)
  - Meta-learner must learn how to train a base-learner based on prior experience
  - Parameterize base-learner model and learn the parameters $\theta_i$

\[
Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)
\]

1-shot, 5-class:

- New classes!
Few-shot learning: approaches

- Existing algorithm as meta-learner:
  - LSTM + gradient descent
  - Learn $\theta_{init} +$ gradient descent
  - kNN-like: Memory + similarity
  - Learn embedding + classifier
  - …

- Black-box meta-learner
  - Neural Turing machine (with memory)
  - Neural attentive learner
  - …

$\text{Cost}(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} \text{loss}(\theta_i, t)$

**Sources:**
- Santoro et al. 2016
- Ravi and Larochelle 2017
- Finn et al. 2017
- Vinyals et al. 2016
- Snell et al. 2017
- Mishra et al. 2018
LSTM meta-learner + gradient descent

- Gradient descent update $\theta_t$ is similar to LSTM cell state update $c_t$
  \[
  \theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \\
  c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t
  \]
- Hence, training a meta-learner LSTM yields an update rule for training $M$
  - Start from initial $\theta_0$, train model on first batch, get gradient and loss update
  - Predict $\theta_{t+1}$, continue to $t=T$, get cost, backpropagate to learn LSTM weights, optimal $\theta_0$
Model-agnostic meta-learning

- Quickly learn new skills by learning a model *initialization* that generalizes better to similar tasks
  - Current initialization $\theta$
  - On K examples/task, evaluate $\nabla_\theta L_{T_i}(f_\theta)$
  - Update weights for $\theta_1, \theta_2, \theta_3$
  - Update $\theta$ to minimize sum of per-task losses
  - Repeat
    $$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_1^*})$$

- More resilient to overfitting
- Generalizes better than LSTM approaches
- *Universality*: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- REPTILE: do SGD for k steps in one task, only then update initialization weights$^3$

---

1. Finn et al. 2017
2. Finn et al. 2018
3. Nichol et al. 2018
1-shot learning with Matching networks

- Don’t learn model parameters, use non-parameters model (like kNN)
- Choose an embedding network \( f \) and \( g \) (possibly equal)
- Choose an attention kernel \( a(\hat{x}, x_i) \), e.g. softmax over cosine distance
- Train complete network in minibatches with few examples per task

\[
\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i
\]

\( \theta = \{\text{VGG, Inception,}...\} \)
Prototypical networks

- Train a “prototype extractor” network
- Map examples to p-dimensional embedding so examples of a given class are close together
- Calculate a prototype (mean vector) for every class
- Map test instances to the same embedding, use softmax over distance to prototype
- Using more classes during meta-training works better!
Learning to reinforcement learn

- Humans often learn to play new games much faster than RL techniques do
- Reinforcement learning is very suited for learning-to-learn:
  - Build a learner, then use performance as that learner as a reward
- Learning to reinforcement learn \(^1,^2\)
  - Use RNN-based deep RL to train a recurrent network on many tasks
  - Learns to implement a ‘fast’ RL agent, encoded in its weights

- Also works for few-shot learning \(^3\)
  - Condition on observation + upcoming demonstration
  - You don’t know what someone is trying to teach you, but you prepare for the lesson
Learning to learn more tasks

- Active learning
  - Deep network (learns representation) + policy network
  - Receives state and reward, says which points to query next
- Density estimation
  - Learn distribution over small set of images, can generate new ones
  - Uses a MAML-based few-shot learner
- Matrix factorization
  - Deep learning architecture that makes recommendations
  - Meta-learner learns how to adjust biases for each user (task)
- Replace hand-crafted algorithms by learned ones.
- Look at problems through a meta-learning lens!
Meta-data sharing building a shared memory

• OK, but how do I get large amounts of meta-data for meta-learning?
• OpenML.org
  • Thousands of uniform datasets
  • 100+ meta-features
  • Millions of evaluated runs
    • Same splits, 30+ metrics
    • Traces, models (opt)
  • APIs in Python, R, Java,…
  • Publish your own runs
  • Never ending learning
  • Benchmarks

import openml as oml
from sklearn import tree

task = oml.tasks.get_task(14951)
clf = tree.ExtraTreeClassifier()
flow = oml.flows.sklearn_to_flow(clf)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()

run locally, share globally

Open positions!
Scientific programmer
Teaching PhD
Towards human-like learning to learn

- Learning-to-learn gives humans a significant advantage
  - Learning how to learn any task empowers us far beyond knowing how to learn specific tasks.
  - It is a universal aspect of life, and how it evolves
- Very exciting field with many unexplored possibilities
  - Many aspects not understood (e.g. task similarity), need more experiments.

- **Challenge:**
  - Build learners that **never stop learning**, that **learn from each other**
  - Build a **global memory** for learning systems to learn from
  - Let them explore by themselves, active learning
Thank you!
Merci!

more to learn
http://www.autml.org/book/
Chapter 2: Meta-Learning

special thanks to
Pavel Brazdil, Matthias Feurer, Frank Hutter, Erin Grant,
Hugo Larochelle, Raghu Rajan, Jan van Rijn, Jane Wang