Scalable Hyperparameter Transfer learning

Valerio Perrone†, Rodolphe Jenatton†, Cédric Archambeau*, Matthias Seeger*

AWS AI†/Amazon Research*, Berlin
Co-authors

R. Jenatton    C. Archambeau    M. Seeger

Most of the material from

V. Perrone, R. Jenatton, M. Seeger, C. Archambeau
Scalable Hyperparameter Transfer learning. NeurIPS 2018
Tuning deep neural nets for optimal performance

LeNet5 [LBBH98]

The search space $\mathcal{X}$ is large and diverse:

- Architecture: # hidden layers, activation functions, ...
- Model complexity: regularization, dropout, ...
- Optimisation parameters: learning rates, momentum, batch size, ...
Two straightforward approaches

(Figure by Bergstra and Bengio, 2012)

- Exhaustive search on a regular or random grid
- Complexity is exponential in $p$
- Wasteful of resources, but easy to parallelise
- Memoryless
Hyperparameter transfer learning
Hyperparameter transfer learning
Hyperparameter transfer learning

HPO Job 1

HPO Job 2

HPO Job K
Hyperparameter transfer learning
Motivation

- **Transfer learning**: Exploit evaluations of related past tasks
  - A given ML algorithm tuned over different datasets
  - Can we do it in absence of meta-data?

- **Scalability**: Both with respect to
  - #evaluations: $\sum_{t=1}^{T} N_t$
  - #tasks: $T$
Black-box global optimisation

- The function $f$ to optimise can be non-convex.
- The number of hyperparameters $p$ is moderate (typically $< 20$).

Evaluating $f(x)$ is expensive.

No analytical form or gradient.
Evaluations may be noisy.
The function $f$ to optimise can be non-convex.

The number of hyperparameters $p$ is moderate (typically $< 20$).

Our goal is to solve the following optimisation problem:

$$x_\star = \arg\min_{x \in \mathcal{X}} f(x) .$$

- Evaluating $f(x)$ is expensive.
- No analytical form or gradient.
- Evaluations may be noisy.
Example: tuning deep neural nets \([\text{SLA12, SRS}^{+15, \text{ KFB}}^{+16}]\)

\[ f(x) \] is the validation loss of the neural net as a function of its hyperparameters \( x \).

Evaluating \( f(x) \) is very costly \( \approx \) up to weeks!
Bayesian (black-box) optimisation \[ \text{[MTZ78, SSW}^+ 16] \]

\[ x_* = \operatorname{argmin}_{x \in \mathcal{X}} f(x) \]
Bayesian (black-box) optimisation [MTZ78, SSW+16]

\[ x_\star = \arg\min_{x \in \mathcal{X}} f(x) \]

**Canonical algorithm:**
- Surrogate model \( \mathcal{M} \) of \( f \) #cheaper to evaluate
- Set of evaluated candidates \( \mathcal{C} = {} \)
Bayesian (black-box) optimisation [MTZ78, SSW+16]

$$x_\star = \arg\min_{x \in \mathcal{X}} f(x)$$

Canonical algorithm:

- **Surrogate model** $\mathcal{M}$ of $f$ \# cheaper to evaluate
- Set of evaluated candidates $\mathcal{C} = \{\}$
- While some BUDGET available:
  - Select candidate $x_{\text{new}} \in \mathcal{X}$ using $\mathcal{M}$ and $\mathcal{C}$ \# exploration/exploitation
Bayesian (black-box) optimisation [MTZ78, SSW+16]

\[ x_\star = \arg\min_{x \in \mathcal{X}} f(x) \]

Canonical algorithm:

- Surrogate model \( \mathcal{M} \) of \( f \) #cheaper to evaluate
- Set of evaluated candidates \( \mathcal{C} = {} \)
- While some BUDGET available:
  - Select candidate \( x_{\text{new}} \in \mathcal{X} \) using \( \mathcal{M} \) and \( \mathcal{C} \) #exploration/exploitation
  - Collect evaluation \( y_{\text{new}} \) of \( f \) at \( x_{\text{new}} \) #time-consuming
Bayesian (black-box) optimisation [MTZ78, SSW\textsuperscript{+}16]

\[ x_\star = \arg\min_{x \in \mathcal{X}} f(x) \]

**Canonical algorithm:**

- **Surrogate model** $\mathcal{M}$ of $f$ \#cheaper to evaluate
- Set of evaluated candidates $\mathcal{C} = \{\}$
- While some BUDGET available:
  - Select candidate $x_{\text{new}} \in \mathcal{X}$ using $\mathcal{M}$ and $\mathcal{C}$ \#exploration/exploitation
  - Collect evaluation $y_{\text{new}}$ of $f$ at $x_{\text{new}}$ \#time-consuming
  - Update $\mathcal{C} = \mathcal{C} \cup \{(x_{\text{new}}, y_{\text{new}})\}$
  - Update $\mathcal{M}$ with $\mathcal{C}$ \#Update surrogate model
  - Update BUDGET
Learn a probabilistic model of $f$, which is cheap to evaluate:

$$y_i | f(x_i) \sim \text{Gaussian} \left( f(x_i), \sigma^2 \right), \quad f(x) \sim \mathcal{GP}(0, K).$$
Bayesian (black-box) optimisation with Gaussian processes

1. Learn a probabilistic model of $f$, which is cheap to evaluate:

$$y_i | f(x_i) \sim \text{Gaussian} \left(f(x_i), \sigma^2 \right), \quad f(x) \sim \mathcal{GP}(0, K).$$

2. Given the observations $y = (y_1, \ldots, y_n)$, compute the predictive mean and the predictive standard deviation:
Bayesian (black-box) optimisation with Gaussian processes

1. Learn a probabilistic model of $f$, which is cheap to evaluate:

$$y_i | f(x_i) \sim \text{Gaussian}\left(f(x_i), \sigma^2 \right), \quad f(x) \sim \mathcal{GP}(0, K).$$

2. Given the observations $y = (y_1, \ldots, y_n)$, compute the predictive mean and the predictive standard deviation:

3. Repeatedly query $f$ by balancing exploitation against exploration.
Where is the minimum of $f(x)$?
Bayesian optimisation in practice

(Image credit: Javier González)
Bayesian optimization with transfer learning

Problem statement:

- $T$ functions $\{f_t(x)\}_{t=1}^T$ with observations $D_t = \{(x^n_t, y^n_t)\}_{n=1}^{N_t}$
- May/may not have meta-data (or contextual features) for $\{f_t(x)\}_{t=1}^T$
- **Goal:** Optimize some fixed $f_{t_0}(x)$ while exploiting $\{D_t\}_{t=1}^T$
- (this is not multi-objective!)
Bayesian optimization with transfer learning

Problem statement:

- \( T \) functions \( \{f_t(x)\}_{t=1}^T \) with observations \( D_t = \{(x^n_t, y^n_t)\}_{n=1}^{N_t} \)
- May/may not have meta-data (or contextual features) for \( \{f_t(x)\}_{t=1}^T \)
- **Goal:** Optimize some fixed \( f_{t_0}(x) \) while exploiting \( \{D_t\}_{t=1}^T \)
- (this is not multi-objective!)

Previous work:

- Multitask GP (Swersky et al. 2013, Poloczek et al. 2016)
- GP + filter evaluations by task similarity (Feurer et al. 2015)
- Various ensemble-based approaches
  - GPs (Feurer et al. 2018)
  - Feedforward NNs (Schilling et al. 2015)
What is wrong with the Gaussian process surrogate?

Scaling is $\mathcal{O}(N^3)$
Adaptive Bayesian linear regression (ABLR) [Bis06]

The model:

\[ P(y|w, z, \beta) = \prod_n \mathcal{N}(\phi_z(x_n)w, \beta^{-1}), \]
\[ P(w|\alpha) = \mathcal{N}(0, \alpha^{-1}I_D). \]

The predictive distribution:

\[ P(y^*|x^*, D) = \int P(y^*|x^*, w)P(w|D)dw = \mathcal{N}(\mu_t(x^*), \sigma_t^2(x^*)) \]
Multi-task ABLR for transfer learning

1. Multi-task extension of the model:

\[ P(y_t|\mathbf{w}_t, z, \beta_t) = \prod_{n_t} \mathcal{N}(\phi_z(x_{nt})\mathbf{w}_t, \beta_t^{-1}), \quad P(\mathbf{w}_t|\alpha_t) = \mathcal{N}(\mathbf{0}, \alpha_t^{-1}I_D). \]

2. Shared features \( \phi_z(x) \):
   - Explicit features set (e.g., RBF)
   - Random kitchen sinks [RR⁺07]
   - Learned by feedforward neural net

3. Multi-task objective:

\[ \rho \left( z, \{ \alpha_t, \beta_t \}_{t=1}^T \right) = -\sum_{t=1}^T \log P(y_t|z, \alpha_t, \beta_t) \]
Examples of $\phi_z$

Feedforward neural networks:

$$\phi_z(x) = a_L(Z_L a_{L-1} (... Z_2 a_1(Z_1 x) ... )).$$

- $z$ consists of all $\{Z_i\}_{i=1}^L$

Random Fourier features:

$$\phi_z(x) = \sqrt{2/D} \cos \left\{ \frac{1}{\sigma} Ux + b \right\}, \text{ with } U \sim \mathcal{N}(0, I) \text{ and } b \sim \mathcal{U}([0, 2\pi]).$$

- $z$ only consists of $1/\sigma$
Pictorial summary of ABLR

\[
\begin{align*}
  w_2 & \sim \mathcal{N}(0, \alpha_2^{-1}I_D) \\
y_2 & \mid X_2, w_2, \beta_2, z \sim \mathcal{N}(\Phi_z(X_2)w_2, \beta_2^{-1}I_{N_2})
\end{align*}
\]

\[
\begin{align*}
  w_1 & \sim \mathcal{N}(0, \alpha_1^{-1}I_D) \\
y_1 & \mid X_1, w_1, \beta_1, z \sim \mathcal{N}(\Phi_z(X_1)w_1, \beta_1^{-1}I_{N_1})
\end{align*}
\]

\[
\begin{align*}
  w_3 & \sim \mathcal{N}(0, \alpha_3^{-1}I_D) \\
y_3 & \mid X_3, w_3, \beta_3, z \sim \mathcal{N}(\Phi_z(X_3)w_3, \beta_3^{-1}I_{N_3})
\end{align*}
\]

\[D_1 = \{(x_1^n, y_1^n)\}_{n=1}^{N_1} \quad D_2 = \{(x_2^n, y_2^n)\}_{n=1}^{N_2} \quad D_3 = \{(x_3^n, y_3^n)\}_{n=1}^{N_3}\]
Posterior inference

Hyperparameters:
- $\{\alpha_t, \beta_t\}_{t=1}^T$ for each task $t$
- $z$ for the shared basis function

Empirical Bayesian approach:
- Marginalize out the Bayesian linear regression parameters $\{w_t\}_{t=1}^T$
- Jointly learn the hyper-parameters of the model $\{\alpha_t, \beta_t\}_{t=1}^T$ and $z$

Minimize

$$\rho \left( z, \{\alpha_t, \beta_t\}_{t=1}^T \right) = - \sum_{t=1}^T \log \{ \mathbb{P}(y_t \mid X_t, \alpha_t, \beta_t, z) \}$$
Posterior inference (cont’d)

We have closed-forms for posterior mean and variance:

\[
\mu_t(x^*_t, D_t, \alpha_t, \beta_t, z) = \frac{\beta_t}{\alpha_t} \phi_z(x^*_t)^\top K_t^{-1} \Phi_t^\top y_t
\]

\[
\sigma_t^2(x^*_t, D_t, \alpha_t, \beta_t, z) = \frac{1}{\alpha_t} \phi_z(x^*_t)^\top K_t^{-1} \phi_z(x^*_t) + \frac{1}{\beta_t}
\]

and marginal likelihood:

\[
\rho(z, \{\alpha_t, \beta_t\}_{t=1}^T) = -\sum_{t=1}^T \left[ \frac{N_t}{2} \log \beta_t - \frac{\beta}{2} \left( \|y_t\|^2 - \frac{\beta}{\alpha_t} \|c_t\|^2 \right) - \sum_{i=1}^D \log([L_t]_{ii}) \right]
\]

- Cholesky for \(K_t = \frac{\beta_t}{\alpha_t} \Phi_t^\top \Phi_t + I_D = L_t L_t^\top\)
- \(c_t = L_t^{-1} \Phi_t^\top y_t\)
Leveraging MXNet

In Bayesian optimization, derivatives needed for

- Posterior inference: \((z, \{\alpha_t, \beta_t\}_t^{T}) \rightarrow \rho(z, \{\alpha_t, \beta_t\}_t^{T})\)
- Acquisition functions \(A\), typically of the form (e.g., EI, PI, UCB, ...):

\[
x^* \mapsto A(\mu_t(x^*; D_t, \alpha_t, \beta_t, z), \sigma_t^2(x^*; D_t, \alpha_t, \beta_t, z))
\]

Leverage MXNet (Seeger et al. 2017):

- Auto-differentiation
- Backward operator for Cholesky
- Can use any \(\phi_z\)
Optimization of the marginal likelihood

Optimization properties:

- Number of tasks: \( T \approx \) few tens
- Number of points per task: \( N_t \gg 1 \)
- Not standard SGD regime
- We apply L-BFGS *jointly* over all parameters \( \mathbf{z} \) and \( \{\alpha_t, \beta_t\}_{t=1}^{T} \)
- Warm-start parameters: Re-convergence in a very few steps
Surrogate models used in Bayesian optimization

Various types of models used:

- Gaussian processes (Jones et al. 1998, Snoek et al. 2012, ...)
- Sparse gaussian processes (McIntire et al. 2016)
- Variants (DKL/KISS-GP) of Gaussian processes (Pleiss et al. 2018)
- Random forests (Hutter et al. 2011)
- (Bayesian) NNs (Snoek et al. 2015, Springenberg et al. 2016)
Contributions:

- Simplicity
- Scalability
- Transfer learning in absence of meta-data
- Extend DNGO (Snoek et al. 2015) with:
  - Joint inference
  - Transfer learning and handling of heterogenous tasks
Warm-start procedure for hyperparameter optimisation (HPO)

Leave-one-task out.
Pictorial view of different transfer learning approaches

1. Single marg. likelihood, stack across tasks

\[
\begin{bmatrix}
X_1 & \text{context}_1 \\
\vdots & \vdots \\
X_T & \text{context}_T \\
\end{bmatrix} \in \mathbb{R}^{\sum_{t=1}^{T} N_t \times (P + |\text{context}|)}
\]

2. One marg. likelihood per \( X_t \) (no context!)

3. One marg. likelihood per \( [X_t, \text{context}_t] \)
Small-scale synthetic example: Transfer learning across quadratic functions

3-dimensional parameterized quadratic functions:

$$f_t(x) = \frac{1}{2} a_t \|x\|_2^2 + b_t 1^T x + c_t,$$

- One task = one function $f_t$
- $(a_t, b_t, c_t) \in [0.1, 10]^3$, contextual information
- $T = 30$ tasks
- “Leave-one-task-out”
Experimental protocol

Comparisons with:
- Random search (Bergstra et al. 2012)
- Gaussian process (based on GPyOpt implementation)
- Gaussian process + “$L_1$ heuristic” (Feurer et al. 2015)
- DNGO$^1$ (Snoek et al. 2015)
- BOHAMIANN$^1$ (Springenberg et al. 2016)

Other considerations:
- Results aggregated over 30 replicates.
- Expected improvement used for all model-based approaches.
- Architecture of ABLR is (50, 50, 50) (following Snoek et al. 2015).

$^1$Implementation from https://github.com/automl/RoBO
Transfer learning across quadratic functions

Transfer learning with baselines [KO11]. Transfer learning with neural nets [SRS⁺15, SKFH16].
Scalability: GP vs ABLR

Scaling in N:
- GP
- ABLR NN

Scaling in N:
- ABLR NN
- ABLR RKS
Transfer learning - OpenML data (Vanschoren et al. 2014)

- One task = one dataset
- Collect \( \{(X_t, y_t)\}_{t=1}^{T} \) from OpenML (Vanschoren et al. 2014)
- SVM: 4 HPs, XGBoost: 10 HPs
- Take \( T=30 \) datasets (flow_ids)
  - \( \sum_t N_t \) up to \( 7.5 \times 10^5 \) evaluations
Transfer learning across OpenML data sets

Transfer learning in SVM.

Transfer learning in XGBoost.
## Transfer learning vs. exploiting side signals

<table>
<thead>
<tr>
<th># active task(s)</th>
<th># optimized task</th>
<th>marg. likelihood</th>
<th>transfer learning</th>
<th>side signals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1 T</td>
</tr>
<tr>
<td>$N_t$</td>
<td>$N_t$</td>
<td></td>
<td>non-active $N_t$ fixed</td>
<td>growing $N_t = N$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a tuning experiment</td>
<td>a signal</td>
</tr>
</tbody>
</table>

### Typical use cases

- **Transfer learning**: Reuse data of previous tuning experiments
- **Side signals**: The training of ML models generate multiple signals
Leveraging multiple signals

**Goal**: Tune feedforward NNs for binary classification

- **Main signal**: Validation accuracy
- **Side signals**: Training accuracy and CPU time ("come for free")

**Idea**: Side signals can help learn $\phi_z$
Leveraging multiple signals

Transfer learning across LIBSVM data sets.
Conclusion

Bayesian optimisation is a model-based approach that **automates** machine learning:
- Algorithm tuning
- Model tuning

**ABLR** [PJSA17]:
- Scalable
- Fully leverages MXNet
- Transfers knowledge across tasks and signals
Thank you!
References

James Bergstra and Yoshua Bengio.
Random search for hyper-parameter optimization.

C. M. Bishop.
*Pattern Recognition and Machine Learning*.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter.
Fast Bayesian optimization of machine learning hyperparameters on large datasets.

Andreas Krause and Cheng S Ong.
Contextual gaussian process bandit optimization.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.


