TRAINING DEEP LEARNERS AND OTHER ITERATIVE MODELS WITH MLJ

Anthony Blaom
Department of Computer Science, University of Auckland.

SUMMARY

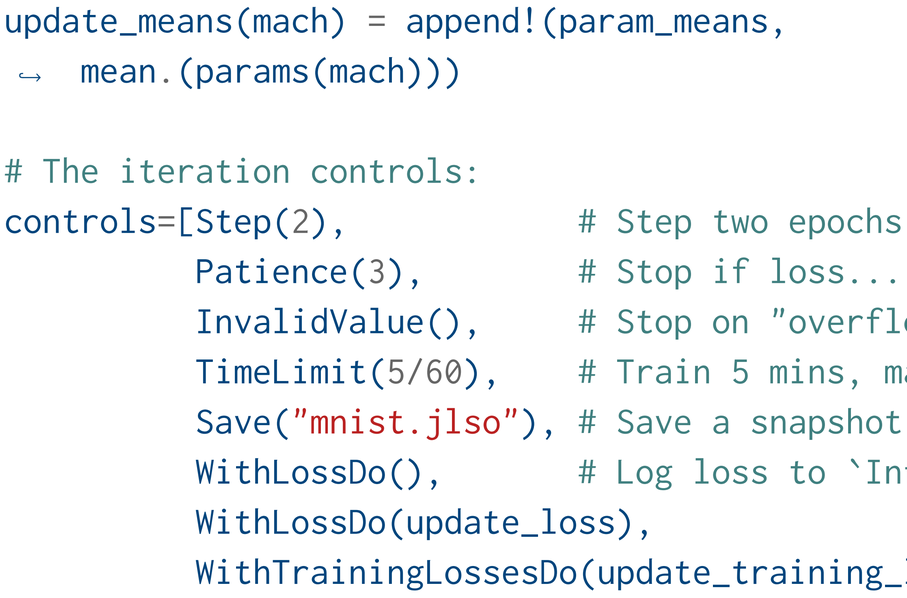
MLJ.jl (A. D. Blaom et al., 2020) is a toolbox written in Julia providing meta-algorithms for selecting, tuning, evaluating, composing, and comparing over 160 machine learning models written in Julia and other languages. We present a new wrapper `IteratedModel` for controlling iterative supervised models, such as a gradient tree booster or a neural network, and indicate some applications.

BASIC DEMONSTRATION

The code below, an excert from (A. Blaom and Shridhar, 2021), wraps an instance, `clf`, of MLJFlux.jl's neural network `ImageClassifier` model type, in a number of iteration controls. Figures 1 and 2, generated by applying the wrapped model, `iterated_clf`, to 500 images of the MNIST data set, show traces initialized in the first code block.

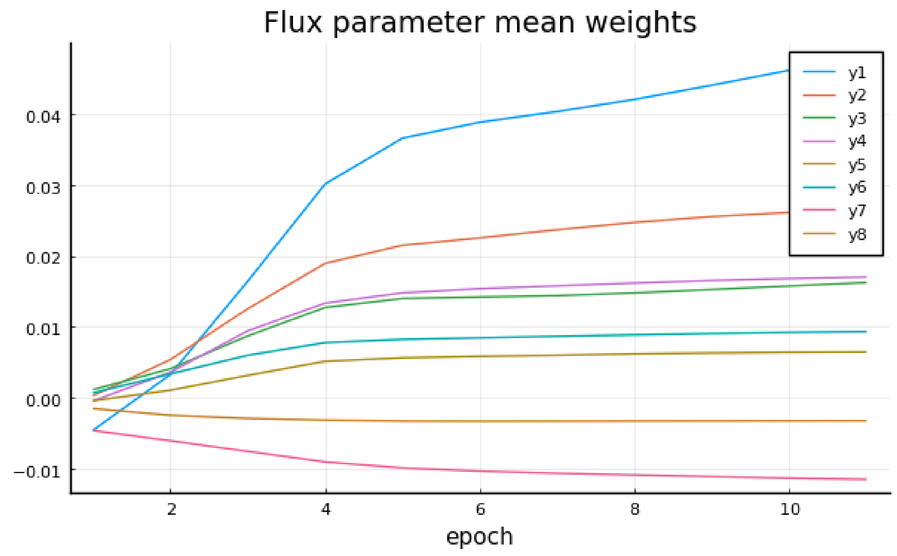
```
losses = []
training_losses = []
param_means = Float32[]

# to extract Flux params from an MLJFlux machine:
make1d(x) = reshape(x, length(x));
params(mach) = make1d.(Flux.params(fitted_params(mach)))

# To update traces:
update_loss(loss) = push!(losses, loss)
update_training_loss(losses) = push!(training_losses,
 ↪  losses[end])
update_means(mach) = append!(param_means,
 ↪  mean.(params(mach)))

# The iteration controls:
controls=[Step(2),              # Step two epochs
          Patience(3),          # Stop if loss...
          InvalidValue(),       # Stop on "overflow"
          TimeLimit(5/60),      # Train 5 mins, max
          Save("mnist.jlso"), # Save a snapshot
          WithLossDo(),         # Log loss to `Info`
          WithLossDo(update_loss),
          WithTrainingLossesDo(update_training_loss),
          Callback(update_means)]

# The wrapped model:
iterated_clf = IteratedModel(model=clf, controls=controls)
```

FIGURE 1



Comparison of training loss and out-of-sample loss in an image classifier with `Patience(3)` early stopping.

FIGURE 2



Evolution of mean weights in the wrapped classifier model of Figure 1. Higher numbers refer to deeper parameters.

IMPLEMENTATION

Iterative models in MLJ support warm restart of training, which makes external control of these algorithms relatively straightforward. In MLJ, an object called a *machine* points to the model hyper-parameters, data, and updatable learned parameters (the `mach` in the code above). A machine is controlled using the author's generic IterationControl.jl package. Stopping criterion, such as `Patience` in the example above, are based on an out-of-sample loss estimate. An optional key-word argument specifies the proportion of data to be held back for the estimate, or if all data is used and the training loss substituted for the out-of-sample loss.

ITERATION CONTROL IS COMPOSITIONAL

All MLJ's functionality, such as performance estimation, tuning and model composition, can be applied to the wrapped model. By default, the atomic iterative model is trained on all data once the number of iterations has been determined in the controlled training phase (which is restricted to a subset of the supplied data). In this way, the iterated model is simply a new version of the original model, but with the iteration parameter transformed from hyper-parameter to *learned* parameter.

HYPER-PARAMETER OPTIMIZATION

In MLJ hyper-parameter optization (tuning) is also implemented as a model wrapper, and is *iterative*. Tuning can therefore be controlled externally using the `IteratedModel` wrapper. For example, one can imagine a relatively straightforward implementation of HyperBand optimization (Li et al., 2018), in which several tuning jobs are run competitively in parallel, with resources being gradually diverted to the best performers. A proof of concept is offered at (A. Blaom, 2021).

REFERENCES

Blaom, Anthony (2021). *A demonstration of controlling models competing in parallel*. Juptyer notebook. URL: https://github.com/JuliaAI/IterationControl.jl/blob/dev/examples/competing_models/competing_models.ipynb.

Blaom, Anthony D. et al. (2020). "MLJ: A Julia package for composable machine learning". In: *Journal of Open Source Software* 5.55, p. 2704. DOI: 10.21105/joss.02704. URL: https://doi.org/10.21105/joss.02704.

Blaom, Anthony and Ayush Shridhar (2021). *Using MLJ to classifiy the MNIST image dataset*. Juptyer notebook. URL: https://github.com/FluxML/MLJFlux.jl/blob/master/examples/mnist/mnist.ipynb.

Li, Lisha et al. (2018). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.185, pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html.