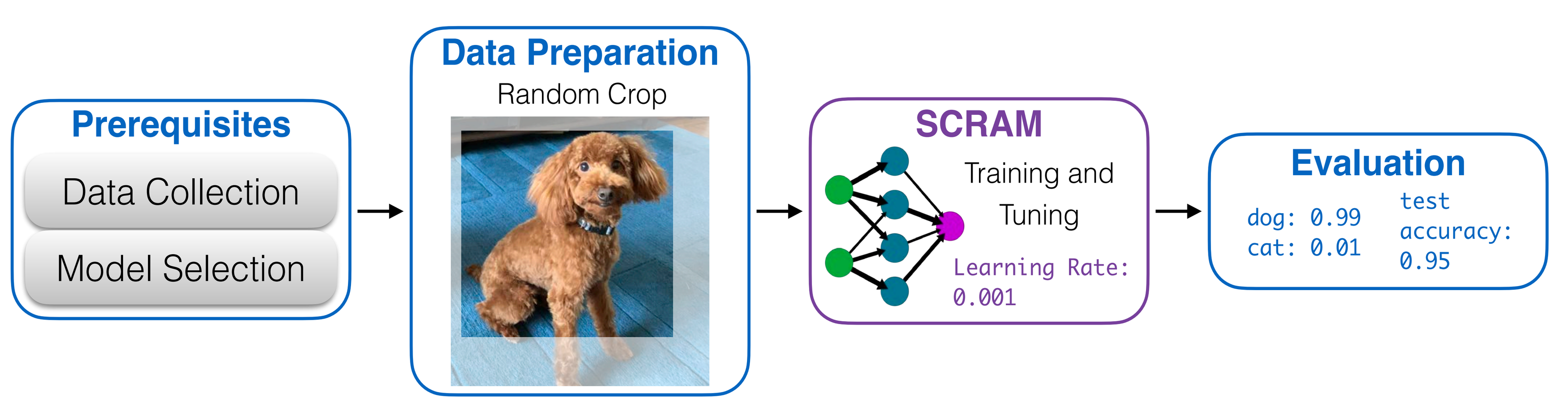
SCRAM Simple Checks for Realtime Analysis of Model Training for Non-Expert ML Programmers

Eldon Schoop, Forrest Huang, Björn Hartmann. UC Berkeley {eschoop, forrest_huang, bjoern}@berkeley.edu

Berkeley institute of design

Abstract

Many non-expert Machine Learning users wish to apply deep learning models to their own domains but encounter hurdles in the model training process. We introduce SCRAM, a tool which uses heuristics to detect potential error conditions in model output and suggest best practices to help such users tune their models. Inspired by metaphors from software engineering, SCRAM extends high-level deep learning development tools to check model metrics during training and produce human-readable error messages. We validate SCRAM through three author-created examples with image and text datasets, and by collecting informal feedback from ML researchers with teaching experience. We reflect upon their feedback for the design of future ML debugging tools.

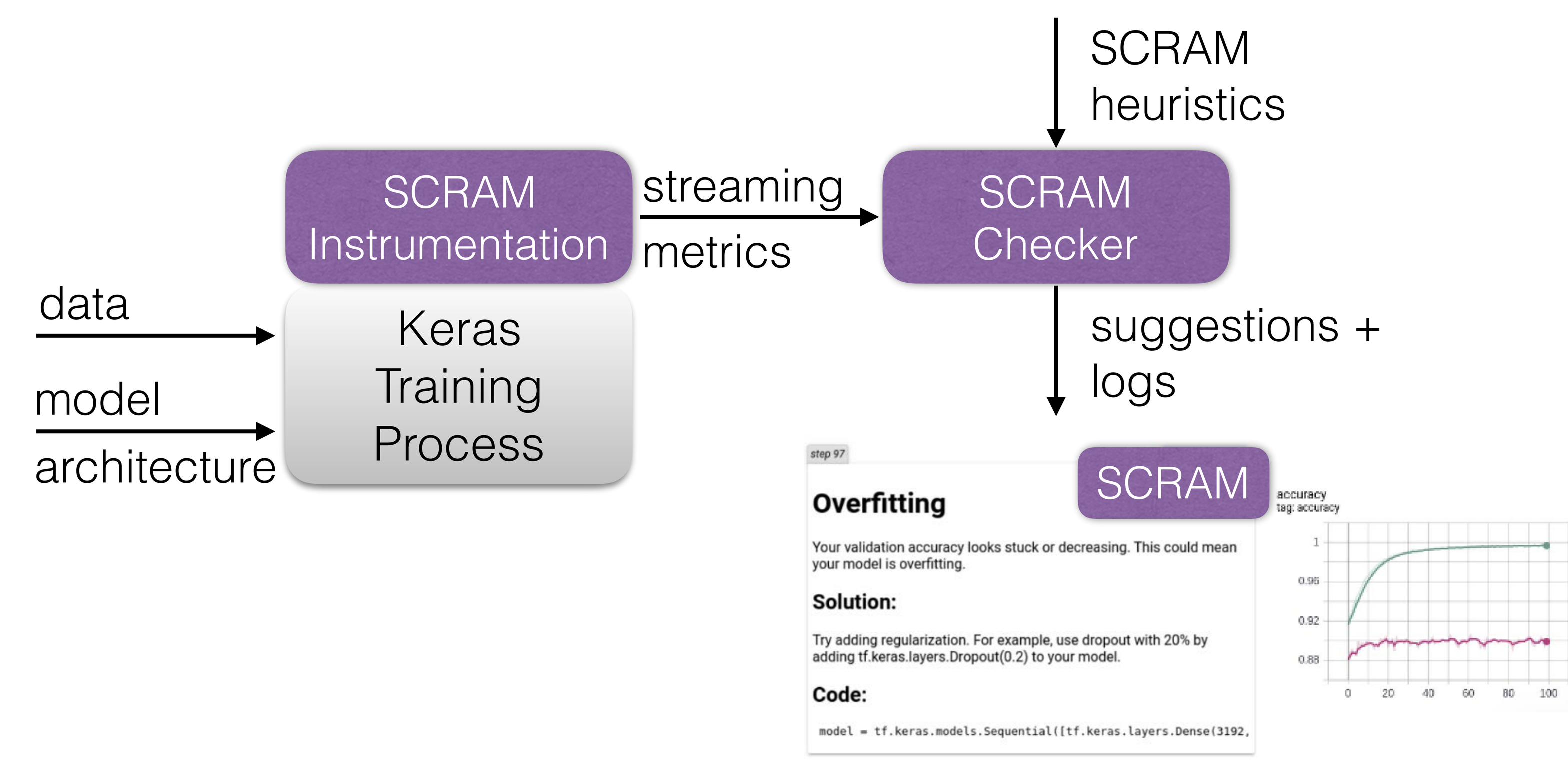


SCRAM instruments the data and model to guide developers through the training and tuning stage of model development.

SCRAM Design Principles

SCRAM targets scenarios when users have an existing problem formulation for applying ML to their applications (i.e., a meaningful problem statement, network architecture, and dataset are prepared).

SCRAM guides users through the model training and tuning phase, where users train their model on new data, tweaking hyperparameters and making other changes to the model to better fit the data. While ML experts can rely on tacit knowledge to interpret model output, many non-experts are confused by model output, leading many to abandon ML approaches altogether.



The Keras framework outputs data batches and model metrics to SCRAM (left), and SCRAM outputs error messages and visualizations to Tensorboard (right).

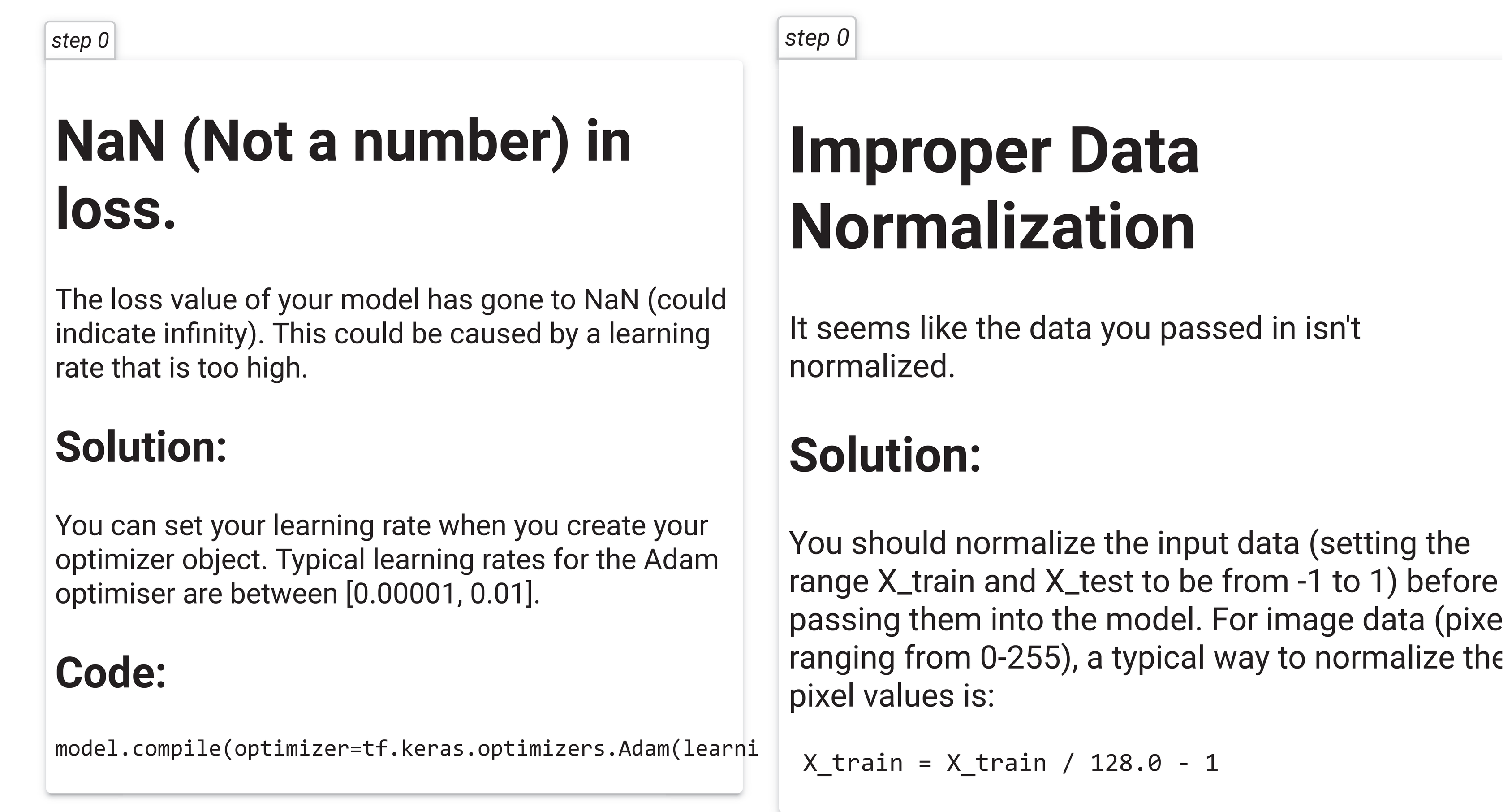
Implementation

SCRAM hooks into the built-in callback mechanism of Keras, which can invoke actions during model training. During training runtime, data batches and model metrics (loss and accuracy) are fed into SCRAM, where they are logged and checked against a list of heuristics to produce error mes- sages. Checks are loaded individually, and can be swapped and customized as needed. Error messages and metrics are emitted directly to Tensorboard via the Summary API.

Model Checks and Errors

Heuristics are collected from literature, lecture slides, blog posts, and other sparse sources. When a condition is detected by a heuristic, it is used to generate a human-readable error message designed to guide the user towards locating and correcting the error. Errors describe the suspected underlying problem in plain language and offer potential solutions (including example code snippets). Checks supported by SCRAM include:

Overfitting: Check if validation accuracy decreases over two epochs while training accuracy increases, indicating potential overfitting (Kaparthy, 2016).



Error messages produced by SCRAM describe the suspected underlying problem and guide the user with potential solutions.

Improper Normalization: Check if the values of input features of cur- rent batch lie within the conventional range of [−1, 1] (Shewchuk, 2019).

Unconventional Hyperparameter Range: Check if the loss value reaches NaN, which indicates a possible incorrect range of hyperparameters (Kaparthy, 2019).

Future Work

Code-Aware Tutorial Content: Making error messages interactive and linking directly to code could help users probe potential issues and identify bugs.

Active Testing: Future versions of SCRAM could run static checks (e.g., inspecting program structure) or even execute its own operations (e.g., trying to overfit on one batch).

Dynamic Error Messages: Dynamically generated error messages could highlight more detailed program context and help users identify the root causes of errors.

Communicating Uncertainty of Heuristics: The heuristics used by SCRAM are designed to detect and explain common errors, but these explanations are assumptions of model behavior and may not always be applicable.