HyperSearch: A Parallel Training Approach For Optimizing Neural Networks Performance

Geraud Nangue Tasse[*,1], **James Connan**[2]

[1]University of the Witwatersrand, Johannesburg, South Africa
[2]Rhodes University, Grahamstown, South Africa

WITS

R A I L
L A B

We provide a method for adaptively focusing on promising regions of the parameter and hyperparameter spaces using the same resources as random search thereby dramatically speeding up training of neural networks.

Introduction

- We want to find the **best neural network** for a given domain **as fast as possible**.
- **Automatic hyperparameter optimisation** and **parallel/distributed computing** are areas of critical importance towards addressing this.

Prior works that attempt to acheive these (partially) are **HyperOpt-TPE** (uses **Bayesian Optimization**) [1], Spearmint (uses Bayesian Optimization with GP), **SMAC** (uses Bayesian Optimization with custom modelling function), and **Hyperband** (uses a Random Search **Bandit-Based Approach**)[2].

HyperSearch works by training **multiple neural networks** with different hyperparameters in parallel while optimizing both parameters and hyperparameters.

[3] proposes a method similar to ours, but differs in that optimize for a **diverse population** of best networks. While we use all the resourses to find a **single best network** (or networks in the neighbourhood of that).

[1] Bergstra, et al. "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms." *Proceedings of the 12th Python in science conference*. 2013.
[2] Li, Lisha, et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization." *arXiv preprint arXiv:1603.06560* (2016).
[3]Jaderberg, Max, et al. "Population based training of neural networks." *arXiv preprint arXiv:1711.09846* (2017).

Preliminaries

Assumptions

- The hyperparameter space is a metric space
- Similar hyperparameters give similar performance

Adam optimizer:

$$W^{[l]} = W^{[l]} - \alpha \frac{v^{corrected}_{dW^{[l]}}}{\sqrt{s^{corrected}_{dW^{[l]}}} + \varepsilon}$$

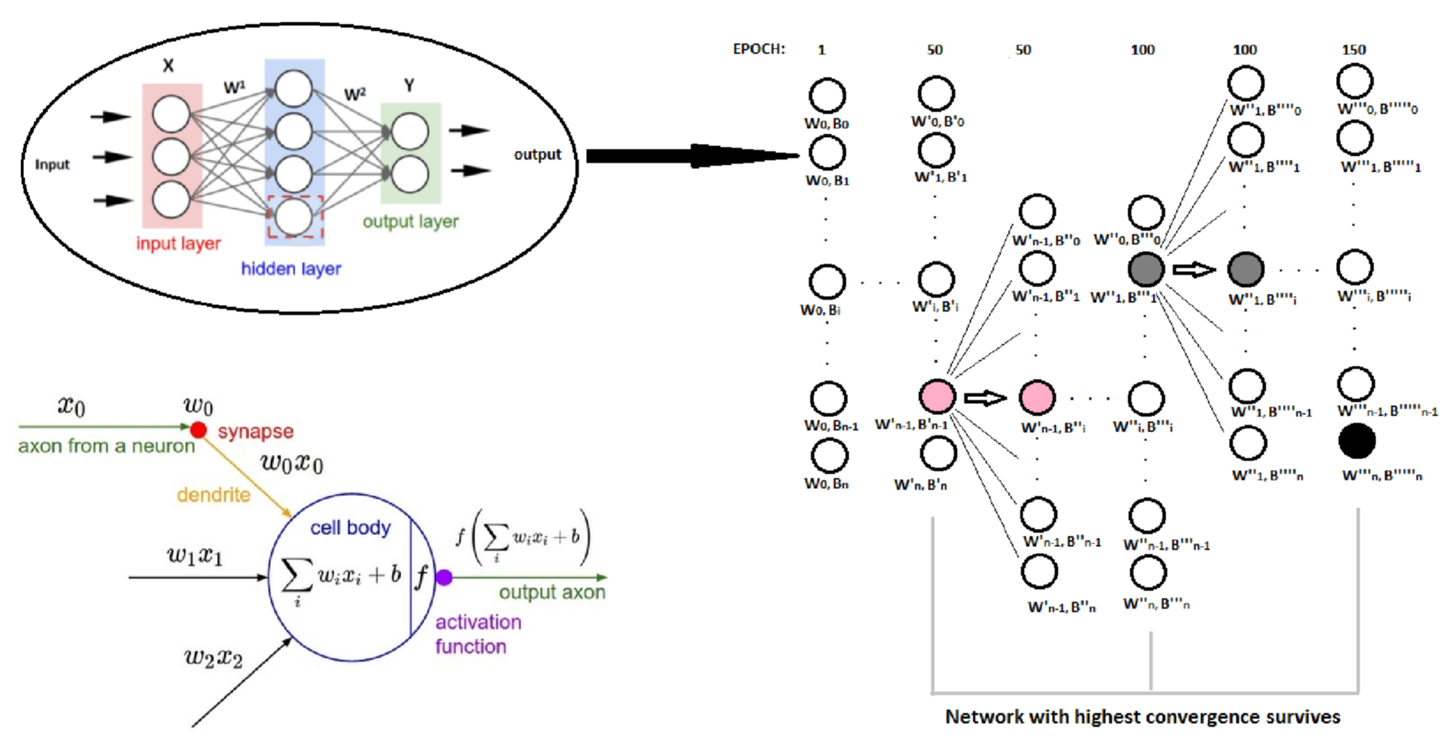
$$s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1-\beta_2)\left(\frac{\partial \mathcal{J}}{\partial W^{[l]}}\right)^2 \qquad v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1-\beta_1)\frac{\partial \mathcal{J}}{\partial W^{[l]}}$$

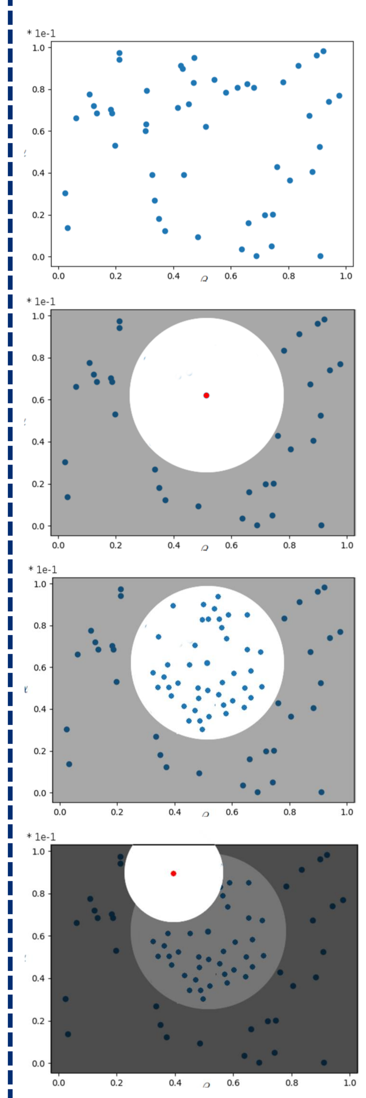
$$s^{corrected}_{dW^{[l]}} = \frac{s_{dW^{[l]}}}{1-(\beta_1)^t} \qquad v^{corrected}_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1-(\beta_1)^t}$$

- **t** counts the number of steps taken of Adam
- **β1 and β2 are hyperparameters** that control the weighted averages
- **ϵ** is a very small number to avoid dividing by zero

Hyperparameter	Type	Range/Examples	Bias/Variance
Learning rate (α)	Continuous	$(0\ \infty)$	Bias
Momentum (β)	Continuous	$(0\ \infty)$	Bias
Adam β_1	Continuous	$(0\ \infty)$	Bias
Adam β_2	Continuous	$(0\ \infty)$	Bias
Dropout	Continuous	[0 1]	Variance
L2-regularization (λ)	Continuous	[0 1]	Variance
Mini Batch Size	Discrete	[0 B]	Bias
Number of epochs	Discrete	$[1\ \infty)$	Bias
Number of Hidden Units	Discrete	$[1\ \infty)$	Bias
Number of Neurons per layers	Discrete	$[1\ \infty)$	Bias
Initializer	Categorical	Random, He, Xavier	Bias
Optimizer	Categorical	Momentum, Adam	Bias
Loss function	Categorical	Cross-Entropy Error, Mean-Squared Error	Both
Activation functions	Categorical	Relu, Sigmoid	Both
Preprocessor	Categorical	None, Normalize	Both

HyperSearch





- **N** = Number of concurrent networks
- **E** = Number of epochs
- **S** = Number of sessions
- **O(N*E)** time complexity
- Explores **N∗S** hyperparameter configurations
- For each session, bad networks **inherit best** networks parameters

Experiments (Tables)

Fix Hyperparameters	Values
Neurons per layers	[20, 20, 10, 2]
Activations per layers	[Relu, Relu, Relu, Sigmoid]
Loss function	Mean-Squared Error
Initializer	He
Optimizer	Adam

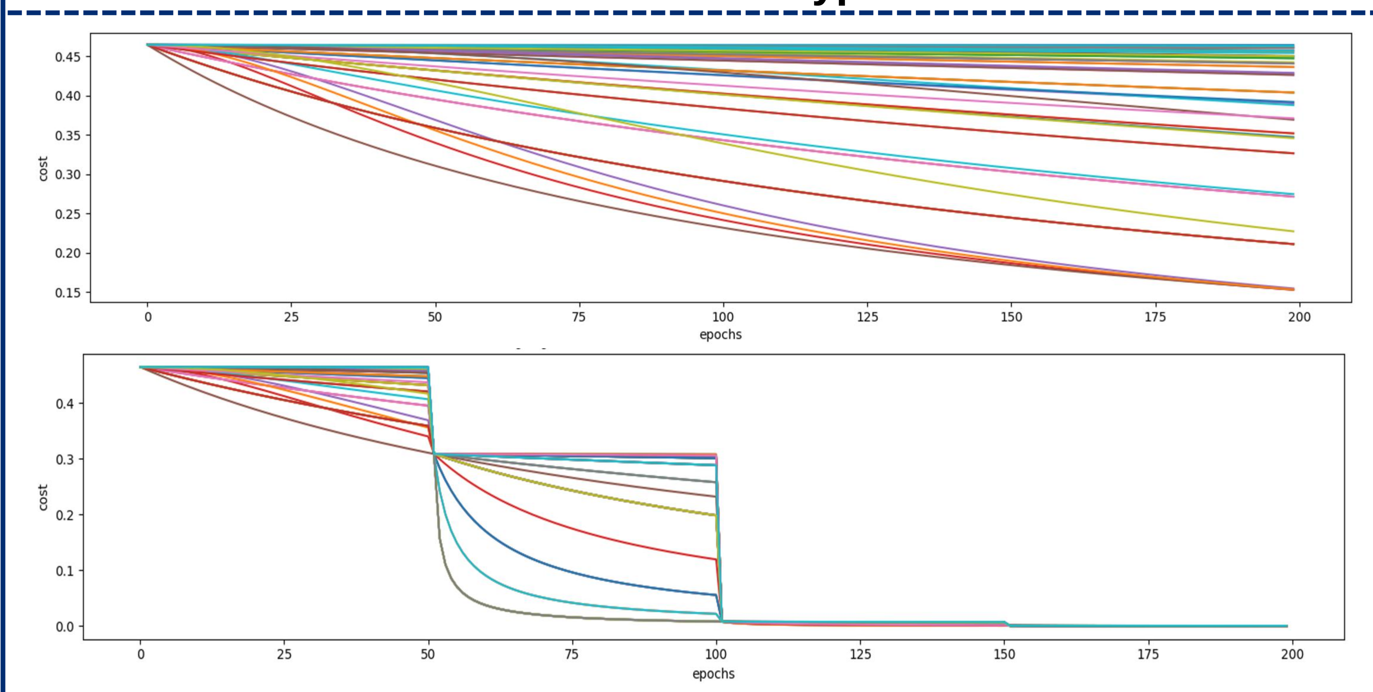
Fixed Hyperparameters	Scale	Range/Values	Boundaries
α	logarithmic	[0.00001 0.1]	$(0,\infty]$
β_1	logarithmic	[0.9 0.999]	[0,1]
β_2	logarithmic	[0.9 0.999]	[0,1]
λ	Normal	[0 0.5]	[0,1]
dropout	Normal	[0.5 1]	[0,1]
Mini-Batch Size (MBS)	Normal	[1 375]	[1,375]

Criterion	*train_cost* (Training error)	*val_cost* (Validation error)	*corrCoef* on training data	*r2Stats* on training data	*corrCoef* on validation data	*r2Stats* on validation data
train_cost	0.002	0.029	0.567	0.339	0.605	0.383
val_cost	0.012	0.008	0.874	0.773	0.887	0.794
mean_cost	0.004	0.01	0.866	0.759	0.879	0.78
r2Stats	0.021	0.01	0.902	0.817	0.904	0.822

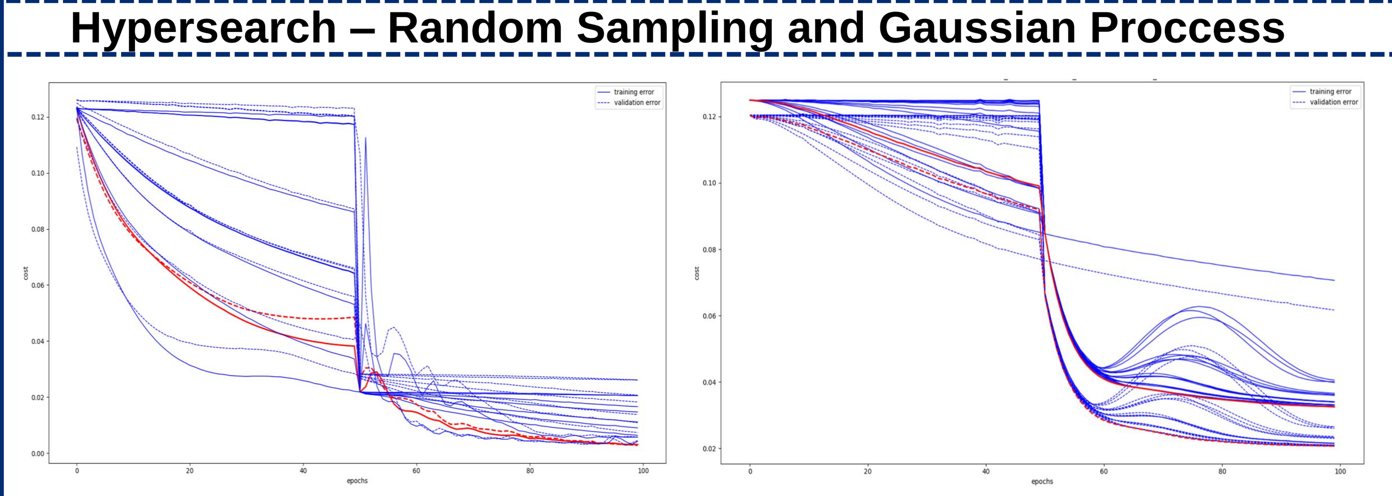
Method	*train_cost* (Training error)	*val_cost* (Validation error)	*corrCoef* on training data	*r2Stats* on training data	*corrCoef* on validation data	*r2Stats* on validation data
Uniform	0.017	0.015	0.807	0.674	0.851	0.737
Sobol	0.006	0.006	0.914	0.837	0.945	0.893
Hammersley	0.004	0.004	0.946	0.895	0.963	0.927
Halton	0.003	0.004	0.968	0.937	0.977	0.954

Experiments (Plots)

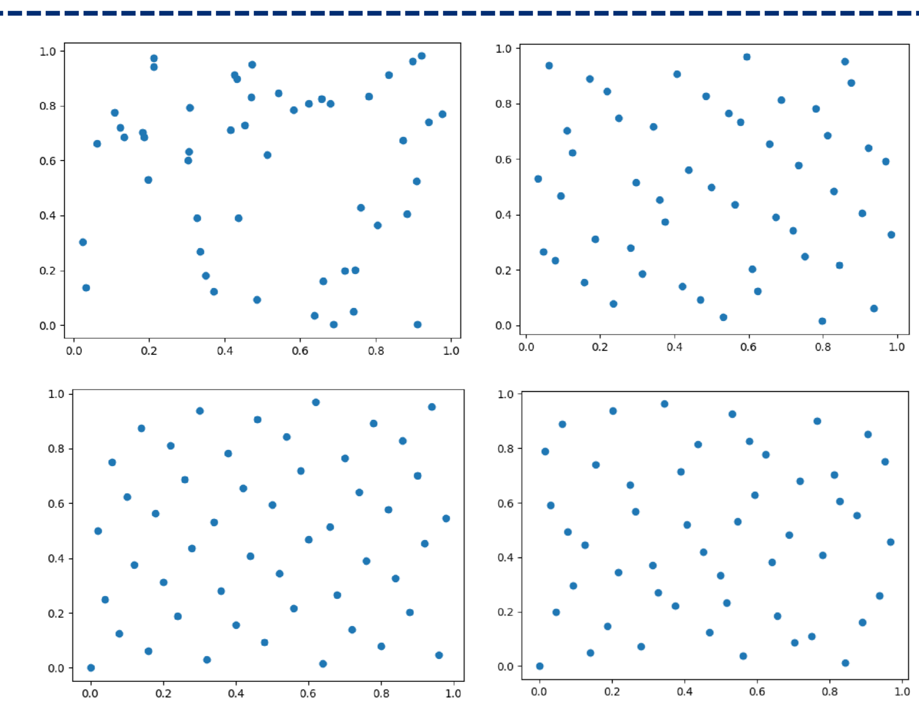
Random Search vs Hypersearch

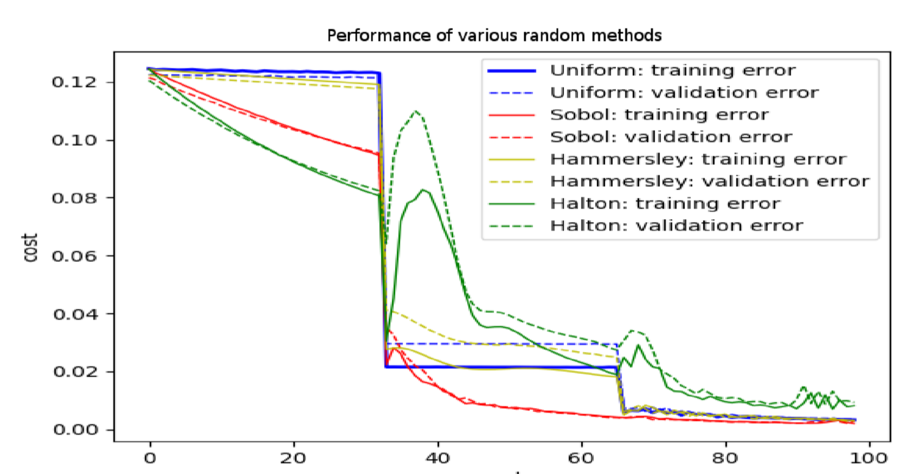


Hypersearch – Random Sampling and Gaussian Proccess

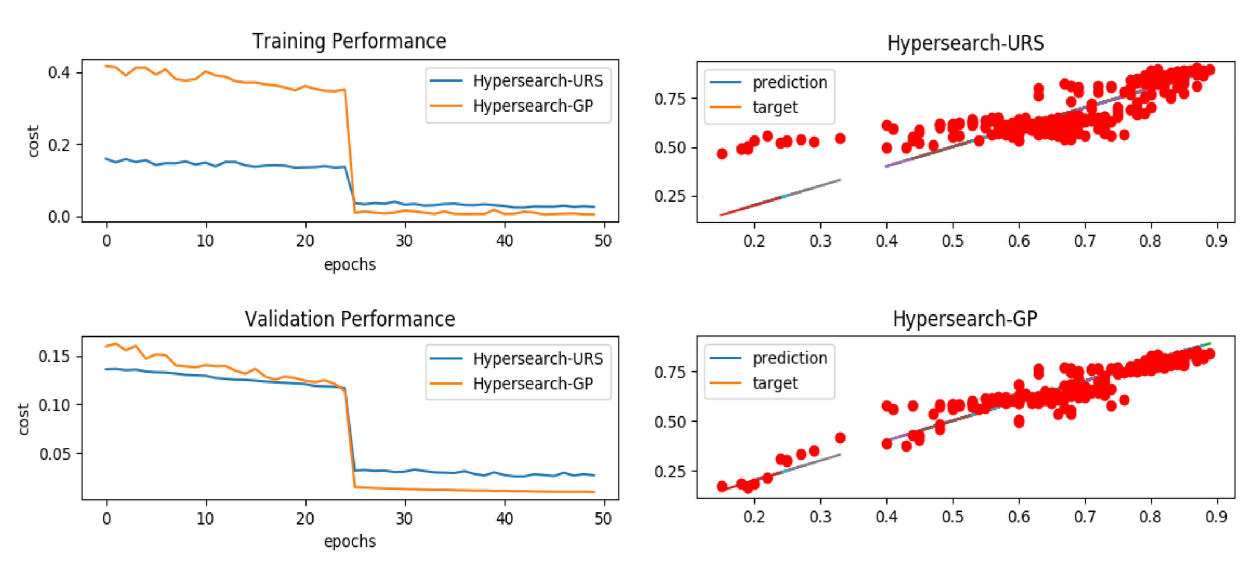


Sampling method





Hypersearch – Random Sampling vs Gaussian Proccess



Hypersearch vs Tree Parsen Estimator