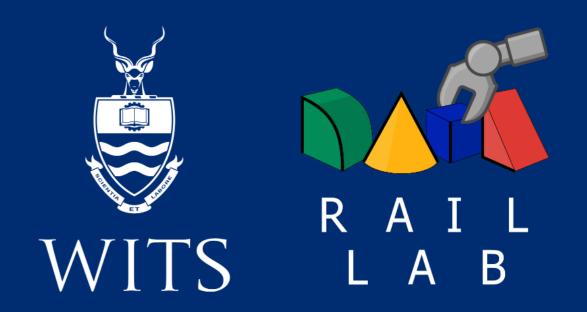
HyperSearch: A Parallel Training Approach For Optimizing Neural Networks Performance

Geraud Nangue Tasse^{*,1}, James Connan²

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



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.

Introd	luction

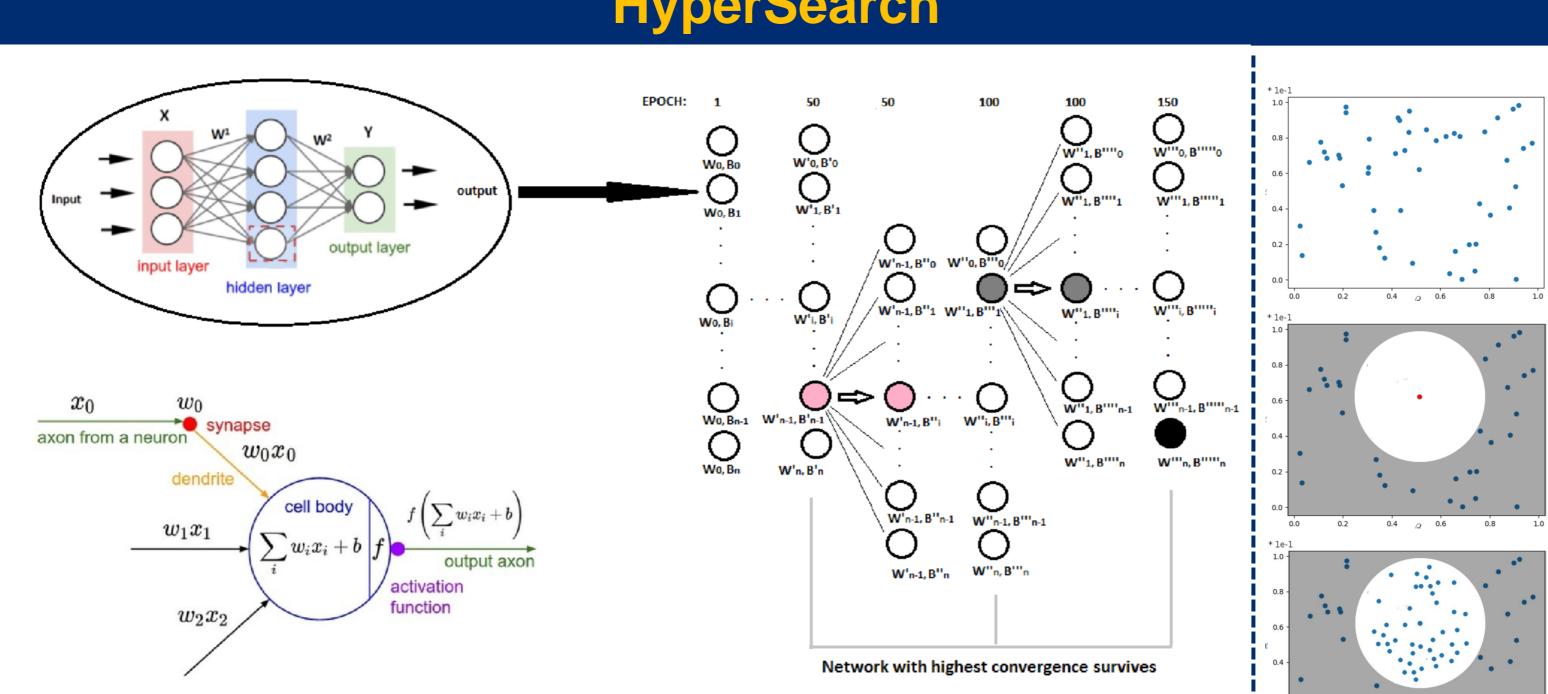
We want to find the **best neural network** for a given domain **as fast** as possible.

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.

• The hyperparameter space is a metric space • Similar hyperparameters give similar performance • Adam optimizer: $W^{[l]} = W^{[l]} - \alpha \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}} + \varepsilon} \qquad s_{dW^{[l]}} = \frac{\beta_2 s_{dW^{[l]}}}{1 - (\beta_1)^t} \qquad v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1 - \beta_1) \frac{\partial \mathcal{J}}{\partial W^{[l]}} \\ \cdot \text{ t counts the number of steps taken of Adam} \\ \cdot \beta_1 \text{ and } \beta_2 \text{ are hyperparameters that control the weighted averages} \\ \end{bmatrix} \frac{Learning rate (\alpha) Continuous (0 \infty) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\beta) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ \frac{Learning rate (\alpha) Continuous (0 \infty) Bias}{Momentum (\lambda) Continuous (0 \infty) Bias} \\ Learning rate (\alpha) Con$	Preliminaries				
Adam optimizer: L^2 -regularization (λ)Continuous $[0\ 1]$ Variance $W^{[l]} = W^{[l]} - \alpha \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}} + \varepsilon}$ $s_{dW^{[l]}} = \beta_2 s_{dW^{[l]}} + (1 - \beta_2) (\frac{\partial \mathcal{J}}{\partial W^{[l]}})^2$ $v_{dW^{[l]}} = \beta_1 v_{dW^{[l]}} + (1 - \beta_1) \frac{\partial \mathcal{J}}{\partial W^{[l]}}$ $N_{unber of epochs}$ Discrete $[1 \ \infty)$ Bias $W^{[l]} = W^{[l]} - \alpha \frac{v_{dW^{[l]}}^{corrected}}{\sqrt{s_{dW^{[l]}}^{corrected}} + \varepsilon}$ $s_{corrected}^{corrected} = \frac{s_{dW^{[l]}}}{1 - (\beta_1)^t}$ $v_{dW^{[l]}}^{corrected} = \frac{v_{dW^{[l]}}}{1 - (\beta_1)^t}$ Number of Neurons per liscrete $[1 \ \infty)$ Bias• t counts the number of steps taken of Adam $v_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1 - (\beta_1)^t}$ $v_{dW^{[l]}} = \frac{v_{dW^{[l]}}}{1 - (\beta_1)^t}$ CategoricalRandom, He, XavierBias• β 1 and β 2 are hyperparameters that control the weighted averages $v_{excorrected}$ $v_{dW^{[l]}} = \frac{1}{1 - (\beta_1)^t}$ $v_{excorrected}$ $Relu, Sigmoid$ Both	 The hyperparameter space is a metric space 	Learning rate (α)Momentum (β)Adam β_1 Adam β_2	ContinuousContinuousContinuousContinuousContinuous	$ \begin{array}{c} (0 \ \infty) \\ (0 \ \infty) \\ (0 \ \infty) \\ (0 \ \infty) \\ (0 \ \infty) \end{array} $	Bias Bias Bias
 t counts the number of steps taken of Adam β1 and β2 are hyperparameters that control the weighted averages Δctivation functions Categorical Momentum, Adam Bias Cotegorical Cross-Entropy Error, Mean-Squared Error Both Activation functions Categorical Relu, Sigmoid Both 	$\partial \mathcal{J}$	L2-regularization (λ) Mini Batch SizeNumber of epochsNumber of Hidden UnitsNumber of Neurons per	ContinuousDiscreteDiscreteDiscrete	$ \begin{array}{c c} [0 1] \\ [0 B] \\ [1 \infty) \\ [1 \infty) \end{array} $	Variance Bias Bias Bias
• ϵ is a very small number to avoid dividing by zero	 t counts the number of steps taken of Adam 	Initializer Optimizer Loss function	Categorical Categorical	Momentum, Adam Cross-Entropy Error, Mean-Squared Error	Bias Both



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

[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).

- **N** = Number of concurrent networks
- **E** = Number of epochs
- **S** = Number of sessions
- O(N*E) time complexity

 \bullet

- Explores **N*****S** hyperparameter configurations
- For each session, bad networks **inherit best** networks parameters

Fixed Hy	perparam	eters	Scale	Range/Va	alues I	Boundaries
α			logarithmic	[0.00001	0.1]	$(0,\infty]$
β_1			logarithmic	0.9 0.99)9]	[0,1]
β_2			logarithmic	$[0.9 \ 0.99]$		[0,1]
$\begin{vmatrix} \beta_2 \\ \lambda \end{vmatrix}$			Normal	[0.0, 0.5]		[0,1]
				L]		
dropout			Normal	$[0.5 \ 1]$		$\begin{bmatrix} 0,1 \end{bmatrix}$
Mini-Batch	Size (MB	S)	Normal	$[1 \ 375]$		[1, 375]
		1 /	Coord f	r2Stats		$f \mid r2Stats$
Criterion $ $	train_cost	val_cost	$\downarrow corr \cup oet$	TZStats	corrCoet	I = TZStats
Criterion	train_cost (Training	val_cost (Vali-	corrCoef on train-	on train-	corrCoef on va	·
			0			
	(Training	(Vali-	on train-	on train-	on va	al- on va
	(Training	(Vali- dation	on train-	on train-	on va idation	al- on va idation
$train_cost$ val_cost	(Training error) 0.002 0.012	(Vali- dation error) 0.029 0.008	on train- ing data 0.567 0.874	on train- ing data 0.339 0.773	on va idation data 0.605 0.887	al- on va idation data 0.383 0.794
train_cost val_cost mean_cost	(Training error) 0.002 0.012 0.004	(Vali- dation error) 0.029 0.008 0.01	on train- ing data 0.567 0.874 0.866	on train- ing data 0.339 0.773 0.759	on value idation data 0.605 0.887 0.879 0.879	on validation idata 0.383 0.794 0.78
train_cost val_cost mean_cost	(Training error) 0.002 0.012	(Vali- dation error) 0.029 0.008	on train- ing data 0.567 0.874	on train- ing data 0.339 0.773	on va idation data 0.605 0.887	al- on va idation data 0.383 0.794
train_cost val_cost mean_cost r2Stats	(Training error) 0.002 0.012 0.004 0.021	(Vali- dation error) 0.029 0.008 0.01 0.01	on train- ing data 0.567 0.874 0.866 0.902	on train- ing data 0.339 0.773 0.759 0.817	on value idation data 0.605 0.887 0.879 0.904	on validation idata 0.383 0.794 0.78 0.822 0.822
train_cost val_cost mean_cost	(Training error) 0.002 0.012 0.004 0.021 train_cost	(Vali- dation error) 0.029 0.008 0.01 0.01 val_cost	on training 0.567 0.874 0.866 0.902 t corrCoef	on train- ing data 0.339 0.773 0.759 0.817 <i>r2Stats</i>	on va idation data 0.605 0.887 0.879 0.904	$\begin{array}{c c} \textbf{al-} & \textbf{on va} \\ \textbf{idation} \\ \textbf{data} \\ \hline 0.383 \\ 0.794 \\ 0.78 \\ 0.822 \\ \hline ef & r2Stats \\ \end{array}$
train_cost val_cost mean_cost r2Stats	(Training error) 0.002 0.012 0.004 0.021 train_cost (Training	(Vali- dation error) 0.029 0.008 0.01 0.01 val_cost	$\begin{array}{c c} \text{on train-}\\ \text{ing data}\\ \hline 0.567\\ 0.874\\ 0.866\\ 0.902\\ \hline t & corrCoef\\ \text{on train-}\\ \end{array}$	on train- ing data 0.339 0.773 0.759 0.817 <i>r2Stats</i> on train-	on va idation data 0.605 0.887 0.879 0.904	onvalidationidation $data$ 0.383 0.794 0.794 0.78 0.822ef $r2Stats$ val-onvalidation
train_cost val_cost mean_cost r2Stats	(Training error) 0.002 0.012 0.004 0.021 train_cost	(Vali- dation error) 0.029 0.008 0.01 0.01 val_cost (Vali-	on training 0.567 0.874 0.866 0.902 t corrCoef	on train- ing data 0.339 0.773 0.759 0.817 <i>r2Stats</i>	on va idation data 0.605 0.887 0.879 0.904 <i>corrCoe</i> on v	onvalidationidation $data$ 0.383 0.794 0.794 0.78 0.822ef $r2Stats$ val-onvalidation
train_cost val_cost mean_cost r2Stats	(Training error) 0.002 0.012 0.004 0.021 train_cost (Training	(Vali- dation error) 0.029 0.008 0.01 0.01 val_cost (Vali- dation	$\begin{array}{c c} \text{on train-}\\ \text{ing data}\\ \hline 0.567\\ 0.874\\ 0.866\\ 0.902\\ \hline t & corrCoef\\ \text{on train-}\\ \end{array}$	on train- ing data 0.339 0.773 0.759 0.817 <i>r2Stats</i> on train-	on value idation data 0.605 0.887 0.879 0.904 corrCoe on idation value	onvalidationidationidata 0.383 0.794 0.794 0.78 0.822 0.822 ef $r2Stats$ ral-onvalidation
$\begin{array}{c c} train_cost \\ val_cost \\ mean_cost \\ r2Stats \end{array}$ $\begin{array}{c} \mathbf{Method} \end{array}$	(Training error) 0.002 0.012 0.004 0.021 train_cost (Training error)	(Vali- dation error) 0.029 0.008 0.01 0.01 val_cost (Vali- dation error)	on train- ing data 0.567 0.874 0.866 0.902 t corrCoef on train- ing data	on train- ing data 0.339 0.773 0.759 0.817 <i>r2Stats</i> on train- ing data	on value idation data 0.605 0.887 0.879 0.904 corrCoe on idation data	al- idation data 0.383 0.794 0.78 0.822 ef r2Stats ral- idation data
train_cost val_cost mean_cost r2Stats Method	(Training error) 0.002 0.012 0.004 0.021 <i>train_cost</i> (Training error) 0.017	(Vali- dation error) 0.029 0.008 0.01 0.01 0.01 val_cost (Vali- dation error) 0.015	on train- ing data 0.567 0.874 0.866 0.902 t corrCoef on train- ing data 0.807	on train- ing data 0.339 0.773 0.759 0.817 r2Stats on train- ing data 0.674	on va idation data 0.605 0.887 0.879 0.904 <i>corrCoe</i> on va idation data 0.851	al- on validation data 0.383 0.794 0.78 0.822 ef $r2Stats$ ral- on validation data 0.737

