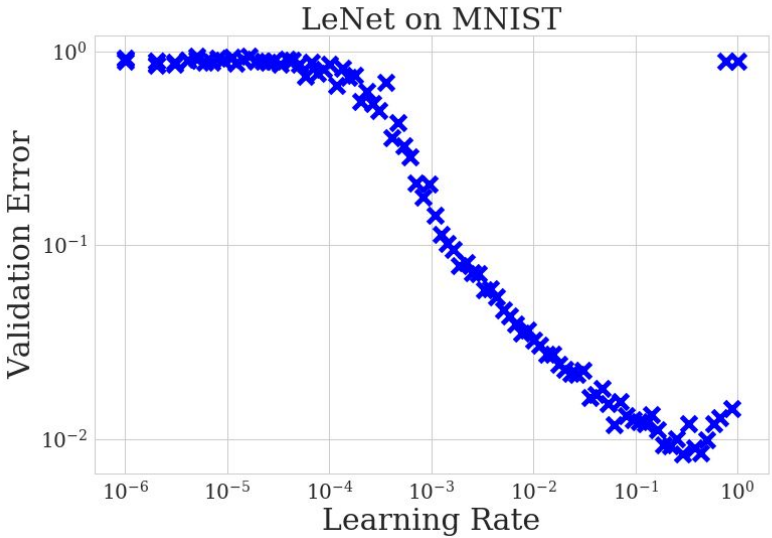
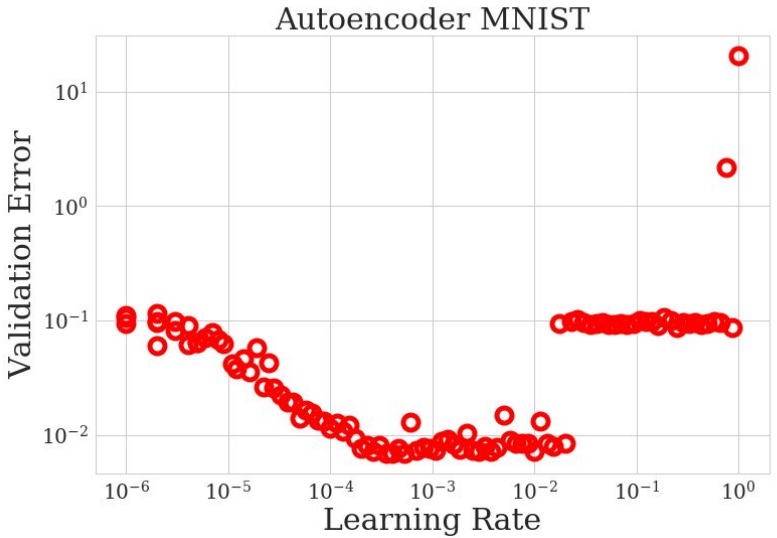
Hyperparameter Optimization

Aaron Klein

Effects of Changing the Learning Rate





Hyperparameter Optimization

Cast the problem as an optimization problem:

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

- **x :** denotes all hyperparameters
- **f(x) :** training and validation of the machine learning algorithm with hyperparameters **x**

Comparison

Hyperparameter optimization:

- very expensive (hours, days)
- no gradient information (at least not easy to compute)
- noisy
- non-convex

SGD optimization:

- single function evaluation are cheap (seconds)
- gradient information
- noisy
- non-convex

Configuration Space

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓

Hyperparameter Optimization

different ways to tackle this optimization problem:

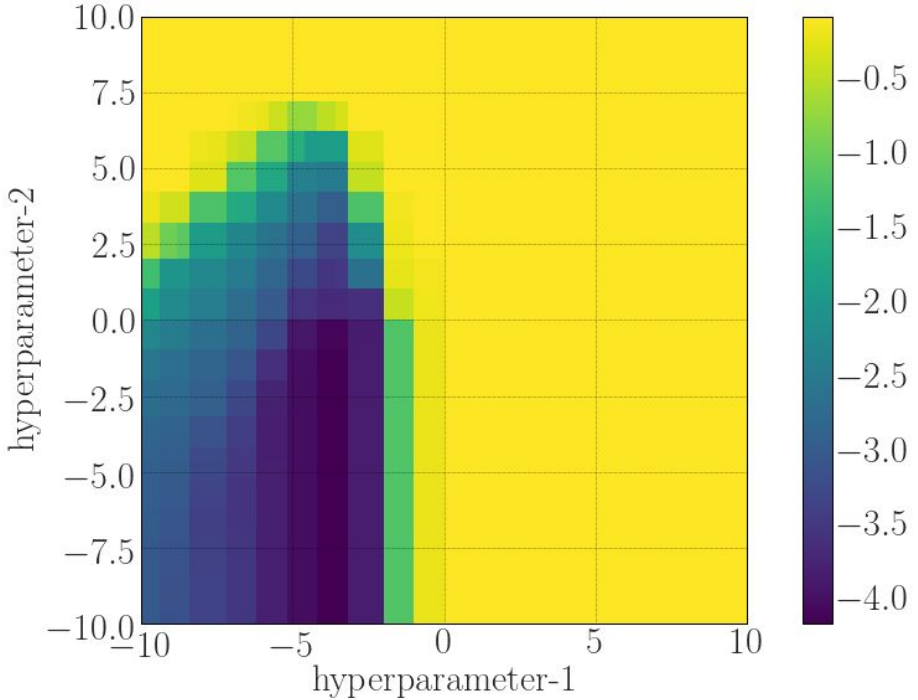
- manual tuning
- grid search
- random search
- Bayesian optimization
- evolutionary algorithms
- reinforcement learning
- hypergradient descent

Hyperparameter Optimization

different ways to tackle this optimization problem:

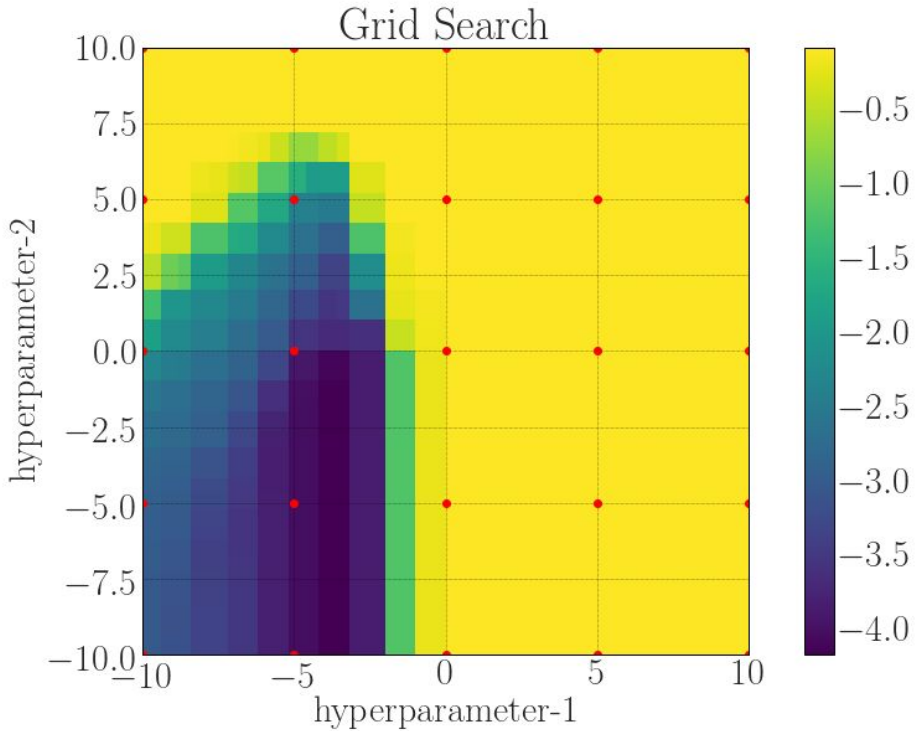
- manual tuning
- grid search
- random search
- Bayesian optimization
- evolutionary algorithms
- reinforcement learning
- hypergradient descent

Example: SVM with 2 hyperparameters



Grid Search

1. select a finite set of values for each hyperparameter
2. evaluate the Cartesian product of all hyperparameters

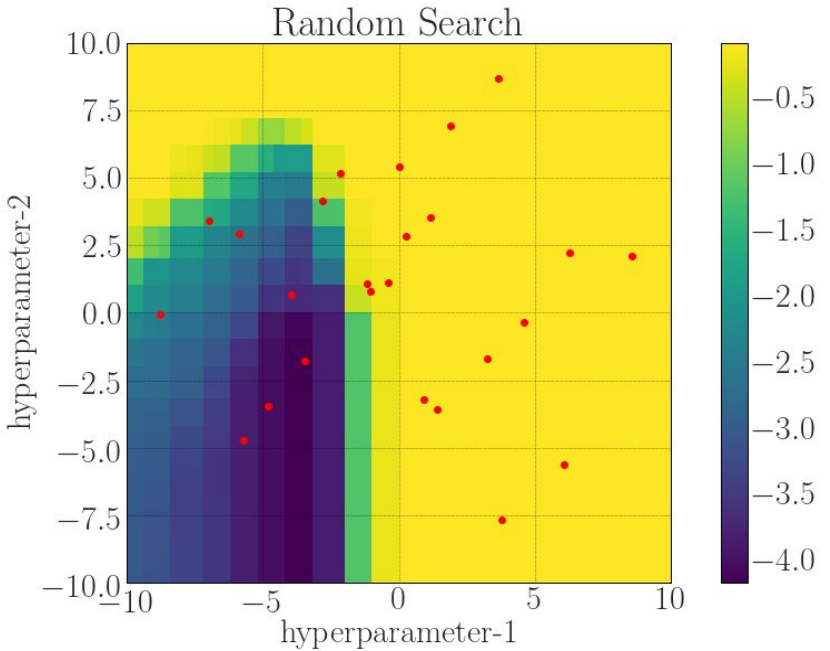


Grid Search

- **Pros:**
  - easy to implement
  - parallelizable
- **Cons:**
  - might never find the true optimum
  - number of function evaluations grows exponentially with the number of dimensions

Random Search

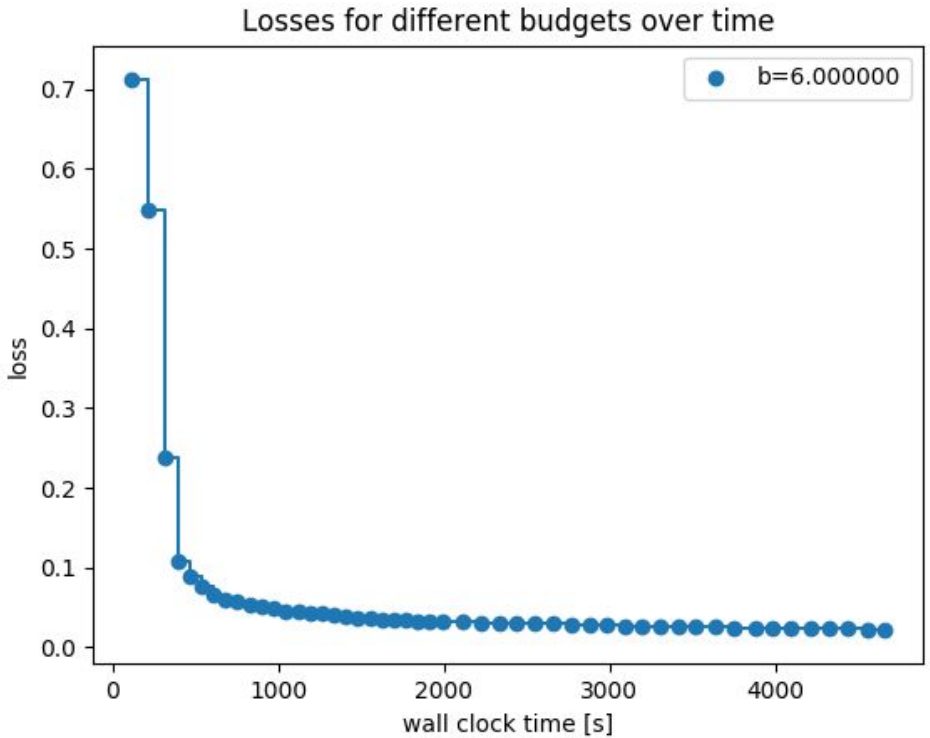
1. Specify a distribution over the input space, e g. uniform
2. draw and evaluate configurations sampled from this distribution until a predefined stopping criterion has been reached



Random Search

- **Pros:**
  - easy to implement
  - parallelizable
  - in theory it always finds the optimum
- **Cons:**
  - can’t make use of previous observations and often needs too long to approach the optimum

Exercise: Incumbent Trajectory



Conclusions

- Hyperparameters are important and need to be set correctly
- Instead of manually tuning them, we can cast it as an optimization problem
- Random Search works better than Grid Search
- In the AutoML track we will learn how to make hyperparameter optimization more efficient