Hyperparameter Optimization with Hyperband Algorithm

Deep Learning Meetup Italy

Gilberto Batres-Estrada

- **Senior Data Scientist** @ Trell Technologies, Stockholm
- **Experience**: Finance, Telecom, Fintech and Energy
- **AIFI**: Graduate teaching fellow
- **Co-author**: Big Data and Machine Learning

  in Quantitative Investment, Wiley (Chapter on LSTM)
- **MSc in Theoretical Physics**, Stockholm University
- **MSc in Engineering:** Applied Mathematics and Statistics ,

  (KTH Royal Institute of Technology) in Stockholm

Goals for today’s talk

1. Make the training process of neural networks faster
2. Get better performance and accurate neural networks (better test error)
3. To get more time for exploring different architectures

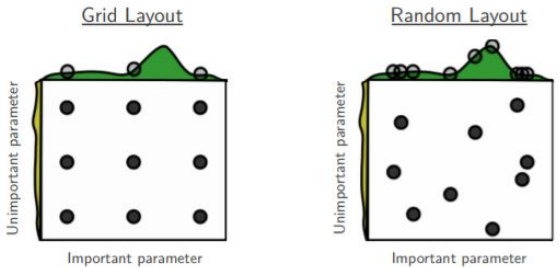
Agenda

- Random Search for Hyper-Parameter Optimization
- Bayesian optimization
- Hyperband
- Other methods
- Implementations and examples

Random Search

Proposed by James Bergstra and Yoshua Bengio

http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf



Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bayesian Optimization

Model the conditional probability

$$p(y|\lambda),$$

Where *y* is an evaluation metric such as test error and

$$\lambda$$

is a set of hyperparameters.

Sequential Model-Based Algorithm Configuration SMAC

SMAC uses random forest to model

$$p(y|\lambda),$$

as a Gaussian Distribution (Hetter et al., 2011)

Tree Structured Parzen Estimator (TPE)

TPE is a non-standard Bayesian optimization algorithm based on tree-structured Parzen density estimators (Bergstra et al., 2011)

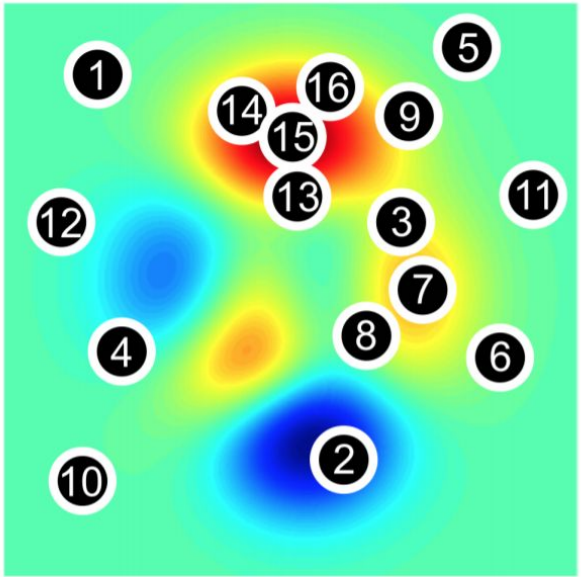
Spearmint

Uses Gaussian Processes (GP) to model

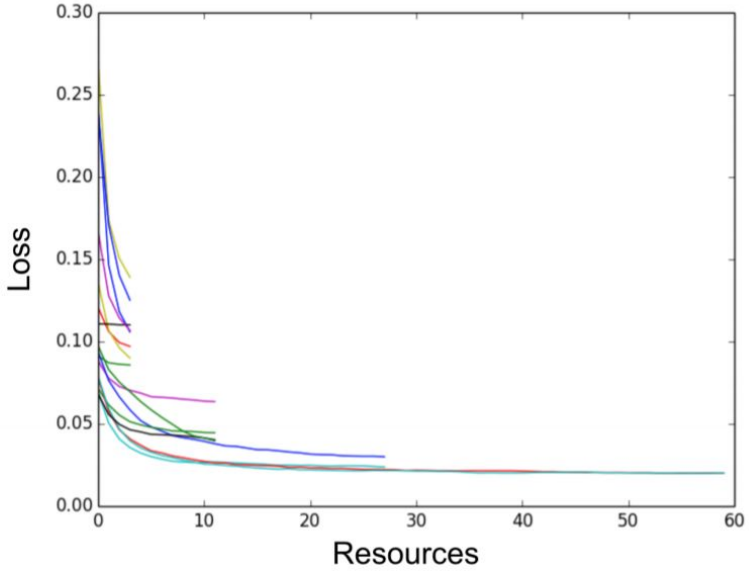
$$p(y|\lambda)\,,$$

And Performs slice sampling over GP (Sonek et al. 2012)

Hyperband



(a) Configuration Selection



(b) Configuration Evaluation

Hyperband

Successive Halving

Hyperband extends Successive Halving (Jamieson and Talwalkar, 2005) and uses it as a subroutine

- Uniformly allocate a budget to a set of hyperparameter configurations
- Evaluate the performance of all configurations
- Throw out the worst half
- Repeat until one configuration remains

The algorithm allocates exponentially more resources to more promising configurations.

Lisha Li et al. (2018) http://jmlr.org/papers/volume18/16-558/16-558.pdf

Hyperband

- **`get_hyperparameter_configuration(n):`** returns a set of *n* i.i.d samples from some distribution defined over the hyperparameter configuration space. Uniformly sample the hyperparameters from a predefined space (hypercube with min and max bounds for each hyperparameter).

- **`run_then_return_val_loss(t, r):`** a function that takes a hyperparameter configuration t and resource allocation *r* as input and returns the validation loss after training the configuration for the allocated resources.

- **`top_k(configs, losses, k):`** a function that takes a set of configurations as well as their associated losses and returns the top *k* performing configurations.

Hyperband: Implementation

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

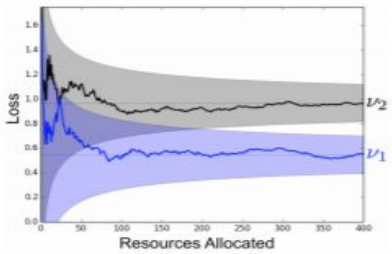
for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with (n, r) inner loop
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Lisha Li et al. (2018) http://jmlr.org/papers/volume18/16-558/16-558.pdf

Finding the right hyperparameter configuration

Takeaways from Figure 2, more resources are needed to differentiate between the two configurations when either:

1. The envelope functions are wider
2. The terminal losses are closer together



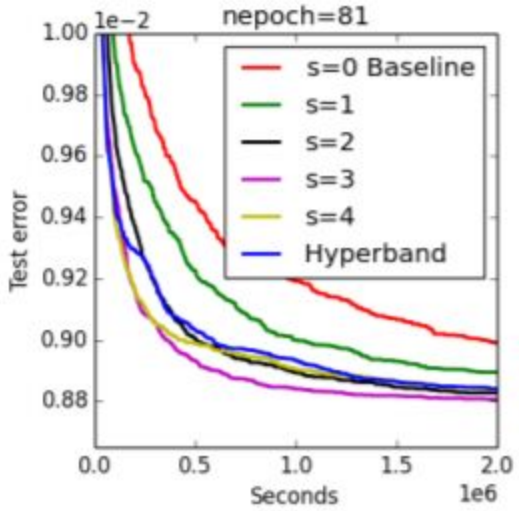
The validation loss as a function of total resources allocated for two configurations is shown. ν_1 and ν_2 represent the terminal validation losses at convergence. The shaded areas bound the maximum distance of the intermediate losses from the terminal validation loss and monotonically decrease with the resource.

Lisha Li et al. (2018) http://jmlr.org/papers/volume18/16-558/16-558.pdf

Example from the Paper: LeNet

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

The values of n_i and r_i for the brackets of HYPERBAND corresponding to various values of s, when $R = 81$ and $\eta = 3$.



Performance of individual brackets s and HYPERBAND.

Example from the Paper: LeNet, Parameter Space

Hyperparameter	Scale	Min	Max
Learning Rate	log	1e-3	1e-1
Batch size	log	1e1	1e3
Layer-2 Num Kernels (k2)	linear	10	60
Layer-1 Num Kernels (k1)	linear	5	k2

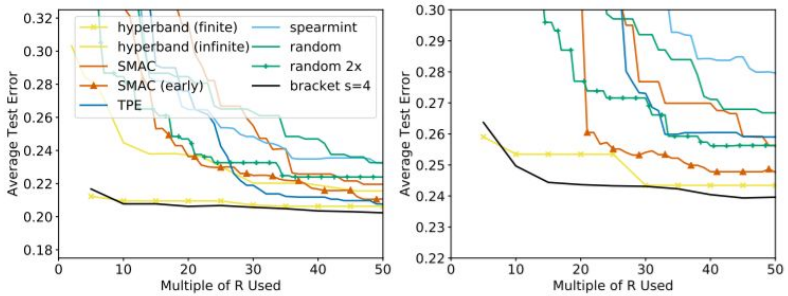
Hyperparameter space for the LeNet application of Section 3.3. Note that the number of kernels in Layer-1 is upper bounded by the number of kernels in Layer-2.

Experiment in the Paper

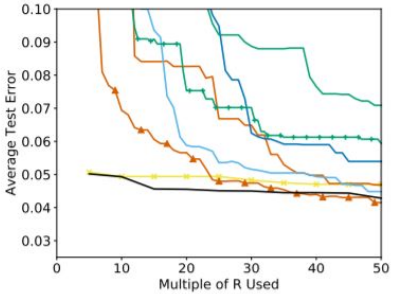
CNN used in Snoek et al. (2012) and Domhan et al. (2015)

Data-sets

- CIFAR-10 (40k, 10k, 10k)
- Rotated MNIST with Background images (MRBI) (Larochelle et al., 2007) (10k, 2k, 50k)
- Street View House Numbers (SVHN) (600k, 6k, 26k)



(a) CIFAR-10 (b) MRBI



(c) SVHN

Keras Tuner: Hyperparameter search

Keras Tuner: Hyperparameter search

- Works with tensorflow/Keras

https://keras-team.github.io/keras-tuner/

Ray Tune:

- Works with XGBoost, Pytorch, Tensorflow/Keras, MXNet, Scikit-Learn

https://docs.ray.io/en/master/tune/index.html

Other Methods: Cyclical Learning Rate

Lesley N. Smith

https://arxiv.org/pdf/1506.01186.pdf

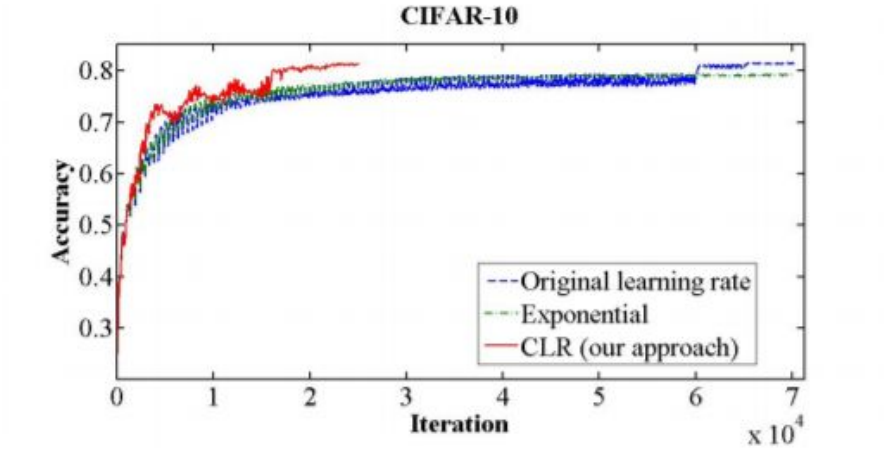


Figure 1. Classification accuracy while training CIFAR-10. The red curve shows the result of training with one of the new learning rate policies.

Cyclical Learning Rate (CLR)

Torch:

```
local cycle = math.floor(1 +
    epochCounter/(2*stepsize))
local x = math.abs(epochCounter/stepsize
    - 2*cycle + 1)
local lr = opt.LR + (maxLR - opt.LR)
    * math.max(0,  (1-x))
```

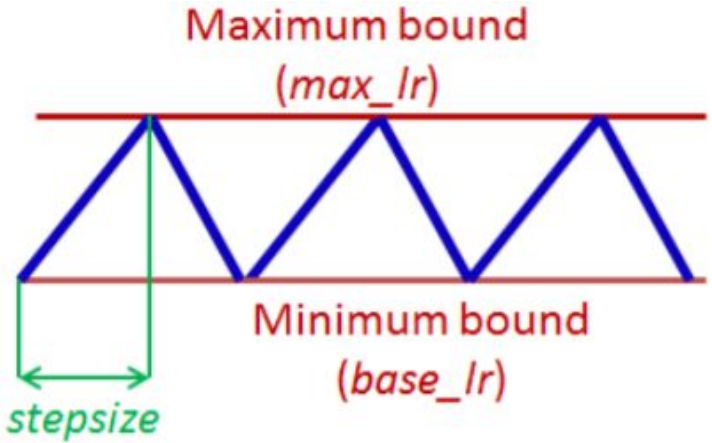


Figure 2. Triangular learning rate policy. The blue lines represent learning rate values changing between bounds. The input parameter *stepsize* is the number of iterations in half a cycle.

Learning Rate Scheduler tf.keras

```
# This function keeps the learning rate at 0.001 for the first ten epochs
# and decreases it exponentially after that.
def scheduler(epoch):
  if epoch < 10:
    return 0.001
  else:
    return 0.001 * tf.math.exp(0.1 * (10 - epoch))

callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
model.fit(data, labels, epochs=100, callbacks=[callback],
          validation_data=(val_data, val_labels))
```

References

Gilberto Batres-Estrada

+46703387868

gilberto.batres-estrada@live.com

Repository https://github.com/gilberto-BE/deep_learning_italia

Cyclical Learning Rate: https://arxiv.org/pdf/1506.01186.pdf

Random Search: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Keras tuner: https://keras-team.github.io/keras-tuner/

Learning Rate Scheduler: fastai (pytorch high level API) https://docs.fast.ai/callbacks.one_cycle.html

Source code for Hyperband: https://github.com/keras-team/keras-tuner/blob/master/kerastuner/tuners/hyperband.py