A Multi-Armed bandit approach to Hyperparameter Tuning

Bhishma Dedhia
Swadha Sanghvi
Santanu Rathod

EE 761 - Advanced Concentration Inequalities

Indian Institute of Technology Bombay

Motivation

- Every model contains a set of parameters and hyperparameters.
- Parameters (essentially the weights) are optimized during training for best performance.
- Hyperparameters are design choices that define the architecture and optimization process to be used.
- Hyperparameters are carefully tuned by practitioners. This is a time consuming and fairly empirical process.

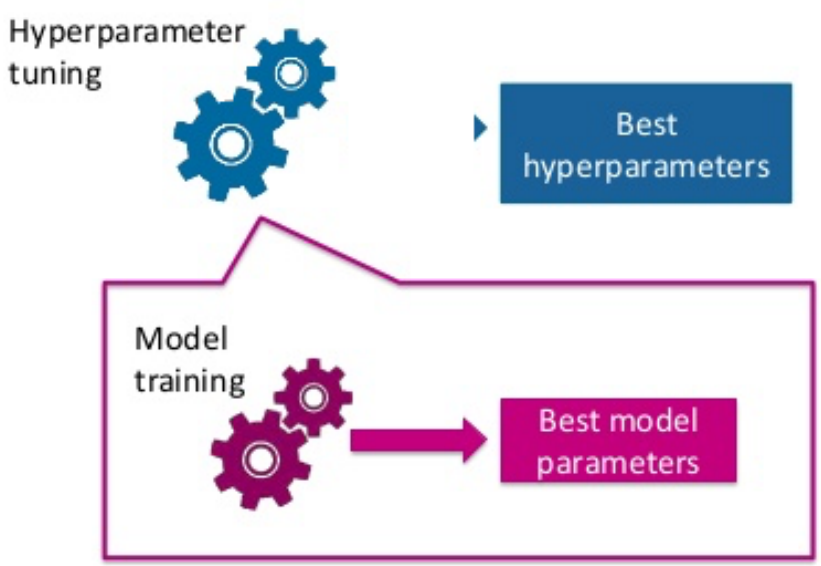


Figure: Hyperparameter tuning vs Model training (Source)

Current Practices

- **Manual search**
- **Grid search:** Divide hyperparameter space into a grid and sequentially sample points on the grid. Suffers from the curse of dimensionality!
- **Random search**
- **Bayesian Optimization:** Start with a prior over the space and update belief after sampling.
- **Gradient based approach:** Use gradient of validation accuracy with respect to hyperparameter.


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Issues and Challenges

Optimal settings for hyperparameters are rarely known a priori and their settings are critical to our models' ability to learn from data. The major issues are:

- As the number of model hyperparameters increases, we face a combinatorial increase in potential model configurations.
- There is an increased chance that models' hyperparameters interact in complex manners that are intuitively unexplainable as the hyperparameters increase in numbers.

Moreover, we don't just want the best arm configuration but also an insight into how the performance changes as the configurations are perturbed.

Hyperband: Hyperparameter Optimization

Hyperband: Hyperparameter Optimization

Hyperband performs a geometric search in the average budget per configuration and removes the need to select n for a fixed budget by exploring across different values of n

Hyperband: Hyperparameter Optimization

Algorithm 1 Hyperband algorithm[1]

1: *Input:* η, R
2: *Initialize:*
3: $s_{max} = \log_\eta R, B = (s_{max} + 1)R$
4: *For* $s \in \{s_{max} \cdots, 0\}$ do
5: $n = \frac{B}{R}\frac{\eta^s}{s+1}, r = R\eta^{-s}$
6: $T =$ get_hyperparameter_configuration(n)
7: For i $\in \{s \cdots, 0\}$ do
8: $n_i = n\eta^{-i}$
9: $r_i = r\eta^i$
10: L $= \{$get_accuracy(t, r_i) : $t \in T\}, T = top\_k\left(T, L, \frac{n_i}{\eta}\right) -$
11: end
12: end
13: Return the arm with the highest accuracy

A Bayesian Optimization based approach

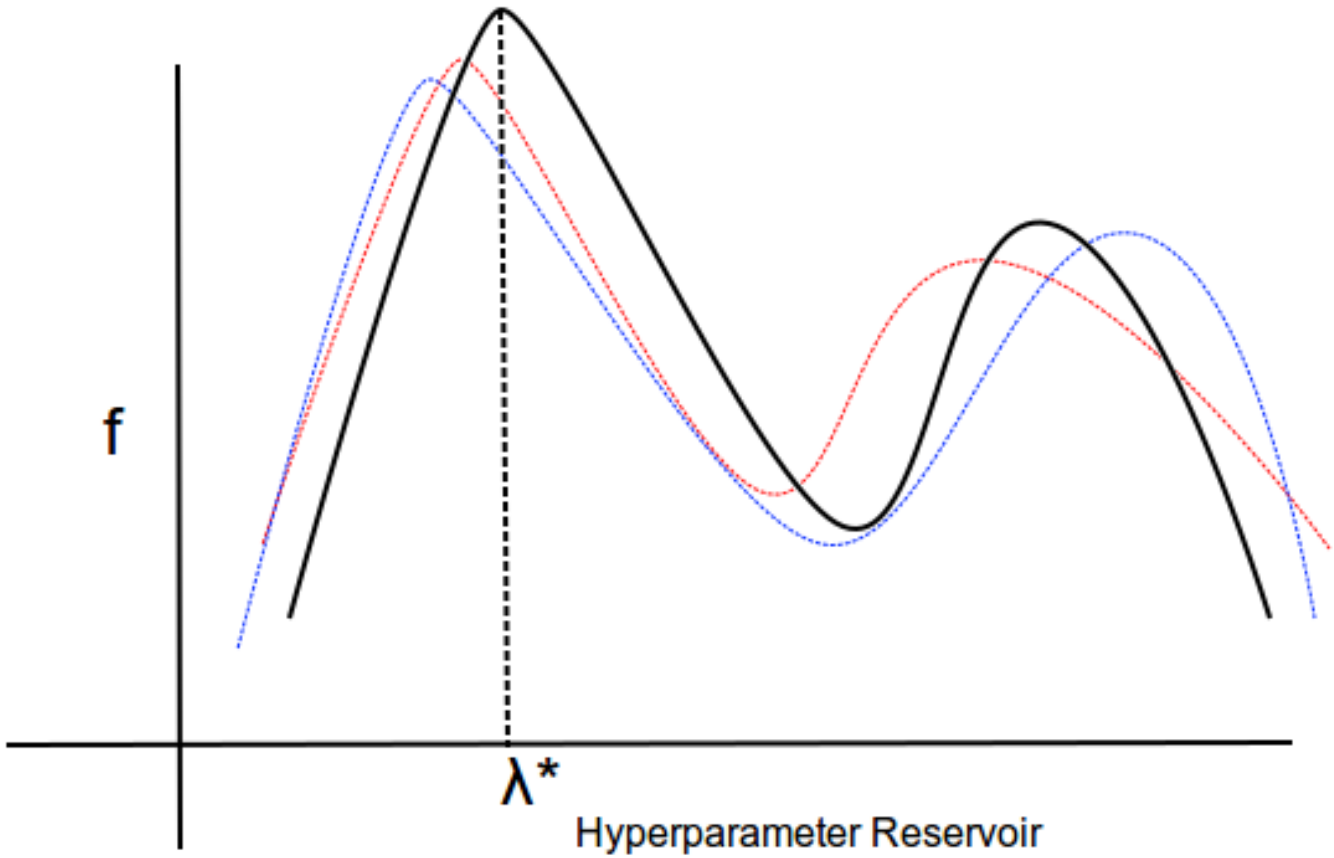
Hyperparameter Optimization objective:

Let Ω be the hyperparameter space, f be a mapping from Ω to a measure of success of an algorithm trained over the configuration. Then we want,

$$\lambda^* = \underset{\lambda}{\text{argmax}}\, f(\lambda)$$

We take f as the validation accuracy. It can also be viewed as the reward for an arm. Explicit evaluation of f is impossible and we have access only to its noisy evaluations.
Hence the goal of a hyperparameter optimizer is to guess configuration $\hat{\lambda^*}$ such that $f(\hat{\lambda^*})$ is not far away from $f(\lambda^*)$

Hyperparameter Optimization objective



Top Two Thompson Sampling (TTTS)

- Extension of Thompson Sampling for best arm identification among finite set of arms.[2]
- Has been shown to be effective in cases where sampling is time consuming and produces only noisy output.
- The algorithm computes the two designs with the highest posterior probability of being optimal, and then randomizes to select among these two.
- **Main result**: There exists an optimal randomizing bernoulli random variable β such that in a Bayesian (asymptotic) sense, it attains the best possible rate of exponential decay of the posterior probability of the set of wrong models

[2] Russo, Daniel *CoRR* 2016.

Top Two Thompson Sampling (TTTS)

Algorithm 2 Top Two Thompson Sampling for K arms

Input: β
Initialize:
    $\Pi_0 \leftarrow \bigotimes_{i=1}^{k} U(0,1)$, For all k: $S_k \leftarrow 0, L_k \leftarrow 0$
For t in 1 $\cdots$ T:
    Sample $\Theta \sim \Pi_{t-1}$
    $I_t^{(1)} = \text{argmax}_{i \in K}\, \Theta_i$
    While $I_t^{(1)} \neq I_t^{(2)}$:
        Sample $\Theta^{'} \sim \Pi_{t-1}$
        $I_t^{(2)} = \text{argmax}_{i \in K}\, \Theta_i^{'}$
    $\text{Arm}_\text{t} = I_t^{(1)}$ with Prob β, $I_t^{(2)}$ with Prob $1-\beta$
    Play Arm_t to get reward R, $S_{Arm_t} \mathrel{+}= R, L_{Arm_t} \mathrel{+}= 1 - R$
    $\Pi_t \leftarrow \bigotimes_{i=1}^{k} Beta(S_k + 1, L_k + 1)$
End for

Dynamic Top Two Thompson Sampling (D-TTTS)

- Shang et. al.[3] proposed a a new adaptive algorithm inspired by TTTS for best arm identification in an infinitely many armed bandit.
- The algorithm dynamically balances between refining the estimate of the quality of arm configurations previously explored and adding arm configurations to the pool of candidates. TTTS is invoked as a subroutine while exploting arms.
- Hence using D-TTTS:
  1) We sample new hyperparameters from the reservoir at every step (exploration) and add it to the sample pool.
  2) We refine the estimates of the sample pool (exploitation) using TTTS.

[3] Shang, Xuedong, Emilie Kaufmann, and Michal Valko 2019.

D-TTTS: A slight modification

- D-TTTS is conservative in exploring and adds only one configuration to the sample pool at every time step. Exploration may not be enough when model complexity is high.
- Hence we generalize D-TTTS where a user gets to decide the number of configurations to be added to the sample pool (Exploration count) and number of configurations to be re-sampled at every time step (Exploitation count).
- We call our generalized version as **D-TTTS with forced exploration**

D-TTTS

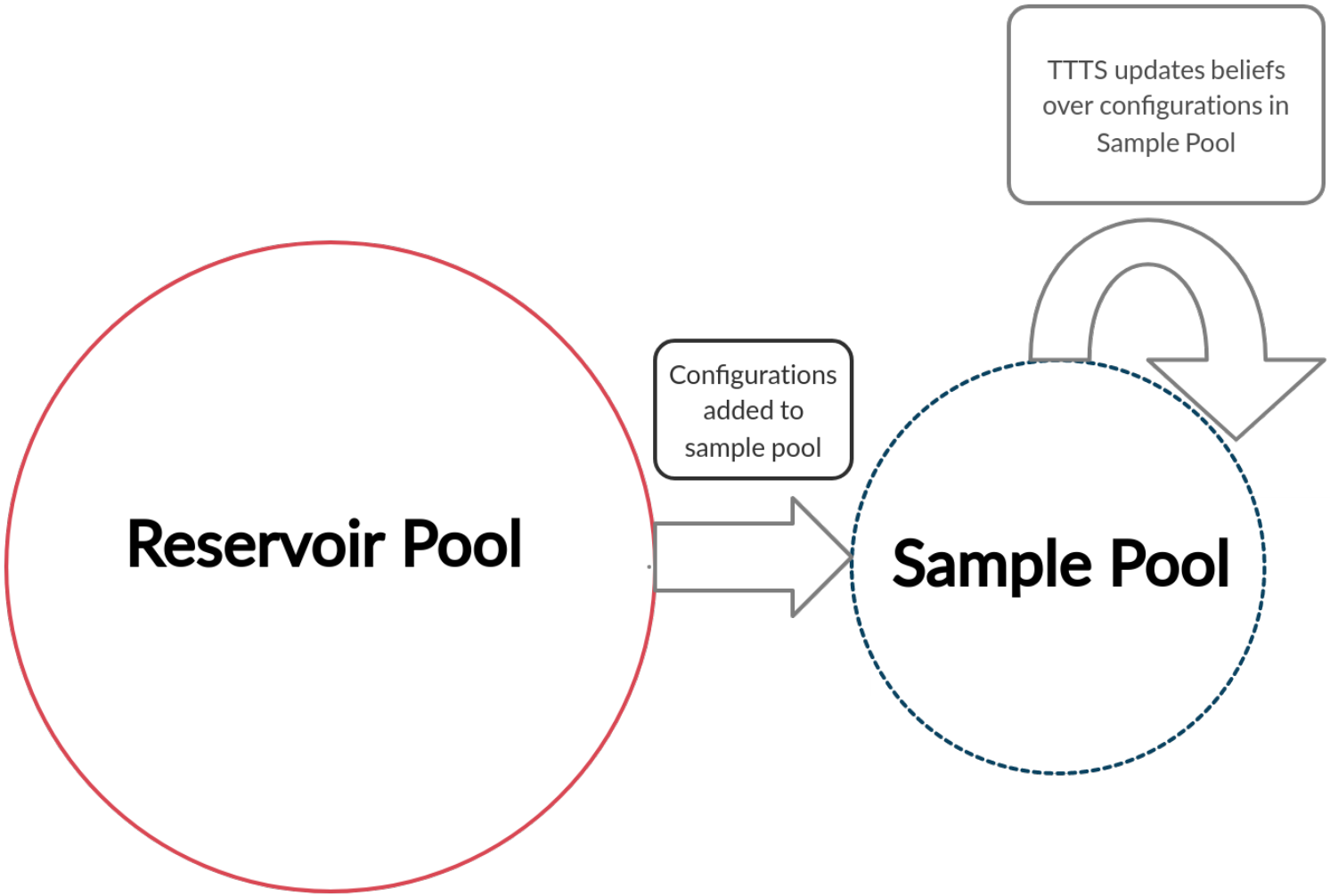


Figure: D-TTTS visualized

D-TTTS with Forced Exploration

Algorithm 3 D-TTTS for infinite armed bandit

Input: β, *Horizon, Exploration Count, Exploitation count*
Initialize:
    Reservoir_pool$\leftarrow \Omega$, Sample_pool$\leftarrow \emptyset$
For t in 1 $\cdots$ *Horizon*:
    For i in 1 $\cdots$ *Exploration Count*:
        Select model M uniformly from reservoir space
        Reservoir_pool.pop(M), Sample_pool.append(M)
        Initialize TTTS parameters for M, Sample M
    End for
    For i in 1 $\cdots$ Exploitation Count:
        Run_TTTS(Sample Pool)
    End for
End for
Best Configuration $\leftarrow$ $argmax_{i \in Sample_pool}$mean reward^i

Gaussian Process Regression

- Gaussian Process allows for a form of non parametric regression. It finds a distribution over the possible functions f(x) that are consistent with the observed data.
- It models the data points as jointly gaussian and returns the posterior mean and variance for query points. A covariance function measures the influence of observed data on the query points.

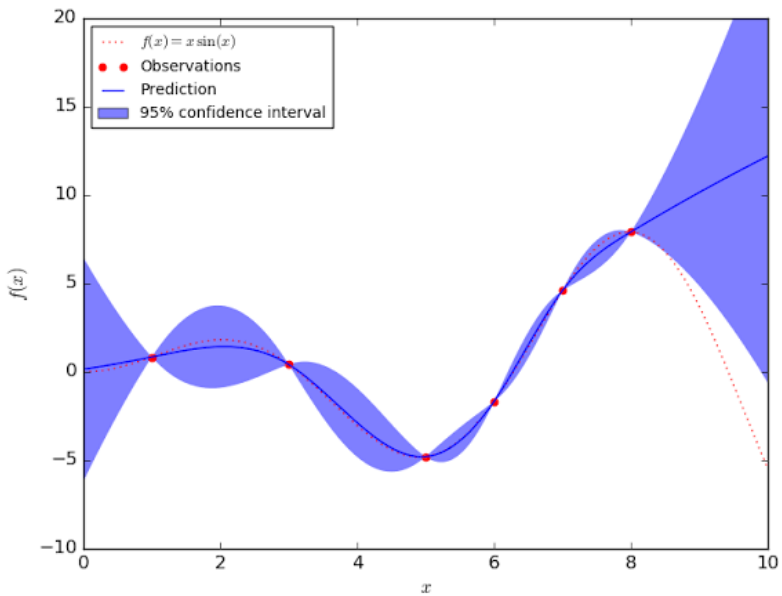


Figure: GP regression over sin(x) (Source)

Accuracy Prediction as GP regression

- We utilize an accuracy predictor to estimate the final accuracy of a model without actually training it. This can significantly speed up best configuration evaluation.
- Dai et al.[4] mention two benefits of using a GP regressor for accuracy prediction:
  - It offers the most reliable predictions when training data is scarce. This has been experimentally shown by Dai et al. by comparing GP regression to 6 other regression models.
  - A GP regressor produces predictions with uncertain estimations. This offers additional guidance for new sample architecture selection for training. This helps boost the convergence speed and improves sample efficiency
- Hence we can train the accuracy predictor over the sample pool and use it to evaluate any configuration in the reservoir pool.

[4] Dai, Xiaoliang et al. 2018.

Accuracy Prediction as GP regression

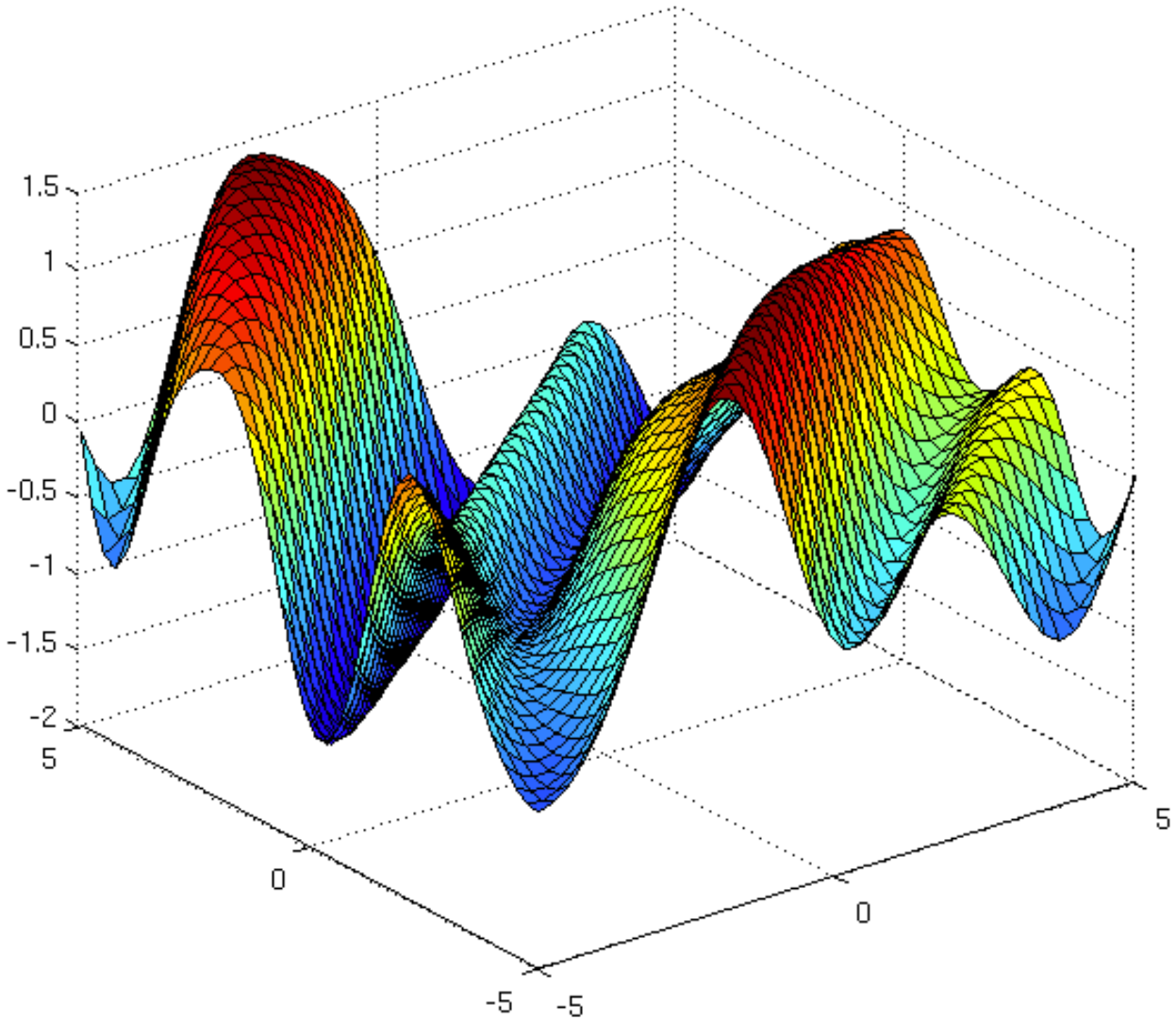


Figure: Accuracy Predictor visualized (Source)

D-TTTS meets Accuracy Predictor

- D-TTTS provides good guarantees on the sample pool but is limited in the exploration it does on the reservoir pool in a fixed horizon.
- Accuracy predictor has the ability of training itself on sample pool configurations in an online manner and provides an insight into the reservoir space.
- Hence we introduce the accuracy predictor as a feedback loop to the D-TTTS algorithm to guide forced exploration. Query points in reservoir space having higher predicted uncertainty (or variance) are added to the sample space.

D-TTTS with Accuracy Predictor in feedback loop

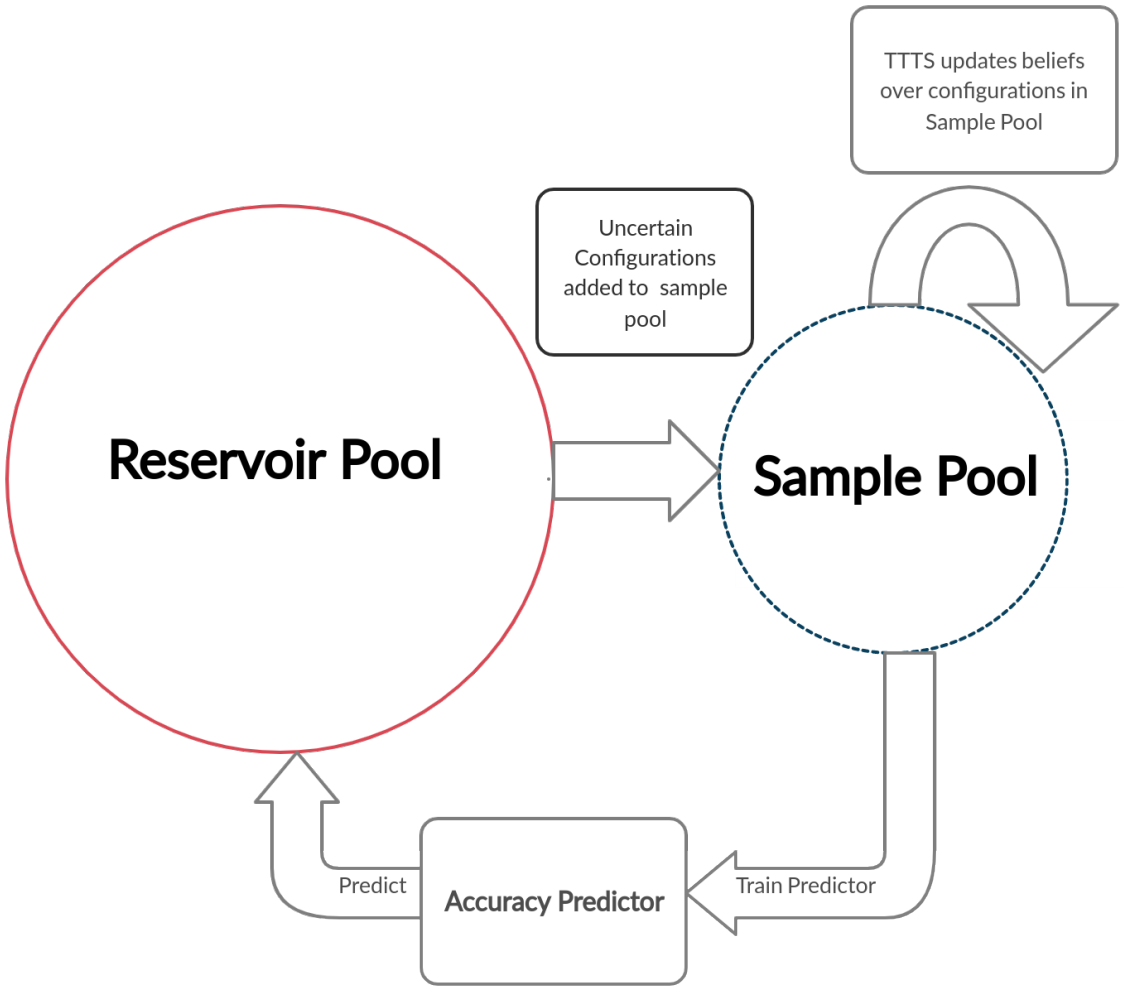


Figure: D-TTTS with Predictor in feedback loop visualized

D-TTTS with Predictor in the Loop

Algorithm 4 D-TTTS with Predictor

Input: β, *Horizon, Exploration Count, Exploitation count*
Initialize: Reservoir_pool$\leftarrow \Omega$, Sample_pool$\leftarrow \emptyset$, AccuracyPredictor
For t in 1 $\cdots$ *Horizon*:
    For i in 1 $\cdots$ *Exploration Count*:
        Select model M = $\text{argmax}_{i \in \text{Reservoir\_pool}}$ $Var(AccuracyPredictor(i))$
        Reservoir_pool.pop(M), Sample_pool.append(M)
        Initialize TTTS parameters for M, Sample M
    End for
    For i in 1 $\cdots$ Exploitation Count:
        Run_TTTS(Sample Pool)
    End for
    Train AccuracyPredictor
Best Sample_Pool Configuration $\leftarrow$ $argmax_{i \in Sample_pool}$mean reward^i
Best Reservoir_Pool Configuration $\leftarrow$ $argmax_{i \in Reservoir_pool}$mean(i)

Implementation Tricks

- **Binarization:** The reward (validation accuracy) observed $Z_t \in [0, 100]$. The algorithm is updated with a fake binary variable $Y_t \sim Ber(Z_t/100) \in \{0, 1\}$
- **Generating Reservoir Pool:** We generate the configurations in the Reservoir Pool using sobol sequence (Quasi Random) sampling. They explore the space more uniformly than pseudo Random number generators. Reservoir pool has $\sim$ 5000 configurations.

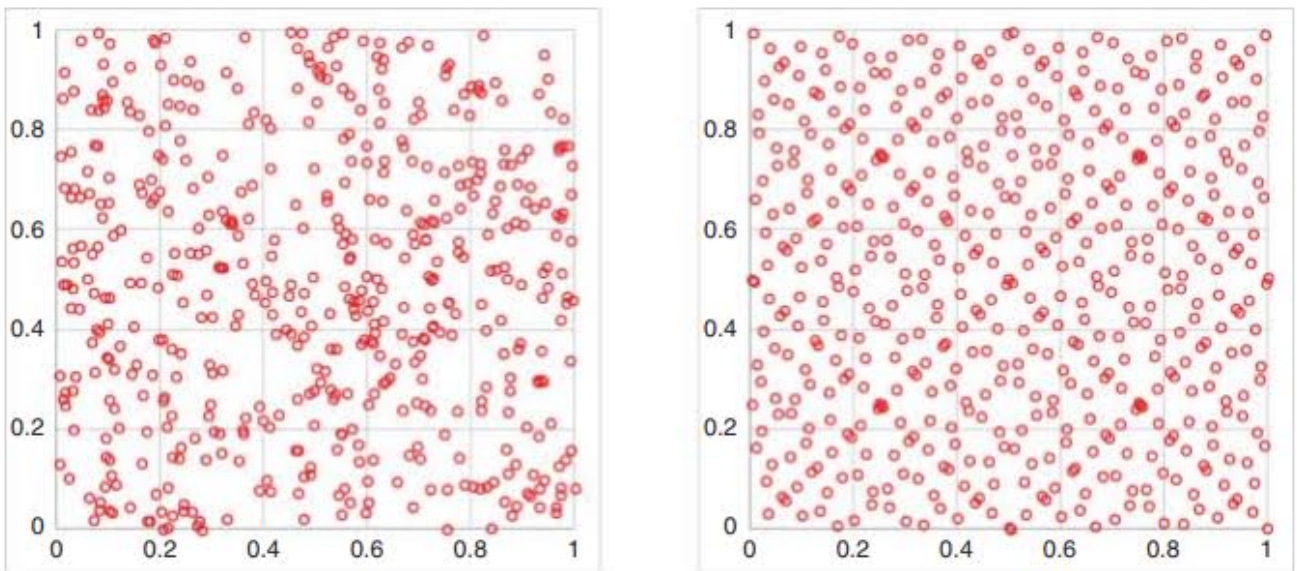


Figure: Random Sampling vs Quasi Random Sampling

Results: Classification using Dense Neural Nets

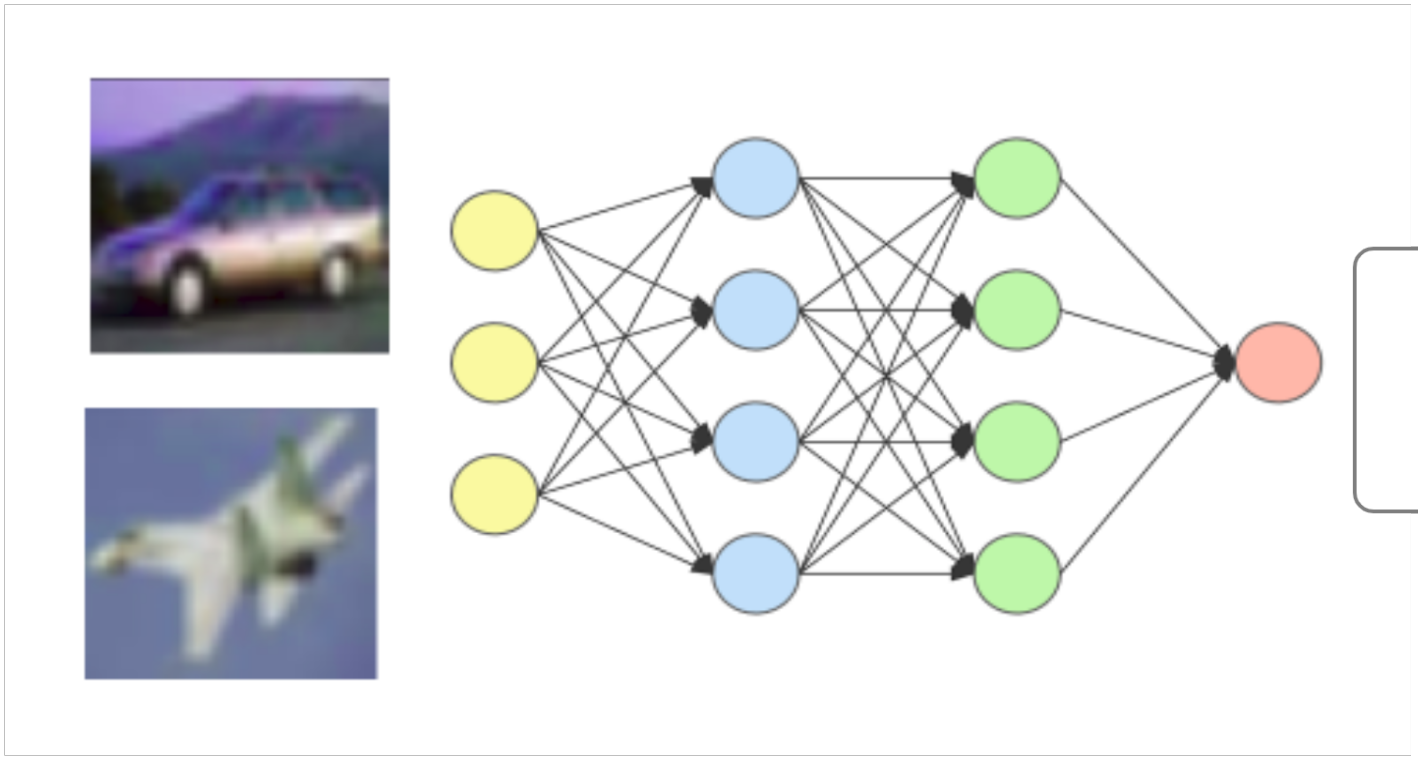


Figure: Architecture of DNN

Results: Classification using Dense Neural Nets

Hyperparameter space:

- Hidden layer 1 : 10 - 100 neurons
- Hidden layer 2 : 5 - 40 neurons
- Learning Rate: $10^{-3} - 10^{-7}$

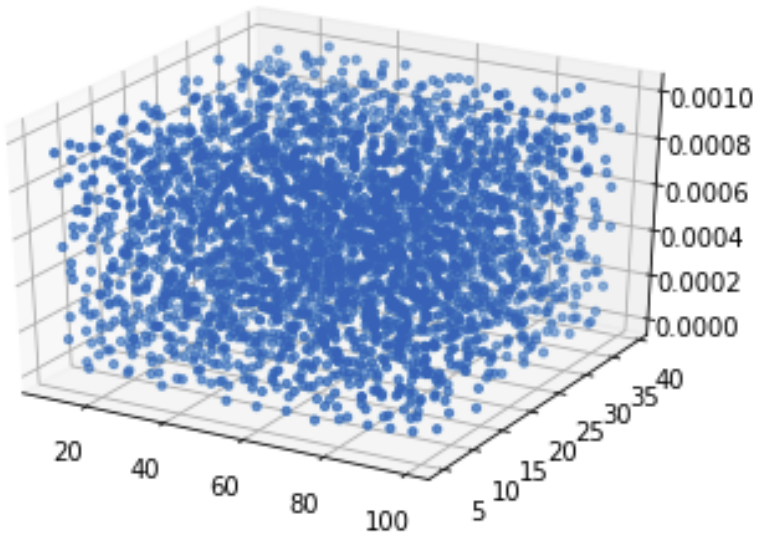


Figure: Reservoir Pool

Horizon: 100 iterations, Exploration count $= 3$, Exploitation Count $= 2$, $\beta = 0.9$

Results: Classification using Dense Neural Nets

D-TTTS with Forced exploration:

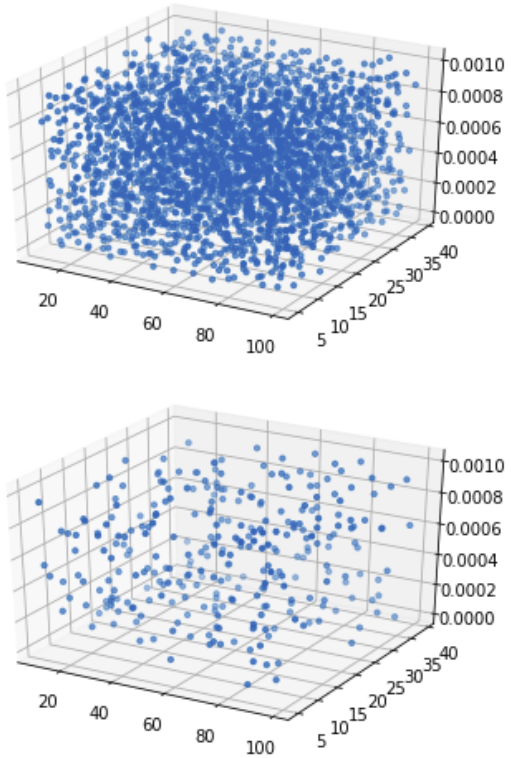


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: Hidden layer 1 = 89 neurons, Hidden layer 2 = 7 neurons, Learning rate = 9.7×10^{-4}. Validation accuracy = 89%

Results: Classification using Dense Neural Nets

D-TTTS with Predictor in the loop:

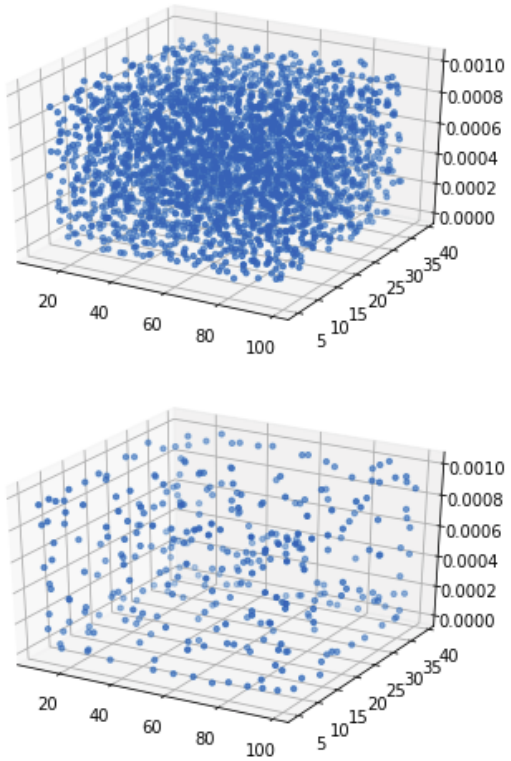


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: Hidden layer 1 = 55 neurons, Hidden layer 2 = 11 neurons, Learning rate = 9.9×10^{-4}. Validation accuracy = 91.9%

Results: Classification using Dense Neural Nets

Geometric Search

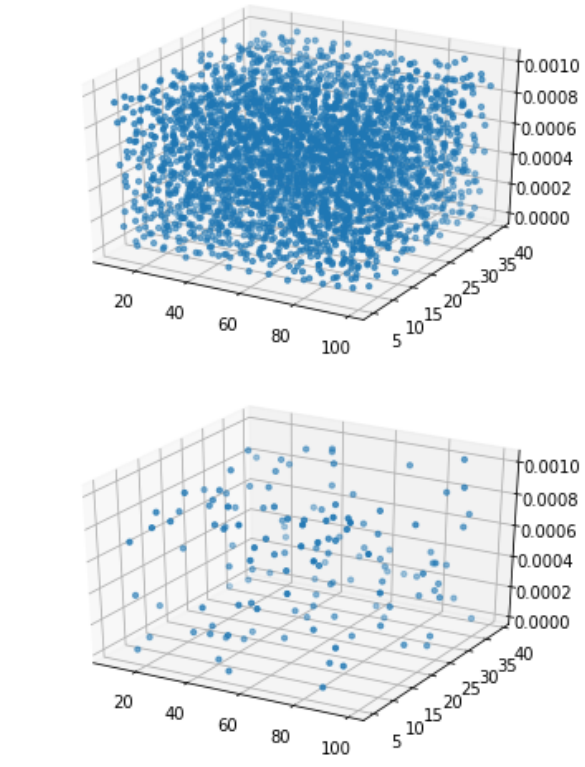


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: Hidden layer 1 = 98 neurons, Hidden layer 2 = 19 neurons, Learning rate = 8.3×10^{-4}. Validation accuracy = 88.6%

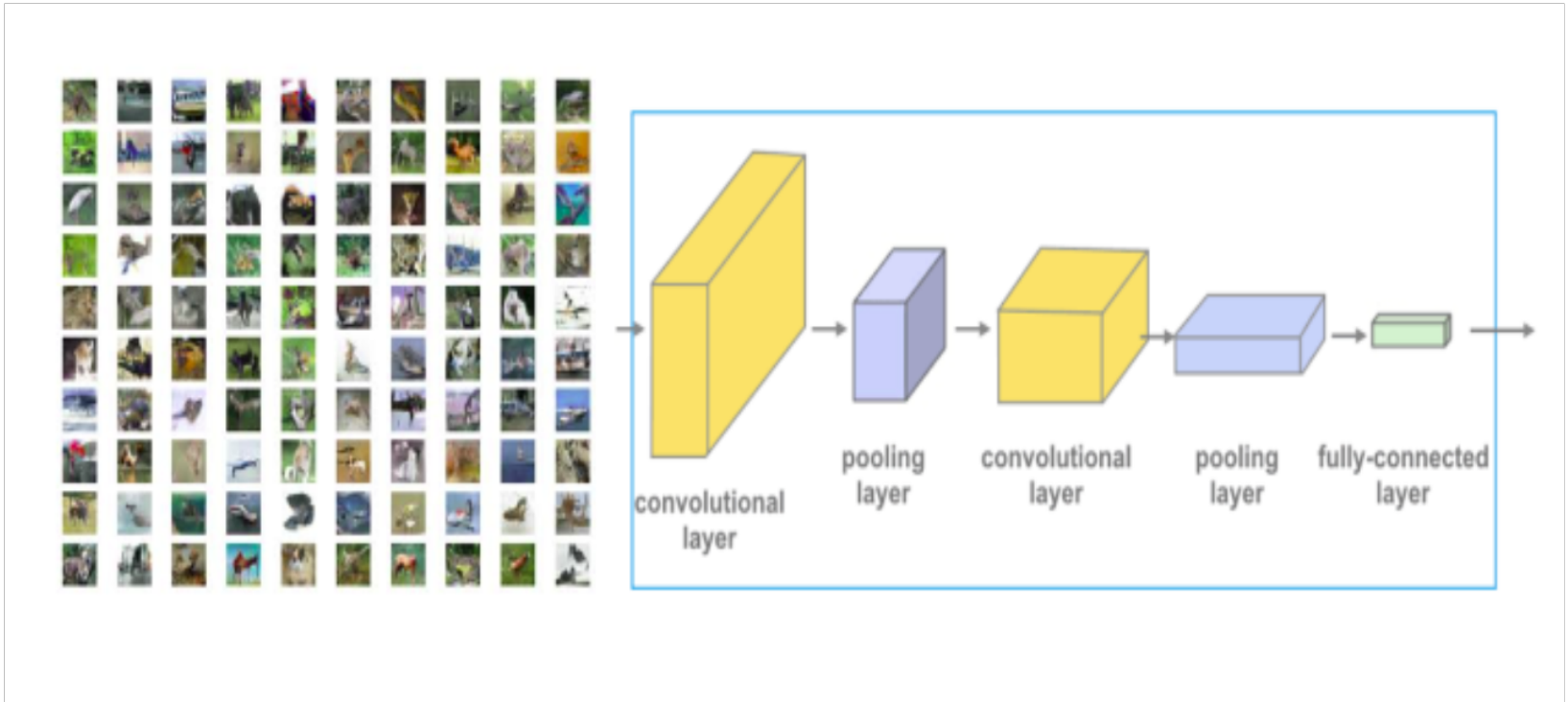
Results: Classification using Dense Neural Nets

Accuracy Predictor works fairly well!

Predicted Accuracy	Actual Accuracy
81.9%	82.1%
67.9%	69.1%
71.3%	71.6 %
77 %	79.4%
87.5%	85.9%

Table: Predicted vs Actual accuracy of some configurations

Results: CIFAR10 Classification using CNNs



Hyperparameters:
1) No. of Feature map layer 1
2) No. of Feature map layer 2
3) Learning rate

Figure: Architecture of CNN

Results: CIFAR10 Classification using CNNs

Hyperparameter space:

- Feature maps in layer 1 : 2 - 10 maps
- Feature maps in layer 2: 5 - 20 maps
- Learning Rate: $10^{-2} - 10^{-6}$

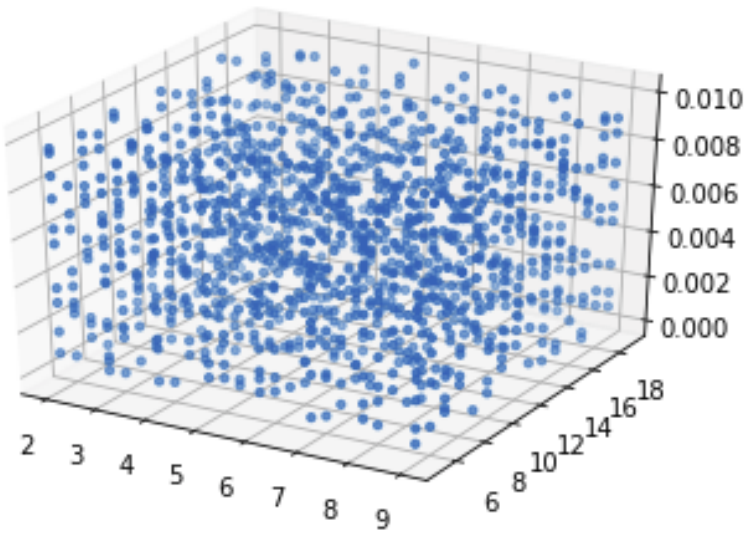


Figure: Reservoir Pool

Horizon: 100 iterations, Exploration count $= 3$, Exploitation Count $= 2$, $\beta = 0.9$

Results: CIFAR10 Classification using CNNs

D-TTTS with Forced exploration:

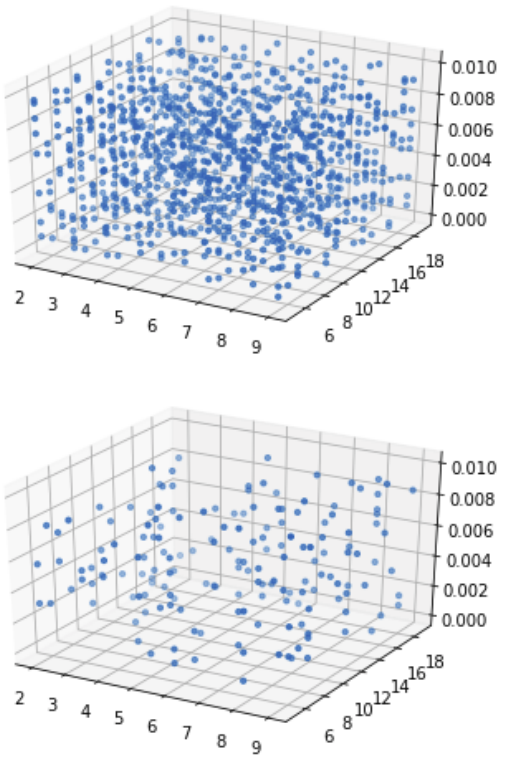


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: No. of Feature maps in Layer 1 = 8 neurons, No. of Feature maps in Layer 2 = 19 neurons, Learning rate = 8.7×10^{-3}.
Validation accuracy = 59.3%

Results: CIFAR10 Classification using CNNs

D-TTTS with Predictor in the loop:

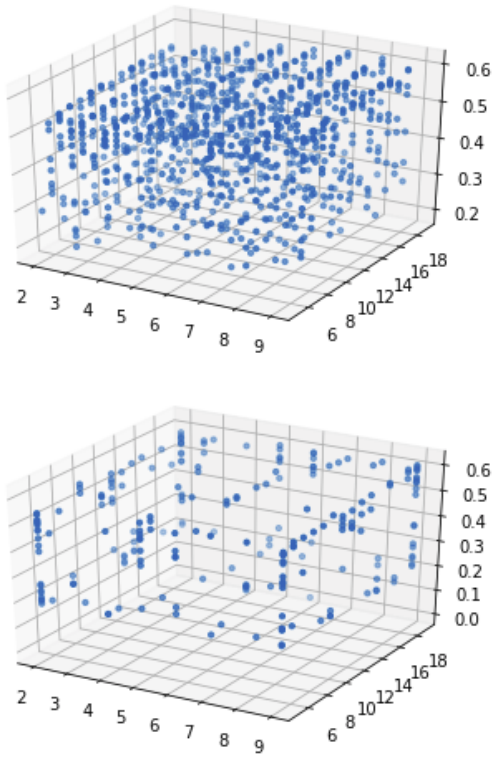


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: No. of Feature maps in Layer 1 = 9 neurons, No. of Feature maps in Layer 2 = 18 neurons, Learning rate = 9.4×10^{-3}.
Validation accuracy = 60.93%

Results: CIFAR10 Classification using CNNs

Geometric search

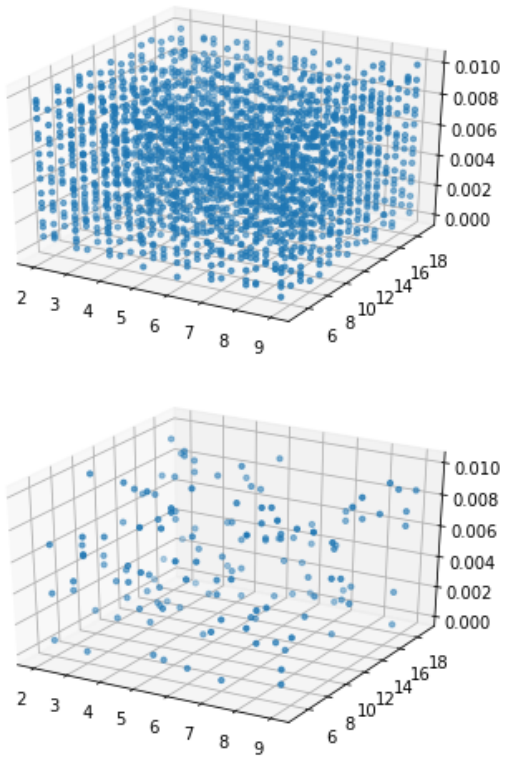


Figure: Reservoir Pool and Sample pools after end of horizon

Best Configuration: No. of Feature maps in Layer 1 = 9 neurons, No. of Feature maps in Layer 2 = 15 neurons, Learning rate = 8.9×10^{-3}.
Validation accuracy = 58.95%

Results: CIFAR10 Classification using CNNs

Accuracy Predictor works fairly well in this case too!

Predicted Accuracy	Actual Accuracy
52.09%	55.54%
57.66%	57.69%
35.7%	32.9 %
46 %	48.92%
52.8%	51.72%

Table: Predicted vs Actual accuracies of some configurations

Future Work

Hyperparameter optimization using finite arms

- Instead of infinte arms to begin with, we explored the possibility of sampling one hyperparameter at a time, and it was primarily motivated to remove the infinte arm consideration in the beginning.
- In this method, instead of considering rewards from each arm we consider the deviation in the prediction accuracy.
- Somewhat inspired by Thompson sampling, we model the expectation of our hyperparameters using a Beta(α, β) distribution and expect it to trickle down to stability eventually.

Figure: Accuracy variation for system with two hyperparameters(h_1, h_2)

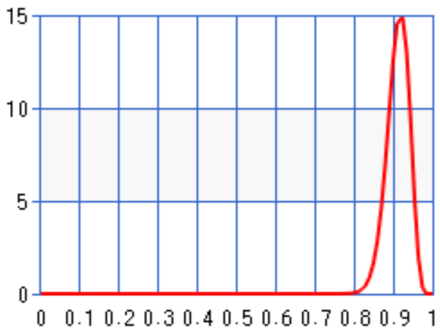
Hyperparameter optimization using finite arms

Algorithm 5

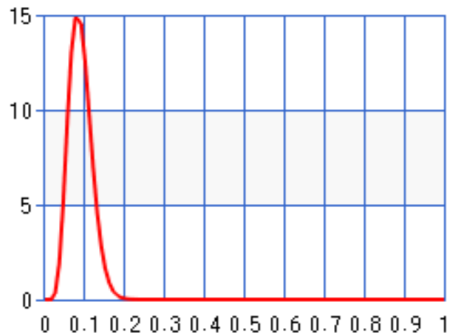
Input: ϵ, n= *no_of_hyperparameters*
Initialize: $\alpha \in R^n, \beta \in R^n$
For t in 1 $\cdots$ *Horizon*:
    accuracy_deviation = []
    For i in 1 $\cdots$ *no_of_hyperparameters*:
        $\delta_1 = |Accuracy(h/h[i] \cup h[i] + \epsilon) - Accuracy(h/h[i] \cup h[i] - \epsilon)|$
        accuracy_deviation.append(δ_1)
    arm = max_index(accuracy_deviation)
    acc_left= Accuracy(h:h[arm]$\sim$Beta(α[arm], β[arm]+1))
    acc_right= Accuracy(h:h[arm]$\sim$Beta(α[arm]+1, β[arm]))
    δ_2= acc_left$-$acc_right
    if ($\delta_2 > 0$):
        $\alpha[arm] += 1$
    else: $\beta[arm] += 1$

Variations in Beta distribution

Beta(α, β)

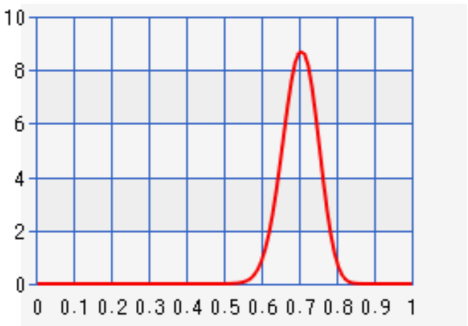


(a) $\alpha = 100, \beta = 10$

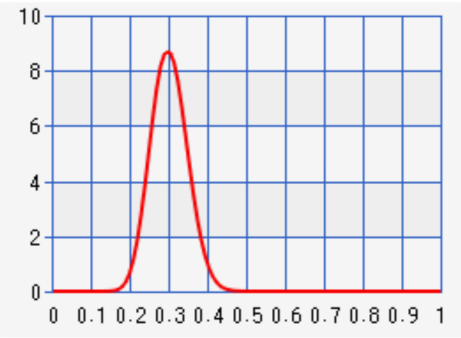


(b) $\alpha = 10, \beta = 100$

Figure: Beta shift comparision



(a) $\alpha = 70, \beta = 30$



(b) $\alpha = 30, \beta = 70$

Insights

- As we see from the previous slide the PDF of Beta distribution is highly sensitive to the values of α, β.
- So with iterations the PDF is expected to trickle down to optimum distribution wrt system.
- However we didn't see such a behaviour exactly, mainly because the sensitivity of the system is different wrt to each hyperparameter, hence in our observation the arm pulling was skewed towards the hypersensitive one.
- Tackling that issue is left for future studies.

Any Questions?

Dai, Xiaoliang et al. "ChamNet: Towards Efficient Network Design through Platform-Aware Model Adaptation". In: *CVPR*. 2018.

Li, Lisha et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2016. arXiv: 1603.06560 [cs.LG].

Russo, Daniel. "Simple Bayesian Algorithms for Best Arm Identification". In: *CoRR* abs/1602.08448 (2016). arXiv: 1602.08448. URL: http://arxiv.org/abs/1602.08448.

Shang, Xuedong, Emilie Kaufmann, and Michal Valko. "A simple dynamic bandit algorithm for hyper-parameter tuning". In: *Workshop on Automated Machine Learning at International Conference on Machine Learning*. AutoML@ICML 2019 - 6th ICML Workshop on Automated Machine Learning. Long Beach, United States, June 2019. URL: https://hal.inria.fr/hal-02145200.