IOWA STATE UNIVERSITY
Department of Computer Science
COMS 641: Advanced Topics in Programming Languages

Group: Certainty

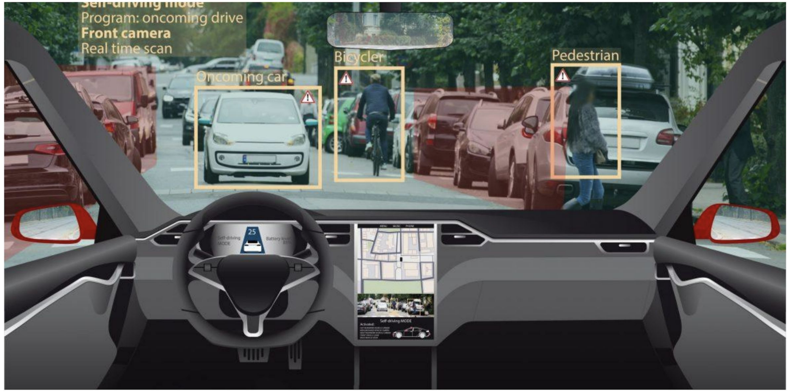
Quantifying Uncertainty of DNN Hyperparameter Optimization using a First Order Type

Sumon Biswas, Sayem Mohammad Imtiaz, Yuepei Li

December 17, 2019

Motivation

How heavy the brakes should be applied?

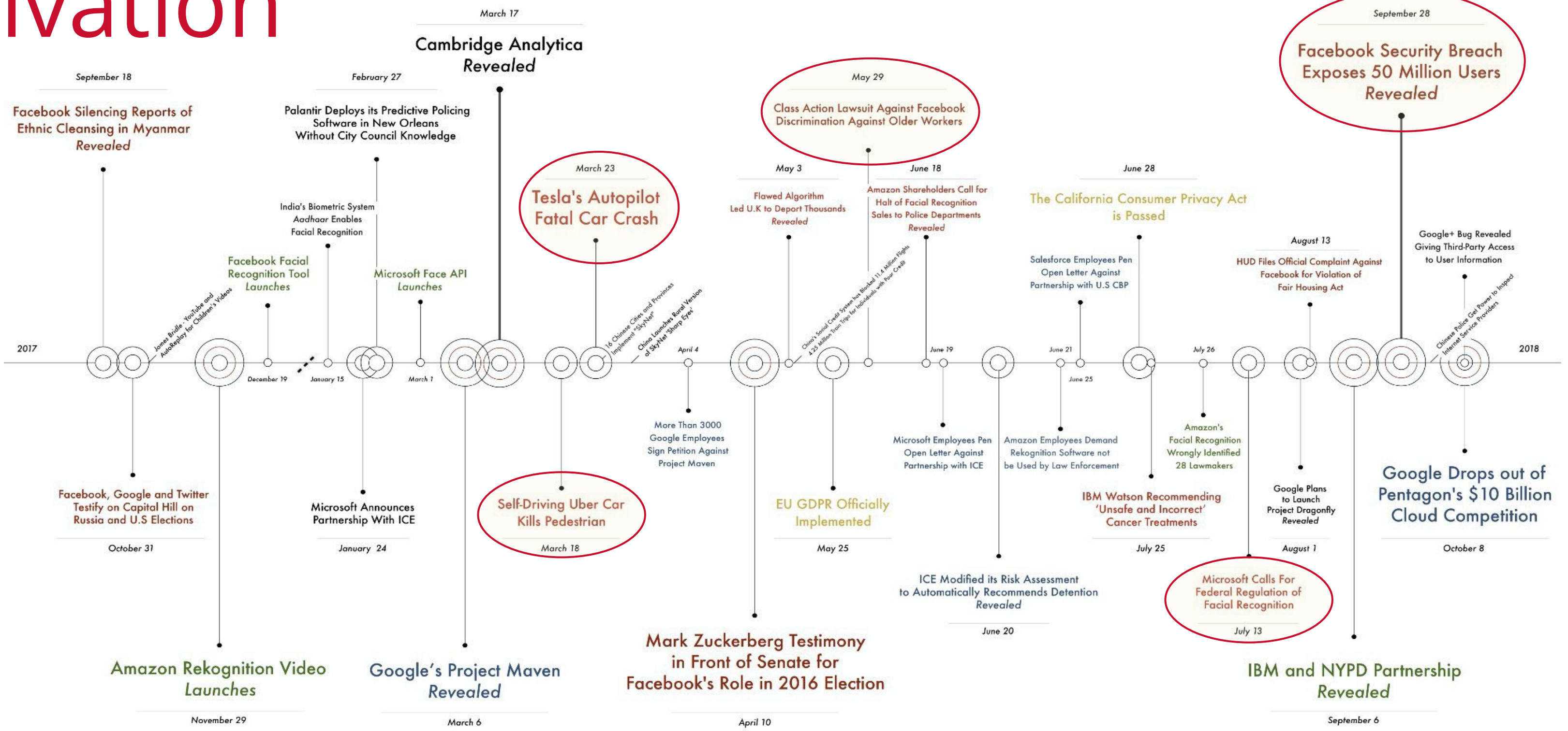


We build models for predictions. Are they certain?

Motivation

- "If I had to guess at what our biggest existential threat is, it's probably AI. So we need to be very careful" - made the comments in an interview in MIT

Motivation



"Right" answer is usually ambiguous

@VaroonMathur, Technology Fellow

Overview

DNN hyperparameter optimization is difficult

Random search strategy has been proven efficient

However, in random search, programmers can not overtly represent uncertainty

We utilize a first order type Uncertain<*T*> to approximate the distributions of the hyperparameters

Contribution

No attempt has been made to leverage probabilistic programming in DNN hyperparameter optimization.

1. Implemented first order type Uncertain<*T*> to hold the distribution of loss values over randomly chosen hyperparameters. The main goal is to help programmers overtly represent uncertainty and write conditional statements.
2. Our method performs significantly better than the random search method
   a. less training data needed
   b. converges quickly
   c. allows programmers to impose confidence over the accuracy

Background

- DNN hypermeter optimization
  - Aims to find best value of the hyperparameters which helps
    - Converge faster
    - Minimize loss function
  - E.g., find the optimal number of hidden units, learning rate etc. in a DNN

- Random search strategy
  - Grid search evaluates exhaustively, which is time-consuming
  - Random search evaluates manageable number of trials picked uniformly

Problem Statement

- If λ is a hyperparameter, then random search draws uniformly from possible trials of $\lambda = \{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ and minimizes loss function:

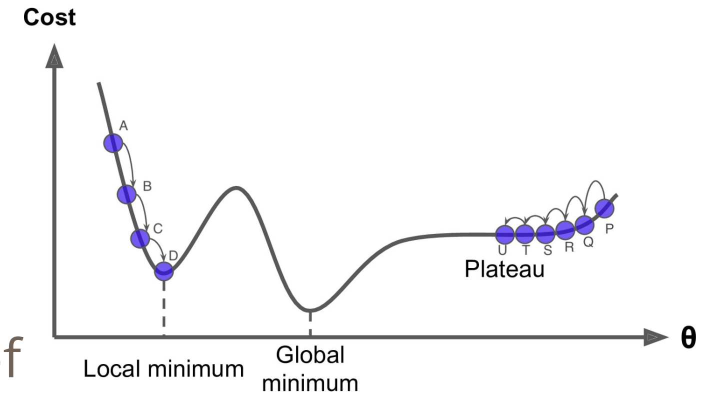
$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, Loss_\lambda(x)$$

- A particular choice of hyperparameter might not work optimally on distribution of X

- What is the uncertainty in choosing optimizing hyperparameters with a certain confidence level?

Motivating Example

- Gradient descent algorithm is widely used as an optimizer in DNN
- A gradient descent works by moving downhill of the loss function curve
- Uncertainty in random initialization of parameters have varying impacts on the cost function

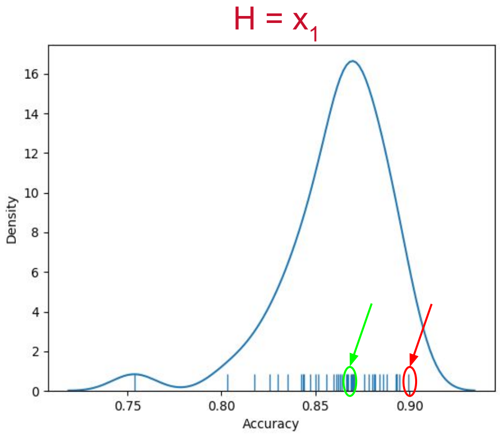


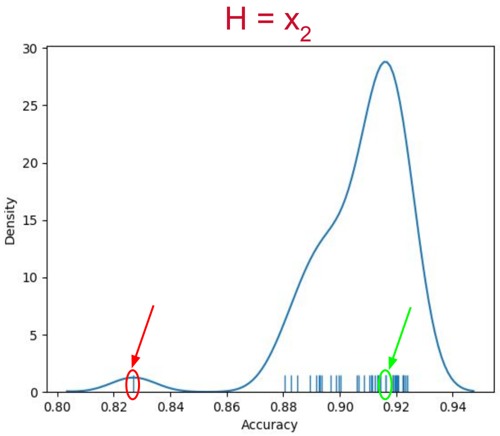
Solution Idea

Random search finds best hyperparameter value

A particular value might not produce best result on unseen wider population (uncertainty)

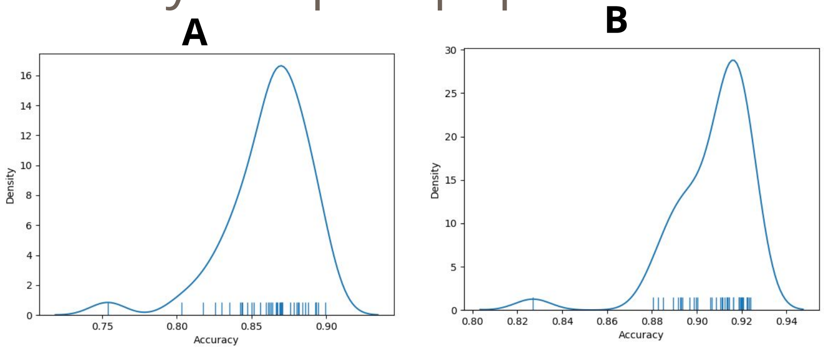
We want to capture this uncertainty into account

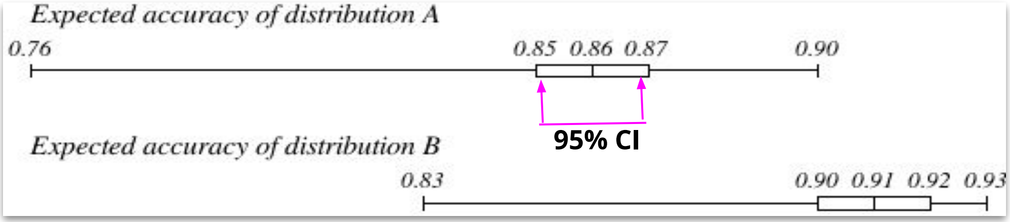




Solution Idea

- Accuracy distributions across randomly sampled population is obtained
- Able to choose a value which is in good terms with overall population
- Expected accuracy in 95% CI is computed





Usability

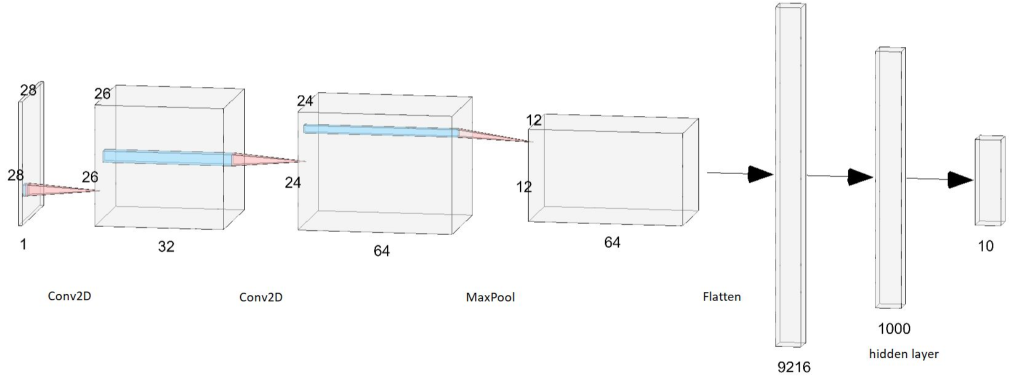
- Other randomly initialized parameters in the mode. E.g., weight vectors when tuning learning rate or hidden unit size.
- Unseen model inputs. Unseen data can be generated with synthetic data generation.
- Large dataset where training on whole dataset takes long time.

[1] Swersky, Kevin, Jasper Snoek, and Ryan P. Adams. "Multi-task bayesian optimization." Advances in neural information processing systems. 2013.

Experiment Design (DNN Model)

MNIST Dataset

- Handwritten digit classification
- 784 inputs | 28x28 = 784 pixels
- 1 output (0, 1, 2 ..., 9)
- Files (70K) | Train (60K) Test (10K)



- 2 convolutional layers
- 1 pooling layer
- 1 hidden layer (goal is to optimize number of hidden units)

Experiment Design (Random Search)

- **Goal: optimize number of hidden units in given model**
- Hidden unit: [1, 1000]
- Number of experiments: 13
- 10 trials in each experiment

Generate	Generate sampling space for hyperparameter h ([1, 1000])
Select	Uniformly select *n* random values from the sample space H = {H_1, H_2, …, H_n }
Generate	Generate *n* corresponding model M = {M_1, M_2, …, M_n} using H
Train and evaluate	Train and evaluate each model M_i
Choose	Choose H_i for which M_i gives best result

Uncertain<*T*> Data Type

Implemented in Python

All comparison operators have been overloaded

Hypothesis test for sample and test distributions

Function *E* approximated population mean

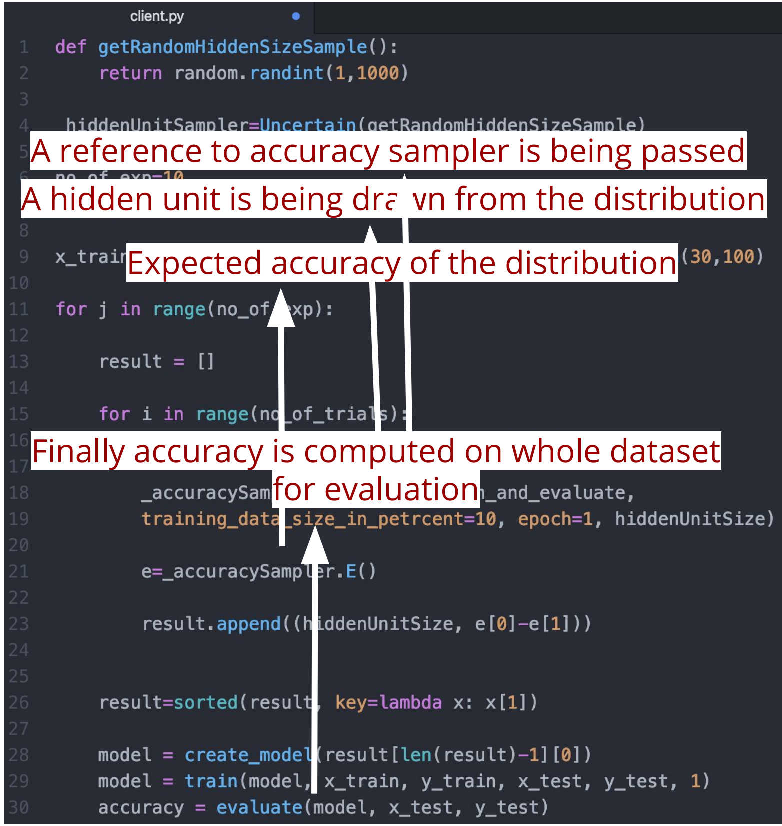
MOE is calculated

uncertain.py

```python
class Uncertain:
    def __init__(self,sampler, *args):
        self.id=''
        self.plotDensity=False
        self.sampleSize=40
        self.samplingFunction = sampler
        self.args = list(args)

    def __lt__(self, other):
        return self.hypothesis_test(other, op.lt)

    ...

    def sample(self):
        return self.samplingFunction(*self.args)

    def hypothesis_test(self, other, H0):
        t1 = self.sample()
        t2 = other.sample()
        return H0(t1,t2)

    #returns sample mean and margin of error in 95% CI
    def E(self):
        data=[]
        for i in range(self.sampleSize):
            data.append(self.sample())

        std=stat.stdev(data)
        moe= (2*std)/math.sqrt(self.sampleSize)

        return [stat.mean(data), moe]
```

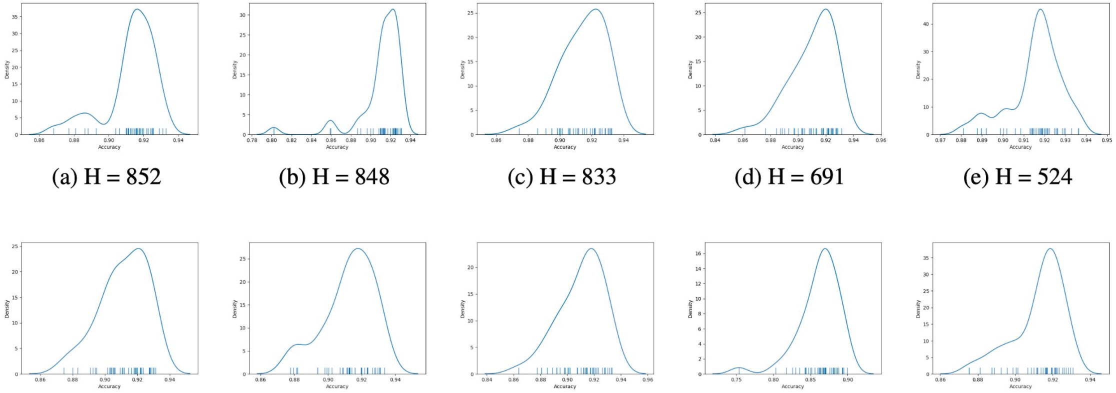
Uncertain Search

- **Draw** — Draw a sample from hidden unit distribution
- **Obtain** — Obtain 40 samples from accuracy distribution
- **Calculate** — Calculate expected accuracy in 95% CI
- **Choose** — Choose best hidden unit



Expected Accuracy

- Population mean is approximated in 95% CI using central limit theorem



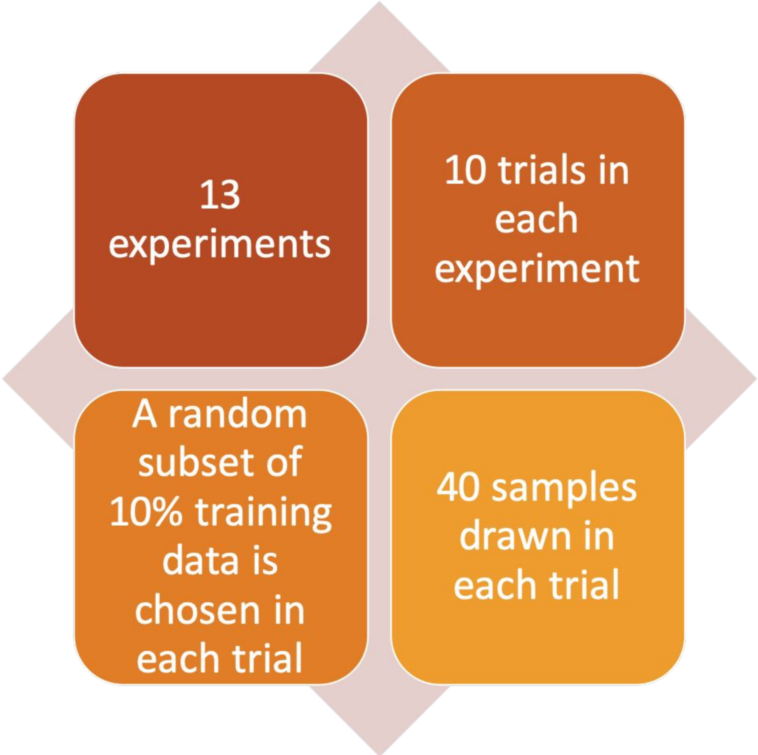
(a) H = 852 (b) H = 848 (c) H = 833 (d) H = 691 (e) H = 524

(f) H = 432 (g) H = 353 (h) H = 297 (i) H = 25 (j) H = 237

uncertain.py

```python
class Uncertain:
    def __init__(self,sampler, *args):
        self.id=''
        self.plotDensity=False
        self.sampleSize=40
        40 samples are being drawn
        self.args = list(args)

    def __lt__(self, other):
        return self.hypothesis_test(other, op.lt)

    ...  Population mean in 95% CI

    def sample(self):
        return self.samplingFunction(*self.args)

    def hypothesis_test(self, other, H0):
        t1 = self.sample()
        t2 = other.sample()
        return H0(t1,t2)

    #returns sample mean and margin of error in 95% CI
    def E(self):
        data=[]
        for i in range(self.sampleSize):
            data.append(self.sample())

        std=stat.stdev(data)
        moe= (2*std)/math.sqrt(self.sampleSize)

        return [stat.mean(data), moe]
```

Experiment Setup (Uncertain Random Search)



Evaluation

- RQ1: How much **uncertainty** remains in hyperparameter optimization?
- RQ2: What is the **performance** improvement when we optimize hyperparameter with our method over the random search?

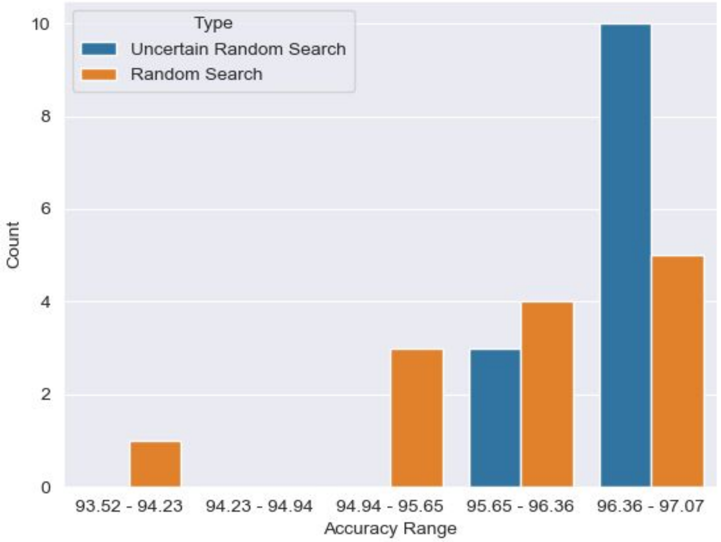
RQ1: How much *uncertainty* remains in hyperparameter optimization?

- Accuracy ranges from **96%** to **97%** in case of uncertain search
- However, random search is more uncertain as the accuracy fluctuates largely
- Uncertain random search provides consistent performance as the best trial generalizes over the population

RQ2: How much is the performance improvement of *Uncertain Random Search*?

- **62%** chance that accuracy will be in lower ranges if random search applied compared to **23%** in uncertain version

- **77%** likely that a randomly initialized model's accuracy will be in the high range



Findings

- In general, uncertain random search produces more consistent accuracies as it evaluates model accuracy on as many diverse population as possible
- Requires less training data for tuning compared to whole dataset needed by random search or grid search

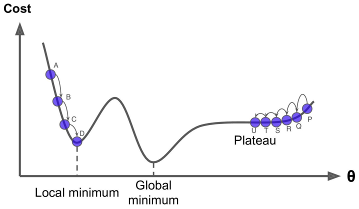
Future Work

- Evaluate our methodology on very large dataset
- Evaluate our methodology on synthetic data

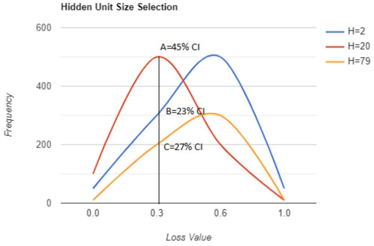
References

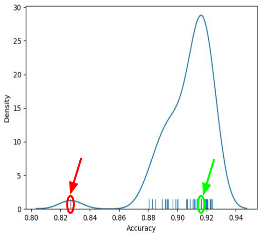
1. Swersky, Kevin, Jasper Snoek, and Ryan P. Adams. "Multi-task bayesian optimization." Advances in neural information processing systems. 2013.
2. Bornholt, James, Todd Mytkowicz, and Kathryn S. McKinley. "Uncertain< T>: A first-order type for uncertain data." ACM SIGPLAN Notices. Vol. 49. No. 4. ACM, 2014.
3. Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research 13.Feb (2012): 281-305.

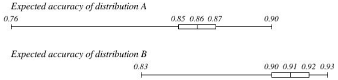
- DNN hyperparameter optimization is difficult
- Random search strategy has been proven efficient
- However, in random search, programmers can not overtly represent uncertainty
- We utilize a first order type Uncertain<*T*> to approximate the distributions of the hyperparameters



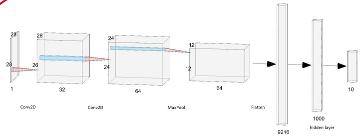
Wrong question: Is 0.3 the lowest, loss value can get or is it optimal?







Right question: How much evidence is there for 0.3 to be the optimal value?



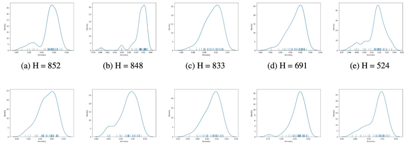
Generate	Generate sampling space for hyperparameter h ([1, 1000])
Select	Uniformly select *n* random values from the sample space H = $\{H_1, H_2, \ldots, H_n\}$
Generate	Generate *n* corresponding model M = $\{M_1, M_2, \ldots, M_n\}$ using H
Train and evaluate	Train and evaluate each model M_i
Choose	Choose H_i for which M_i gives best result

uncertain.py

```
class Uncertain:
    def __init__(self,sampler, *args):
        self.id=''
        self.plotDensity=False
        self.sampleSize=40
        self.samplingFunction = sampler
        self.args = list(args)

    def __lt__(self, other):
        return self.hypothesis_test(other, op.lt)


    ...

    def sample(self):
        return self.samplingFunction(*self.args)

    def hypothesis_test(self, other, H0):
        t1 = self.sample()
        t2 = other.sample()
        return H0(t1,t2)

    #returns sample mean and margin of error in 95% CI
    def E(self):
        data=[]
        for i in range(self.sampleSize):
            data.append(self.sample())

        std=stat.stdev(data)
        moe= (2*std)/math.sqrt(self.sampleSize)

        return [stat.mean(data), moe]
```

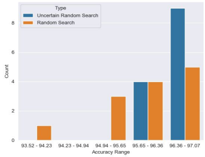
Draw	Draw a sample from hidden unit distribution
Obtain	Obtain 40 samples from accuracy distribution
Calculate	Calculate expected accuracy in 95% CI
Choose	Choose best hidden unit

- 13 experiments
- 10 trials in each experiment
- A random subset of 10% training data is chosen in each trial
- 40 samples drawn in each trial



(a) H = 852 (b) H = 848 (c) H = 833 (d) H = 691 (e) H = 524

(f) H = 432 (g) H = 353 (h) H = 297 (i) H = 25 (j) H = 237



Thank You

Questions?