Hyper-parameter optimization for convolutional neural networks using a Gaussian Process and simple partial randomization applied to the MNIST dataset [5]

Steven Smith

P13167208

Thursday 18th May – 15:40

What is to come?

- Problem definition
- Summary of existing literature
- Proposed solution
- Methodology
- Experimental design
- Data collection
- Experimental results
- Conclusions
- Future work
- Lessons learned
- References

Problem Definition

- Convolutional Neural Network
  - Convolutional Layers
    - Feature maps
  - Pooling Layers
    - Reduce features using maximum or average pooling
  - Fully Connected Layers
    - Traditional neural network layers

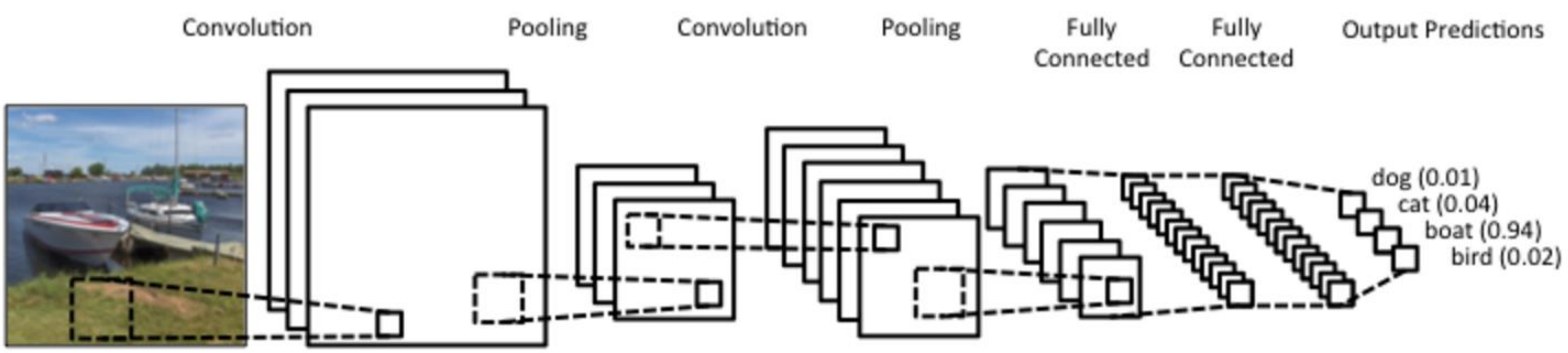


Image accessed on 09/05/2017 from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Problem Definition (continued)

- Hyper-parameters
  - Number of layers and filters
  - Size of filters and stride when applying filters
  - Regularization, e.g. L1, L2
  - Max pooling kernel size and strides
  - Number of neuronal units
  - Dropout percentage
  - Learning rate and decay
  - Activation functions, e.g. RELU, Linear, tanh, softmax, sigmoid
  - Optimization functions, e.g RMSProp, Stochastic Gradient Descent & ADAM
  - and many many more!
- Grid Search
  - Large search space
  - The experiments uses 1,555,200 possible combinations
- Random Search
  - Not guided

Problem Definition (continued)

- MNIST [5]
- Digitised handwritten digits 0 to 9
- 70000 Digits in total
  - 60000 train/validate
  - 10000 test
- Best CNN error rate is 0.23% (Acurracy 99.77%)

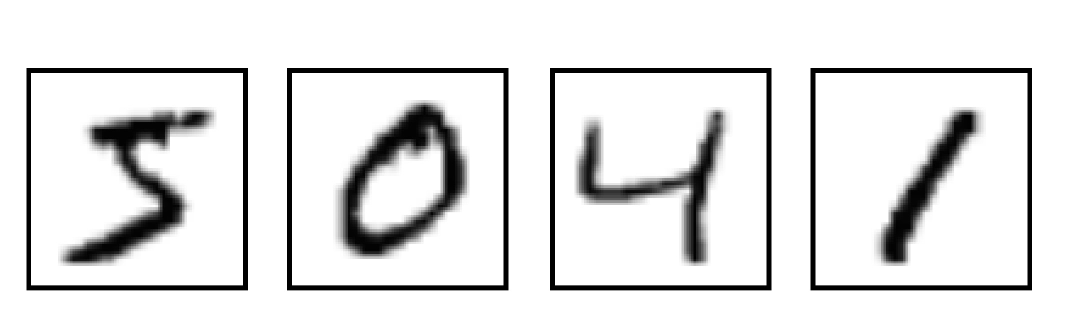


Image accessed on 09/05/2017 from https://www.tensorflow.org/get_started/mnist/mechanics

Summary of existing literature

- Sequential Model-based Global Optimization
  - Iteratively maximize a ‘cheap’ surrogate function using an acquisition function such as Expected Improvement
  - At the maximum is where the parameters should now be tested against the real ‘expensive’ function
- Gaussian Process Regression [2]
  - Updating Multi-Variate Gaussian
  - Bayesian - Prior (existing knowledge) to Posterior (prediction)
  - Maximise Expected Improvement $EI(x) = current\ best\ min - (\mu + 1.96\sigma)$
  - “All hyper-parameters are not relevant for each point” [1, pp.3]
    - Multiple GPs for each layer
  - Issues modelling categorical data such as activation function
  - Matern 5/2 Kernel
    - As opposed to Squared Exponential Kernel

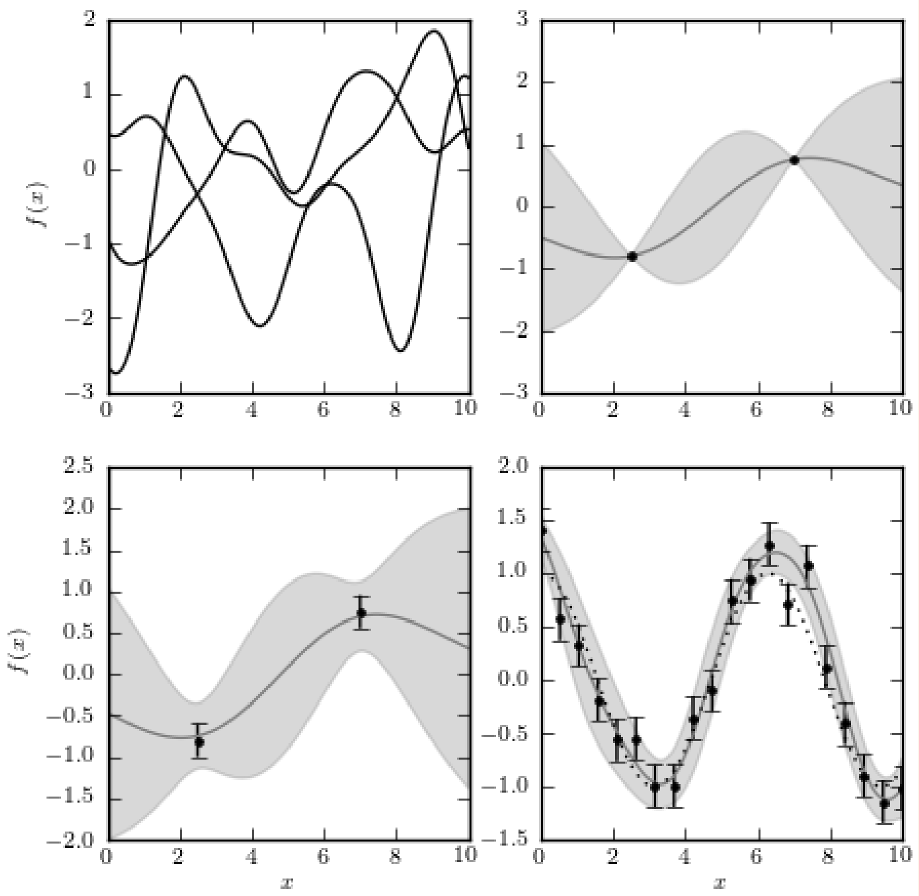


Image accessed on 09/05/2017 from http://www.astroml.org/book_figures/chapter8/fig_gp_example.html

Summary of existing literature (continued)

- Tree-Structured Parzen Estimator Approach [1]
  - Parameters modelled by separate distributions
  - Gather some random samples and split into best and worst observations for a parameter
  - Determine which group parameter sets are likely to belong to.
  - Expected Improvement $EI(x) = \frac{P(x\ is\ in\ best\ group)}{P(x\ is\ in\ worst\ group)}$
  - Parzen-window density estimator, non-parametric probability density function estimator
  - Tree based due to splitting on a hierarchy of parameters
- Transfer Learning
  - Use Gaussian Processes and learning built on multiple training datasets
  - One approach has access to the entire datasets [7]
  - The other uses statistics about the datasets already trained [8]
- Covariance Matrix Adaptation-Evolution Strategy
  - Preprint paper [10], uses an evolution based algorithm to determine the hyper-parameters
  - Found to have better results than both TPE and GP.
- Auto-Weka [11]
  - Implemented both GP and TPE based methods to determine which of 39 classifiers to use
  - Hyper-parameter optimization for those classifiers also.

Proposed solution

- In [1] authors suggest not all parameters need to be set, e.g. if layer 2 or 3 does not exists
  - The proposed approach uses the same hyper-parameters for different layers
- [1] also suggests that TPE outperformed GP due to “the exploration induced by the TPE’s lack of accuracy turned out to be a good heuristic for search” [1](pp.7)
  - The proposed approach adds partial randomization after GPR
- Gaussian Process Regression (GPR) optimizing using expected improvement of the log loss (or cross entropy) metric and random search
- Convolutional hyper-parameters vs fully connected hyper-parameters
- GPR to determine initial parameters then perturb some of the parameters for additional randomness.
  - This allows GPR to use all parameters to guide overall direction.
  - Only sets either convolutional or fully connected, the others are randomly generated.
  - Can this improve traversal of the search space?

Methodology

- Scope - 12 hyper-parameters converted to a $(1\ x\ 12)$ array for randomization or the Gaussian Regression Process
  - Number of conv. Layers, (1,3), n = 1, 2, 3
  - Number filters, (1,4), 2^{n+3} = 16, 32, 64, 128
  - Filter size, (3,10), n = 3, 4, 5, 6, 7, 8, 9, 10
  - Stride, (1,5), n = 1, 2, 3, 4, 5
  - Regularizer, (1,2), Ln = L1, L2
  - Local Response Normalization, (0,1), 0 = off or 1 = on
  - Use Max Pooling, (0,1), 0 = off or 1 = on
  - Max Pool kernel size, (2,4), n = 2, 3, 4
  - Number of fully connected layers (1,3), n = 1, 2, 3
  - Number of neuronal units, (1,3), 2^{n+6} = 128, 256, 512
  - Drop out keep probability (1,5), $0.1\,n\ +\ 0.4$ = 0.5, 0.6, 0.7, 0.8, 0.9
  - Learning rate (1,3), $10^{(-n)}$ = 0.1, 0.01, 0.001
- For multiple layers, the same parameters are used
- Fixed parameters (taken from TFLearn defaults)
  - RELU for convolutional layer activation, Tanh for Fully connected layer activations
  - Weight are initialized using a truncated normal, $N(0, 0.02)$ and < 2 standard deviations from the mean
- Mini batches of size, 64
- All other network hyper-parameters are static between runs.
- Limit training to 20 epochs (50 epochs for the final-full tests)

Methodology (continued)

Four separate methods tested:

1. Random search
2. GPR without further randomization
3. GPR with 8 Convolutional parameters randomized

   [1,2,3,4,5,6,7,8,1,2,3,1] by GPR

   [5,6,7,8,1,2,3,4,1,2,3,1] further randomized
4. GPR with 3 Fully Connected Parameters randomized

   [1,2,3,4,5,6,7,8,1,2,3,1] by GPR

   [1,2,3,4,5,6,7,8,9,8,7,1] further randomized

Experimental design

- The four experiments in each batch
- Each batch has
  - Number of iterations – 20 or 50
  - Number of epochs – 20
  - Validation set train/validate ratio – fixed at 70/30
  - Groups of MNIST [5] digits to train on
    - 4,7
    - 4,7,9
    - 0, 4, 7 ,9
    - 0, 2, 4, 7 ,9
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Ten combinations in all (Training 1400 models, 350 per algorithm, for 20 epochs each)

Data collection

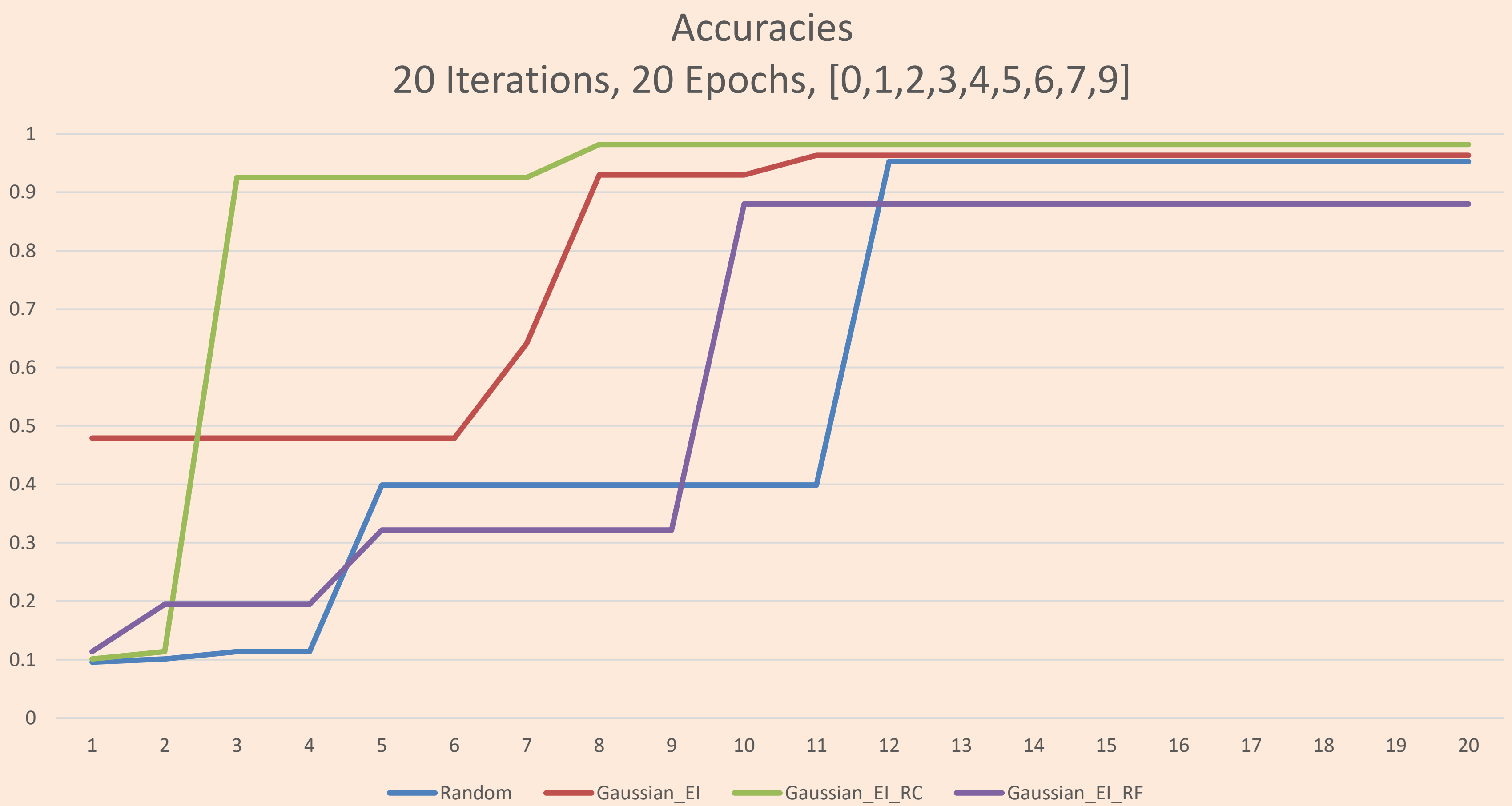
- Using Python, TFLearn, Tensorflow, NumPy, Sci-kit Learn, Matplotlib to train the models and generate data.
- Generate plots of accuracy and log loss to compare visually.
- Accuracy, Log Loss, Confusion Matrix for every trained model recorded in a log of each experiment.
- Processed by jupyter notebook to extract data
- Excel to create tables and graphs

Experimental results

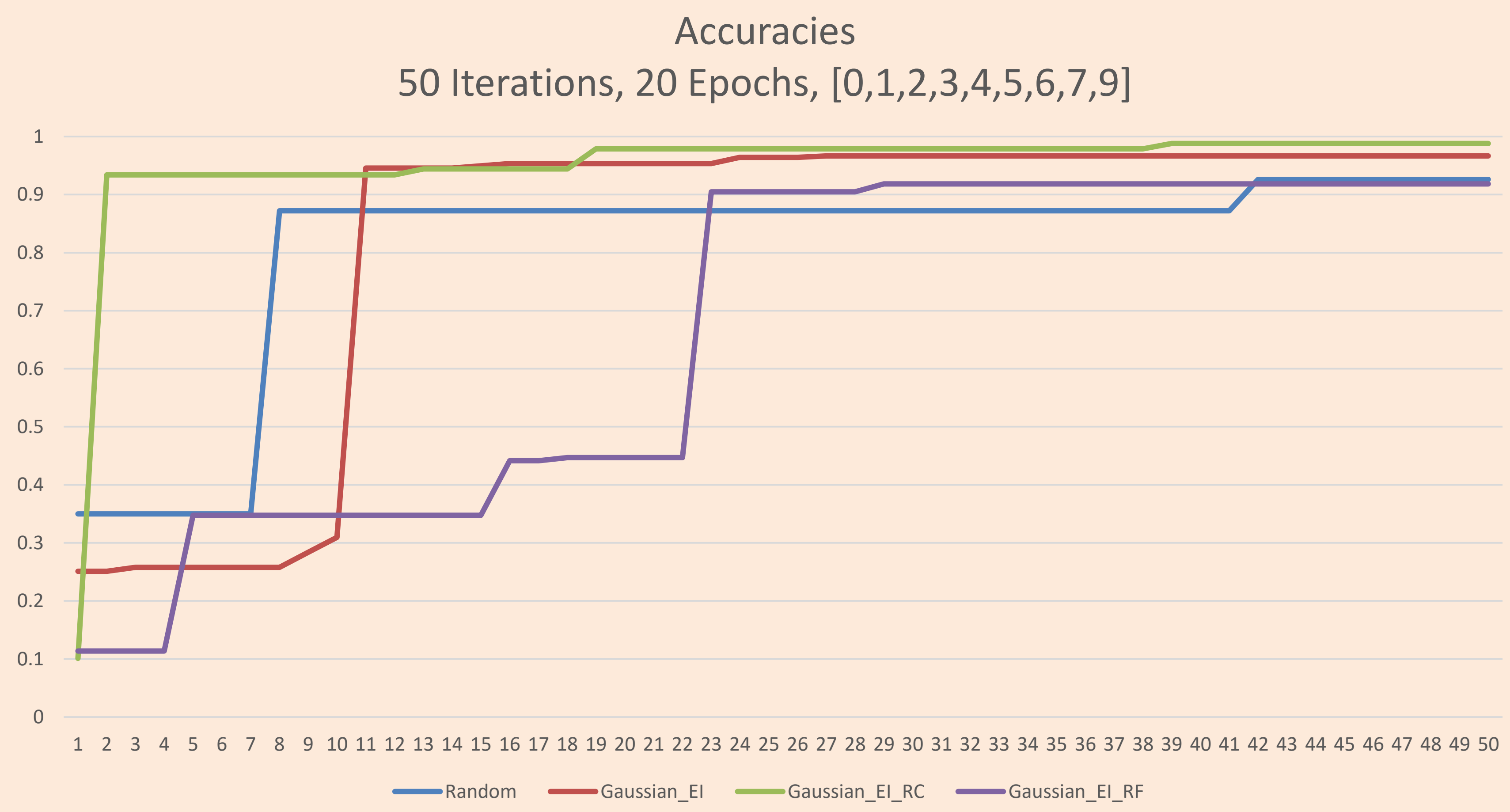
	Maximum Accuracy			
	Random	Gaussian_EI	Gaussian_EI_RC	Gaussian_EI_RF
2n_20	0.995	***0.996***	0.979	0.954
2n_50	***0.998***	0.981	0.983	0.988
3n_20	0.984	***0.986***	0.985	0.341
3n_50	***0.974***	0.958	0.907	0.861
4n_20	0.953	0.978	***0.988***	0.469
4n_50	0.982	***0.995***	0.984	0.857
5n_20	***0.965***	0.386	0.204	0.593
5n_50	0.949	***0.988***	0.854	0.980
10n_20	0.952	0.963	***0.982***	0.880
10n_50	0.926	0.967	***0.988***	0.918
Max	***0.998***	0.996	0.988	0.988
Count	3	***4***	3	0

	Count of Accuracy > 0.6			
	Random	Gaussian_EI	Gaussian_EI_RC	Gaussian_EI_RF
2n_20	***4***	1	2	1
2n_50	***11***	4	4	9
3n_20	***5***	2	2	0
3n_50	4	***6***	1	3
4n_20	***2***	1	***2***	0
4n_50	6	***16***	5	4
5n_20	***2***	0	0	0
5n_50	3	***12***	3	3
10n_20	1	***11***	8	2
10n_50	4	***35***	26	9
Total	42	***88***	53	31
Count	***5***	***5***	1	0

Experimental Results (continued)



Experimental Results (continued)



Experimental Results (continued)

- Best parameters for [0,1,2,3,4,5,6,7,8,9] and 50 iterations
  - Accuracy 98.79% (State of the art CNN Acurracy 99.77%)
  - [3, 1, 10, 1, 2, 1, 1, 3, 1, 3, 5, 3]
  - 3 Convolutional layers
    - filters=16
    - size=10
    - stride=1
    - L2 regularizer
    - local response normalization layer
    - max pooling
      - kernel size=3
  - 1 Fully Connected Layer
    - 512 units
    - Drop out keep probability = 0.9
  - Learning rate = 0.001

Conclusions

- Random, Gaussian_EI and Gaussian_EI_RC all had best performance in 3, 4 and 3 tests respectively.
- Therefore Gaussian Process was better than Random in 7 out of 10 tests.
  - However 8 of the 12 parameters are randomized so is that more a random process?
- Gaussian RC performed best on the Full Dataset on both 20 and 50 iterations
- When the parameters are far from optimum the time to complete epochs can be 4/5 times longer than other iterations
- Would a better approach have been to perturb the parameters to a lesser degree?
  - Only perturbing some of the parameters
  - Not perturbing over the whole range but close to the Gaussian process generated hyper-parameters.

Future work

- Incremental initialisation
  - Start with two digit classes, use GPR to find good models
  - Use these to initialise GPR against three digits, then four digits ...
- Time limited test runs as opposed to epochs or number of mini batches
- Can the optimizing of the Convolutional Layers be treated independently of the Fully Connected layers and vice versa?
- Random weights for feature learning [6]
- Create repository of good configurations from other papers
- Test against CIFAR-10 and CIFAR-100 [3,4]
- Reinforced Learning optimizing hyper-parameters on 2 classes then 3 classes then 4 …

Lessons learned

- Start small, start simple
  - Difficult to make a complex solution simpler
- Use relational database to store experiment logs and results
- Split the learning across all available GPUs/CPUs
- Don’t close the program when you mean to minimise it! (Twice!)

References

[1] Bergstra, J.S., Bardenet, R., Bengio, Y. and Kégl, B., 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems* (pp. 2546-2554).

[2] Snoek, J., Larochelle, H. and Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951-2959).

[3] Thornton, C., Hutter, F., Hoos, H.H. and Leyton-Brown, K., 2013, August. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 847-855). ACM.

[4] CIFAR-10 and CIFAR-100 datasets, https://www.cs.toronto.edu/~kriz/cifar.html last accessed 09/05/2017

[5] The MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/ last accessed 09/05/2017

[6] Saxe, A., Koh, P.W., Chen, Z., Bhand, M., Suresh, B. and Ng, A.Y., 2011. On random weights and unsupervised feature learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 1089-1096).

[7] Bardenet, R., Brendel, M., Kégl, B. and Sebag, M., 2013, June. Collaborative hyperparameter tuning. In ICML (2) (pp. 199-207).

[8] Yogatama, D. and Mann, G., 2014. Efficient transfer learning method for automatic hyperparameter tuning. Transfer, 1, p.1

[9] Rasmussen, C.E. and Williams, C.K., 2006. Gaussian processes for machine learning. 2006. The MIT Press, Cambridge, MA, USA, 38.

[10] Loshchilov, I. and Hutter, F., 2016. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. arXiv preprint arXiv:1604.07269.

[11] Thornton, C., Hutter, F., Hoos, H.H. and Leyton-Brown, K., 2013, August. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 847-855). ACM.