Scalable Machine Learning: Laboratory 2

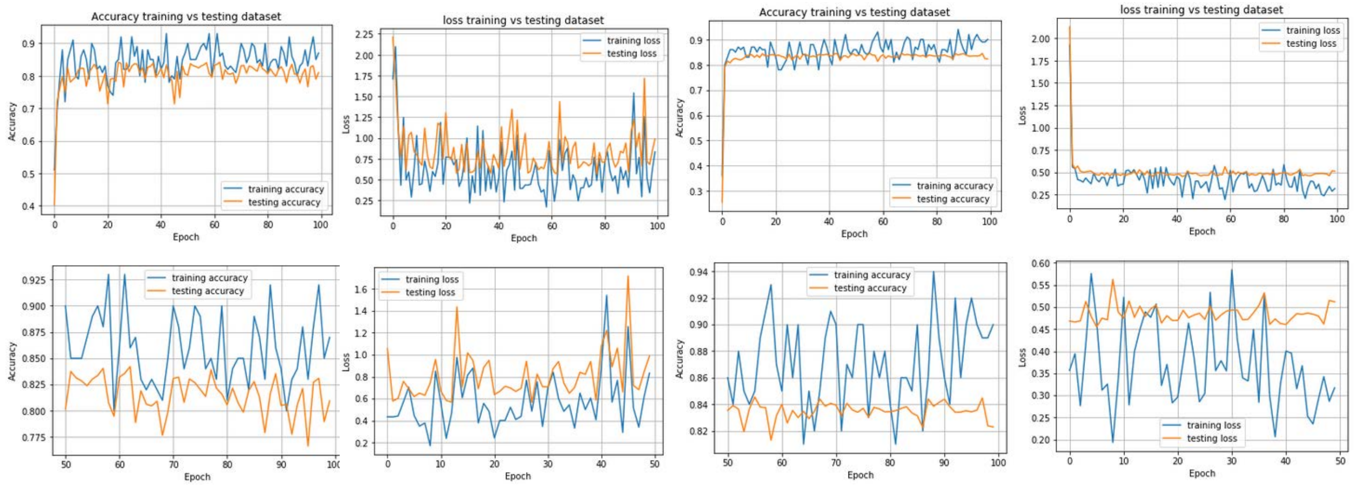
Deep Learning with TensorFlow

By Douglas Garcia Torres

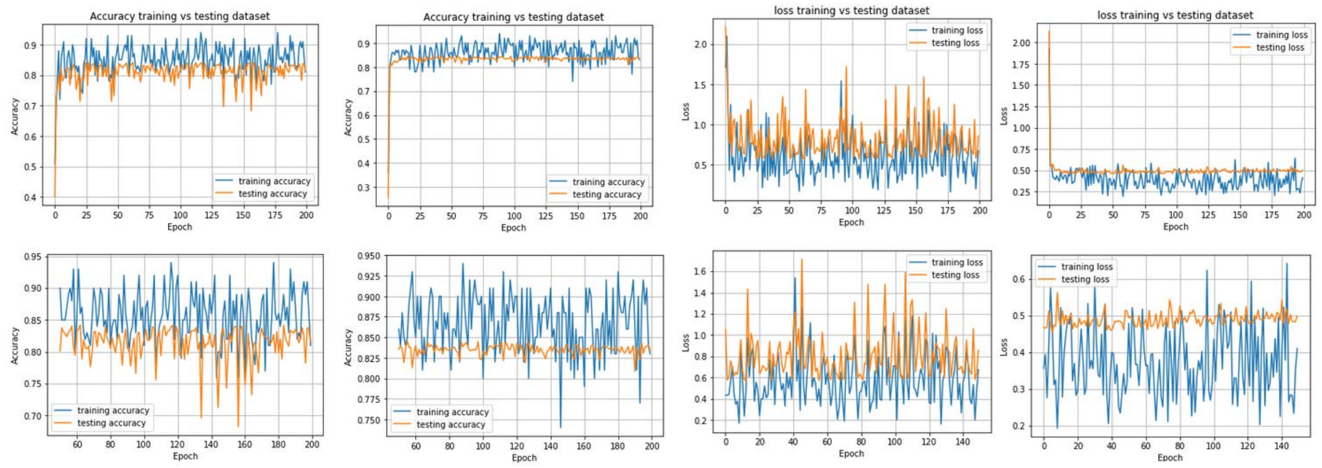
November 2017

Task 1: One-Layer Softmax Regression

- Gradient Descent Optimizer with learning rate = 0.5 (left) Adam Optimizer with learning rate = 0.005 (right)



- What loss and accuracy do you get when training over 10.000 iterations (with 20.000 iterations)?
  Gradient Descent (left) vs Adam Optimizer (right)



Task 2: Feed-Forward NN with 4 layers

- Is there a big difference between the convergence rate of the sigmoid and the ReLU?
  Yes, ReLU converges faster than SIGMOID.

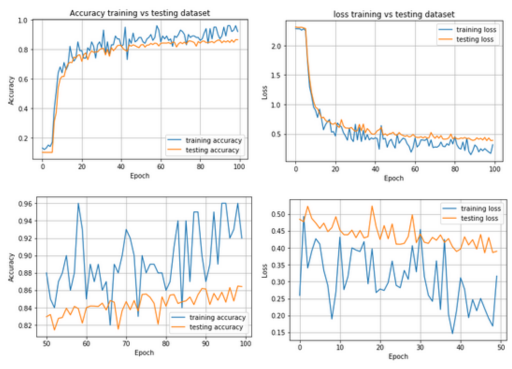
- If yes, what is the reason for the difference?
  In comparison with SIGMOID, ReLU is less likely to suffer from Vanishing gradient and its constant gradient produces a faster learning. SIGMOID weights becomes smaller as the value of X increases.

- What is the reason that we use the Softmax in our output layer?
  Because we are performing multinomial classification and SOFTMAX layer outputs a vector of probabilities instead of a scalar, while SIGMOID is not very well suited for multinomial classification.

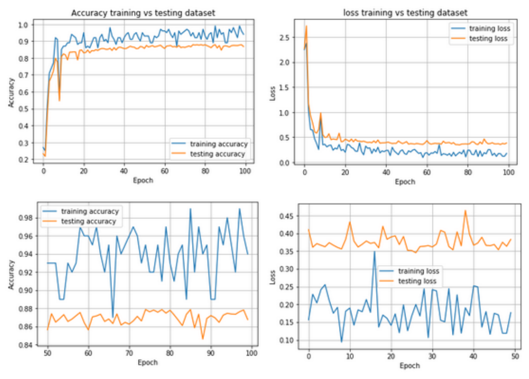
- By zooming into the second half of the epochs in accuracy and loss plot, do you see any strange behavior?
  Yes, there is some volatility and sudden drops. The trend is especially evident using Gradient Descent in both training and testing accuracy and loss (with both Sigmoid or ReLU). With ADAM optimizer, the testing accuracy and loss are more stable.

- What is the reason and how you can overcome them? (e.g., look at fluctuations or sudden loss increase after a period of decreasing loss).
  The reason might be the that it reaches a local minima, a situation that could be fixed by setting a lower learning rate. If the situation would only occur in the test data, that would be an indicator of over-fitting.

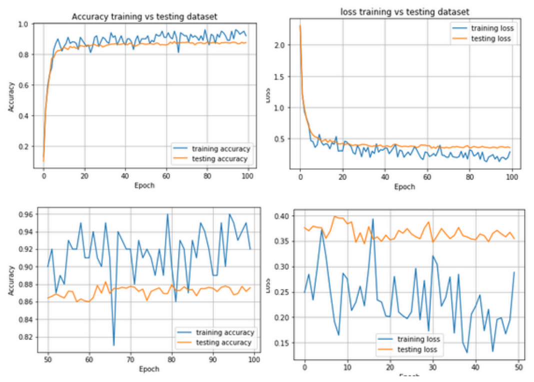
Sigmoid activation + Softmax with Gradient Descent
Max.train accuracy:0.96
Max.test accuracy:0.8649



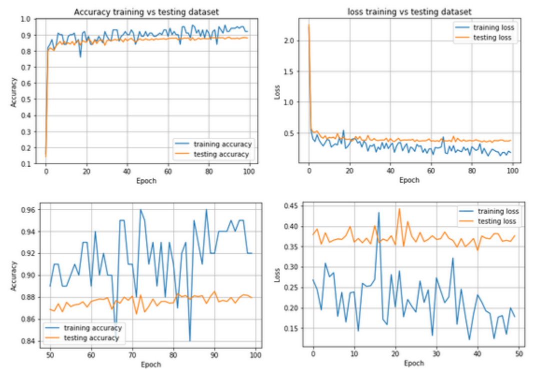
ReLU activation + Softmax with Gradient Descent
Max.train accuracy:0.99
Max.test accuracy:0.8788



Sigmoid activation + Softmax with ADAM Optimizer
Max.train accuracy:0.96
Max.test accuracy:0.8828



ReLU activation + Softmax with ADAM Optimizer
Max.train accuracy:0.96
Max.test accuracy:0.8853



Task 3: Regularization and Tuning

- What is the motivation behind learning rate decay?
  When training a model, it is often recommended to lower the learning rate as the training progresses. Learning rate decay applies exponential decay to the learning rate (i.e. applies an exponential decay function to a provided initial learning rate). The function returns the decayed learning rate which is computed as:

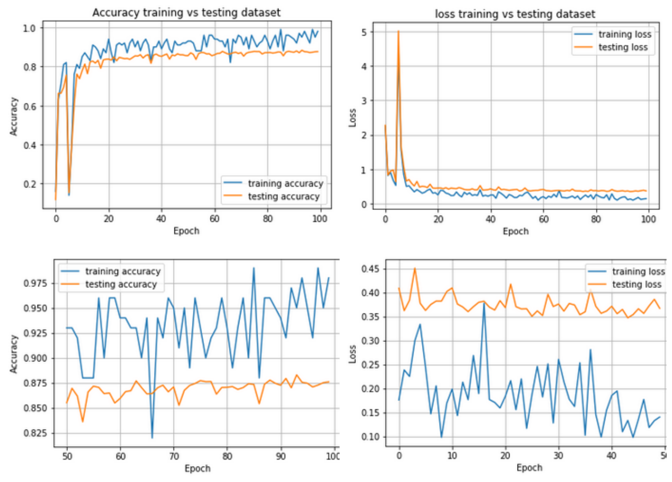
```
decayed_learning_rate = learning_rate *
                        decay_rate ^ (global_step / decay_steps)
```

- Why dropout can be an effective regularization technique?
  Dropout works as follows: one or more neural network nodes are switched off once in a while so that it will not interact with the network (it weights cannot be updated, nor affect the learning of the other network nodes). With dropout, the learned weights of the nodes become somewhat more insensitive to the weights of the other nodes and learn to decide somewhat more by their own (and less dependent on the other nodes they're connected to). In general, dropout helps the network to generalize better. With dropout, neurons are prevented from co-adapting too much which makes overfitting less likely.

  <u>NOTE</u>: These experiments are done with ReLU activation and 10000 iterations (starting_rate = 0.5, decay_rate = 0.96, decay_steps = 100000) with Gradient Descent optimization.

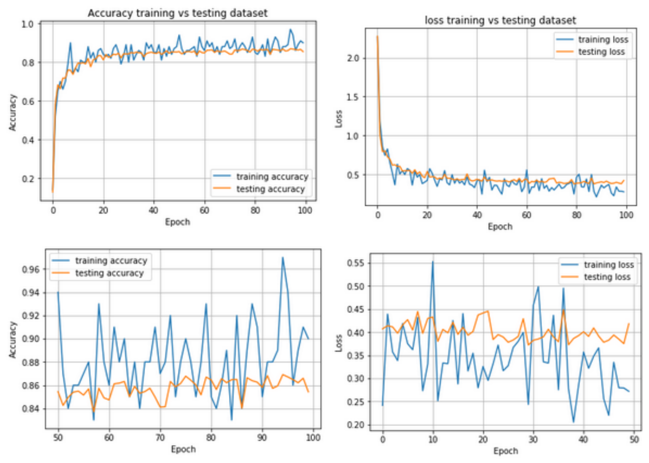
With Learning Rate Decay *(this one is with decay_steps = 1000)*

Max.train accuracy:0.99
Max.test accuracy:0.8831



With Regularization such as Dropout

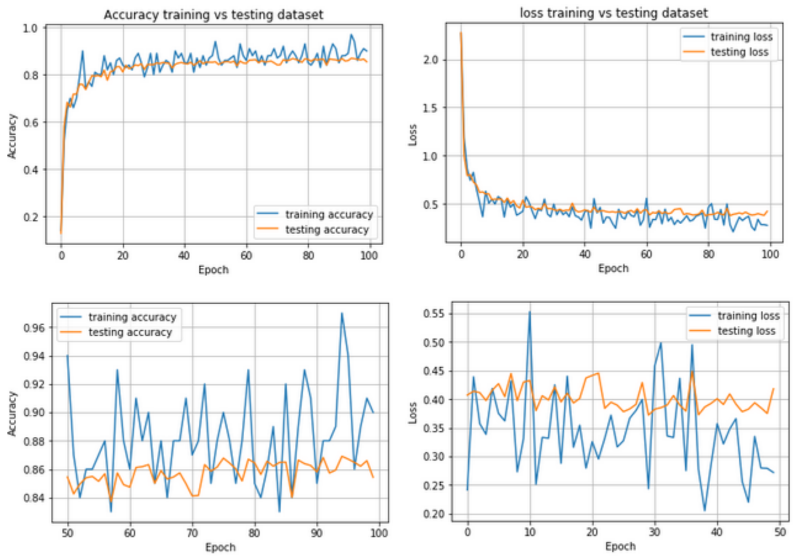
Max.train accuracy:0.97
Max.test accuracy:0.8691



With Learning Rate Decay and Dropout Regularization

<u>NOTE:</u> If I change the decay steps to 100 or 1000, it will train faster, the training accuracy will drop (to 0.94) but the test accuracy remains almost the same (0.8778). Also the testing accuracy will be more stable.

Max.train accuracy:0.97
Max.test accuracy:0.8691

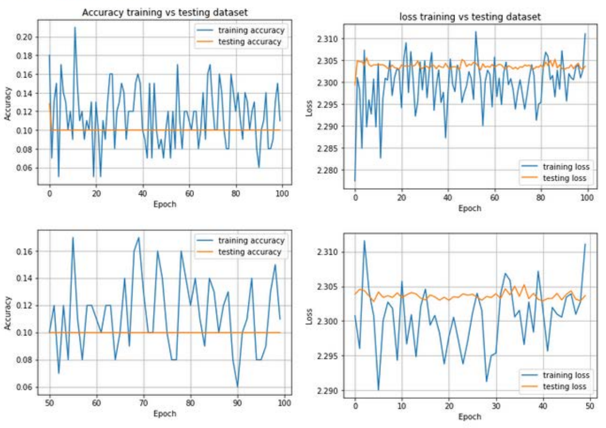


Task 4: Convolutional Neural Network

- Network layer with 3 conv. layers, 1 ReLU layer and 1 Softmax layer with Gradient Descent Optimizer (left) and with Adam optimizer (right).

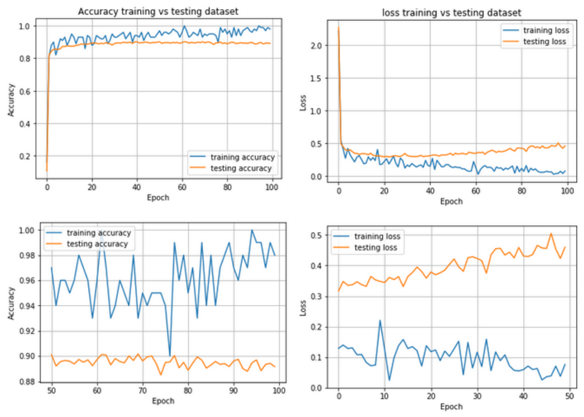
<u>Note</u>: I'm using a 0.5 learning rate

```
Max.train accuracy:0.21
Max.test accuracy:0.1281
```



```
Max.train accuracy:1.0
Max.test accuracy:0.9017
```

This results are way better than the previous ones. For the first time in the lab I reached 90% max.testing accuracy



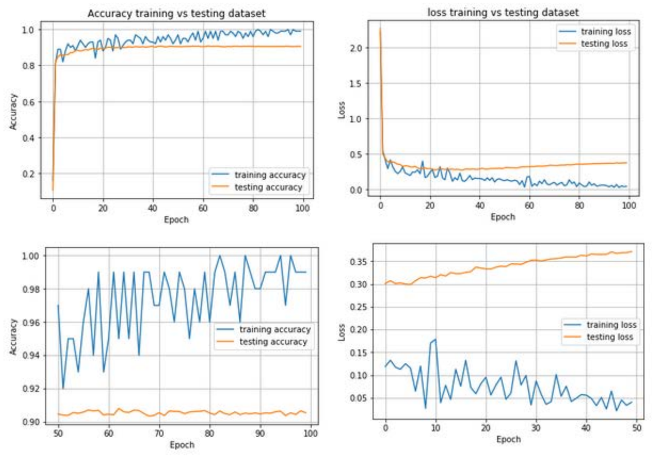
- What is the output structure of the convolutional layers based on the given stride?
  Keeping in mind that if the stride has any element bigger than 1 in its structure it will reduce that particular dimension in the output of the layer, then:
  - The 1st convolutional layer outputs an image of 28x28 (after using stride of [1,1,1,1])
  - The 2nd convolutional layer outputs an image of 14x14 (after using stride of [1,2,2,1])
  - The 3rd convolutional layer outputs an image of 7x7 (after using stride of [1,2,2,1])

  Therefore the output structure will be a vector of 200 elements of 7x7.

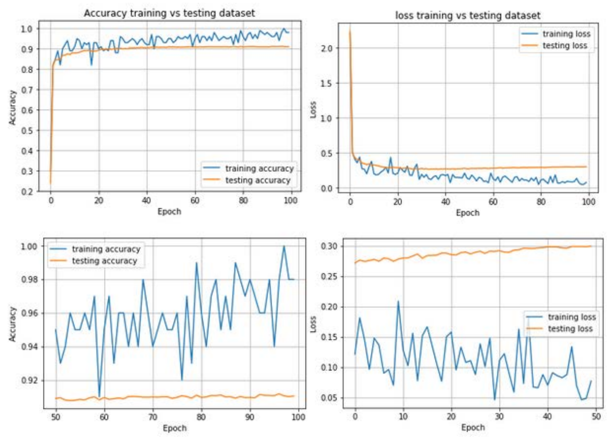
- After adding learning rate decay (left) and after adding Drop-out regularization:

  <u>NOTE</u>: the learning rate decay did not have much influence in the accuracy results. But the accuracy improved 1% after adding drop-out regularization (and got way more stable than before).

```
Max.train accuracy:1.0
Max.test accuracy:0.9081
```

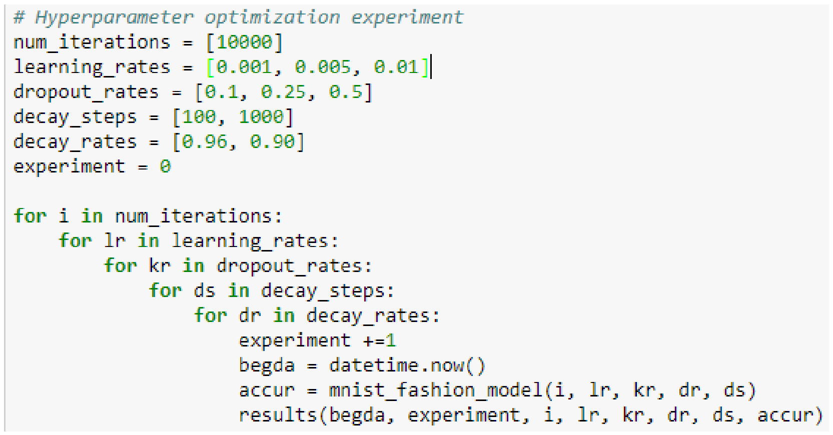


```
Max.train accuracy:1.0
Max.test accuracy:0.9118
```



Task 5: Improving Predictions with Hyper-parameter Optimization

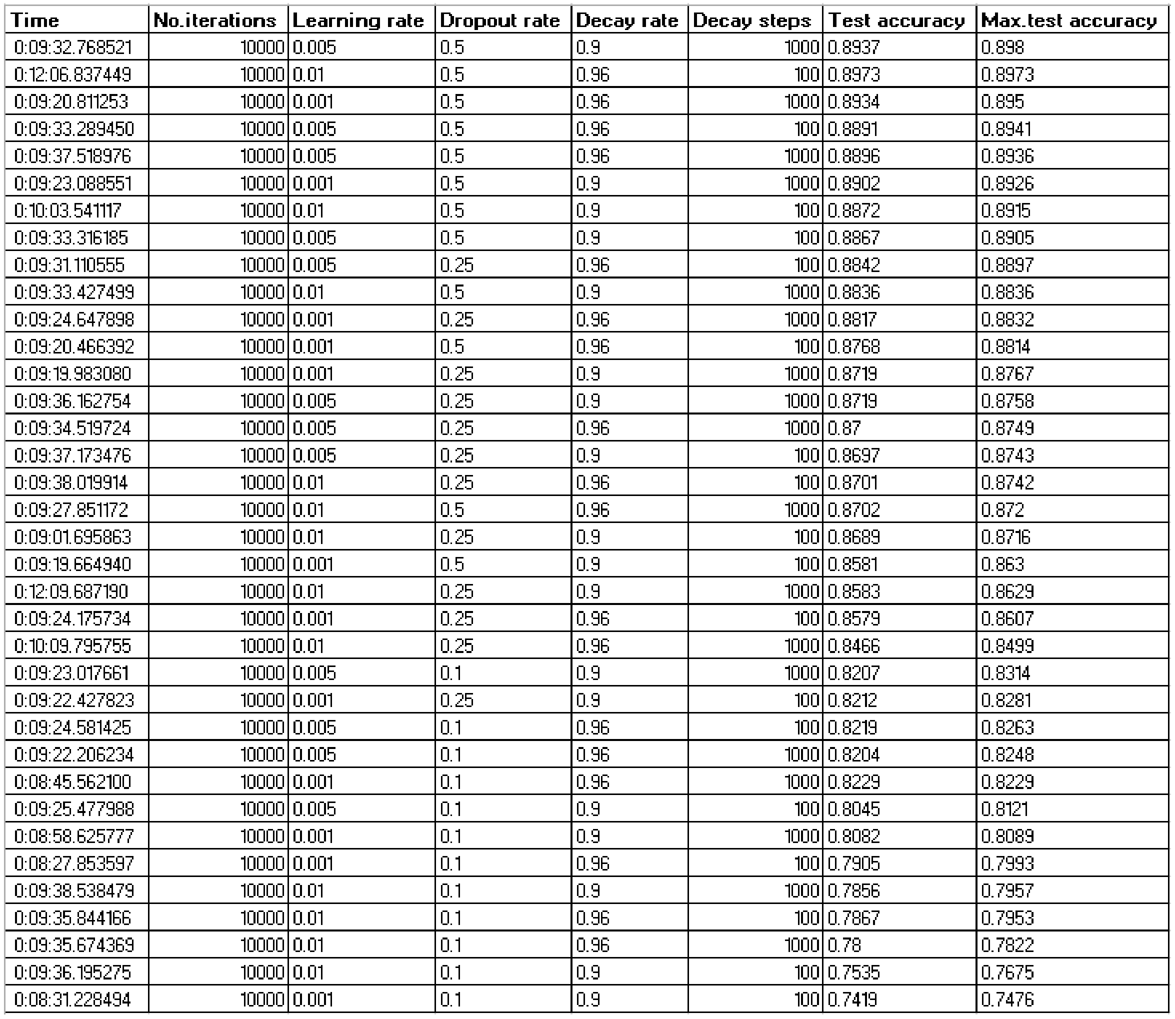
- Experiment (36 different combinations of hyper-parameters)



```
# Hyperparameter optimization experiment
num_iterations = [10000]
learning_rates = [0.001, 0.005, 0.01]
dropout_rates = [0.1, 0.25, 0.5]
decay_steps = [100, 1000]
decay_rates = [0.96, 0.90]
experiment = 0

for i in num_iterations:
    for lr in learning_rates:
        for kr in dropout_rates:
            for ds in decay_steps:
                for dr in decay_rates:
                    experiment +=1
                    begda = datetime.now()
                    accur = mnist_fashion_model(i, lr, kr, dr, ds)
                    results(begda, experiment, i, lr, kr, dr, ds, accur)
```

- Results (using CPU)
  <u>NOTE</u>: I used the dropout rate incorrectly in the code



Time	No.iterations	Learning rate	Dropout rate	Decay rate	Decay steps	Test accuracy	Max.test accuracy
0:09:32.768521	10000	0.005	0.5	0.9	1000	0.8937	0.898
0:12:06.837449	10000	0.01	0.5	0.96	100	0.8973	0.8973
0:09:20.811253	10000	0.001	0.5	0.96	1000	0.8934	0.895
0:09:33.289450	10000	0.005	0.5	0.96	100	0.8891	0.8941
0:09:37.518976	10000	0.005	0.5	0.96	1000	0.8896	0.8936
0:09:23.088551	10000	0.001	0.5	0.9	1000	0.8902	0.8926
0:10:03.541117	10000	0.01	0.5	0.9	100	0.8872	0.8915
0:09:33.316185	10000	0.005	0.5	0.9	100	0.8867	0.8905
0:09:31.110555	10000	0.005	0.25	0.96	100	0.8842	0.8897
0:09:33.427499	10000	0.01	0.5	0.9	1000	0.8836	0.8836
0:09:24.647898	10000	0.001	0.25	0.96	1000	0.8817	0.8832
0:09:20.466392	10000	0.001	0.5	0.96	100	0.8768	0.8814
0:09:19.983080	10000	0.001	0.25	0.9	1000	0.8719	0.8767
0:09:36.162754	10000	0.005	0.25	0.9	1000	0.8719	0.8758
0:09:34.519724	10000	0.005	0.25	0.96	1000	0.87	0.8749
0:09:37.173476	10000	0.005	0.25	0.9	100	0.8697	0.8743
0:09:38.019914	10000	0.01	0.25	0.96	100	0.8701	0.8742
0:09:27.851172	10000	0.01	0.5	0.96	1000	0.8702	0.872
0:09:01.695863	10000	0.01	0.25	0.9	100	0.8689	0.8716
0:09:19.664940	10000	0.001	0.5	0.9	100	0.8581	0.863
0:12:09.687190	10000	0.01	0.25	0.9	1000	0.8583	0.8629
0:09:24.175734	10000	0.001	0.25	0.96	100	0.8579	0.8607
0:10:09.795755	10000	0.01	0.25	0.96	1000	0.8466	0.8499
0:09:23.017661	10000	0.005	0.1	0.9	1000	0.8207	0.8314
0:09:22.427823	10000	0.001	0.25	0.9	100	0.8212	0.8281
0:09:24.581425	10000	0.005	0.1	0.96	100	0.8219	0.8263
0:09:22.206234	10000	0.005	0.1	0.96	1000	0.8204	0.8248
0:08:45.562100	10000	0.001	0.1	0.96	1000	0.8229	0.8229
0:09:25.477988	10000	0.005	0.1	0.9	100	0.8045	0.8121
0:08:58.625777	10000	0.001	0.1	0.9	1000	0.8082	0.8089
0:08:27.853597	10000	0.001	0.1	0.96	100	0.7905	0.7993
0:09:38.538479	10000	0.01	0.1	0.9	1000	0.7856	0.7957
0:09:35.844166	10000	0.01	0.1	0.96	100	0.7867	0.7953
0:09:35.674369	10000	0.01	0.1	0.96	1000	0.78	0.7822
0:09:36.195275	10000	0.01	0.1	0.9	100	0.7535	0.7675
0:08:31.228494	10000	0.001	0.1	0.9	100	0.7419	0.7476