Artificial Intelligence Report 7th Week-23rd Problem

Le Trung Kien
03-120291, 3rd Year
Department of Mechano-Informatics
The University of Tokyo

January 14, 2013

1 Backpropagation Neural Network

The prototype for Backpropagation Neural Network (BnnnnPNN) is shown in Listing 1.

```
class BPNN{
private:
    vector<MatrixXd> weight;
    vector<MatrixXd> weightGrad;
    vector<MatrixXd> prevWeightGrad;
    vector<MatrixXd> delta;
    vector<MatrixXd> output;
    vector<MatrixXd> buffer;
    vector<int> layerSize;
    void Update(double learningRate, double momentum, int iteration, int maxIteration);
    bool CheckConvergence(double prev, double current, double min, double threshold);
public:
    enum WeightInitType {RANDOM, ZEROS};
    BPNN();
    BPNN(const vector<int> &layerSize);
    BPNN(int inputLayerSize, int hiddenLayerSize, int outputLayerSize);
    void Initialize(const vector<int>& layerSize);
    void Compute(const MatrixXd &in, MatrixXd& out);
    void Compute(const MatrixXd &in);
    void ComputeAll(const MatrixXd &in, MatrixXd& out);
    double Run(const MatrixXd &in, const MatrixXd &out);
    double RunEpoch(const MatrixXd &trainInput, const MatrixXd &trainOutput, double lambda);
    BPNN Train(const MatrixXd &trainInput, const MatrixXd &trainOuput,
               double learningRate, double lambda, double momentum,
               int maxIteration, double threshold);
    void Predict(const MatrixXd &in, MatrixXd &out, MatrixXd &outputLabel);
    void Save(const string& filename);
    void Load(const string& filename);
    void SetWeight(const vector<MatrixXd> &weight);
    void SetWeight(WeightInitType type = RANDOM);
    vector<MatrixXd> Weight() const;
    MatrixXd Output() const;
    vector<int> LayerSize() const;
    int InputLayerSize() const ;
    int OutputLayerSize() const ;
    int LayerNum() const ;
};
```

Listing 1: Backpropagation Neural Network Class (bpnn.hpp)

I am not going to talk much about BPNN's implementation, since it is very straightforward. Details about BPNN's theory can be seen in [1]. I just want to mention some important notes about the implementation:

- The training error and cross validation error is calculated as following:

$$E_{train} = \frac{1}{2mK}\sum_{1}^{m}\sum_{1}^{K}(y_k^{(l)} - t_k^{(l)})^2 + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$$E_{cross} = \frac{1}{2pK}\sum_{1}^{p}\sum_{1}^{K}(y_k^{(l)} - t_k^{(l)})^2$$

  In which, m is the size of training set, p is the size of cross validation set, K is the size of output, L is the number of neural network layers, Θ is neural network's weight, λ is regularization's parameter, y is output vector, t is desired output vector. When calculating training error, the bias is ignored.

- The most important method and its parameters of BPNN's class:

```
BPNN Train(const MatrixXd &trainInput, const MatrixXd &trainOuput,
           double learningRate, double lambda, double momentum,
           int maxIteration, double threshold);
```

  1. **trainInput** input for training
  2. **trainOuput** desired output for training
  3. **learningRate** decides convergence's speed
  4. **lambda** regularized parameter
  5. **momentum** increases convergence's speed
  6. **maxIteration** biggest number of iterations
  7. **threshold** threshold to decide when error function converges

  This method **returns** a trained BPNN which is ready for pattern's recognition.

- The usage of $momentum$ factor to accelarate convergence. The weight of neural network is updated as following:

$$\Theta_{new} = \Theta_{old} + \alpha\Delta_{new} + \alpha\beta\Delta_{old}$$

  In which, α is learning rate, β is momentum factor, Θ is weight, Δ is weight's gradient.

- The weight is randomly initialized before training.

2 Digit Recognition Application

2.1 Data preparation

All the training and testing data which consists of 900 grayscale images are pulled from database in [1].

http://research.microsoft.com/en-us/um/people/cmbishop/prml/

All images have size of 20x20. Pixel's value is scaled from [0,255] range to [0,1] range. They are put into a single xml file along side with a xml file containing respective labels. **Noise** is not directly generated

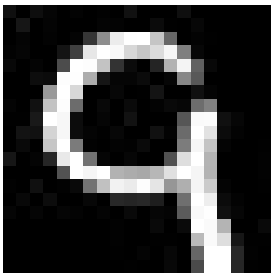

Figure 1: An image of 9

from data but labels. In this report, noise should be solely thought as mislabels not distorted imgages.

2.2 Experiments

In my experiements, data are divided into the following three sets.

1. **Training Set** is used for BPNN's training. Its size varies from 20 to 700.
2. **Cross Validation Set** is used to check for overfitting and optimal parameters' selection. It includes 100 images.
3. **Test Set** is used to check trained network's precision. It includes 100 images.

Firstly, I exam how training set's size affects BPNN's performance. In this experiment, hidden layer has 10 neurons, and regularized parameter is 0. As shown in Figure 2 and 3, as the size of the training set increases, the training error increases while the cross validation error decreases. Figure 2 shows that relationship when no noise involves, while Figure 3 shows it with 25% of noise. In both cases, the precision is going up as the training set's size increases, though in different trajectory. Secondly, we examine how BPNN's precision is reduced after adding noise to training set. Hidden layer size is 10, training set size is 300. According to Figure 4, the more noise the training set has, the larger the cross validation error is and the lower the precision is.
Finally, we try to find out how the size of hidden layer affects BPNN's precision and training error. Training set's size is 300, and noise is ignored. Figure 5 and 6 shows BPNN's performance according to the total number of neurons in hidden layer with or without noise, respectively. Finally, the usage of regularlization is considered. Figure 7 shows the BPNN's performance as the regularized parameter changes. Because regularlization is designed to counter overfit, only consider training set with noise in this case. Without noise, our training set is unlikely to spawn overfit. In this experiment, the training set size is 300, noise's ratio is 25%, hidden layer's size is 50.

3 Discussion

Noise in training set has great influence on neural network performance. With the presence of noise, a good decision boundary is harder to find, which leads to the decline of precision. In order to counter noise, complexity of the recognition's problem should be considered. If the network's complexity is lower than the problem's complexity, then it is likely to suffer underfit. On the contrary, if the network's complexity is higher than the problem's complexity, then it will cause overfit with the presence of noise.

According to Figure 3, one of the great way to suppress noise is to increase the size of training set. However big the noise's ratio in training set might be, if a large training set is available, we should be able to efficiently train the neural network. Another way to restrain noise's impact is to reduce the network's complexity, in other words, to have fewer neurons in hidden layer. The problem with this approach is that the neural network will become more susceptible to outliers or noise. Fortunately, we can use regularization to counter this trouble. As shown in Figure 7, the presence regularized parameter's existence alliviate noise impact. When regularized parameter is 0.1, we gain the highest precision for test's set, which is 68%. Notice that the growth of training error which is resulted from the increase of regularized parameter does not imply precision's reduction.

All the experiments suggest that not training error but cross validation error is the factor to affect precision in test set. While precision increases, cross validation error always decreases while training error may or may not. That emphasizes the importance of cross validation method. By looking at the learning curve, which consists of cross validation error and training error, underfit or overfit problem can be resolved.

References

[1] Christopher M. Bishop *Pattern Recognition and Machine Learning*, Springer 1st Edition, 2006

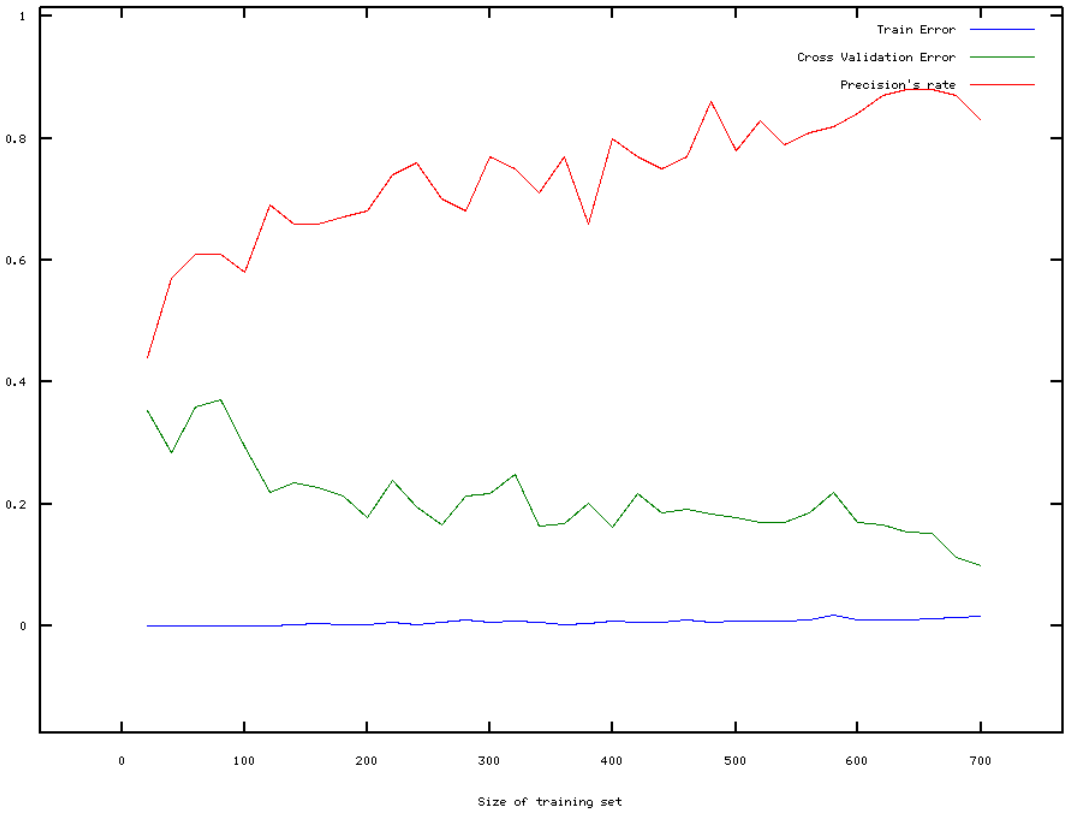

Figure 2: Training error and cross validation error as functions of training set's size without noise.

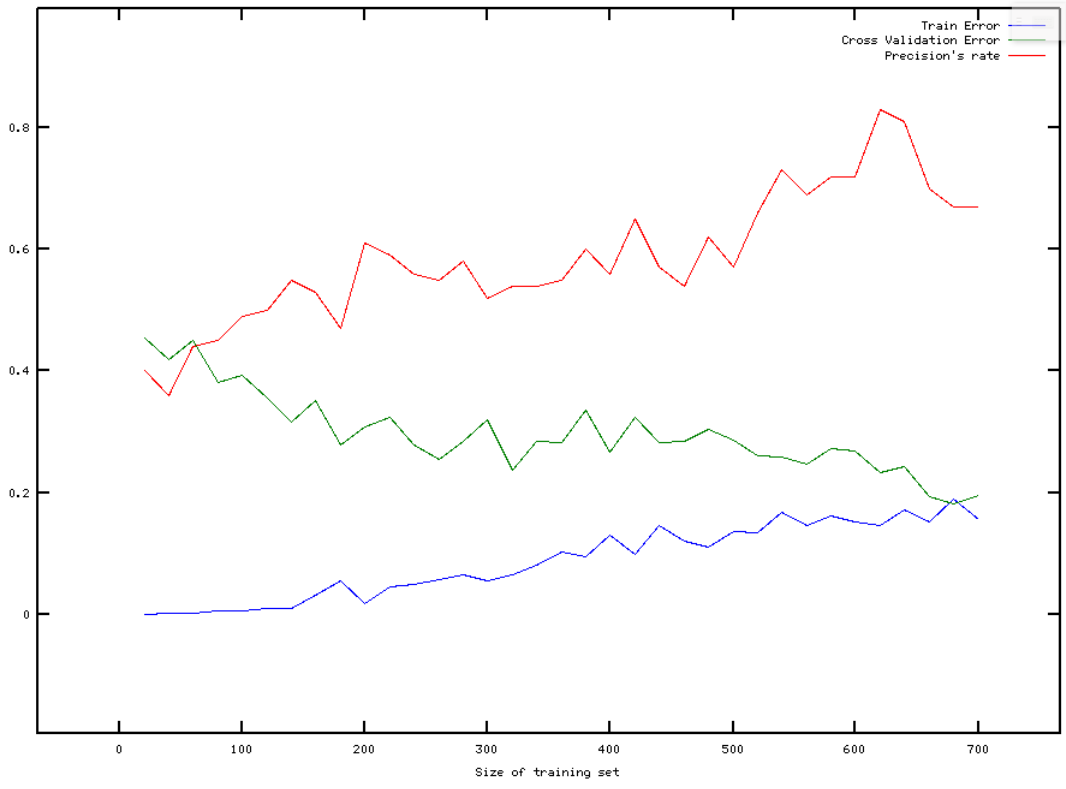

Figure 3: Training error and cross validation error as functions of training set's size with 25% of noise.

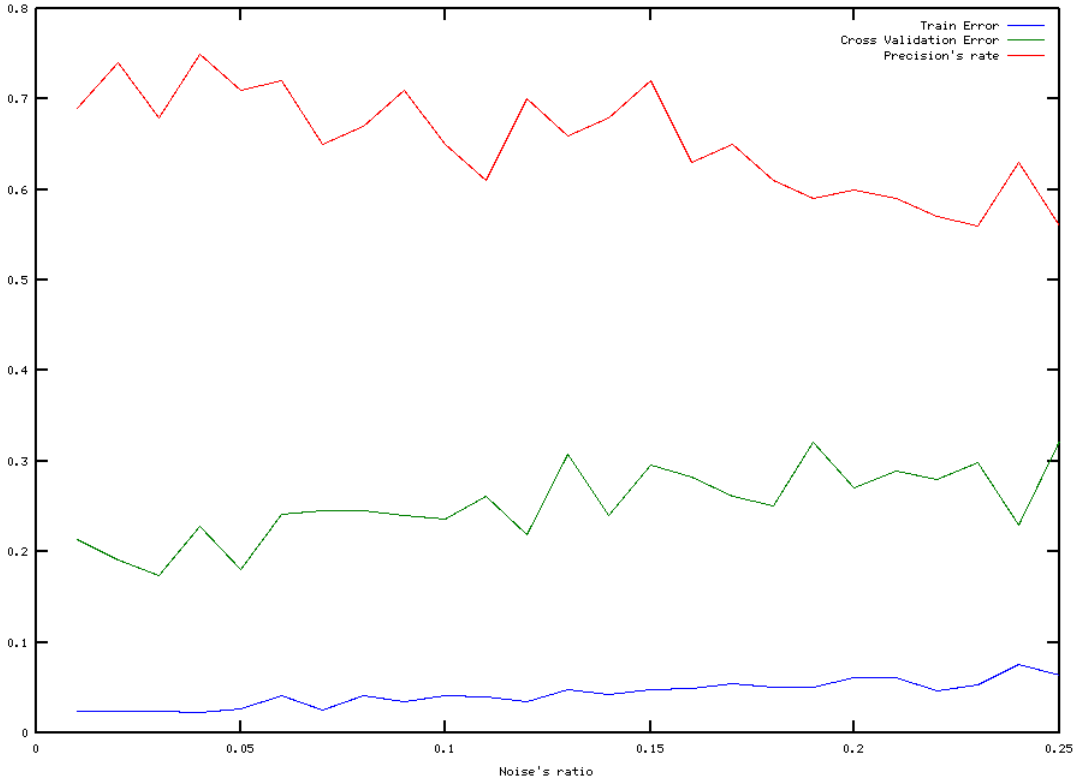

Figure 4: Training, cross validation errors and precision's relationships with noise's ratio in training set.

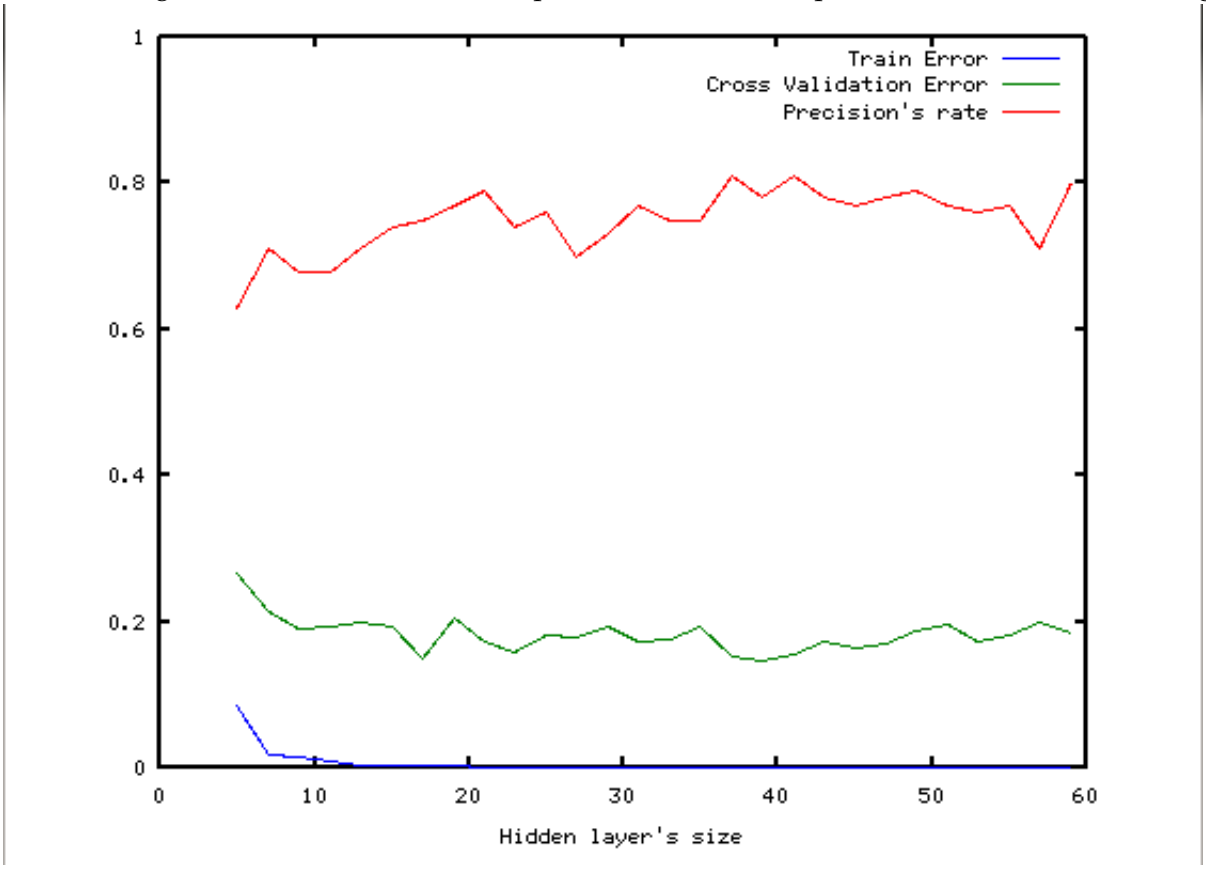

Figure 5: Training, cross validation errors and precision's relationships with hidden layer's size.

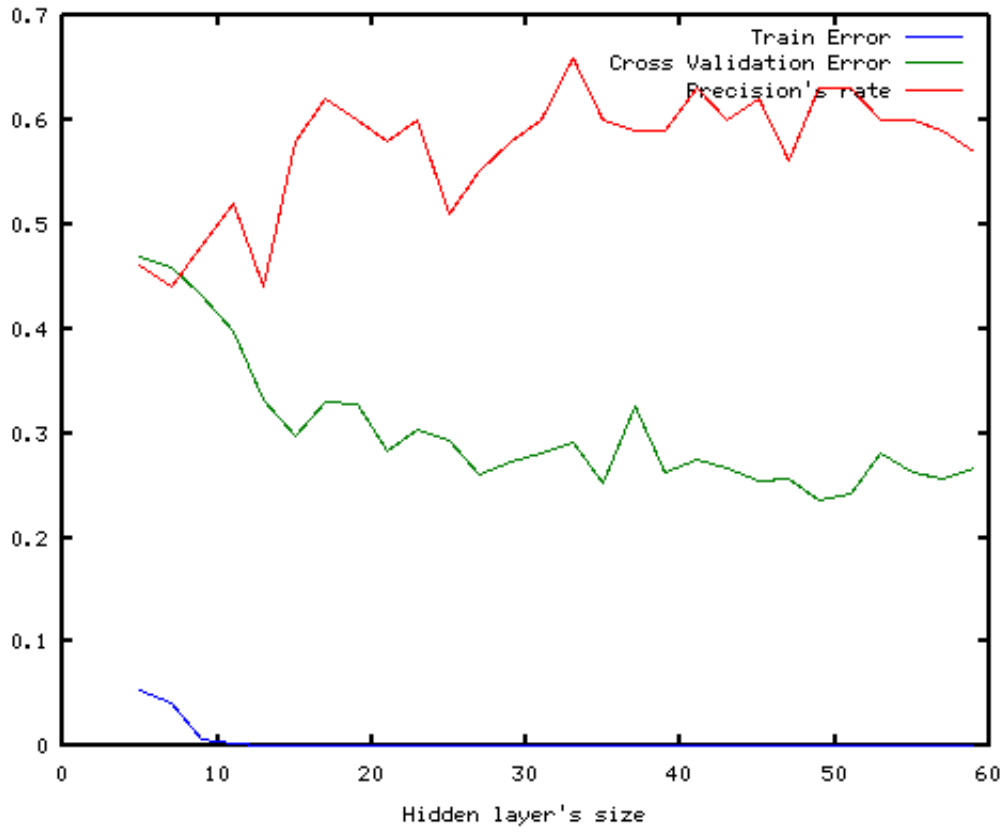

Figure 6: Training, cross validation errors and precision's relationships with hidden layer's size.

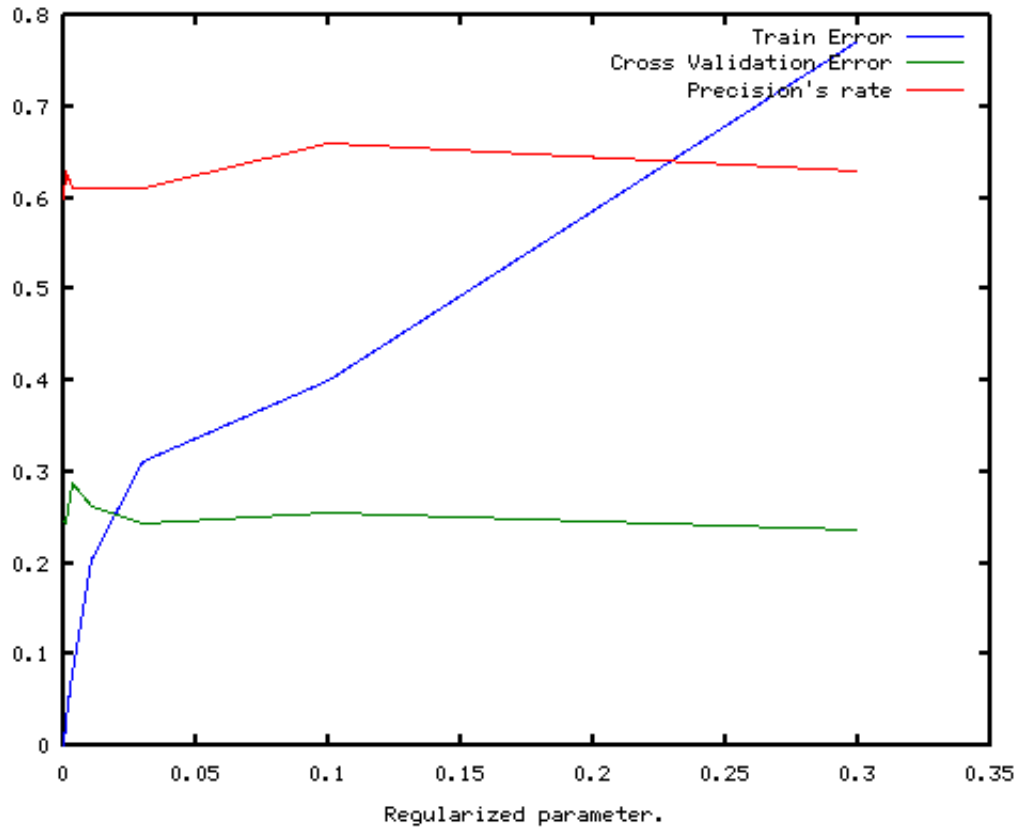

Figure 7: Training, cross validation errors and precision's relationships with regularized parameter.