CS 5350/6350: Machine Learning Fall 2022

Homework 2

Handed out: 29 Sep, 2022
Due date: 11:59pm, 21 Oct, 2022

1 Paper Problems [40 points + 8 bonus]

1. [5 points] We have derived the PAC guarantee for consistent learners (namely, the learners can produce a hypothesis that can 100% accurately classify the training data). The PAC guarantee is described as follows. Let H be the hypothesis space used by our algorithm. Let C be the concept class we want to apply our learning algorithm to search for a target function in C. We have shown that, with probability at least $1-\delta$, a hypothesis $h \in H$ that is consistent with a training set of m examples will have the generalization error $\text{err}_D(h) < \epsilon$ if

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big).$$

   (a) [2 points] Suppose we have two learning algorithms L_1 and L_2, which use hypothesis spaces H_1 and H_2 respectively. We know that H_1 is larger than H_2, i.e., $|H_1| > |H_2|$. For each target function in C, we assume both algorithms can find a hypothesis consistent with the training data.

      i. [1 point] According to Occam's Razor principle, which learning algorithm's result hypothesis do you prefer? Why?
      I would prefer using algorithm L2's hypothesis because it has a smaller hypothesis space. Occam's Razor states that we should use simpler explanations over complex ones and since $|H_1| > |H_2|$ the result hypothesis from L2 would be preferable.

      ii. [1 point] How is this principle reflected in our PAC guarantee? Please use the above inequality to explain why we will prefer the corresponding result hypothesis.
      In the above inequality we can see that as the size of the hypothesis space increases, the number of examples required to guarantee a generalization below ϵ also increases. Therefore if we choose an algorithm with a smaller hypothesis space, we are more likely to get a lower generalization error which would mean that the hypothesis result from L2 would be preferable.

   (b) [3 points] Let us investigate algorithm L_1. Suppose we have n input features, and the size of the hypothesis space used by L_1 is 3^n. Given $n = 10$ features,

if we want to guarantee a 95% chance of learning a hypothesis of at least 90% generalization accuracy, how many training examples at least do we need for L_1? Given $n = 10$ input features, $|H_1| = 3^{10} = 59049$. From the problem we can see that $\epsilon = 0.1$ and $\delta = 0.05$. Plugging these into the equation we get

$$\frac{1}{0.1}(log(59049) + log(\frac{1}{0.05})) = 202$$

This means that we would need to have at least 202 examples to have a 95% chance of getting accuracy of at least 90%.

2. [5 points] In our lecture about AdaBoost algorithm, we introduced the definition of weighted error in each round t,

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\big)$$

where $D_t(i)$ is the weight of i-th training example, and $h_t(x_i)$ is the prediction of the weak classifier learned round t. Note that both y_i and $h_t(x_i)$ belong to $\{1, -1\}$. Prove that equivalently,

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

We know that both y_i and $h_t(x_i)$ belong to $\{1, -1\}$ which means that $y_i * h_t(x_i)$ will always be equal to 1 when $y_i = h_t(x_i)$ and -1 when $y_i \neq h_t(x_i)$. Therefore we can rewrite the original definition as

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\big) = \frac{1}{2} - \frac{1}{2}\big(\sum_{y_i = h_t(x_i)} D_t(i) - \sum_{y_i \neq h_t(x_i)} D_t(i)\big)$$

$$= \frac{1}{2} - \frac{1}{2}\big(\sum_{y_i = h_t(x_i)} D_t(i)\big) + \big(\frac{1}{2}\sum_{y_i \neq h_t(x_i)} D_t(i)\big)$$

Because the all of the weights add up to 1, 1 minus the sum of weights when $y_i = h_t(x_i)$ will be equal to the sum of the weights when $y_i \neq h_t(x_i)$ Therefore the above equation is equal to

$$= \frac{1}{2}\sum_{y_i \neq h_t(x_i)} D_t(i) + \frac{1}{2}\sum_{y_i \neq h_t(x_i)} D_t(i) = \sum_{y_i \neq h_t(x_i)} D_t(i) = \epsilon_t$$

3. [20 points]

(a) [2 point] $f(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge \neg x_3$
$y = 1$ if $x_1 + (1 - x_2) + (1 - x_3) \geq 3$
$y = 1$ if $x_1 - x_2 - x_3 \geq 1$
$f(x_1, x_2, x_3) = sgn(x_1 - x_2 - x_3 - 1)$
weight vector = [1,-1,-1], bias = -1

(b) [2 point] $f(x_1, x_2, x_3) = \neg x_1 \vee \neg x_2 \vee \neg x_3$
$y = 1$ if $(1 - x_1) + (1 - x_2) + (1 - x_3) \geq 1$
$f(x_1, x_2, x_3) = sgn(-x_1 - x_2 - x_3 + 2)$
weight vector = [-1,-1,-1], bias = 2

(c) [8 points] $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$
This is not linearly separable because there is no linear combination of the given variables that can exactly partition the data according to y. In order to make this function linearly seperable we need to map the given features to a higher dimensional space. The feature mapping that I came up with is as follows:

$$z_1 = sgn(x_1 + x_2 - 1), \quad z_2 = sgn(x_3 + x_4 - 1)$$

After mapping the existing features to these new features, we can get the following hyper plane that well separates the inputs:

$$z_1 + z_2 \geq 2 \quad f(z_1, z_2) = sign(z_1 + z_2 - 2)$$

(d) [8 points] $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
This is not linearly separable because there is no linear combination of the given variables that can exactly partition the data according to y. In order to make this function linearly seperable we need to map the given features to a higher dimensional space. The feature mapping that I came up with is as follows:

$$z_1 = sgn(x_1 + x_2 - 2), \quad z_2 = sgn(-x_1 - x_2)$$

After mapping the existing features to these new features, we can get the following hyper plane that well separates the inputs:

$$z_1 + z_2 \geq 1 \quad f(z_1, z_2) = sign(z_1 + z_2 - 1)$$

4. [**Bonus**] [8 points] Given two vectors $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$, find a feature mapping $\phi(\cdot)$ for each of the following functions, such that the function is equal to the inner product between the mapped feature vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$. For example, $(\mathbf{x}^\top \mathbf{y})^0 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1]$ and $\phi(\mathbf{y}) = [1]$; $(\mathbf{x}^\top \mathbf{y})^1 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = \mathbf{x}$ and $\phi(\mathbf{y}) = \mathbf{y}$.

(a) [2 points] $(\mathbf{x}^\top \mathbf{y})^2$

(b) [2 points] $(\mathbf{x}^\top \mathbf{y})^3$

(c) [4 points] $(\mathbf{x}^\top \mathbf{y})^k$ where k is any positive integer.

5. [10 points]

(a) [1 point] Write down the LMS (least mean square) cost function $J(\mathbf{w}, b)$.

$$J(\mathbf{w}) = \frac{1}{2}(\sum_{i=1}^{m}(y_i - \mathbf{w}^T x_i + b)^2$$

(b) [3 points] Calculate the gradient $\frac{\nabla J}{\nabla \mathbf{w}}$ and $\frac{\nabla J}{\nabla b}$ when $\mathbf{w} = [-1, 1, -1]^\top$ and $b = -1$.

From class we learned that we can use the following equation to solve for the gradient assuming that we add an extra variable that is always equal to 1 for the bias.

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

I used python to implement the above formula to get the following results.
$\frac{\partial J}{\partial w_1} = -22$
$\frac{\partial J}{\partial w_2} = 16$
$\frac{\partial J}{\partial w_3} = -56$
$\frac{\partial J}{\partial b} = -10$
The gradient of the weight vector is $\frac{\nabla J}{\nabla \mathbf{w}} = [-22, 16, -56]$ and the gradient of the bias is $\frac{\nabla J}{\nabla b} = -10$

(c) [3 points]

We can solve for the optimal weight vector and bias that minimize the LMS cost function analytically via the equation $(XX^T)^{-1}XY$. In order to get the bias we add another variable to the X matrix where every value is 1 so that the weight multiplied by that new variable will be the bias. Using python I plugged in the training data into the above equation and found that the optimal weight vector was $\mathbf{w}^T = [1, 1, 1]$ and the optimal bias was $\mathbf{b} = -1$

(d) [3 points]

In stochastic gradient descent we only take the gradient using one example at a time and update the weights and bias after every example we go through. The update equation for stochastic gradient descent is

$$w_j^{t+1} = w_j^t + r(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

Using python I performed the above formula for five examples and computed the following gradients and updated weights and bias.
After 1st example: Stochastic gradient of weights: $[1, -0.9, 1.62]$, Stochastic gradient of bias: 0.486, $\mathbf{w} = [0.1, -0.09, 0.162]$, $\mathbf{b} = 0.049$
After 2nd example: Stochastic gradient of weights: $[3.46, 3.11, 8.4]$, Stochastic gradient of bias: 0.28, $\mathbf{w} = [0.445, 0.221, 1.0]$, $\mathbf{b} = 0.08$
After 3rd example: Stochastic gradient of weights: $[0.852, -0.767, 0]$, Stochastic gradient of bias: -0.69, $\mathbf{w} = [0.531, 0.144, 1.0]$, $\mathbf{b} = 0.008$
After 4th example: Stochastic gradient of weights: $[1.18, 3.12, -2.55]$, Stochastic gradient of bias: -0.38, $\mathbf{w} = [0.649, 0.357, 0.747]$, $\mathbf{b} = -0.031$
After 5th example: Stochastic gradient of weights: $[-2.44, 0.081, 0.073]$, Stochastic gradient of bias: -0.066, $\mathbf{w} = [0.405, 0.365, 0.754]$, $\mathbf{b} = -0.037$

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of decision trees in HW1 to your GitHub repository. Remember last time you created a folder "Decision Tree". You can commit your code into that folder. Please also supplement README.md with concise descriptions about how to use your code to learn decision trees (how to call the command, set the parameters, etc). Please create two folders "Ensemble Learning" and "Linear Regression" in the same level as the folder "Decision Tree".

2. [36 points]

   (a) [8 points] The first figure displaying training and testing errors shows that as the number of iterations increases, the training error decreases. The testing error begins to increase after about 50 iterations. This means that after about 50 iterations the AdaBoost algorithm begins overfitting to the training data. The next figure displays all of the training and test error for every stump in every run of the AdaBoost algorithm. Do to the high level of crowding in this plot, I also plotted the means of the stump training and test error for every iteration in the next figure. From these plots we can see that as the number of stumps increases, the training and testing error of the stumps also increases which is in line with what we learned in class. On this data, a fully expanded tree had a testing error of 0.155 and a training error of 0.013. The training error of a fully expanded tree is significantly better than that of AdaBoost but the testing error was significantly worse, even when AdaBoost was ran for 500 iterations. This tells me that the single tree overfits much more to the data and that for making new predictions, AdaBoost will be better.

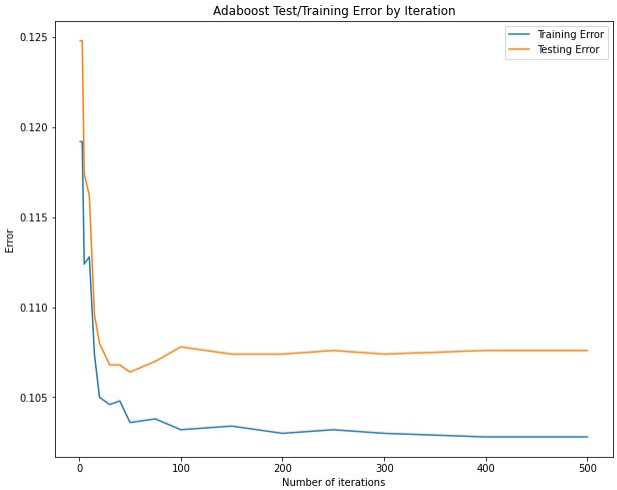

Figure 1: Training and testing errors of the AdaBoost algorithm. Problem 2a

   (b) [8 points]

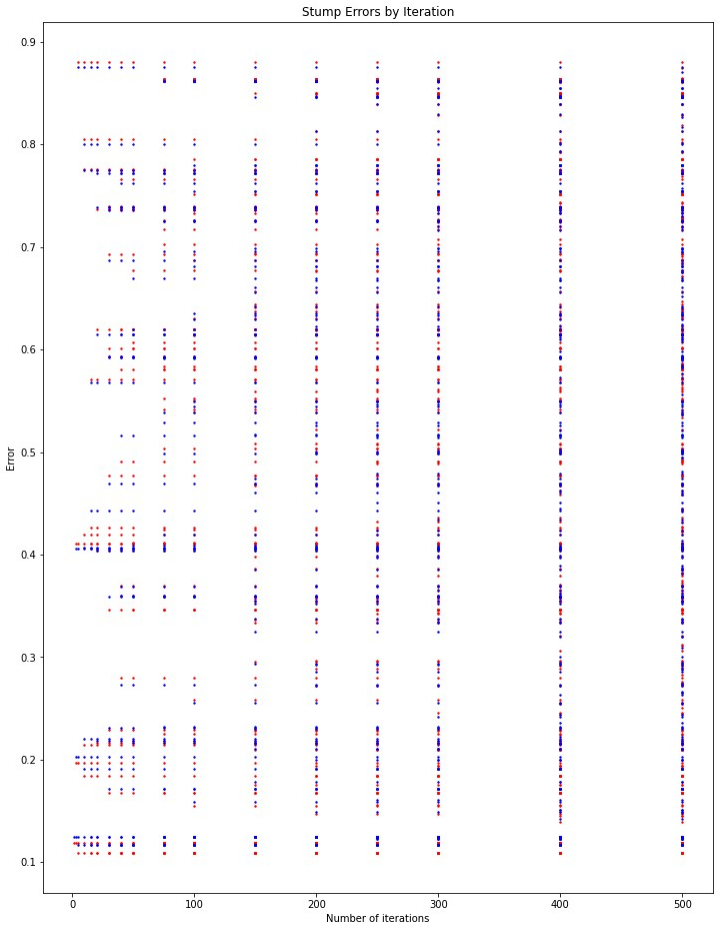

Figure 2: All stump errors by iteration. Problem 2a

From this figure I can conclude that the training error decreases until about 50 iterations and then remains fairly constant. The testing error seems to reach its lowest point at about 20 iterations and then increases which tells us that after about 20 iterations, the bagging trees algorithm begins to overfit to the data. The training error was 0.016 and the testing error reached a low of 0.139. Like the fully expanded tree, the bagged trees learn the training data much better than the AdaBoost algorithm but also begins to overfit more than AdaBoost. The low testing error of the bagged trees algorithm was better than that of a single fully expanded tree but was still worse than the AdaBoost algorithm. I believe out of the three methods tested so far AdaBoost is best since it generalizes the best to new data.

(c) [6 points]

I got the following results for bias/variance decomposition of the bagged trees:
Single Trees Bias: 0.355553
Single Trees Variance: 0.364100
Single Trees Estimated Squared Error: 0.719653
Bagged Trees Bias: 0.356796

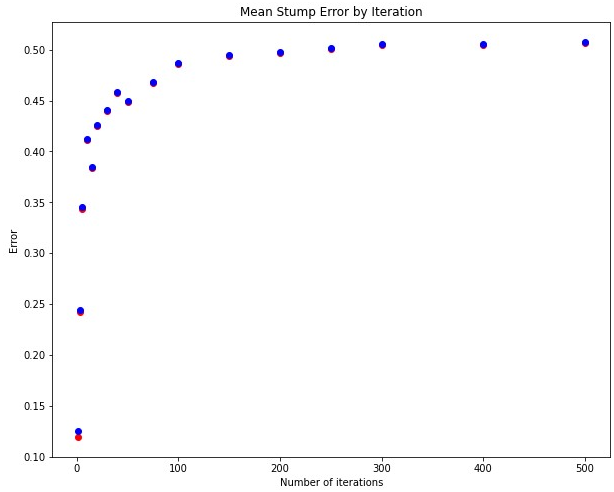

Figure 3: Mean stump errors by iteration. Problem 2a

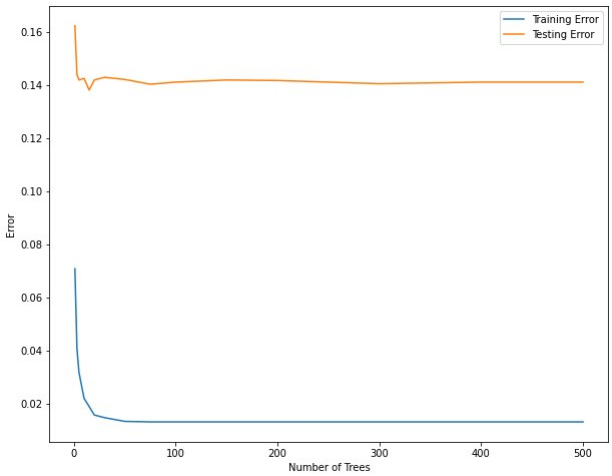

Figure 4: Mean stump errors by iteration. Problem 2b

Bagged Trees Variance: 0.212884
Bagged Trees Estimated Squared Error: 0.569680

The bias of the bagged trees was slightly higher than that of the single trees but the variance and estimated squared error of the bagged trees was significantly lower than that of the single trees. This is because bagging averages a set of many decision trees which largely reduces the variance as the number of decision trees grows.

(d) [8 points]

From this figure we can see that all 3 of the subset sizes perform similarly with a subset size of two performing slightly better on the training data, and a subset size of 4 performing slightly better on the testing data. We can also see that there does not seem to be any significant overfitting with the random forest algorithm.

While the testing errors do seem to plateau after about 120 iterations, it does not ever go up significantly. The best training error of the random forest was 0.173 and the best testing error, with a subset size of 4, was 0.122. This is a better testing performance than with bagged trees, likely due to the increased variance in the trees built due to random subsets of attributes.

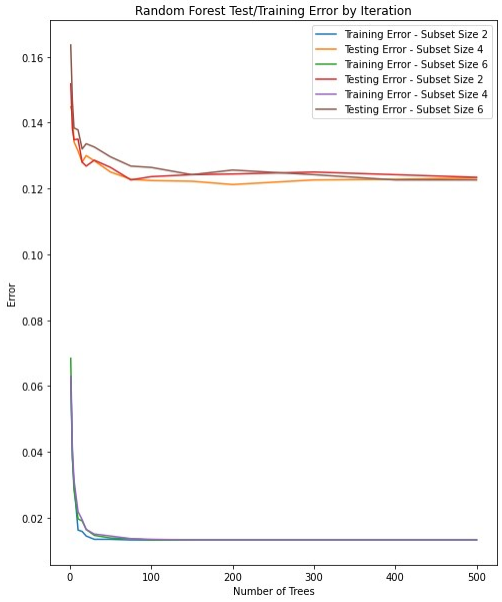

Figure 5: Random Forest training and testing errors. Problem 2d

(e) [6 points]

For Random forest I got the following results for bias/variance decomposition: Single Trees Bias: 0.365423 Single Trees Variance: 0.318543 Single Trees Estimated Squared Error: 0.683966 Random Forest Bias: 0.430535 Random Forest Variance: 0.061495 Random Forest Estimated Squared Error: 0.492030

In this experiment with a random forest, I found that the bias for the random forest was somewhat significantly higher than the bias for the single tree and for the bagged trees. However the variance for the random forest was significantly lower than that of both the single tree and even the bagged trees. From this I would say that a random forest is a better overall predictor because variance of it's error and it's estimated squared error is significantly lower than that of bagged trees.

3. [**Bonus**][10 points] In practice, to confirm the performance of your algorithm, you need to find multiple datasets for test (rather than one). You need to extract and process data by yourself. Now please use the credit default dataset in UCI repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Randomly choose 24000 examples for training and the remaining 6000 for test. Feel free to deal with continuous features. Run bagged trees, random forest, and Adaboost with decision stumps algorithms for 500 iterations. Report in a figure how the training and

test errors vary along with the number of iterations, as compared with a fully expanded single decision tree. Are the results consistent with the results you obtained from the bank dataset?

4. [22 points]

   (a) [8 points]
   After experimenting with the learning rate I found an appropriate r so that the algorithm converges was $r = 0.01$. The resulting weight vector was **w**= [0.9002 0.7859 0.8507 1.2986 0.1298 1.5718 0.9983] and the bias was -0.015204. The final cost of the learned function on the test data was 23.36.

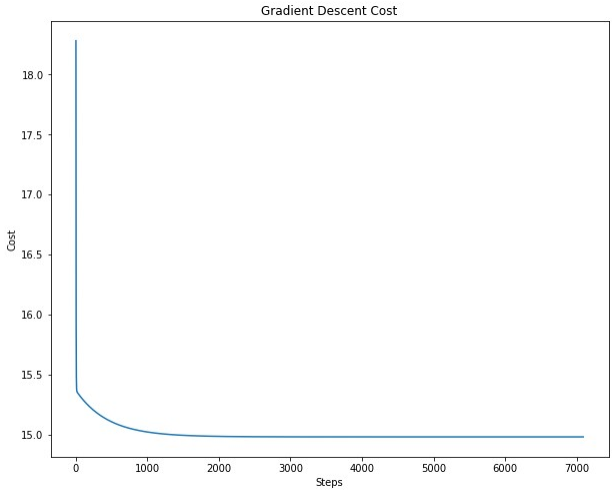

Figure 6: Costs over time for gradient descent. Problem 4a

   (b) [8 points] I found that by using a learning rate of 0.003 and 100 iterations, I was able to converge with stochastic gradient descent. The resulting weight vector was [-0.0714 -0.2142 -0.2266 0.5146 -0.0251 0.2627 0.0102] and the bias was -0.044279. The testing cost after applying the final learned weight vector and bias was 22.78.

   (c) [6 points]
   Using Python I used the following formula on our data to get the optimal weight vector

$$(XX^T)^{-1}XY$$

   This produced the following weight vector: **w**= [0.9006, 0.7863, 0.851, 1.2989, 0.1299, 1.5722, 0.9987] with an optimal bias of -0.015197. This weight vector was very similar to the one learned in batch gradient descent which leads me to believe that this algorithm converged to the optimum. However, the learned weight vector from stochastic gradient descent was almost entirely different, even though the costs on the test and training data were very similar to those of batch gradient descent. This leads me to believe that stochastic gradient descent did not ultimately converge to the optimal weights. Maybe with a decaying learning

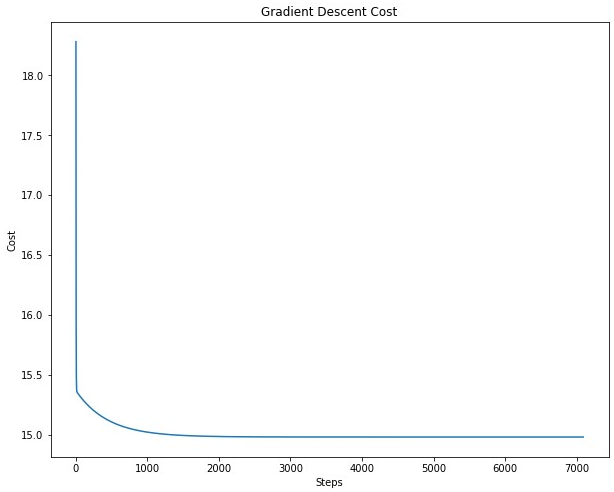

Figure 7: Costs over time for stochastic gradient descent. Problem 4b

rate and significantly more iterations stochastic gradient descent would converge on the optimum.