CS 361: Probability and Statistics for Computer Science **(Spring 2025)**

Stochastic Optimization Project

1 Stochastic Optimization Theory Part

1.1 A common stochastic optimization task

In many machine learning problems, we are trying to minimize function $\boldsymbol{f(\theta)}$ in the following format.

$$f(\theta) = \frac{1}{k}\sum_{j=1}^{k} Q(\theta, j) \tag{1.1}$$

where $Q(\boldsymbol{\theta}, j)$ is the loss function for j^{th} data point where we have k training data points in the data set D. Here, $\boldsymbol{\theta}$ is the set of parameters of the model that we try to optimize for, that is we want to find $\boldsymbol{\theta^*} = \arg\min_\theta f(\theta)$. However, f is either unknown or very expensive to collect, but we have access to the noisy version $\boldsymbol{g}(\theta)$. Using $\boldsymbol{g}(\theta)$ to find the optimized $\boldsymbol{\theta^*}$ for $\boldsymbol{f(\theta)}$ would be a stochastic optimization task.

Exercise 1.
(4 points) Define the random variable $\boldsymbol{g(\theta)}$ specifically here to be $Q(\boldsymbol{\theta, i})$, where $\boldsymbol{i}$ is a random variable that is of uniform distribution in the integer set $[1:k]$. So, $\nabla g(\theta)$ is to be $\nabla Q(\theta, i)$. Further, we define the noise $z = Q(\theta, i) - f(\theta)$, hence $g(\theta) = Q(\theta, i) = f(\theta) + z$. That's why we call g is the noisy version of f. Given this definition of f and g, prove that equations $E[g(\theta)] = f(\theta)$, $E[\nabla g(\theta)] = \nabla f(\theta)$, and $E[z] = 0$ are satisfied.

1.2 Review of GD and SGD

The goal of optimization is to find the θ^* that minimizes $f(\theta)$, which could be more general than that in (1.1). Again sometimes f is either unknown or very expensive to collect, but we have access to the noisy version $g(\theta)$. The following are two approximation methods to find θ^* iteratively.

1.2.1 Gradient Descent Algorithm (GD)

- Initialize θ_1
- For $t = 1, 2, \cdots$ Then do the following update

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t) \tag{1.2}$$

1.2.2 Stochastic Gradient Descent Algorithm (SGD)

- Initialize θ_1
- For $t = 1, 2, \cdots$. Obtain a noisy version of the gradient (i.e. $\nabla g(\theta_t)$). Then do the following update

$$\theta_{t+1} = \theta_t - \eta_t \nabla g(\theta_t) \tag{1.3}$$

1.3 Convergence rates for SGD and GD

Suppose we want to minimize function f with learning rate sequence η_t, We have the following theorems:

Theorem 1.1 *Assume ∇f has a unique root θ^*. If f is twice continuously differentiable and strongly convex, and $\eta_t = O(t^{-1})$, then to reach an approximation error $\epsilon = |E[f(\theta_t)] - f(\theta^*)|$, **stochastic gradient descent** requires $t = O(\frac{1}{\epsilon})$ updates.*

Theorem 1.2 *Assume ∇f has a unique root θ^*. If either the smoothness assumptions about f in theorem 1.1 or $\eta_t = O(t^{-1})$ is not met, then to reach an approximation error $\epsilon = |E[f(\theta_t)] - f(\theta^*)|$, **stochastic gradient descent** requires $t = O(\frac{1}{\epsilon^2})$ updates.*

Theorem 1.3 *Assume ∇f has a unique root θ^*. To reach an approximation error $\epsilon = |E[f(\theta_t)] - f(\theta^*)|$, **gradient descent** requires $t = O(\ln(\frac{1}{\epsilon}))$ updates.*

2 Comparing SGD vs GD for their running time

Let us consider the f minimization task discussed earlier in (1.1).

- $f(\theta)$ and $g(\theta)$ are as described in exercise 1, $\nabla f(\theta^*) = 0$, where θ^* is the unique solution.
- Let's assume that f is strongly convex, and is twice continuously differentiable.
- We use the $\eta_t = \frac{1}{t}$ learning rate sequence.
- We have k data points.
- We want to achieve a training precision of $\boldsymbol{\epsilon} = |E[f(\theta_t)] - f(\theta^*)|$.
- Assume finding $\nabla Q(\theta, j)$ takes c time, and finding $\nabla f(\theta)$ takes kc time.

Exercise 2.
(15 points) Fill in table 1, with complexity orders in terms of k and ϵ

Table 1 Comparing SGD vs GD in terms of training precision

	SGD	GD
Computational Cost per Update	$O(1)$	
Number of updates to reach ϵ		
Total Cost		

Explain your answer for each element of the second row by providing a reference to the theorem mentioned in this document. For every other element, **explain your answer**.

3 Comparing SGD vs GD with respect to test loss

The last exercise was about finding the computational complexity required to reach a specific precision with respect to the optimal **training** loss. However, a specific precision of training loss does not translate to the same precision of **test** loss. Here are some helpful facts:

- To achieve a specific test loss precision ρ, you need a large enough number of training samples k. The relation
$$\rho = O(k^{-\gamma})$$
must hold for most loss functions where $0 < \gamma \leq 1$ is an unknown constant.
- Let's assume $\epsilon = \Theta(\rho)$; for simplicity, you can assume that it is as if ϵ is $c \times \rho$ where c is a positive constant.

Exercise 3.
(16 points) 1) Fill in table 2, with complexity orders in terms of ρ and γ. You can refer to the previous table, and replace the elements with appropriate values. Make sure you **state the reason for each element** even if it looks obvious.

Table 2 Comparing SGD vs GD in terms of testing precision

	SGD	GD
Computational Cost per Update	$O(1)$	
Number of updates to reach ρ		
Total Cost		

2) For a typical $\gamma = 0.2$, Explain why choosing GD over SGD can be very unwise.

CS 361: Probability and Statistics for Computer Science **(Spring 2025)**

Stochastic Optimization Implementation

Objective: Implement state-of-the-art stochastic optimization algorithms for Machine Learning Problems, Linear Regression and Classification (using Neural Networks).

3.1 Adaptive Momentum Estimation (ADAM) Gradient Descent Algorithm

SGD can be ineffective when the function is highly non-convex. Unfortunately, most modern applications of Machine Learning involve non-convex optimization problems. One stochastic optimization algorithm that works well even for non-convexity is ADAM [KB14]. ADAM uses past gradient information to "speed" up optimization by smoothing, and the algoirthm has been further improved [SSS18]. You will implement the ADAM stochastic optimization algorithm for a Linear Regression problem.

The pseudo-code for ADAM has been reproduced here from this paper. Credits to [KB14].

Disclaimer: The textbook, in Chapter 13, uses β for parameters but we will be using θ.

Algorithm 1: g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t, we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\epsilon$: Division-by-zero control parameter
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$.
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
      $t \leftarrow t + 1$
      $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
      $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
      $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
      $\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate).
      $\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-correct second raw moment estimate).
      $\hat{\theta}_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Exercise 4.
Consider the following problem setting:

- The number of training data points is $k = 1000$. The number of test data points is 50.
- Generation of the data set to fit for the model: the j^{th} data point is represented by $\boldsymbol{x_j}$ and is a 10-dimensional vector. Each element of $\boldsymbol{x_j}$ is being generated from a uniform distribution over the interval $[-1, 1]$.
- $\boldsymbol{\theta}_{true}$ (i.e. the true parameter set) and $\boldsymbol{\theta}$ (i.e. the variable indicating a candidate parameter set) are also 10-dimensional vectors. Assume that $\boldsymbol{\theta}_{true}$ is all ones. However, we will pretend that we do not know it and we want to estimate it using the training data.
- Data is being generated according to $y_j = \boldsymbol{x_j}^T \boldsymbol{\theta}_{true} + 0.1 \cdot \xi$ (where $\xi \sim \mathcal{N}(0, 1)$ is a sample from the normal distribution). The label y_j is a scalar.

- Let us assume that all the algorithm variants start from the same initial $\boldsymbol{\theta}$, whose elements are picked uniformly in the interval $[0, 0.1]$.
- Use 1000 updates.

Since this is an optimization problem, we need to define the loss function. Let's use the following format

$$F(\boldsymbol{\theta}) = \frac{1}{k}\sum_{j=1}^{k} Q(\boldsymbol{\theta}, j) \tag{3.1}$$

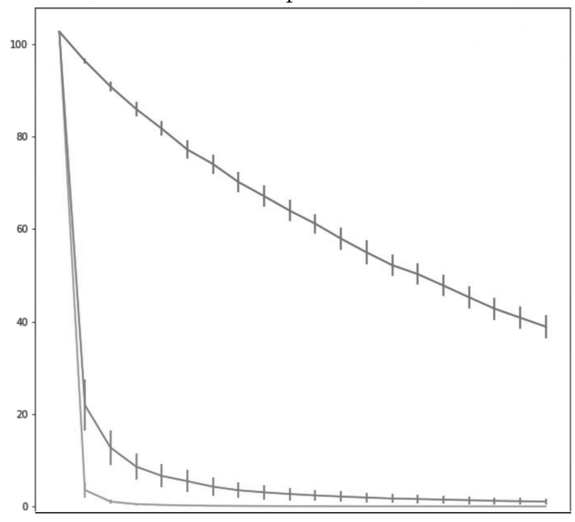
$$Q(\boldsymbol{\theta}, j) = \left| \boldsymbol{x_j}^T\boldsymbol{\theta} - y_j \right|^{\gamma} \tag{3.2}$$

Where γ is a hyper-parameter that we use to control and define the loss function.

For the following problems, state findings, provide analysis, and substantiate them in your write-up with screenshots of the plots from the **Gradient Descent Python notebook** code. Please ***save the notebook to your own Google drive*** so that your work could be preserved after the runtime expires.

1. **(5 points)** For $\gamma = 2$, the problem becomes the least squares regression that you learned from Chapter 13. State the closed form solution (i.e. $\boldsymbol{\theta} = ...$) in your report, and then implement the solution to solve for $\boldsymbol{\theta}$. Provide the results of the experiment and state whether it is close to the true value.
2. **(5 points)** For $Q(\boldsymbol{\theta}, j)$, find an expression that gives you the gradient $\nabla Q(\boldsymbol{\theta}, j)$. Report this expression, and implement it in the appropriate function.
   hint: For $r(\boldsymbol{\theta}) = h(e(\boldsymbol{\theta})) = h(\boldsymbol{x_j}^T\boldsymbol{\theta} - y_j)$, the gradient can be written as $\nabla r(\boldsymbol{\theta}) = \frac{\partial h}{\partial e} \cdot \nabla e(\boldsymbol{\theta}) = \frac{\partial h}{\partial e} \cdot \boldsymbol{x_j}$ according to the chain rule.
   hint: The sign function, $sgn(x)$, may be useful.
3. **(5 points)** Code the ADAM optimization algorithm (with default hyper-parameters such as the learning rate as mentioned in the pseudocode above) and SGD to find the best parameter $\boldsymbol{\theta}$. Use a batch size of 1 for this problem, and perform 1000 parameter updates. Report the final set of parameters.
4. **(5 points)** Update your code to compute the average test loss at every 50th update, and plot these values. You might notice that the error bars of ADAM and SGD overlap. This is due to the inherent randomness from sampling values. To avoid this probabilistic overlap, increase the number of replicates (num_rep in the starter code) until the error bars between ADAM and SGD do not overlap. Report this curve.
5. **(9 points)** Run ADAM method for each of $\gamma = 0.4, 0.7, 1, 2, 3,$ and 5. Report your observations including all the plots clearly, and analyze the trends you are seeing. State whether ADAM consistently out performs SGD. Your analysis should include the reason why one method outperforms the other under each setting.

Example Plot



Note: Your plots will probably not end up looking exactly like this one.

3.2 Classifying Handwritten Digits with Neural Networks

Next, we'll use Neural Networks to classify handwritten digits from MNIST dataset (dataset of handwritten digits). The objective is to use different stochastic optimization algorithms that you have seen so far and compare their performances (GD, SGD, ADAM). You will train the model and then classify handwritten digits.

Fun fact: One of the co-creators of MNIST dataset (Dr. Yann LeCun) is also the co-recicpient of 2018 ACM Turing award for his work on Neural Networks.

Exercise 5.
For the following problems, please modify the **Neural Networks Python notebook** code.

1. To run the Gradient Descent (GD) Algorithm: Set the (mini) batch size to 60000 (the size of the MNIST dataset). Run the SGD optimizer with a learning rate of 0.003.

   (1 points) Why is this the same as the GD algorithm if we are using SGD optimizer?

   (2 points) What do you observe? Report the accuracy.

   (2 points) List at least 2 ways to improve the accuracy.

2. To run the Stochastic Gradient Descent (SGD) Algorithm: Set the (mini) batch size to 64. Run the SGD optimizer with a learning rate of 0.003.

   (2 points) What do you observe? Report the accuracy.

   (1 points) List at least one way to improve accuracy further.

3. To run the ADAM algorithm: Set the (mini) batch size to 64. Run the ADAM optimizer with default learning rates (Algorithm 1)

   Hint: You can use Pytorch (or any other library in any language) for setting up the ADAM optimizer.

   (2 points) What do you observe? Report the accuracy.

   (1 points) Why does ADAM converge faster than SGD?

References

[RM51] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The annals of mathematical statistics* (1951), pp. 400–407.

[Sac58] Jerome Sacks. “Asymptotic distribution of stochastic approximation procedures”. In: *The Annals of Mathematical Statistics* 29.2 (1958), pp. 373–405.

[NY83] Semenovich Nemirovsky Arkadi and David Borisovich Yudin. “Problem complexity and method efficiency in optimization.” In: *Wiley-Interscience series in discrete mathematics.* (1983).

[CZ07] Felipe Cucker and Ding Xuan Zhou. *Learning theory: an approximation theory viewpoint.* Vol. 24. Cambridge University Press, 2007.

[Nem+09] Arkadi Nemirovski et al. “Robust stochastic approximation approach to stochastic programming”. In: *SIAM Journal on optimization* 19.4 (2009), pp. 1574–1609.

[Bot10] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010.* Springer, 2010, pp. 177–186.

[Bot12] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural networks: Tricks of the trade.* Springer, 2012, pp. 421–436.

[DGL13] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition.* Vol. 31. Springer Science & Business Media, 2013.

[JZ13] Rie Johnson and Tong Zhang. “Accelerating stochastic gradient descent using predictive variance reduction”. In: *Advances in neural information processing systems.* 2013, pp. 315–323.

[Vap13] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

[KB14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. arXiv: `1412.6980 [cs.LG]`.

[MV18] Pierre Moulin and Venugopal Veeravalli. *Statistical inference for engineers and data scientists.* Cambridge University Press, 2018.

[SSS18] Sashank, Satyen, and Sanjiv. *On the Convergence of Adam and Beyond.* 2018.

Acknowledgements

Ehsan Saleh, Anay Pattanik created the first draft of the project outline.
Hongye Liu, Ajay Fewell, Chenhui Zhang, Muhammed Imran, Brian Yang, and Jinglin contributed to the newest edition.
This document was compiled and inspired by the work and ideas shown in [RM51; Sac58; NY83; Nem+09; Bot10; JZ13; Bot12; Vap13; DGL13; CZ07; MV18]