Bryan Morales & David Schulz
CS 3851 Algorithms - Final Project
April 27th, 2021

Final Project Proposal

3 Chosen Algorithms:

1. **Adagrad**
   a. **References**
      i. http://d2l.ai/chapter_optimization/adagrad.html#implementation-from-scratch
      ii. https://databricks.com/glossary/adagrad
      iii. https://golden.com/wiki/Adagrad
      iv. http://d2l.ai/chapter_optimization/adadelta.html
      v. http://d2l.ai/chapter_optimization/rmsprop.html

   b. **State the computational problem solved by algorithm. Give an example of a problem instance.**
      i. An optimization problem is solved: best learning rate for each dimension in a Neural Network rather than a general learning rate.
      ii. Scales learning rate with respect to accumulated squared gradient at each iteration in each dimension

   c. **List 2 applications for the algorithm with a brief description (2-3 sentences) of each application.**
      i. Training Language Model
         1. In the natural language, there are various words that have the same meaning, but the amount of times each one is used varies. Typically, people use the simplest word out of all the synonyms. With Adagrad, a model training on natural languages can adjust the learning rate by either decreasing it for features (words) that are used frequently or increasing it for features that are rarely used. Assigning "personalized" weight to each feature prevents the model from converging too fast on frequently used features while not converging for less-frequently used features.

      ii. Unsupervised Learning
         1. An unsupervised learning model is not given labeled/classified data in order to check if what it has learned is correct or incorrect: they are trained to provide information about a dataset. By using Adagrad, the trainer wouldn't have to manually adjust the learning rate for each feature in a large dataset. Additionally, it would also help identify infrequent features in the data that wouldn't be noticed by a model without Adagrad due to it only focusing on the most frequent features.

d. Describe the worst-case and average-case asymptotic time complexity for the algorithm.

```
def adagrad(params, states, hyperparams):
    eps = 1e-6
    for p, s in zip(params, states):
        with torch.no_grad():
            s[:] += torch.square(p.grad)
            p[:] -= hyperparams['lr'] * p.grad / torch.sqrt(s + eps)
        p.grad.data.zero_()
```

i. Worst-Case time complexity: $O\ (n^2)$

ii. Average-Case time complexity: $O\ (n^2)$

e. Determine if there are other well-known algorithms for the same computational problem. Compare and contrast with those algorithms in terms of run-time complexity, complexity of the algorithm itself, and other factors. (e.g., Fibonacci heaps allow merging of heaps in O(1) time but are much more complicated to implement than binary heaps like the one we implemented as a lab.)

i. RMSProp - Shares square of gradient with Adagrad / Keeps track of extra variable (gamma)

```
def rmsprop(params, states, hyperparams):
    gamma, eps = hyperparams['gamma'], 1e-6
    for p, s in zip(params, states):
        s[:] = gamma * s + (1 - gamma) * np.square(p.grad)
        p[:] -= hyperparams['lr'] * p.grad / np.sqrt(s + eps)
```

Worst-Case time complexity: $O\ (n^2)$ || Average-Case time complexity: $O\ (n^2)$

ii. Adadelta - Decreases amount of how adaptive the learning rate is to coordinates. / Doesn't use learning rate

```
def adadelta(params, states, hyperparams):
    rho, eps = hyperparams['rho'], 1e-5
    for p, (s, delta) in zip(params, states):
        with torch.no_grad():
            # In-place updates via [:]
            s[:] = rho * s + (1 - rho) * torch.square(p.grad)
            g = (torch.sqrt(delta + eps) / torch.sqrt(s + eps)) * p.grad
            p[:] -= g
            delta[:] = rho * delta + (1 - rho) * g * g
        p.grad.data.zero_()
```

Worst-Case time complexity: $O\ (n^3)$ || Average-Case time complexity: $O\ (n^3)$

f. Estimate how easy or difficult it would be to implement the algorithm and perform a demonstration. Indicate what you might need, including any potential data sets. If you do need to use a data set, provide a link to a data set that you would use and a description.

i. We expect the difficulty of implementing and performing a demonstration of the algorithm to be medium. We understand how networks work due to our Deep Learning class/labs and we also understand how the algorithm works, but we are also aware how time-consuming and detailed our debugging needs to be for a network.

ii. We can use our Deep Learning datasets to see if we can improve a network using Adagrad. We will train it on the Fashion MNIST and CIFAR-10 datasets.

2. A* (A star)

a. References

i. https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

ii. https://www.geeksforgeeks.org/a-search-algorithm/

iii. https://www.redblobgames.com/pathfinding/a-star/implementation.html

b. State the computational problem solved by algorithm. Give an example of a problem instance.

i. Shortest path to get from point A to point B while a certain part of the search space is not available

c. List 2 applications for the algorithm with a brief description (2-3 sentences) of each application.

i. Finding shortest path/route in map:

1. The A* algorithm can be used in a web-based mapping application, such as Google Maps, where it's necessary to give the user the shortest path to their destination, but that go around "obstacles" (e.g. houses, dead ends, bridges).

ii. Video Games (Speed running):

1. If a player is competing for the fastest speedrun of a video game, the A* algorithm would be advantageous as the player would know which path is the quickest to the end. A more specific example would be doing a speedrun on the good old Super Mario Bros game where the user has to figure out if it's faster to go through the course as intended, or if it's faster for them to go underground through the tunnels instead if it's more linear than outside.

d. Describe the worst-case and average-case asymptotic time complexity for the algorithm.

i. Worst-Case time complexity: $Infinite$

If no destination exists, the algorithm will not stop.

ii. Average-Case time complexity: $O(b^d)$

b = branching factor

d = depth of solution (# of nodes in path)

e. Determine if there are other well-known algorithms for the same computational problem. Compare and contrast with those algorithms in terms of run-time complexity, complexity of the algorithm itself, and other factors. (e.g., Fibonacci heaps allow merging of heaps in O(1) time but are much more complicated to implement than binary heaps like the one we implemented as a lab.)

i. Breadth-First Search

1. Average and worst-case are both O(V+E), where V is the number of vertices (nodes) in the graph and E is the number of edges (connections) in the graph. I believe it's slightly easier to implement than A*, but much less versatile.

ii. Depth-First Search

1. Average-case is also O(V+E), where V is the number of vertices (nodes) in the graph and E is the number of edges (connections) in the graph, but worst-case is O(infinity), since it's possible to get stuck in a loop where the goal node doesn't exist. Again, I believe it's slightly easier to implement than A*, but much less versatile.

f. Estimate how easy or difficult it would be to implement the algorithm and perform a demonstration. Indicate what you might need, including any potential data sets. If you do need to use a data set, provide a link to a data set that you would use and a description.

i. We have implemented A* before in a previous class, and it wasn't very difficult to learn. It should also be very easy to demonstrate. We don't need any data sets for this algorithm.

3. k-Means Clustering

a. References

i. https://realpython.com/k-means-clustering-python/

ii. https://www.geeksforgeeks.org/k-means-clustering-introduction/

b. State the computational problem solved by algorithm. Give an example of a problem instance.

i. We are given a data set of items, with certain features and values for these features. The task is to categorize those items into groups. The algorithm will categorize the items into k groups of similarity. To calculate that similarity, the euclidean distance is used as measurement.

c. List 2 applications for the algorithm with a brief description (2-3 sentences) of each application.

i. Document Clustering
   1. Let's say you have multiple documents and you need to cluster similar documents together. Clustering helps us group these documents such that similar documents are in the same clusters.

ii. Image Segmentation
   1. We can use clustering to perform image segmentation. We try to club similar pixels in the image together. We can apply clustering to create clusters having similar pixels in the same group.

d. Describe the worst-case and average-case asymptotic time complexity for the algorithm.

i. Worst-case: O(tknd)
   1. t = # of iterations
   2. k = # of classes/clusters
   3. n = # of data points to be categorized
   4. d = # of dimensions of each data point

ii. Average-case: O(tknd)
   1. t = # of iterations
   2. k = # of classes/clusters
   3. n = # of data points to be categorized
   4. d = # of dimensions of each data point

e. Determine if there are other well-known algorithms for the same computational problem. Compare and contrast with those algorithms in terms of run-time complexity, complexity of the algorithm itself, and other factors.

i. <u>k-Medians</u>
   1. Average and worst-case are both still O(tknd) because the only difference is what metric is being used to find the centroids. This also means the complexity of implementing the algorithm itself should be no different from implementing k-Means.

ii. k-Medoids
    1. Average and worst-case are both O(tk(n-k)2), where t is the number of iterations, k is the number of classes/clusters, and n is the number of data points to be categorized. This is probably more complex to implement.

f. Estimate how easy or difficult it would be to implement the algorithm and perform a demonstration. Indicate what you might need, including any potential data sets. If you do need to use a data set, provide a link to a data set that you would use and a description.

i. The algorithm is a very basic concept that shouldn't be difficult to implement and demonstrate at all. The first data set that came to mind that we could use is the Iris Data Set.

ii. https://archive.ics.uci.edu/ml/datasets/iris - Iris Data Set, 3 different types of iris plants, 50 samples of each, with 4 features for each sample (sepal length/width, petal length/width)