Homework 2
600.682 Deep Learning
Spring 2022

February 17, 2022

Ting He

Instructions. Please submit the following two items to Gradescope (entry code WYPD4D): 1) your report (LaTeX generated PDF) to `homework2-report`, and 2) a zip file containing your Python Jupyter Notebook (.ipynb) and an exported PDF of the notebook (with all cell outputs) to `homework2-notebook`.

1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computational graph. The function is defined as:

$$f(x_1, x_2, w_1, w_2) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}} + 0.5(w_1^2 + w_2^2)\,.$$

   (a) Please calculate $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}$.

   (b) Start with the following initialization: $w_1 = 0.3, w_2 = -0.5, x_1 = 0.2, x_2 = 0.4$, draw the computational graph. Please use backpropagation as we did in class.
   You can draw the graph on paper and insert a photo into your report.
   The goal is for you to practice working with computational graphs. As a consequence, you must include the intermediate values during the forward and backward pass.

A. given

$$\begin{aligned} f(x_1, x_2, w_1, w_2) &= \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}} + 0.5(w_1^2 + w_2^2) \\ &= [1 + e^{-(w_1 x_1 + w_2 x_2)}]^{-1} + 0.5(w_1^2 + w_2^2) \end{aligned}$$

we have

$$\begin{aligned} \frac{\partial f}{\partial w_1} &= [1 + e^{-(w_1 x_1 + w_2 x_2)}]^{-2} e^{-(w_1 x_1 + w_2 x_2)}(-x_1) + w_1 \\ &= \frac{-x_1 e^{-(w_1 x_1 + w_2 x_2)}}{[1 + e^{-(w_1 x_1 + w_2 x_2)}]^2} + w_1 \\ \frac{\partial f}{\partial w_2} &= \frac{-x_2 e^{-(w_1 x_1 + w_2 x_2)}}{[1 + e^{-(w_1 x_1 + w_2 x_2)}]^2} + w_2 \\ \frac{\partial f}{\partial x_1} &= [1 + e^{-(w_1 x_1 + w_2 x_2)}]^{-2} e^{-(w_1 x_1 + w_2 x_2)}(-w_1) \\ &= \frac{-w_1 e^{-(w_1 x_1 + w_2 x_2)}}{[1 + e^{-(w_1 x_1 + w_2 x_2)}]^2} \\ \frac{\partial f}{\partial x_2} &= \frac{-w_2 e^{-(w_1 x_1 + w_2 x_2)}}{[1 + e^{-(w_1 x_1 + w_2 x_2)}]^2} + w_2 \end{aligned}$$

B.the computational graph can be checked in the attached pic, as well as the intermediate values during forward and backward pass.

2. Doing the computations above is a lot of work, already for such a simple function as in Problem 1. Clearly, it will be desirable to automate the forward and backward propagation process. In contemporary frameworks, such as TensorFlow and PyTorch, backpropagation is handled automatically for all standard operations, which is clearly very convenient. This feature is commonly referred to as "AutoGrad" and becomes possible due to one of the key observations we learned in class: difficult expressions can be broken down into simple subproblems, the analytic gradients of which can be synthesized via chain rule. In this problem, you will implement your own AudoGrad structure.

   (a) Consider the Jupyter Notebook we provide and follow the instructions therein to implement the missing operations and functionality. (To use Jupyter Notebook, we highly recommend uploading the .ipynb file to Google Colaboratory (https://colab.research.google.com/), where you may edit and run your notebook online. If that option is not feasible for you, you can create a local Python 3.6+ environment. The only required external dependencies are `numpy`, `matplotlib`, and `jupyter` which you can install using the package manager of your choosing (e.g. `pip`, `conda`))

   (b) Use the function and output derived in Problem 1 to test whether your AutoGrad structure works as intended.

   Attach a screen shot of your notebook's printed intermediate values for this test case (both during the forward and backward pass) to your report.

   A. the code can be checked at my python notebook.
   B. My autograd function works and you can check at my notebook. The intermediate value can be checked at following attached screenshot.

3. Your goal in this exercise is to implement a linear classifier using the AutoGrad class you implemented in Problem 2, train it on a 2-dimensional toy dataset, and show the results tested on the **same** dataset. Recall that we formulate a linear classifier as $f(x, W, b) = Wx + b$. To quantify the model's performance, you will pass its output through a softmax function and then use Cross Entropy loss to measure agreement with the target output.

   Write a Python program based on your AutoGrad structure that iteratively computes a small step in the direction of negative gradient. The dataset can be downloaded here (https://piazza.com/class_profile/get_resource/kwo1bkotxk1ja/kz5xf09eb9ry). **Please go through the README.txt carefully before you start.** Initialize your optimization with all elements of W being small random numbers and $b = 0$. As shown in the visualization of the data provided in the link above, you can reasonably expect to see your algorithm converge to a solution that linearly separates the given data samples.

   (a) Please plot the following:
      i. Cross Entropy loss with respect to training iterations (loss curve)
      ii. Training accuracy with respect to training iterations (accuracy curve)
      iii. Visualization of the ground truth labels and your model's decision boundaries (decision boundary visualization)

   (b) Report your classification accuracy. Briefly discuss your result. Does it match your expectation?

   A. the loss curve, accuracy curve and decision boundary visualization can be checked at following screenshots.
   B. The final accuracy is 1 at 500th iteration, while the final loss is 0.001. It meets my expectations since when getting more iterations, we have less loss, increased accuracy and they become no huge differences later on. More specifically, After having 79 iterations, our accuracy reached at 1 with out change. And the loss is around 0.002 after around 239 iterations.

4. In this problem, we would like you to compare the performance of single-layer and multi-layer classifiers, again using your AutoGrad structure.

   (a) Please download the dataset for this question here (https://piazza.com/class_profile/get_resource/kwo1bkotxk1ja/kz5xf096cvg1zm). This data is two dimensional. Create another linear model (same as Problem 3) for this dataset and run your classification training.

   Report your classification accuracy. As in Problem 3, please attach plots of the 1) loss curve, 2) accuracy curve and 3) decision boundary visualization. Is this dataset linearly separable?

   (b) Now add one more layer of perceptrons.* Use ReLU as activation functions for the first hidden layer.** Connect outputs with a softmax function. Initialize your optimization procedure with all elements of W being small random numbers and $b = 0$. Use Cross Entropy loss. Train this model on the same two dimensional dataset in (a).

   Report your classification accuracy. Attach plots of the 1) loss curve, 2) accuracy curve and 3) decision boundary visualization. Does it perform better than (a)?

   (c) Briefly discuss your results.

   * If the output of a single layer is $f(x, W_1, b_1)$, then adding another layer basically means that the output is $g(f(x, W_1, b_1), W_2, b_2)$: the second layer uses the first layer's output as its input. Therefore, the output dimension of f should match the expected input dimension of g. In our case, the output dimension from the ReLU activation in the first layer should match the input dimension of the second layer.

   ** The ReLU activation function is a nonlinear function defined as:

$$\text{ReLU}(x) = \max(0, x)$$

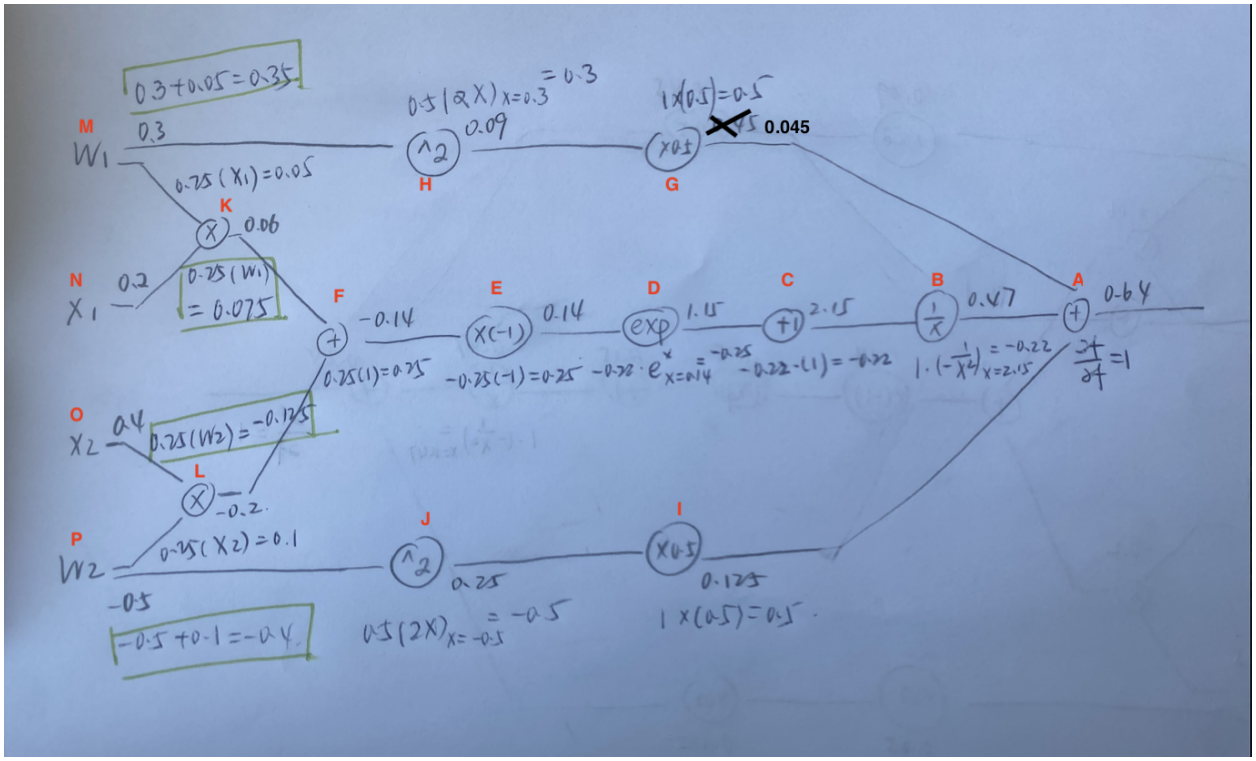
   If we use ReLU as our activation function for the first layer, instead of $f(x, W_1, b_1) = W_1x+b$, we have

$$f(x, W_1, b_1) = \text{ReLU}(W_1x + b) = \max(0, W_1x + b)$$

   The non-linearity of the ReLU function adds non-linearity to the model.

A. the loss curve, accuracy curve and decision boundary visualization can be checked at screenshots. The classification accuracy is 0.85 at 500th iteration. This dataset can't perfectly linear separate since we can see from decision boundary visualization that near the boundaries of four sections, there are some mixed/worry data points which are hard to seperate
B. the classification accuracy is 0.9 at 500th iteration which is better than previous result. Also the loss is much more smaller than the one in (a), 10 vs 40. The other 2 plots are attached in screenshots.
C.Even though by using MLP, we still have some points which were wrongly labeled, but the overall results are better than single linear model. By using MLP, we created a non-linear boundary and reached accuracy at 0.9 which performed better than linear classifier. However, we used a smaller learning rate in MLP (0.001 vs 1 in the previous question). It might cause the problem can't converge in a fixed time of interation.

Problem 1. B. computational graph



Problem 2.B

(1) Final output

```
(Value(data=0.3, grad=0.34975579811673396),
 Value(data=-0.5, grad=-0.400488403766532),
 Value(data=0.2, grad=0.074633697175101),
 Value(data=0.4, grad=-0.124389495291835))
```

(2) intermediate value for both forward and backward pass (name of each node can be checked at pic)

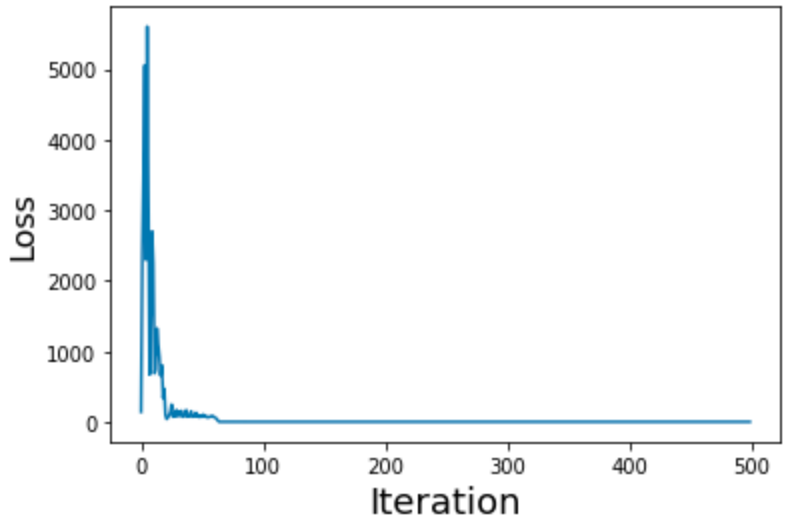
```
Intermediate (gradient) results for backward process:
node A = 1
node B = -0.21627806425811547
node C = -0.21627806425811547
node D = -0.24877899058367
node E = 0.24877899058367
node F = 0.24877899058367
node G = 0.5
node H = 0.3
node N - K = 0.074633697175101
node O - L = -0.124389495291835
node I = 0.5
node J = -0.5
node M - K = 0.049755798116734
node M = node M - K + node H = 0.34975579811673396
node P - L = 0.099511596233468
node P = node P - L + node J = -0.400488403766532
```

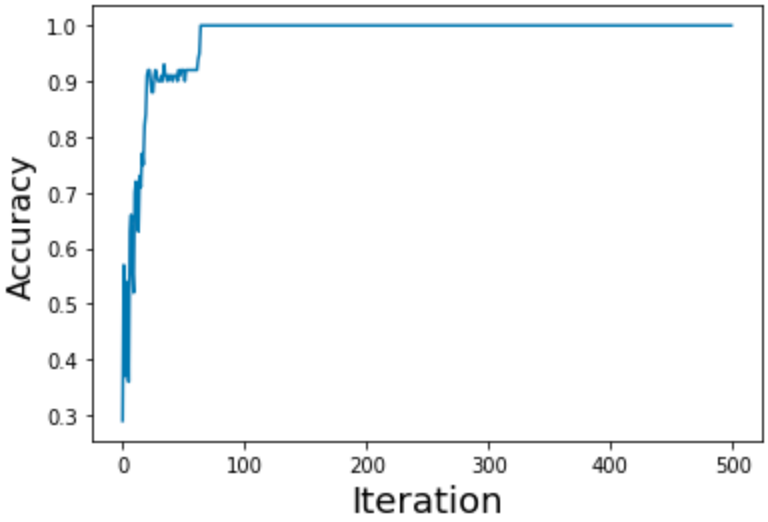
```
Intermediate (value) results for forward process:
node K = 0.06
node L =-0.2
node F =-0.14
node E =0.14
node D =1.1502737988572274
node C =2.1502737988572274
node B =0.4650570548417855
node H =0.09
node G =0.045
node J =0.25
node I =0.125
node A  =0.6350570548417855
```

Problem 3.A

Loss curve, accuracy curve, decision boundary visualization

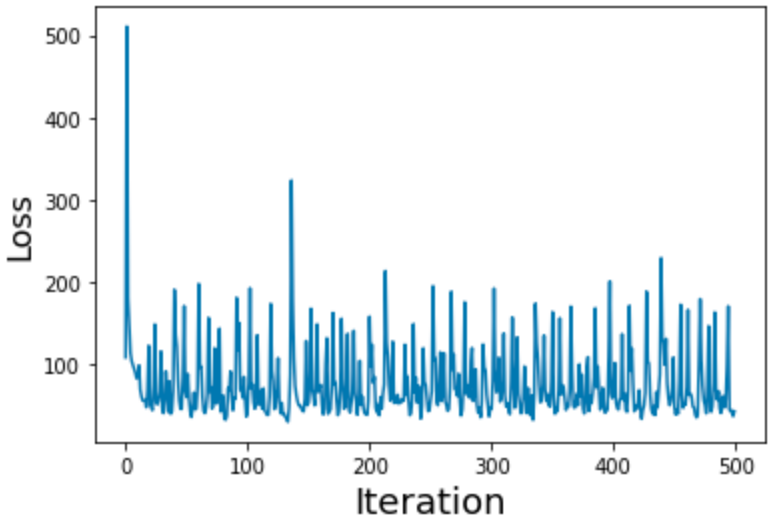
```
(100, 2) (100,) [0 1 2 3]
iteration 19 loss: 468.4891448403702 accuracy: 0.84
iteration 39 loss: 95.9251385050858 accuracy: 0.91
iteration 59 loss: 82.91304813672961 accuracy: 0.92
iteration 79 loss: 0.023319009717379675 accuracy: 1.0
iteration 99 loss: 0.011831923295343074 accuracy: 1.0
iteration 119 loss: 0.007974324818637017 accuracy: 1.0
iteration 139 loss: 0.006026546371341371 accuracy: 1.0
iteration 159 loss: 0.0048505283025551535 accuracy: 1.0
iteration 179 loss: 0.004063507200662139 accuracy: 1.0
iteration 199 loss: 0.0035001204494061024 accuracy: 1.0
iteration 219 loss: 0.0030771521512492914 accuracy: 1.0
iteration 239 loss: 0.0027481315172612633 accuracy: 1.0
iteration 259 loss: 0.002485058402228892 accuracy: 1.0
iteration 279 loss: 0.0022700523334184225 accuracy: 1.0
iteration 299 loss: 0.0020911580664840492 accuracy: 1.0
iteration 319 loss: 0.001940078272934344 accuracy: 1.0
iteration 339 loss: 0.0018108735985545603 accuracy: 1.0
iteration 359 loss: 0.0016991804290073251 accuracy: 1.0
iteration 379 loss: 0.0016017207458284266 accuracy: 1.0
iteration 399 loss: 0.001515984259111083 accuracy: 1.0
iteration 419 loss: 0.001440016071214109 accuracy: 1.0
iteration 439 loss: 0.0013722711330895435 accuracy: 1.0
iteration 459 loss: 0.0013115121984235399 accuracy: 1.0
iteration 479 loss: 0.001256736827389874 accuracy: 1.0
iteration 499 loss: 0.0012071242275551795 accuracy: 1.0
```

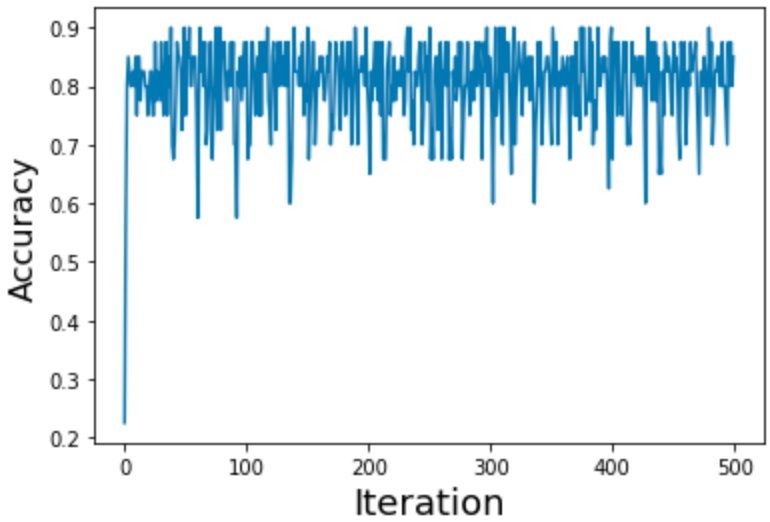

Loss
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
Iteration


Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
100
200
300
400
500
Iteration

Problem 4.A. plots for linear model

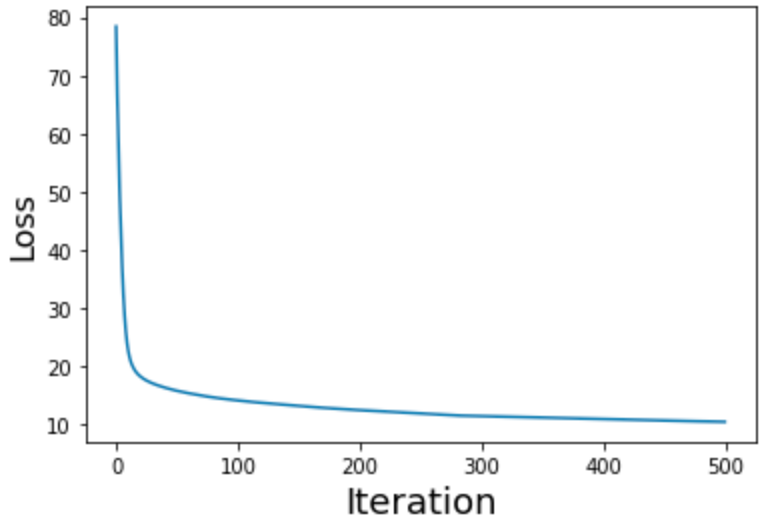
```
iteration 19 loss: 122.2739843385347 accuracy: 0.75
iteration 39 loss: 67.03160609505399 accuracy: 0.7
iteration 59 loss: 72.35680391503996 accuracy: 0.75
iteration 79 loss: 46.41444423244899 accuracy: 0.725
iteration 99 loss: 36.04412487526879 accuracy: 0.8
iteration 119 loss: 173.24390394852568 accuracy: 0.75
iteration 139 loss: 73.2726044281514 accuracy: 0.9
iteration 159 loss: 71.01091392253859 accuracy: 0.8
iteration 179 loss: 45.52960797645645 accuracy: 0.875
iteration 199 loss: 55.03882855015052 accuracy: 0.8
iteration 219 loss: 127.22908422272916 accuracy: 0.8
iteration 239 loss: 53.232420619917086 accuracy: 0.8
iteration 259 loss: 54.98486271561006 accuracy: 0.875
iteration 279 loss: 63.5025771684901 accuracy: 0.8
iteration 299 loss: 46.678467344709354 accuracy: 0.85
iteration 319 loss: 46.514839485279666 accuracy: 0.9
iteration 339 loss: 74.51488471647393 accuracy: 0.875
iteration 359 loss: 72.13171937399537 accuracy: 0.85
iteration 379 loss: 108.41411401784686 accuracy: 0.775
iteration 399 loss: 58.36729772540227 accuracy: 0.9
iteration 419 loss: 45.956406304063435 accuracy: 0.825
iteration 439 loss: 229.07802275829633 accuracy: 0.675
iteration 459 loss: 50.801502353318945 accuracy: 0.875
iteration 479 loss: 46.31692033871879 accuracy: 0.9
iteration 499 loss: 42.427461284854964 accuracy: 0.85
```

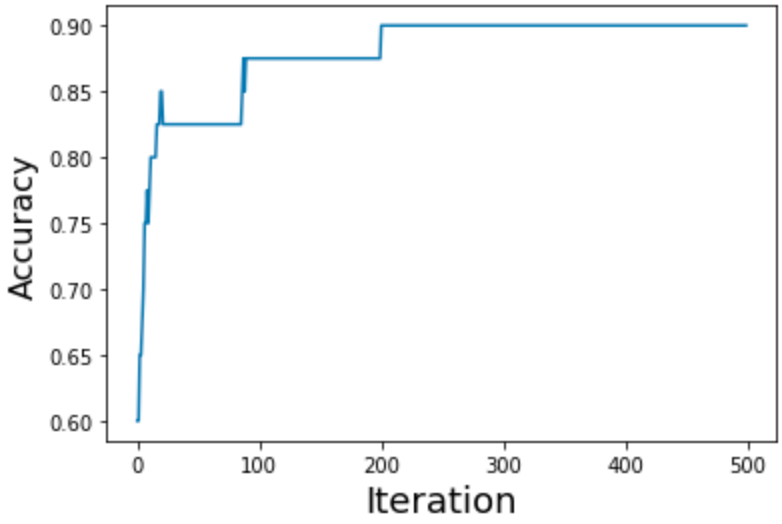

Loss
500
400
300
200
100
0
100
200
300
400
500
Iteration


Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500
Iteration

problem 4.B plots for MLP model

```
iteration 19 loss: 18.285040925042193 accuracy: 0.85
iteration 39 loss: 16.263418420234547 accuracy: 0.825
iteration 59 loss: 15.251651353047853 accuracy: 0.825
iteration 79 loss: 14.532364247307214 accuracy: 0.825
iteration 99 loss: 14.009800609918212 accuracy: 0.875
iteration 119 loss: 13.587374215121958 accuracy: 0.875
iteration 139 loss: 13.246290541504704 accuracy: 0.875
iteration 159 loss: 12.89174239163063 accuracy: 0.875
iteration 179 loss: 12.59001938947722 accuracy: 0.875
iteration 199 loss: 12.33086181252935 accuracy: 0.875
iteration 219 loss: 12.0918702391982 accuracy: 0.9
iteration 239 loss: 11.879124028170414 accuracy: 0.9
iteration 259 loss: 11.632529497045816 accuracy: 0.9
iteration 279 loss: 11.400754344133277 accuracy: 0.9
iteration 299 loss: 11.275988670158899 accuracy: 0.9
iteration 319 loss: 11.16004805123973 accuracy: 0.9
iteration 339 loss: 11.048377180955027 accuracy: 0.9
iteration 359 loss: 10.94463078979013 accuracy: 0.9
iteration 379 loss: 10.84288821732265 accuracy: 0.9
iteration 399 loss: 10.747286598675762 accuracy: 0.9
iteration 419 loss: 10.655595943159502 accuracy: 0.9
iteration 439 loss: 10.567824396274254 accuracy: 0.9
iteration 459 loss: 10.476629682956794 accuracy: 0.9
iteration 479 loss: 10.388995226654654 accuracy: 0.9
iteration 499 loss: 10.308441473459842 accuracy: 0.9
```


Loss
Iteration
80
70
60
50
40
30
20
10
0
100
200
300
400
500


Accuracy
Iteration
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
100
200
300
400
500