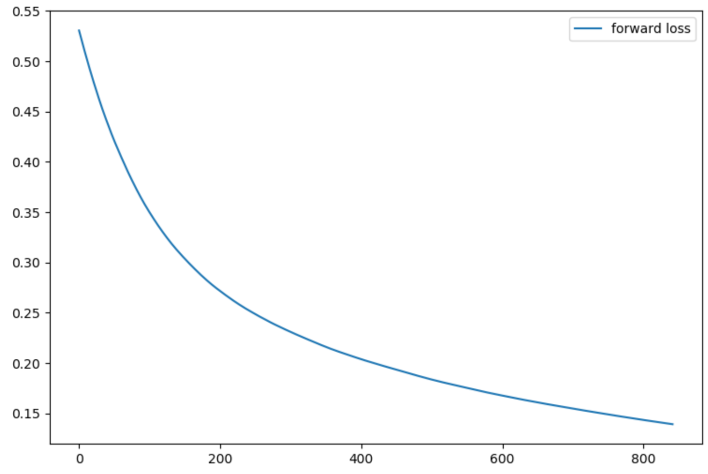
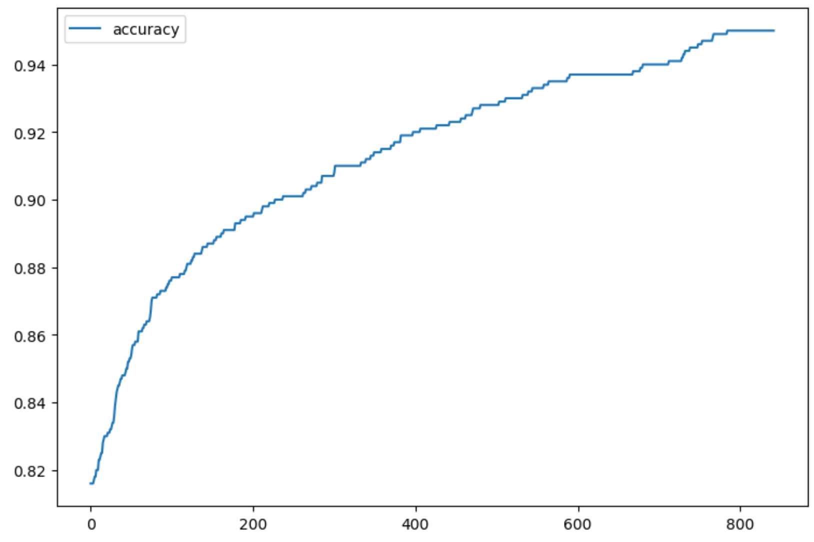
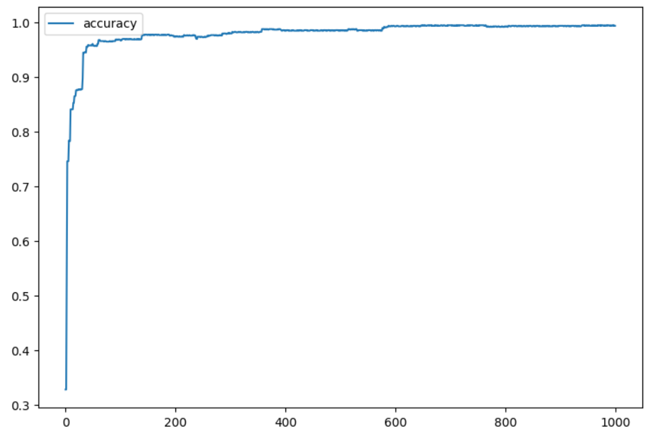
1. (1.25 points) Visualizing Gradient Descent

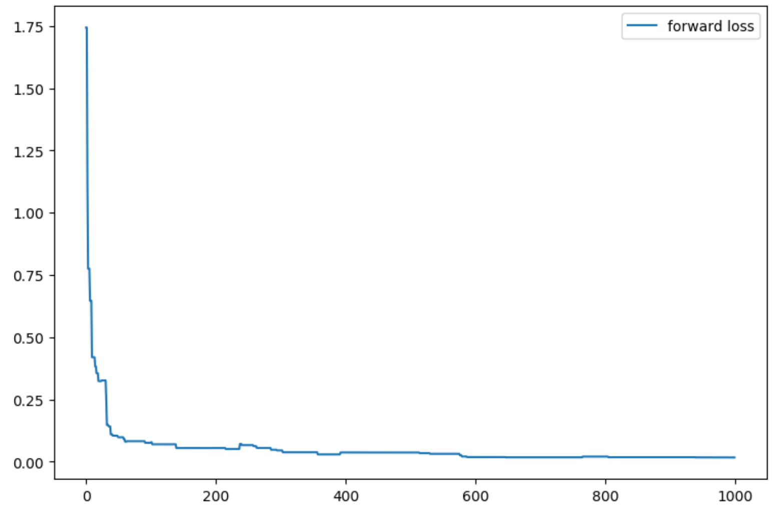
a. (0.5 points) Setup an experiment to classify the `mnist-binary` dataset (see `load_data.py` for more on that dataset) using the gradient descent learner. Use `fraction=1` when calling `load_data`. For your loss function, use `hinge` loss and set the learning rate (i.e., lambda) to `1e-4`. Keep all other gradient descent parameters as their defaults. After each training iteration through the dataset (using batch gradient descent on just the training data), compute both the loss and accuracy of the model on the full training dataset. Terminate after convergence or after 1000 iterations (i.e., `max_iter = 1000`). Construct and include two (labeled) plots of the loss and accuracy at each iteration. Note that when `batch_size` is not set, it trains on all available training data for each step.





b. (0.5 points) Repeat the previous experiment, but this time using stochastic gradient descent (i.e., we accomplish this by changing `batch_size`). Compute and plot the loss and accuracy over the entire dataset after each *epoch*. One epoch is one iteration through the entire dataset. Note that you will need to increase the value of `max_iter` to do this. For example, if we have 500 examples, setting `max_iter=500` and `batch_size=1` would result in one epoch of stochastic gradient descent. Terminate after convergence or after 1000 epochs. Construct and include the two (labeled) plots of the loss and accuracy at each epoch.





c. (0.25 points) Compare the plots generated from batch gradient descent and stochastic gradient descent. Which one seems to be faster to train? Which one converges to a lower average loss on the training data? Are there differences in the behavior of the loss and accuracy throughout training?

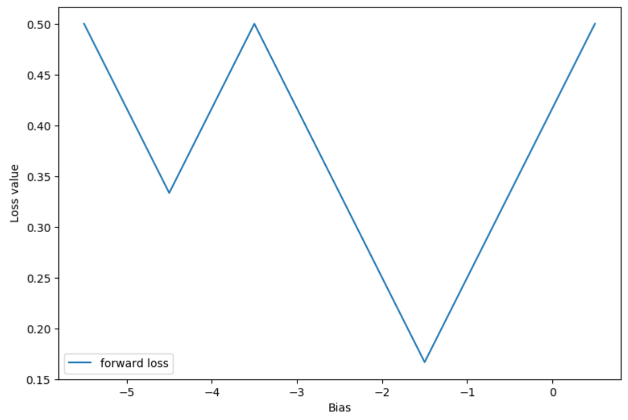
Training stochastic gradient descent is faster than training gradient descent with all available examples.

0.1392611132482182 Vs 0.01666254882535325. stochastic gradient descent has smaller loss value.

Yes, the difference is that the tendency of loss is decrease, the tendency of accuracy is increase.

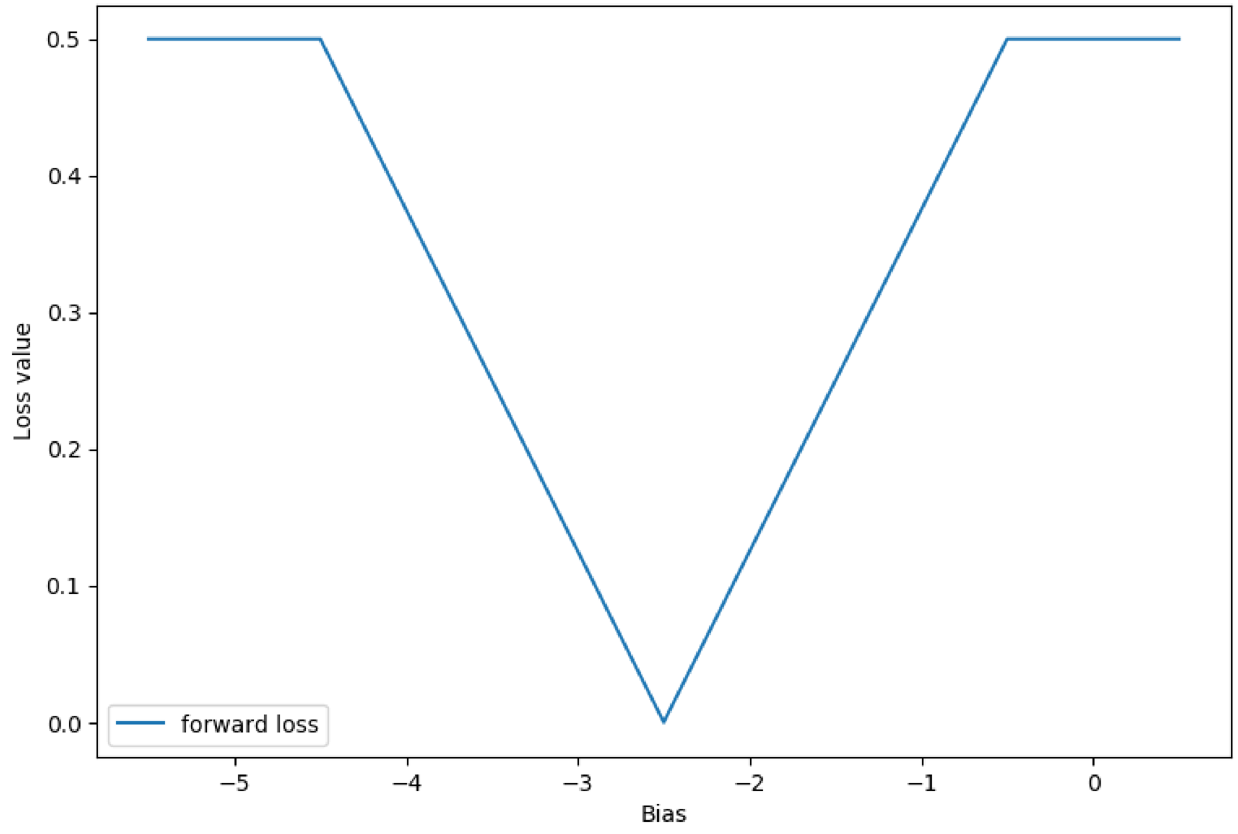
2. (1.25 points) Loss Landscapes and the Effects of Batching

a. (0.5 points) Here we will setup an experiment that will allow us to visualize the loss landscape of the `synthetic` dataset (see `load_data.py` for more on that dataset). Here, we define the "loss landscape" as the loss at every value value of the model parameters with respect to our input data. For more information about loss landscapes see the lectures slides for gradient descent. For this experiment, we will be looking at the loss landscape of a model with only bias terms, meaning all other parameters have values of exactly 1. Note that this model does not do any "learning" because we are setting the weights explicitly (this means you will not be using your `fit()` function). In your experiment, first load the `synthetic`dataset (with `fraction=1`). Using only the bias term, determine the 0-1 loss over the entire dataset by explicitly setting your bias values to 0.5, -0.5, -1.5, -2.5, -3.5, -4.5, and -5.5. Construct a plot of the loss landscape by plotting the bias on the X axis and the loss on the Y axis. Include your (labeled) plot and describe the minima of the loss landscape.



We can find two local minimum at -4.5 and -1.5 and one global minimum at -1.5.

b. (0.5 points) Now we will examine how the loss landscape changes when you select data differently. By analyzing the `synthetic` dataset and the loss landscape plot you have generated, select a set of 4 points from the dataset that, if you optimize considering only that set, cause the global minimum of the loss landscape to shift. Repeat your experiment from part (a) on this set of 4 points, plot the resulting loss landscape and compare it to the loss landscape from part (a).



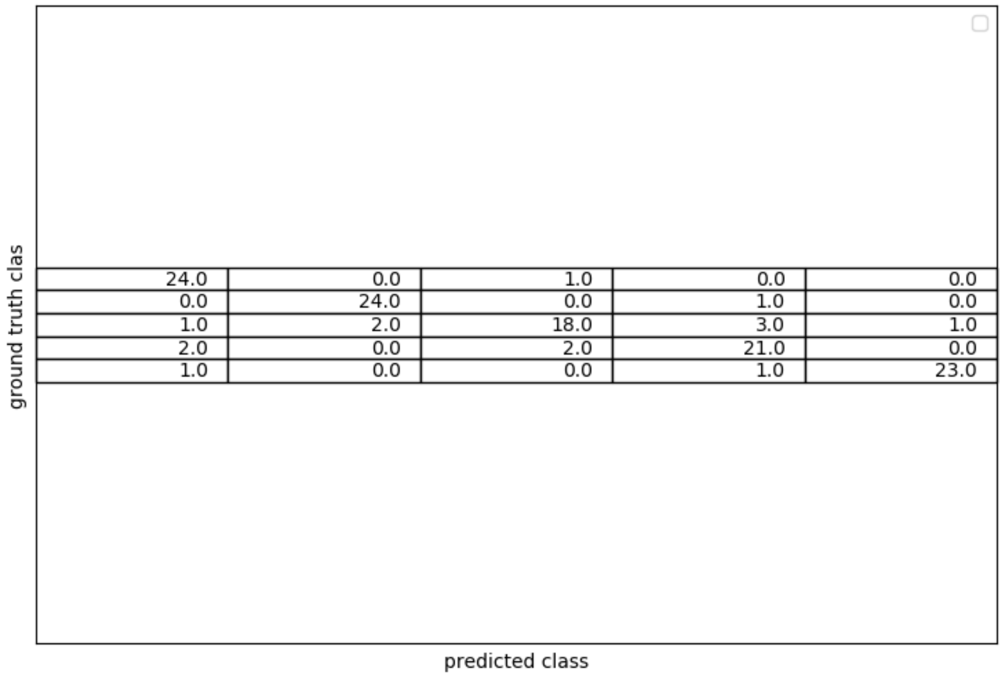
Choosing the last 4 data as input, then I can get global minimum at -2.5 which in part a is at -1.5.

c. (0.25 points) Based on your answers from part (a) and (b), explain what effect batching can have on the loss landscape and the convergence of gradient descent.

Batching will affect the global minimum of the loss landscape. Therefore, if the global minimum changes, it might be more difficult for gradient descent to converge.

3. (1 point) Multiclass Classification with Gradient Descent

a. (0.5 points) Setup an experiment to classify the `mnist-multiclass` dataset using the One-vs-All gradient descent learner (see `load_data.py` for more on that dataset). Use `fraction=0.75` when calling `load_data`. Use `squared` loss and `l1`regularization with the default learning rate and regularization parameters. Train your learner on the training partition (report what values you used for `batch_size` and `max_iter`), and run prediction on the testing partition. Generate the confusion matrix of the testing partition (using the `confusion_matrix` function in `metrics.py`) and include the confusion matrix as a table. Label the rows with the ground truth class and the columns with the predicted class. Describe the most common misclassifications made by your system. Which numbers appear to be most difficult to distinguish from one another? Which appear to be easiest to distinguish?


ground truth clas
predicted class

24.0	0.0	1.0	0.0	0.0
0.0	24.0	0.0	1.0	0.0
1.0	2.0	18.0	3.0	1.0
2.0	0.0	2.0	21.0	0.0
1.0	0.0	0.0	1.0	23.0

I use all available example, so the batch size is 350. Max_iter is 1000.
Class 2 is the most difficult to be distinguished from one another.
Class 0 and 1 are the easiest to be distinguished from one another.

b. (0.25 points) In this assignment, we used One-vs-All (OVA) with linear decision boundaries to handle multiclass classification. For a dataset with *c* classes and *d* features, how many linear binary classifiers are needed in OVA classification? What is the space complexity of the OVA model (i.e., the set of learned decision boundaries)?

We will need c linear binary classifiers to classify c classes.

The space complexity is O(c * d + c), additional c stands for bias.

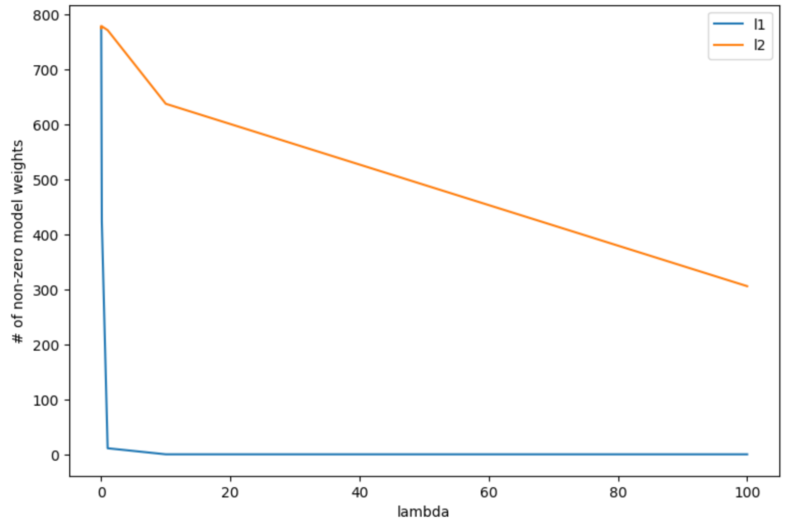
c. (0.25 points) An alternative to OVA classification is One-vs-One (OVO) classification, in which a binary classifier is trained to discriminate between each pair of classes, and the final prediction is decided via majority vote. For the same dataset considered in part (b), how many binary classifiers are need in OVO classification? What is the space complexity of our OVO model?

We will need c * (c-1) / 2 binary classifiers.

We will need O(c * (c-1) / 2 * d + c * (c-1) / 2)

4. (1.5 points) Regularization and Feature Selection

a. (0.5 points) Here we will explore the use of regularization as a means of feature selection. Setup an experiment using the `mnist-binary` dataset. Use `fraction=1` when calling `load_data`. Run gradient descent on the `mnist-binary` training dataset using squared loss, using both `l1` and `l2` regularization. Set the step size, or `learning_rate`, to `1e-5` and the maximum number of iterations to `2000`. Report your values for how many digits from each class you used, what your batch size was, and how many iterations you used. For each regularizer, run the algorithm once for each of the following values for lambda: [1e-3, 1e-2, 1e-1, 1, 10, 100]. Plot the number of non-zero model weights from the learned model for each value of lambda. (Here we define non-zero values as values whose absolute value is greater than some `epsilon`, where `epsilon = 0.001`.) Plot both regularizers on one plot. Include your (labeled) plot and describe the trend in non-zero parameters for each regularizer.
Barch_size = 50, max_iteration = 2000, iteration = 2000* (1000/50)=4000
Class 0, label = -1: 500
Class 1, label = 1: 500



b. (0.5 points) Compared to the L2 regularizer, what property of the L1 regularizer allows it to promote sparsity in the model parameters? Describe a situation in which this sparsity is useful.

the gradient of L1-regularization moves model parameters towards 0 at a constant rate.

L1-regularization encourages the model parameters to be sparse

sparsity can avoid over-fitting

Only features with non-zero coefficients contribute to the model's prediction

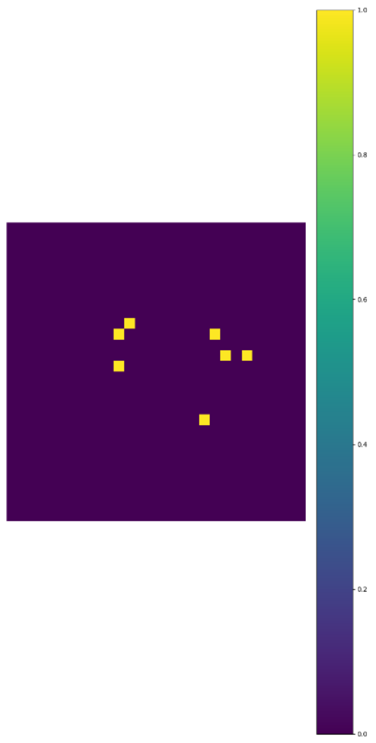
This is a form of feature selection

when you are solving a large vector x with less training data. The solutions to x could be a lot.

Ax = b

By L1 regularization, you essentially make the vector x smaller (sparse), as most of its components are useless (zeros), and at the same time, the remaining non-zero components are very "useful".

c. (0.5 points) Using L1 regularization with `reg_param=1`, make a 2D heatmap that shows which weights in a trained model have non-zero values (as defined in problem 4a). Your heatmap should only have two colors--one color for non-zero values and another color for zero values. Make sure to clearly label which color corresponds to zero and non-zero values. This heatmap should have the same shape as your input images. Is there any discernible pattern as to which values are zero?



Yellow is non-zero. Purple is zero.

My input images are range from 0 to 1.

The non-zero weight shows the shape of 0 and some 1.

So the pattern match to my input data.