520.638 Deep Learning
Homework 2 - MNIST Handwritten Digits

Nicholas Hinke

March 15, 2022

1 Part 1: Written Questions

1. *What kind of decision boundary is learned by perceptrons?*

   Using gradient descent to minimize an appropriate loss function, a perceptron will find a set of weights and biases that define a linear hyperplane that separates–or attempts to separate–two classes. Thus, the learned decision boundary is linear. In fact, if two classes are indeed linearly separable, a perceptron is *guaranteed* to find a linear hyperplane that separates them.

2. *How can we use perceptrons to do C-class classification?*

   One method to use perceptrons for C-class classification involves constructing and training C different "sub-classifier" perceptrons, where the k-th perceptron ($k = \{1, ..., C\}$) performs binary classification to identify whether or not a given input sample is from class k. The C different perceptrons can then be combined into a single model, whose output is equivalent to the class of the most confident sub-classifier. Alternatively, a "Multi-Layer Perceptron" (*i.e.* a feed-forward neural network) can be used with a final softmax layer containing C neurons.

3. *Briefly explain stochastic gradient descent.*

   In classical gradient descent (GD), the weight parameters are updated at each training iteration using the sum of the loss function gradients over **all** of the available training samples. Since this can be very slow for large datasets, stochastic gradient descent (SGD) instead only incorporates the loss function gradient of a **single** training sample which is randomly chosen from the training dataset at each iteration. While much faster than classical GD, SGD typically leads to error curves that are much noisier as training progresses.

4. *Briefly explain mini-batch gradient descent.*

   Mini-batch GD is an attempt to blend the best of both worlds between classical GD and SGD. In mini-batch GD, the weight parameters are updated at each training iteration using the sum of the loss function gradients over a "batch" of some of the available training samples. Thus, mini-batch GD is still substantially faster than classical GD, but incorporates much more of the available information at each iteration

than SGD. It should also be noted that the batch size in mini-batch GD can be considered as another tunable hyperparameter.

5. *Let $\sigma(a)$ be the sigmoid function defined as $\sigma(a) = \frac{1}{1+e^{-a}}$. Show that $\sigma'(a) = \sigma(a)\left[1 - \sigma(a)\right]$.*

$$\sigma'(a) = \frac{d}{da}(\sigma(a)) \tag{1}$$
$$\sigma'(a) = \frac{d}{da}(\frac{1}{1+e^{-a}}) \tag{2}$$
$$\sigma'(a) = \frac{d}{da}((1+e^{-a})^{-1}) \tag{3}$$
$$\sigma'(a) = (-(1+e^{-a})^{-2})(-e^{-a}) \tag{4}$$
$$\sigma'(a) = \frac{e^{-a}}{(1+e^{-a})^2} \tag{5}$$
$$\sigma'(a) = \frac{1}{1+e^{-a}}\left[\frac{e^{-a}}{1+e^{-a}}\right] \tag{6}$$
$$\sigma'(a) = \frac{1}{1+e^{-a}}\left[\frac{1+e^{-a}-1}{1+e^{-a}}\right] \tag{7}$$
$$\sigma'(a) = \frac{1}{1+e^{-a}}\left[\frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}}\right] \tag{8}$$
$$\sigma'(a) = \frac{1}{1+e^{-a}}\left[1 - \frac{1}{1+e^{-a}}\right] \tag{9}$$
$$\sigma'(a) = \sigma(a)\left[1 - \sigma(a)\right] \tag{10}$$

6. *What is an "epoch"?*

   An "epoch" refers to a complete pass through **all** of the available training data during training. In other words, for a MLP this corresponds to one forward pass and one backward pass (backpropagation) for every sample within the training dataset (not including the validation dataset).

7. *If the dataset size is 2000 samples and the batch size is 50, then an epoch will consists of how many iterations?*

   Since the entire dataset has 2000 samples and each batch contains 50 samples, it will take 40 iterations to cover all of the available data (since $50 * 40 = 2000$). Thus, the epoch consists of 40 iterations.

8. *Briefly describe dropout.*

   "Dropout" is a regularization technique in which some randomly-selected neurons (and their connections) are dropped from a neural network at every iteration during training (creating a "thinned network" at each training step). This technique prevents complex co-adaptations from forming that allow nearby neurons to account for one anothers' mistakes, thus forcing each neuron to learn more robust parameters and helping to reduce model overfitting.

9. *What is early stopping?*

   "Early stopping" is a type of cross-validation strategy (since it relies on a validation set) that also helps to combat model overfitting to the training dataset (*i.e.* regularization). The idea behind early stopping is to monitor the validation error during training, and to terminate the training process once the validation error begins to increase. The motivation behind this idea is that increasing validation or test error is generally a sign of model overfitting to the training set.

10. *Why is regularization used when training a neural network?*

    Regularization is extremely important when training a neural network, as it helps to prevent the network from overfitting to the training dataset. Without regularization techniques, a sufficiently complex model will overfit to the training data, and consequently perform very poorly when introduced to any previously unseen data. Indirectly, regularization also helps with the issues of vanishing or exploding gradients when performing the backpropagation algorithm during training of a neural network (since the magnitudes of the weight parameters will generally be smaller).

11. *What are different ways in which we can reduce overfitting in a neural network?*

    There are many different regularization techniques that can be used to reduce overfitting in a neural network, several of which have been briefly described above. These include early stopping and dropout, as well as batch normalization which will be discussed below. Additionally, data augmentation techniques can be employed to both increase the size of the dataset and introduce more variations among samples

of the same class. Data augmentation techniques often lead to more robust model performance (*e.g.* robust to noise, scale, occlusions, rotations, translations, *etc.*), and can even be done using additional generative models such as GANs (generative adversarial networks) or VAEs (variational autoencoders).

12. *Why do we use batch normalization?*

    "Batch normalization" is yet another regularization technique which attempts to combat bias that is introduced when training very deep networks. The motivation behind this technique can be seen by considering how *un*-meaningful weight updates to the first layer of a deep neural network will be when they've been affected by all of the gradients from the **many** layers after them. By transforming the inputs to a given layer, batch normalization helps to combat this problem, and also helps to prevent errors from blowing up without bound during backpropagation. This, in turn, helps to overcome the problem of vanishing or exploding gradients, and even allows for the utilization of higher learning rates during training.

2 Part 2: Perceptrons

For this section, perceptrons were used as binary classifiers on the MNIST dataset of handwritten digits ({0,...,9}) which is available at `http://yann.lecun.com/exdb/mnist/`. In the interest of reducing computational time, only the first 20,000 training samples and the first 2,000 test samples were used in this exercise. Within the datasets, each 28x28 image was first reshaped to a 1x785 vector (*i.e.* $x^{(i)} \in \mathbb{R}^{785}$), which was comprised of the 784 pixel intensities preceded by a '1' corresponding to the bias term. Ten perceptrons were then trained to perform binary classification using the gradient descent algorithm, where the output of the k-th perceptron was as follows:

$$y_k^{(i)} = \begin{cases} 1, & \text{predicted class}\big(x^{(i)}\big) = k \\ 0, & \text{predicted class}\big(x^{(i)}\big) \neq k \end{cases} \tag{11}$$

These ten perceptrons can then be combined into a single "10-way" model, where the output of the combined model for a given input $x^{(i)}$ is equivalent to the corresponding class of the most confident sub-classifier among the ten binary classification perceptrons. This can be defined as

$$y^{(i)} = \arg\max_k P\big(y_k^{(i)} = 1 \big| x^{(i)}\big) \tag{12}$$

where $P\big(y_k^{(i)} = 1 \big| x^{(i)}\big)$ represents the probability (or confidence) of the k-th perceptron that the class of the input sample $x^{(i)}$ is k. It should also be noted that throughout this section, all code used to complete these problems was written in Python (and is publicly available at `https://github.com/nhinke/deeplearning-repo/tree/master/Homework/hw2`) with the use of the following libraries:

- **gzip**
- **numpy**
- **matplotlib.pyplot**
- **sklearn.linear_model.Perceptron**

1. Before evaluating the performance of the combined 10-way model on the test set, each of the sub-classifier perceptrons was first trained and

tested individually. To test the accuracy of the k-th sub-classifier, the test labels y_t were modified as follows:

$$\left(y_t^{(i)}\right)_k = \begin{cases} 1, & y_t^{(i)} = k \\ 0, & y_t^{(i)} \neq k \end{cases} \tag{13}$$

The performance of each sub-classifier was assessed using the 2,000 unseen test samples, and the results were both plotted and printed to the active terminal as shown below:

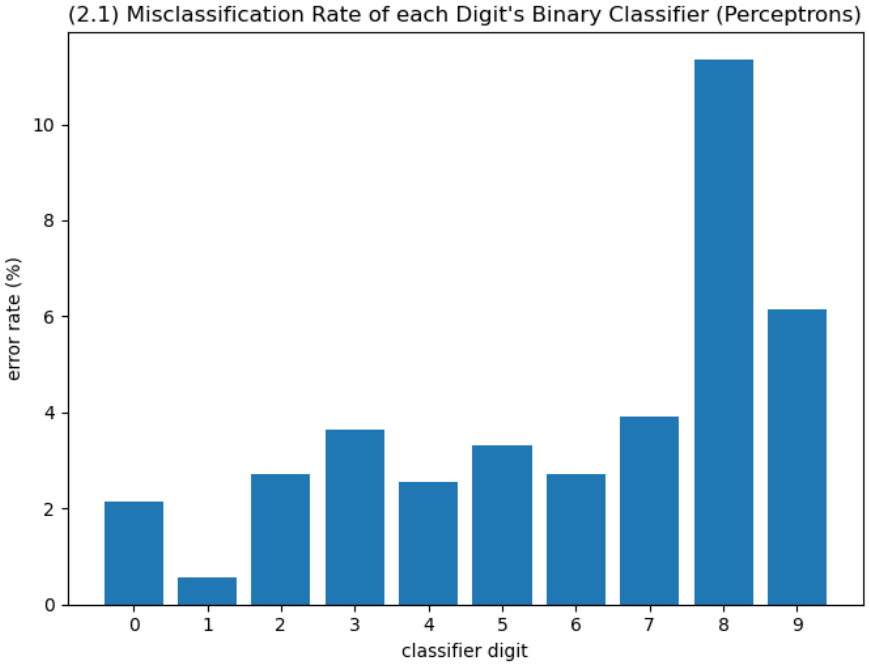

Figure 1: Problem (2.1) Sub-classifier Misclassification Rates

```
(2.1) Misclassification Rate (%) of each Binary Classifier (Perceptrons):
Digit 0:  2.15
Digit 1:  0.55
Digit 2:  2.70
Digit 3:  3.65
Digit 4:  2.55
Digit 5:  3.30
Digit 6:  2.70
Digit 7:  3.90
Digit 8:  11.35
Digit 9:  6.15
```

Figure 2: Problem (2.1) Sub-classifier Misclassification Rates

As can be seen from the results provided above, each of the individual sub-classifiers performed quite well on the unseen test samples, with only one classifier exceeding an error rate of 10%. The next section will discuss the performance of the combined 10-way model.

2. After training and testing each of the sub-classifiers as outlined above, the combined 10-way model was then evaluated in the previously described manner. To briefly restate, the output of the combined model for a given test sample is equivalent to the class of the most confident sub-classifier. The performance of the combined model was assessed using the 2,000 unseen test samples, and the results were printed to the active terminal as shown below:

```
(2.2) 10-Way Model Performance on Test Set (Perceptrons):
Erroneous classifications:   316
Successful classifications:  1684
Misclassification rate (%):  15.8
```

Figure 3: Problem (2.2) Combined Model Misclassification Rates

As can be seen from the results above, while the combined model is still successful roughly 85% of the time, it does not quite achieve the performance of any of the individual sub-classifiers. This is mainly due to the much higher misclassification rates among the small percentage of test samples where *none* of the sub-classifiers output a positive class–and to a lesser degree, the test samples where *multiple* sub-classifiers output a positive class. Note that for these instances, the most confident sub-classifier is determined by finding the maximum signed distance of the test sample from each of the ten decision boundaries.

3 Part 3: Logistic Regression

After constructing and evaluating the ten binary classification perceptrons and the combined 10-way model as described above in part 2, the same procedure was then repeated using logistic regression for the sub-classifiers. In other words, rather than using ten perceptrons to perform binary classification for each digit within the combined model, ten logistic regression models were instead used for the same task. It should also be noted that throughout this section, all code used to complete these problems was written in Python (and is publicly available at `https://github.com/nhinke/deeplearning-repo/tree/master/Homework/hw2`) with the use of the following libraries:

- **gzip**
- **numpy**
- **matplotlib.pyplot**
- **sklearn.linear_model.LogisticRegression**

Before continuing on to the discussion of model performance, it should be noted that logistic regression is a binary classification method which can be modeled as using a single neuron reading in an input vector $x \in \mathbb{R}^d$ and parameterized by weight vector $w \in \mathbb{R}^d$, where the neuron outputs the probability of the class being $y = 1$ given x as follows:

$$P(y = 1|x) = g_w(x) = \frac{1}{1 + \exp(-w^T x)} = \sigma(w^T x) \tag{14}$$

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - g_w(x) \tag{15}$$

Given $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, the Cross Entropy Loss function is defined as

$$J(w) = -\sum_{i=1}^{N} \left(y^{(i)} \log(g_w(x^{(i)})) + (1 - y^{(i)}) \log(1 - g_w(x^{(i)})) \right) \tag{16}$$

where N denotes the total number of training samples. This loss function can then be minimized using the technique of gradient descent.

1. The gradient of the Cross Entropy Loss function given above was first

computed with respect to the parameter w as follows:

$$\frac{\partial J(w)}{\partial w_j} = -\left[\frac{\partial}{\partial w_j}\Big(\sum_{i=1}^{N} y^{(i)} \log(g_w(x^{(i)}))\Big) + \frac{\partial}{\partial w_j}\Big(\sum_{i=1}^{N}(1-y^{(i)}) \log(1-g_w(x^{(i)}))\Big)\right] \tag{17}$$

$$\frac{\partial J(w)}{\partial w_j} = -\left[\sum_{i=1}^{N} \frac{y^{(i)}}{g_w(x^{(i)})} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big) + \sum_{i=1}^{N} \frac{1-y^{(i)}}{1-g_w(x^{(i)})} \frac{\partial}{\partial w_j}\Big(1-g_w(x^{(i)})\Big)\right] \tag{18}$$

$$\frac{\partial J(w)}{\partial w_j} = -\left[\sum_{i=1}^{N} \frac{y^{(i)}}{g_w(x^{(i)})} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big) - \sum_{i=1}^{N} \frac{1-y^{(i)}}{1-g_w(x^{(i)})} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big)\right] \tag{19}$$

$$\frac{\partial J(w)}{\partial w_j} = -\left[\sum_{i=1}^{N} \frac{y^{(i)}(1-g_w(x^{(i)})) - g_w(x^{(i)})(1-y^{(i)})}{g_w(x^{(i)})(1-g_w(x^{(i)}))} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big)\right] \tag{20}$$

$$\frac{\partial J(w)}{\partial w_j} = -\left[\sum_{i=1}^{N} \frac{y^{(i)} - g_w(x^{(i)})}{g_w(x^{(i)})(1-g_w(x^{(i)}))} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big)\right] \tag{21}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{g_w(x^{(i)}) - y^{(i)}}{g_w(x^{(i)})(1-g_w(x^{(i)}))} \frac{\partial}{\partial w_j}\Big(g_w(x^{(i)})\Big) \tag{22}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\sigma(w^T x^{(i)}) - y^{(i)}}{\sigma(w^T x^{(i)})(1-\sigma(w^T x^{(i)}))} \frac{\partial}{\partial w_j}\Big(\sigma(w^T x^{(i)})\Big) \tag{23}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{(\sigma(w^T x^{(i)}) - y^{(i)})\left[\sigma(w^T x^{(i)})(1-\sigma(w^T x^{(i)}))\right]}{\sigma(w^T x^{(i)})(1-\sigma(w^T x^{(i)}))} \frac{\partial}{\partial w_j}\Big(w^T x^{(i)}\Big) \tag{24}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} \big(\sigma(w^T x^{(i)}) - y^{(i)}\big) \frac{\partial}{\partial w_j}\Big(w^T x^{(i)}\Big) \tag{25}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} \big(\sigma(w^T x^{(i)}) - y^{(i)}\big) x_j^{(i)} \tag{26}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} x_j^{(i)} \big(\sigma(w^T x^{(i)}) - y^{(i)}\big) \tag{27}$$

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{N} x_j^{(i)} \big(g_w(x^{(i)}) - y^{(i)}\big) \tag{28}$$

2. In the same manner as in part 2, each of the sub-classifier logistic regression models was first trained and tested individually before evaluating the performance of the combined 10-way model. To test the accuracy of the k-th sub-classifier, the test labels y_t were modified as follows:

$$\left(y_t^{(i)}\right)_k = \begin{cases} 1, & y_t^{(i)} = k \\ 0, & y_t^{(i)} \neq k \end{cases} \tag{29}$$

The performance of each sub-classifier was assessed using the 2,000 unseen test samples, and the results were both plotted and printed to the active terminal as shown below:

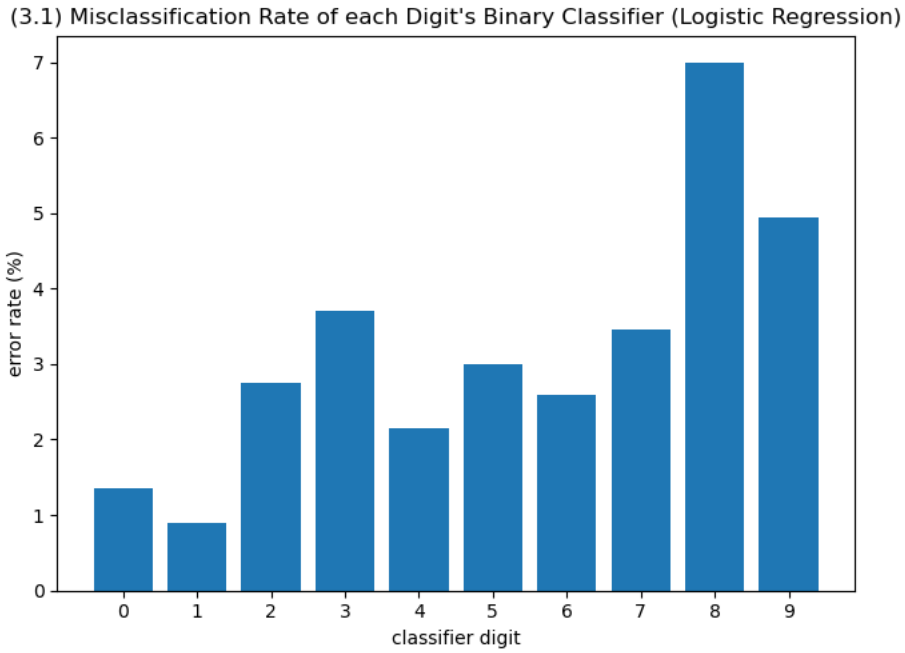

Figure 4: Problem (3.1) Sub-classifier Misclassification Rates

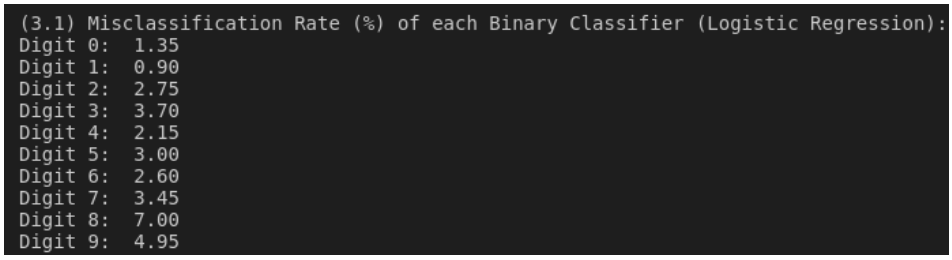

Figure 5: Problem (3.1) Sub-classifier Misclassification Rates

As can be seen from the results provided above, each of the individual sub-classifiers performed quite well on the unseen test samples, with an average error rate of 3.19% and no classifiers exceeding an error rate of 10% . It is also worth mentioning that the average misclassification rate of the sub-classifiers using logistic regression (3.19%) was slightly superior to that of the sub-classifiers using perceptrons (3.51%) as described in part 2. The next section will discuss the performance of the combined 10-way model.

3. In the same manner as in part 2, after training and testing each of the sub-classifiers as outlined above, the combined 10-way model was then evaluated in the previously described manner. To briefly restate, the output of the combined model for a given test sample is equivalent to the class of the most confident sub-classifier. The performance of the combined model was assessed using the 2,000 unseen test samples, and the results were printed to the active terminal as shown below:

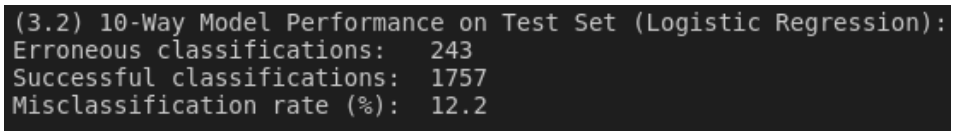

Figure 6: Problem (3.2) Combined Model Misclassification Rates

As can be seen from the results above, while the combined model is still successful roughly 88% of the time, it does not quite achieve the performance of any of the individual sub-classifiers. It is also worth mentioning that the misclassification rate of the combined model using logistic regression sub-classifiers (12.2%) was slightly superior to that

of the combined model using perceptron sub-classifiers (15.8%) as described in part 2. It should also be noted that unlike with perceptrons, the probability of a positive classification output for each sub-classifier can be directly computed (as previously described), which makes for an easier comparison of the logistic regression models when zero or multiple sub-classifiers return a positive output for a given input sample.