CS 5350/6350: Machine Learning Fall 2021

Homework 5

Handed out: 23 Nov, 2021
Due date: 11:59pm, 10 Dec, 2021

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 20 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* **You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.**

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

1 Paper Problems [40 points]

1. [5 points] (Warm up) Suppose we have a composite function, $z = \sigma(y_1^2 + y_2 y_3)$, where $y_1 = 3x$, $y_2 = e^{-x}$, $y_3 = \sin(x)$, and $\sigma(\cdot)$ is the sigmoid activation function . Please use the chain rule to derive $\frac{\partial z}{\partial x}$ and compute the derivative at $x = 0$.

   Answer

$$z = \sigma(y_1^2 + y_2 y_3) = (1 + e^{-y_1^2 - y_2 y_3})^{-1}$$

$$\frac{dz}{dx} = \frac{dz}{dy_1}\frac{dy_1}{dx} + \frac{dz}{dy_2}\frac{dy_2}{dx} + \frac{dz}{dy_3}\frac{dy_3}{dx}$$

   Solving for each term,

$$\frac{dz}{dy_1} = \frac{2y_1 e^{-y_1^2 - y_2 y_3}}{(1 + e^{-y_1^2 - y_2 y_3})^2}$$

$$\frac{dy_1}{dx} = 3$$

$$\frac{dz}{dy_2} = \frac{y_3 e^{-y_1^2 - y_2 y_3}}{(1 + e^{-y_1^2 - y_2 y_3})^2}$$

$$\frac{dy_2}{dx} = -e^{-x}$$

$$\frac{dz}{dy_3} = \frac{y_2 e^{-y_1^2 - y_2 y_3}}{(1 + e^{-y_1^2 - y_2 y_3})^2}$$

$$\frac{dy_3}{dx} = cos(x)$$

Plugging these terms back in,

$$\frac{dz}{dx} = \frac{e^{-9x^2 - e^{-x} sin(x)}}{(1 + exp(-9x^2 - e^{-x} sin(x)))^2}(18x - e^{-2x} + sin(x)cos(x))$$

Checking at 0,

$$\frac{dz(0)}{dx} = \frac{1}{4}(0 - 1 + 0) = -\frac{1}{4}$$

2. [5 points] Suppose we have a three-layered feed-forward neural network in hand. The architecture and the weights are defined in Figure 1. We use the sigmoid activation function. Note that the shaded variables are the constant feature 1, i.e., $x_0 = z_0^1 = z_0^2 = 1$. As we discussed in the class, they are used to account for the bias parameters. We have the values of all the edge weights in Table 1. Now, given a new input example $\mathbf{x} = [1, 1, 1]$. Please use the forward pass to compute the output y. Please list every step in your computation, namely, how you calculate the variable value in each hidden unit, and how you combine the variables in one layer to compute each variable in the next layer. Please be aware of the subtle difference in computing the variable value in the last layer (we emphasized it in the class).

   Answer

   Let's define some terms.

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{w}_1^1 = \begin{bmatrix} -1 \\ -2 \\ -3 \end{bmatrix}, \mathbf{w}_2^1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{z}_1 = \begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \end{bmatrix}, \mathbf{w}_1^2 = \begin{bmatrix} -1 \\ -2 \\ -3 \end{bmatrix}, \mathbf{w}_2^2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{z}_2 = \begin{bmatrix} z_0^2 \\ z_1^2 \\ z_2^2 \end{bmatrix}$$

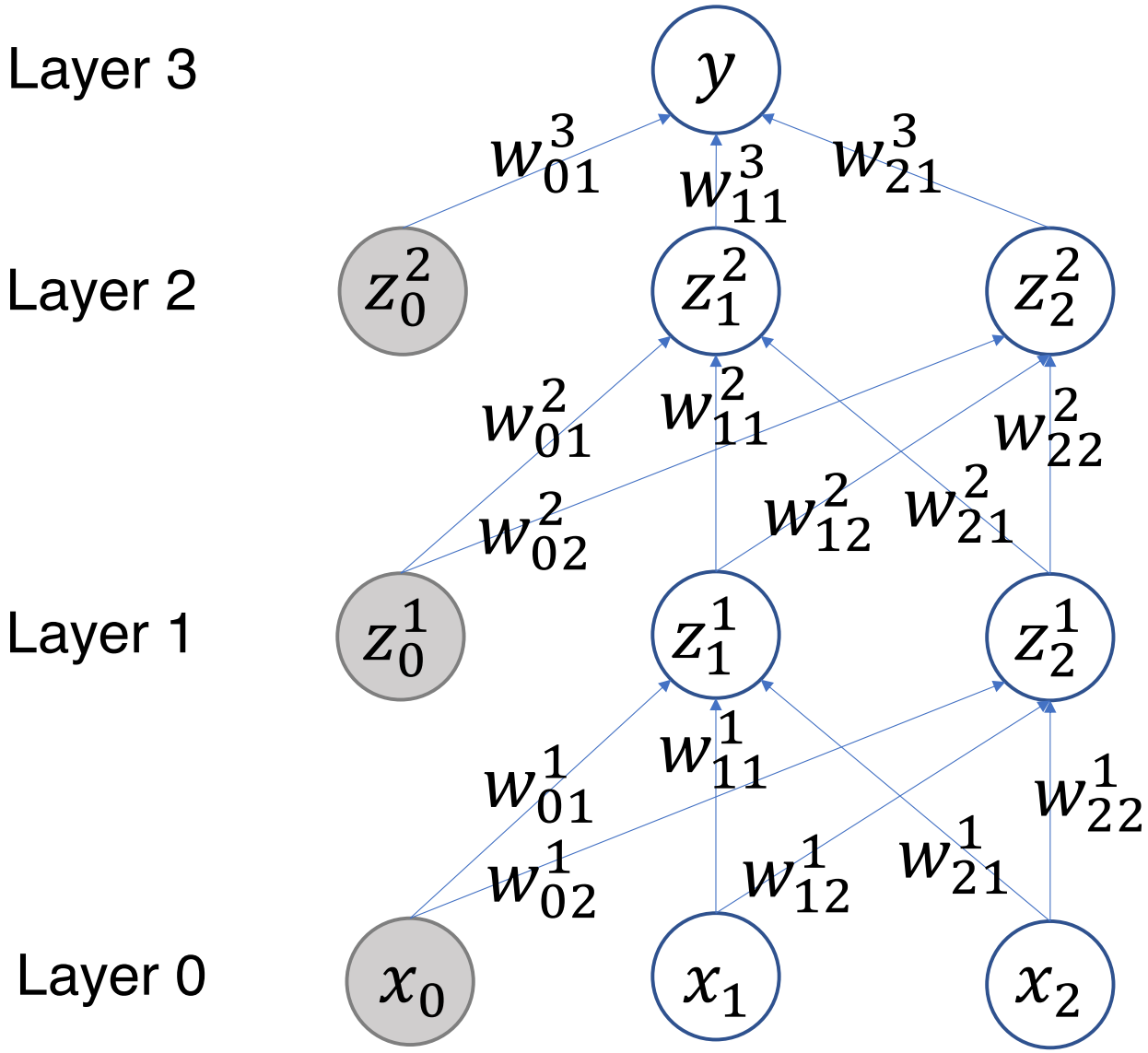

Figure 1: A three layer artificial neural network.

Layer	weigth	value
1	w_{01}^1	-1
1	w_{02}^1	1
1	w_{11}^1	-2
1	w_{12}^1	2
1	w_{21}^1	-3
1	w_{22}^1	3
2	w_{01}^2	-1
2	w_{02}^2	1
2	w_{11}^2	-2
2	w_{12}^2	2
2	w_{21}^2	-3
2	w_{22}^2	3
3	w_{01}^3	-1
3	w_{11}^3	2
3	w_{21}^3	-1.5

Table 1: Weight values.

$$\mathbf{w}_1^3 = \begin{bmatrix} -1 \\ 2 \\ -1.5 \end{bmatrix}$$

Then,

$$z_1^1 = \sigma(\mathbf{x}^\top \mathbf{w}_1^1) = 0.0025$$
$$z_2^1 = \sigma(\mathbf{x}^\top \mathbf{w}_2^1) = 0.9975$$
$$z_1^2 = \sigma(\mathbf{z}_1^\top \mathbf{w}_1^2) = 0.0180$$
$$z_2^2 = \sigma(\mathbf{z}_1^\top \mathbf{w}_2^2) = 0.9820$$
$$y = \mathbf{z}_2^\top \mathbf{w}_1^3 = -2.4369$$

The code for computing this can be found in the function q2 of Listing 1.

Listing 1: Functions used to generate results for Part 1 Questions 2 and 3

```
import math
import numpy as np


def q2():
    x = np.array([1, 1, 1])

    w11 = np.array([-1, -2, -3])
    w12 = np.array([1, 2, 3])
```

```
    w21 = np.array([-1, -2, -3])
    w22 = np.array([1, 2, 3])

    w31 = np.array([-1, 2, -1.5])

    z11 = sigmoid(np.dot(x.T, w11))
    print('z11:', z11)

    z12 = sigmoid(np.dot(x.T, w12))
    print('z12:', z12)

    z1 = np.array([1, z11, z12])

    z21 = sigmoid(np.dot(z1.T, w21))
    print('z21:', z21)

    z22 = sigmoid(np.dot(z1.T, w22))
    print('z22:', z22)

    z2 = np.array([1, z21, z22])

    y = np.dot(z2.T, w31)
    print('y:', y)


def q3():
    # FORWARD PASS CODE
    print('\nFORWARD PASS')
    x = np.array([1, 1, 1])

    w11 = np.array([-1, -2, -3])
    w12 = np.array([1, 2, 3])

    w21 = np.array([-1, -2, -3])
    w22 = np.array([1, 2, 3])

    w31 = np.array([-1, 2, -1.5])

    z11 = sigmoid(np.dot(x.T, w11))
    print('z11:', z11)

    z12 = sigmoid(np.dot(x.T, w12))
    print('z12:', z12)
```

```
z1 = np.array([1, z11, z12])

z21 = sigmoid(np.dot(z1.T, w21))
print('z21:', z21)

z22 = sigmoid(np.dot(z1.T, w22))
print('z22:', z22)

z2 = np.array([1, z21, z22])

y = np.dot(z2.T, w31)
print('y:', y)

# BP
print('\nBP')
print('\nLAYER 3')
dL_dy = y - 1
dy_dw31 = z2
dL_dw31 = dL_dy * dy_dw31
print('dL_dw31', dL_dw31)

print('\nLAYER 2')
dy_dz21 = w31[1]
dz21_dw21 = z1
dL_dw21 = dL_dy * dy_dz21 * dz21_dw21
print('dL_dw21', dL_dw21)

dy_dz22 = w31[2]
dz22_dw22 = z1
dL_dw22 = dL_dy * dy_dz22 * dz22_dw22
print('dL_dw22', dL_dw22)

print('\nLAYER 1')
dz21_dz11 = w21[1]
dz22_dz11 = w22[1]
dz11_dw11 = x
dL_dw11 = dL_dy * (
    dy_dz21 * dz21_dz11 + dy_dz22 * dz22_dz11
) * dz11_dw11
print('dL_dw11', dL_dw11)

dz21_dz12 = w21[2]
dz22_dz12 = w22[2]
dz12_dw12 = x
dL_dw11 = dL_dy * (
```

```
        dy_dz21 * dz21_dz12 + dy_dz22 * dz22_dz12
    ) * dz12_dw12
    print('dL_dw12', dL_dw11)


def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

3. [20 points] Suppose we have a training example where the input vector is $\mathbf{x} = [1, 1, 1]$ and the label $y^* = 1$. We use a square loss for the prediction,

$$L(y, y^*) = \frac{1}{2}(y - y^*)^2.$$

To make the prediction, we will use the 3 layer neural network shown in Figure 1, with the sigmoid activation function. Given the weights specified in Table 1, please use the back propagation (BP) algorithm to compute the derivative of the loss L over all the weights, $\{\frac{\partial L}{\partial w_{ij}^m}\}$. Please list every step of your BP calculation. In each step, you should show how you compute and cache the new (partial) derivatives from the previous ones, and then how to calculate the partial derivative over the weights accordingly.

Answer

We use the terms and results found in Part 1 Question 2.

We know that $\frac{dL}{dy} = y - y^*$ and $\frac{dL}{dw_{ij}} = \frac{dL}{dy}\frac{dy}{dw_{ij}}$, so we easily find that,

$$\frac{dy}{d\mathbf{w}_1^3} = \begin{bmatrix} z_0^2 \\ z_1^2 \\ z_2^2 \end{bmatrix}$$

Combining these, we find,

$$\frac{dL}{d\mathbf{w}_1^3} = \frac{dL}{dy}\frac{dy}{d\mathbf{w}_1^3} = (y-1)\begin{bmatrix} z_0^2 \\ z_1^2 \\ z_2^2 \end{bmatrix} = \begin{bmatrix} -3.4368 \\ -0.0620 \\ -3.3750 \end{bmatrix}$$

Moving onto Layer 2 from Layer 3, we know that $\frac{dy}{dz_1^2} = w_{11}^3$ and $\frac{dL}{d\mathbf{w}_1^2} = \frac{dL}{dy}\frac{dy}{dz_1^2}\frac{dz_1^2}{d\mathbf{w}_1^2}$. We find that,

$$\frac{dz_1^2}{d\mathbf{w}_1^2} = \begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \end{bmatrix} => \frac{dL}{d\mathbf{w}_1^2} = (y-1)(w_{11}^3)\begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \end{bmatrix} = \begin{bmatrix} -6.8738 \\ -0.0170 \\ -6.8568 \end{bmatrix}$$

We also know that $\frac{dy}{dz_2^2} = w_{21}^3$ and $\frac{dL}{d\mathbf{w}_2^2} = \frac{dL}{dy}\frac{dy}{dz_2^2}\frac{dz_2^2}{d\mathbf{w}_2^2}$. We find that,

$$\frac{dz_2^2}{d\mathbf{w}_2^2} = \begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \end{bmatrix} => \frac{dL}{d\mathbf{w}_2^2} = (y-1)(w_{21}^3) \begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \end{bmatrix} = \begin{bmatrix} 5.1553 \\ 0.0127 \\ 5.1426 \end{bmatrix}$$

Finally, for Layer 3 we know that $\frac{dz_1^2}{dz_1^1} = w_{11}^2$, $\frac{dz_2^2}{dz_1^1} = w_{12}^2$, and $\frac{dL}{d\mathbf{w}_1^1} = \frac{dL}{dy}(\frac{dy}{dz_1^2}\frac{dz_1^2}{dz_1^1} + \frac{dy}{dz_2^2}\frac{dz_2^2}{dz_1^1})\frac{dz_1^1}{d\mathbf{w}_1^1}$. We find that,

$$\frac{dz_1^1}{d\mathbf{w}_1^1} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} => \frac{dL}{d\mathbf{w}_1^1} = (y-1)(w_{11}^3 w_{11}^2 + w_{21}^3 w_{12}^2) \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 24.0583 \\ 24.0583 \\ 24.0583 \end{bmatrix}$$

We also know that $\frac{dz_1^2}{dz_2^1} = w_{21}^2$, $\frac{dz_2^2}{dz_2^1} = w_{22}^2$, and $\frac{dL}{d\mathbf{w}_2^1} = \frac{dL}{dy}(\frac{dy}{dz_1^2}\frac{dz_1^2}{dz_2^1} + \frac{dy}{dz_2^2}\frac{dz_2^2}{dz_2^1})\frac{dz_2^1}{d\mathbf{w}_2^1}$. We find that,

$$\frac{dz_2^1}{d\mathbf{w}_2^1} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} => \frac{dL}{d\mathbf{w}_2^1} = (y-1)(w_{11}^3 w_{21}^2 + w_{21}^3 w_{22}^2) \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 36.0874 \\ 36.0874 \\ 36.0874 \end{bmatrix}$$

The code to get these results can be found in function q3 of Listing 1.

4. [10 points] Suppose we have the training dataset shown in Table 2. We want to learn a logistic regression model. We initialize all the model parameters with 0. We assume each parameter (i.e., feature weights $\{w_1, w_2, w_3\}$ and the bias w_0) comes from a standard Gaussian prior distribution,

$$p(w_i) = \mathcal{N}(w_i|0,1) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}w_i^2) \;\; (0 \leq i \leq 3).$$

- [7 points] We want to obtain the maximum a posteriori (MAP) estimation. Please write down the objective function, namely, the log joint probability, and derive the gradient of the objective function.
- [3 points] We set the learning rates for the first three steps to $\{0.01, 0.005, 0.0025\}$. Please list the stochastic gradients of the objective w.r.t the model parameters for the first three steps, when using the stochastic gradient descent algorithm.

x_1	x_2	x_3	y
0.5	-1	0.3	1
-1	-2	-2	-1
1.5	0.2	-2.5	1

Table 2: Dataset

2 Practice [62 points + 60 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of SVM algorithms. Remember last time you created the folders "SVM". You can commit your code into the corresponding folders now. Please also supplement README.md with concise descriptions about how to use your code to run these algorithms (how to call the command, set the parameters, etc). Please create new folders "Neural Networks" and "Logistic Regression" in the same level as these folders. *After the completion of the homework this time, please check in your implementation accordingly.*

   Answer

   The code for this project can be found in https://github.com/Paul-Wissler/cs-6350-hw5.

2. [58 points] Now let us implement a three-layer artificial neural network for classification. We will use the dataset, "bank-note.zip" in Canvas. The features and labels are listed in the file "classification/data-desc.txt". The training data are stored in the file "classification/train.csv", consisting of 872 examples. The test data are stored in "classification/test.csv", and comprise of 500 examples. In both the training and test datasets, feature values and labels are separated by commas. The architecture of the neural network resembles Figure 1, but we allow an arbitrary number of units in hidden layers (Layer 1 and 2). So please ensure your implementation has such flexibility. We will use the sigmoid activation function.

   (a) [25 points] Please implement the back-propagation algorithm to compute the gradient with respect to all the edge weights given one training example. For debugging, you can use the paper problem 3 and verify if your algorithm returns the same derivatives as you manually did.

   Answer

   I verified this in QuestionAnswers.part2.q2a of my code. See the repo.

   (b) [17 points] Implement the stochastic gradient descent algorithm to learn the neural netowrk from the training data. Use the schedule of learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{d}t}$. Initialize the edge weights with random numbers generated from the standard Gaussian distribution. We restrict the width, i.e., the number of nodes, of each hidden layer (i.e., Layer 1 & 2) to be identical. Vary the width from $\{5, 10, 25, 50, 100\}$. Please tune γ_0 and d to ensure convergence. Use the curve of the objective function (along with the number of updates) to diagnosis the convergence. Don't forget to shuffle the training examples at the start of each epoch. Report the training and test error for each setting of the width.

   Answer

   ```
   Q2b

   LAYERS 5
   ```

```
TRAINING ERROR: 0.9357798165137615
TEST ERROR: 0.93

LAYERS 10
TRAINING ERROR: 0.9736238532110092
TEST ERROR: 0.968

LAYERS 25
TRAINING ERROR: 0.9518348623853211
TEST ERROR: 0.946

LAYERS 50
TRAINING ERROR: 0.9403669724770642
TEST ERROR: 0.952

LAYERS 100
TRAINING ERROR: 0.6410550458715596
TEST ERROR: 0.642
```

(c) [10 points]. Now initialize all the weights with 0, and run your training algorithm again. What is your training and test error? What do you observe and conclude?
Answer

```
Q2c

LAYERS 5
TRAINING ERROR: 0.786697247706422
TEST ERROR: 0.794

LAYERS 10
TRAINING ERROR: 0.9288990825688074
TEST ERROR: 0.928

LAYERS 25
TRAINING ERROR: 0.9357798165137615
TEST ERROR: 0.934

LAYERS 50
TRAINING ERROR: 0.948394495412844
TEST ERROR: 0.932

LAYERS 100
TRAINING ERROR: 0.8692660550458715
TEST ERROR: 0.856
```

Comparing the results, they both have similar performance (¿90% accuracy) when

the width is between 10 and 50, though the Gaussian initialization seems to edge out the 0 initialization by a few points on average. Both perform poorly with a width of 100, though the Gaussian initialization performs quite a bit worse, but also quite a bit better with a width of 5. I would conclude that initialization matters less than creating a good structure for your neural net, but if the structure is relatively thin, then a proper initialization can help find the true minimum of the objective function.

(d) [6 points]. As compared with the performance of SVM (and the logistic regression you chose to implement it; see Problem 3), what do you conclude (empirically) about the neural network?

Answer

It is easier to tune hyperparameters for a neural net, though the different structure introduces a host of different design problems. Also, when the SVM performed well, it managed to reach perfect separability, whereas the neural net did not with my specific implementation. Based purely on these results, I would conclude that the SVM has a better potential to reach perfect separability if tuned properly.

(e) [**Bonus**] [30 points] Please use PyTorch (or TensorFlow if you want) to fulfill the neural network training and prediction. Please try two activation functions, "tanh" and "RELU". For "tanh", please use the "Xavier' initialization; and for "RELU", please use the "he" initialization. You can implement these initializations by yourselves or use PyTorch (or TensorFlow) library. Vary the depth from $\{3, 5, 9\}$ and width from $\{5, 10, 25, 50, 100\}$. Pleas use the Adam optimizer for training. The default settings of Adam should be sufficient (*e.g.,* initial learning rate is set to 10^{-3}). Report the training and test error with each (depth, width) combination. What do you observe and conclude? Note that, we won't provide any link or manual for you to work on this bonus problem. It is YOUR JOB to search the documentation, find code snippets, test, and debug with PyTorch (or TensorFlow) to ensure the correct usage. This is what all machine learning practitioners do in practice.

Answer

The results can be found in Table 3. Based on these results, it is obvious that PyTorch neural nets perform quite a bit better than mine, sometimes reaching perfect separability on the test data. This clearly indicates that, as a non-linear classifier, neural nets are quite robust. However, adding the additional complexities of varying widths, depths, and activation functions clearly indicate that in practice neural nets take quite a bit of engineering and tuning to get working well.

3. [**Bonus**] [30 points] We will implement the logistic regression model with stochastic gradient descent. We will use the dataset "bank-note.zip" in Canvas. Set the maximum number of epochs T to 100. Don't forget to shuffle the training examples at the start of each epoch. Use the curve of the objective function (along with the number of updates) to diagnosis the convergence. We initialize all the model parameters with 0.

(a) [10 points] We will first obtain the MAP estimation. In order for that, we assume

Activation	Depth	Width	Train Error	Test Error
Tanh	3	5	91.51%	91.80%
Tanh	3	10	98.50%	98.40%
Tanh	3	25	99.31%	99.40%
Tanh	3	50	100.0%	100.0%
Tanh	3	100	100.0%	100.0%
Tanh	5	5	95.76%	93.80%
Tanh	5	10	98.51%	97.80%
Tanh	5	25	99.77%	99.80%
Tanh	5	50	99.08%	99.80%
Tanh	5	100	97.25%	96.80%
Tanh	9	5	94.15%	94.80%
Tanh	9	10	99.31%	99.20%
Tanh	9	25	97.71%	98.20%
Tanh	9	50	97.59%	99.77%
Tanh	9	100	96.79%	96.40%
ReLU	3	5	99.08%	98.40%
ReLU	3	10	90.37%	89.80%
ReLU	3	25	100.0%	100.0%
ReLU	3	50	100.0%	100.0%
ReLU	3	100	100.0%	100.0%
ReLU	5	5	93.12%	93.20%
ReLU	5	10	100.0%	100.0%
ReLU	5	25	99.89%	100.0%
ReLU	5	50	97.13%	96.40%
ReLU	5	100	83.83%	82.20%
ReLU	9	5	94.27%	95.20%
ReLU	9	10	98.85%	98.80%
ReLU	9	25	99.77%	99.60%
ReLU	9	50	100.0%	99.80%
ReLU	9	100	98.85%	98.80%

Table 3: Error results for Part 2 Question 2e.

each model parameter comes from a Gaussian prior distribution,

$$p(w_i) = \mathcal{N}(w_i|0, v) = \frac{1}{\sqrt{2\pi v}} \exp(-\frac{1}{2v} w_i^2)$$

where v is the variance. From the paper problem 4, you should be able to write down the objective function and derive the gradient. Try the prior variance v from $\{0.01, 0.1, 0.5, 1, 3, 5, 10, 100\}$. Use the schedule of learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{d}t}$. Please tune γ_0 and d to ensure convergence. For each setting of variance, report your training and test error.

(b) [5 points] We will then obtain the maximum likelihood (ML) estimation. That is, we do not assume any prior over the model parameters, and just maximize the logistic likelihood of the data. Use the same learning rate schedule. Tune γ_0 and d to ensure convergence. For each setting of variance, report your training and test error.

(c) [3 points] How is the training and test performance of the MAP estimation compared with the ML estimation? What can you conclude? What do you think of v, as compared to the hyperparameter C in SVM?

4. [2 Points] After the completion, please upload the implementation to your Github repository immediately. How do you like your own machine learning library? *Although it is still light weighted, it is the proof of your great efforts and achievement in this class! It is an excellent start of your journey to machine learning. Wish you further success in your future endeavours!*

Answer

I like knowing that I am capable of implementing so many different algorithms. However, I wonder if perhaps this class wouldn't benefit from being split into two parts, as the workload over the short span of time felt extreme. Thank you for all your hard work this semester, and good luck!