CS 5350/6350: Machine Learning Fall 2022

Homework 5

Handed out: 22 Nov, 2022
Due date: 11:59pm, 9 Dec, 2022

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 $x = 0$.

   To find the derivative we can first exapand the equation according to the chain rule:

$$\frac{\partial z}{\partial x} = \big(\sigma(y_1^2 + y_1 y_2) * (1 - \sigma(y_1^2 + y_1 y_2)\big) * \big(2y_1 * \frac{\partial y_1}{\partial x} + (\frac{\partial y_2}{\partial x} * y_3 + y_2 * \frac{\partial y_3}{\partial x})\big)$$

$$= \big(\sigma((3x)^2 + e^{-x}\sin(x)) * (1 - \sigma((3x)^2 + e^{-x}\sin(x))\big) * \big(2(3x) * 3 + (-e^{-x}\sin(x) + e^{-x}\cos(x)\big)$$

   Now we can plug in $x = 0$ to find the derivative:

$$\frac{\partial z}{\partial x} = \big(\sigma(0 + e^0\sin(0)) * (1 - \sigma(0 + e^0\sin(0))\big) * \big(0 + (-e^0\sin(0) + e^0\cos(0)\big)$$

$$= \sigma(0) * (1 - \sigma(0)) * 1 = 0.5 * 0.5 * 1 = 0.25$$

2. [5 points]

$$z_0^1 = 1$$

$$z_1^1 = \sigma(w_{01}^1 x_0 + w_{11}^1 x_1 + w_{21}^1 x_2) = \sigma(-1 * 1 + -2 * 1 + -3 * 1) = \sigma(-6) = 0.0025$$

$$z_2^1 = \sigma(w_{02}^1 x_0 + w_{12}^1 x_1 + w_{22}^1 x_2) = \sigma(1 * 1 + 2 * 1 + 3 * 1) = \sigma(6) = 0.9975$$

$$z_0^2 = 1$$

$$z_1^2 = \sigma(w_{01}^2 z_0^1 + w_{11}^2 z_1^1 + w_{21}^2 z_2^1) = \sigma(-1*1 + -2*0.0025 + -3*0.9975) = \sigma(-3.9975) = 0.018$$

$$z_2^2 = \sigma(w_{02}^2 z_0^1 + w_{12}^2 z_1^1 + w_{22}^2 z_2^1) = \sigma(1*1 + 2*0.0025 + 3*0.9975) = sigma(3.9975) = 0.982$$

$$y = w_{01}^3 z_0^2 + w_{11}^3 z_1^2 + w_{21}^3 z_2^2 = -1 * 1 + 2 * 0.018 + -1.5 * 0.982 = -2.437$$

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

Table 1: Weight values.

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 **??**, 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, $\{\frac{\partial L}{\partial w^m_{ij}}\}$. 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.

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.*

2. [58 points]

   (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.

   *See output from code

   (b) [17 points]

   Note: Because of the randomness in shuffling the data, consecutive runs do not produce the same error values so there may be variation in my results.

   Training error of NN with a width of 5 : 0.048165
   Test error of NN with a width of 5 : 0.062000
   Training error of NN with a width of 10 : 0.030963
   Test error of NN with a width of 10 : 0.040000
   Training error of NN with a width of 25 : 0.017202
   Test error of NN with a width of 25 : 0.020000
   Training error of NN with a width of 50 : 0.006881
   Test error of NN with a width of 50 : 0.016000
   Training error of NN with a width of 100 : 0.020642
   Test error of NN with a width of 100 : 0.022000

   (c) [10 points].

   Training error of NN with a width of 5 : 0.446101
   Test error of NN with a width of 5 : 0.442000
   Training error of NN with a width of 10 : 0.446101

Test error of NN with a width of 10 : 0.442000
Training error of NN with a width of 25 : 0.446101
Test error of NN with a width of 25 : 0.442000
Training error of NN with a width of 50 : 0.446101
Test error of NN with a width of 50 : 0.442000
Training error of NN with a width of 100 : 0.446101
Test error of NN with a width of 100 : 0.442000

(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?

I found that when comparing the results of the neural network with Gaussian initialization to the results of SVM using the Gaussian kernel, SVM tended to perform better across the board. Compared to SVM without the Gaussian kernel, though, I found that the neural network performed comparably and, in some cases, better than SVM. I think that the reason for this is that the problem we are solving is not very complex, as the data set is relatively small and has low dimensionality. Because of this SVM is better suited to solve the problem however, were the data set larger and with higher dimensionality, we would likely see better results out of the neural network.

(e) [**Bonus**] [30 points]

For these results I used a learning rate of 0.01, Adam optimization, and 20 epochs. Using the ReLu activation function with He weight initialization:

Model with depth = 3 and width = 5:
Training error: 0.119266 - Test error 0.122000
Training loss: 0.409450 - Test loss 0.416117

Model with depth = 3 and width = 10:
Training error: 0.001147 - Test error 0.004000
Training loss: 0.075366 - Test loss 0.079842

Model with depth = 3 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.050840 - Test loss 0.056322

Model with depth = 3 and width = 50:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.026100 - Test loss 0.038678

Model with depth = 3 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.057228 - Test loss 0.064813

Model with depth = 5 and width = 5:
Training error: 0.018349 - Test error 0.018000
Training loss: 0.129987 - Test loss 0.149010

Model with depth = 5 and width = 10:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.051453 - Test loss 0.057860

Model with depth = 5 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.028150 - Test loss 0.032893

Model with depth = 5 and width = 50:
Training error: 0.000000 - Test error 0.002000
Training loss: 0.032320 - Test loss 0.051215

Model with depth = 5 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.023819 - Test loss 0.028742

Model with depth = 9 and width = 5:
Training error: 0.077982 - Test error 0.076000
Training loss: 0.327198 - Test loss 0.332288

Model with depth = 9 and width = 10:
Training error: 0.043578 - Test error 0.054000
Training loss: 0.134775 - Test loss 0.184193

Model with depth = 9 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.049835 - Test loss 0.061753

Model with depth = 9 and width = 50:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.004979 - Test loss 0.008210

Model with depth = 9 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.008816 - Test loss 0.013475

Using TanH activations with Xavier initialization:

Model with depth = 3 and width = 5:
Training error: 0.004587 - Test error 0.004000
Training loss: 0.036218 - Test loss 0.039202

Model with depth = 3 and width = 10:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.011056 - Test loss 0.010795

Model with depth = 3 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.011187 - Test loss 0.012004

Model with depth = 3 and width = 50:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.006518 - Test loss 0.007689

Model with depth = 3 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.004723 - Test loss 0.006303

Model with depth = 5 and width = 5:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.006438 - Test loss 0.008248

Model with depth = 5 and width = 10:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.002368 - Test loss 0.004639

Model with depth = 5 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000796 - Test loss 0.001499

Model with depth = 5 and width = 50:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000494 - Test loss 0.001037

Model with depth = 5 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000165 - Test loss 0.000506

Model with depth = 9 and width = 5:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000371 - Test loss 0.000553

Model with depth = 9 and width = 10:
Training error: 0.000000 - Test error 0.002000
Training loss: 0.000282 - Test loss 0.002386

Model with depth = 9 and width = 25:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000173 - Test loss 0.000486

Model with depth = 9 and width = 50:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000075 - Test loss 0.000251

Model with depth = 9 and width = 100:
Training error: 0.000000 - Test error 0.000000
Training loss: 0.000124 - Test loss 0.000386

While activation functions performed very well, I found that the performance using the TanH activation function with Xavier initialization worked slightly better. Only three combinations of depth and width using the ReLu activation produced training and testing errors above zero, those were depth 3 with width 5, depth 3 with width 10, and depth 5 with width 5. Using the TanH activation function only the model with depth 3 and width five produced test and training error above 0. Overall I found that increasing both the depth and width improved results.

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 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!*

While I think I could improve upon the flexibility of the models that I have implemented, I am proud of what I have been able to accomplish in this class with this library. I learned a great deal making it.