CS 5350/6350: Machine Learning Spring 2019

Homework 5

Handed out: 9 Apr, 2019
Due date: 11:59pm, 24 Apr, 2019

- 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 15 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]

Answers & solutions are in blue fonts.

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 $x = 0$.
   The derivative for sigmoid function $\sigma(s) = \frac{1}{1+e^{-s}}$ is $\frac{\partial \sigma}{\partial s} = \sigma(s)(1 - \sigma(s))$. Using chain rule, we get

$$
\begin{aligned}
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x} + \frac{\partial z}{\partial y_3}\frac{\partial y_3}{\partial x} && (1)\\
&= 3\sigma(y_1)(1-\sigma(y_1)) + -e^{-x}\sigma(y_2)(1-\sigma(y_2)) + \cos(x)\sigma(y_3)(1-\sigma(y_3)) && (2)\\
&= 3\sigma(3x)(1-\sigma(3x)) - e^{-x}\sigma(e^{-x})(1-\sigma(e^{-x})) && (3)\\
&\quad + \cos(x)\sigma(\sin(x))(1-\sigma(\sin(x))) && (4)
\end{aligned}
$$

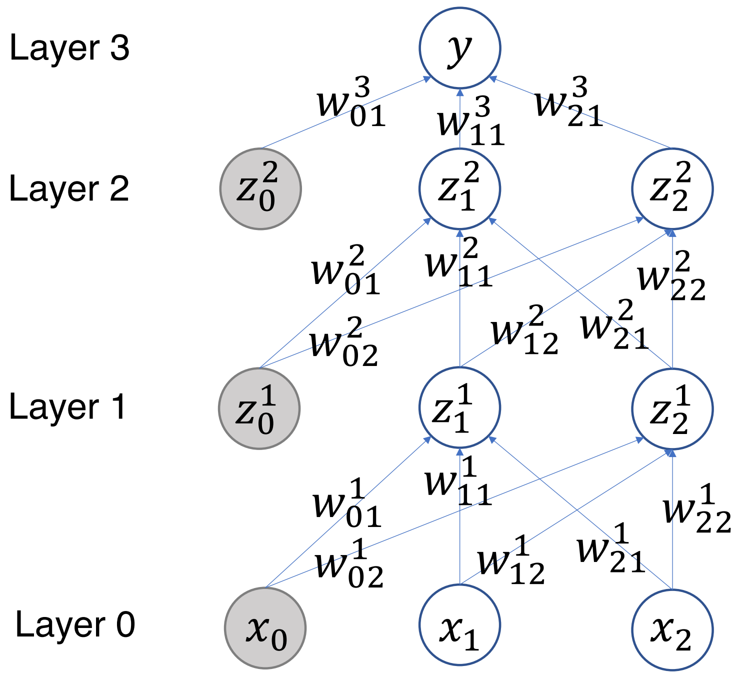

Figure 1: A three layer artificial neural network.

and

$$\left.\frac{\partial z}{\partial x}\right|_{x=0} = 3.3034 \tag{5}$$

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).
(Forward Pass) In the output layer there is no activation.
For layer 1, we have

$$z_1^1 = \sigma(\sum_{i=0}^{2} w_{i1}^1 x_i) = \sigma(-1-2-3) = 0.0025 \tag{6}$$

$$z_2^1 = \sigma(\sum_{i=0}^{2} w_{i2}^1 x_i) = \sigma(1+2+3) = 0.9975 \tag{7}$$

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.

For layer 2 and 3,

$$z_1^2 = \sigma(\sum_{i=0}^{2} w_{i1}^2 z_i^1) = \sigma(-1 - 2 \times 0.0025 - 3 \times 0.9975) = 0.0180 \tag{8}$$

$$z_2^2 = \sigma(\sum_{i=0}^{2} w_{i2}^2 z_i^1) = \sigma(1 + 2 \times 0.0025 + 3 \times 0.0075) = 0.9820 \tag{9}$$

$$y = \sum_{i=0}^{2} w_{i1}^3 z_i^2 = -1 + 2 \times 0.018 - 1.5 \times 0.982 = -2.4370 \tag{10}$$

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.

(BP) For layer 2 weights, we have

$$\frac{\partial L}{\partial w_{01}^3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^3} = \underbrace{(y - y*)}_{=-3.4370\,,cached} z_0^2 = -2.4370 - 1 = -3.4370 \tag{11}$$

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^3} = -3.4370 z_1^2 = -3.4370 \times 0.018 = -0.0619 \tag{12}$$

$$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{21}^3} = -3.4370 z_2^2 = -3.4370 \times 0.982 = -3.3751 \tag{13}$$

For layer 1 weights,

$$\begin{aligned}
\frac{\partial L}{\partial w_{01}^2} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{01}^2} && (14)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial \sigma(s)}{\partial s}\frac{\partial s}{\partial w_{01}^2} && (15)\\
&= \underbrace{(y - y*) w_{11}^3 \sigma(s)(1-\sigma(s))}_{cached} z_0^1 && (16)\\
&= \underbrace{(y - y*) w_{11}^3 z_1^2 (1-z_1^2)}_{cached} z_0^1 && (17)\\
&= -3.4370 \times 2 \times 0.018 \times (1 - 0.018) \times 1 && (18)\\
&= -0.1215 && (19)\\
\frac{\partial L}{\partial w_{11}^2} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2} && (20)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial \sigma(s)}{\partial s}\frac{\partial s}{\partial w_{11}^2} && (21)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial \sigma(s)}{\partial s} z_1^1 && (22)\\
&= -0.1215 \times 0.0025 = -3.0375e - 04 && (23)\\
\frac{\partial L}{\partial w_{21}^2} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{21}^2} && (24)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial \sigma(s)}{\partial s}\frac{\partial s}{\partial w_{21}^2} && (25)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_1^2}\frac{\partial \sigma(s)}{\partial s} z_2^1 && (26)\\
&= -0.1215 \times 0.0075 = -0.1212 && (27)
\end{aligned}$$

where $s = \sum_{i=0}^{2} w_{i1}^2 z_i^1 = -1 - 2 \times 0.0025 - 3 \times 0.9975$ and $\sigma(s) = z_1^2 = 0.018$. Similarly, for the other layer 1 weights, we have

$$\begin{aligned}
\frac{\partial L}{\partial w^2_{02}} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z^2_2}\frac{\partial z^2_2}{\partial w^2_{02}} && (28)\\
&= \frac{\partial L}{\partial y}\frac{\partial y}{\partial z^2_2}\frac{\partial \sigma(s)}{\partial s}\frac{\partial s}{\partial w^2_{02}} && (29)\\
&= \underbrace{(y-y*)w^3_{21}z^2_2(1-z^2_2)}_{cached}\, z^1_0 && (30)\\
&= -3.4370 \times (-1.5) \times 0.982 \times (1-0.982) \times 1 = 0.0911 && (31)\\
\frac{\partial L}{\partial w^2_{12}} &= \underbrace{(y-y*)w^3_{21}z^2_2(1-z^2_2)}_{cached}\, z^1_1 && (32)\\
&= 0.0911 \times 0.0025 = 2.2775e-04 && (33)\\
\frac{\partial L}{\partial w^2_{22}} &= \underbrace{(y-y*)w^3_{21}z^2_2(1-z^2_2)}_{cached}\, z^1_2 && (34)\\
&= 0.0911 \times 0.9975 = 0.0909 && (35)
\end{aligned}$$

For layer 0 weights, there are two paths corresponding to each weight.

$$\begin{aligned}
\frac{\partial L}{\partial w^1_{01}} &= \frac{\partial L}{\partial y}\left(\frac{\partial y}{\partial z^2_1}\frac{\partial z^2_1}{\partial z^1_1} + \frac{\partial y}{\partial z^2_2}\frac{\partial z^2_2}{\partial z^1_1}\right)\frac{\partial z^1_1}{\partial w^1_{01}} && (36)\\
&= \underbrace{\left(\frac{\partial L}{\partial z^2_1}z^2_1(1-z^2_1)w^2_{11} + \frac{\partial L}{\partial z^2_2}z^2_2(1-z^2_2)w^2_{12}\right) z^1_1(1-z^1_1)}_{cached}\, x_0 && (37)\\
&= \underbrace{\left((y-y*)w^3_{11}z^2_1(1-z^2_1)w^2_{11} + (y-y*)w^3_{21}z^2_2(1-z^2_2)w^2_{12}\right) z^1_1(1-z^1_1)}_{cached}\, x_0 && (38)\\
&= \underbrace{(-0.1215 \times (-2) + 0.0911 \times 2) \times 0.0025 \times (1-0.0025)}_{cached}\, x_0 && (39)\\
&= 0.0011 && (40)\\
\frac{\partial L}{\partial w^1_{11}} &= 0.0011x_1 = 0.0011 && (41)\\
\frac{\partial L}{\partial w^1_{21}} &= 0.0011x_2 = 0.0011 && (42)
\end{aligned}$$

Similarly, for the other 3 layer 0 weights, we have

$$\begin{aligned}
\frac{\partial L}{\partial w^1_{02}} &= \frac{\partial L}{\partial y}\left(\frac{\partial y}{\partial z^2_1}\frac{\partial z^2_1}{\partial z^1_2} + \frac{\partial y}{\partial z^2_2}\frac{\partial z^2_2}{\partial z^1_2}\right)\frac{\partial z^1_1}{\partial w^1_{01}} && (43)\\
&= \underbrace{\left(\frac{\partial L}{\partial z^2_1} z^2_1(1-z^2_1)w^2_{21} + \frac{\partial L}{\partial z^2_2} z^2_2(1-z^2_2)w^2_{22}\right) z^1_2(1-z^1_2)}_{cached} x_0 && (44)\\
&= \underbrace{\left((y-y*)w^3_{11}z^2_1(1-z^2_1)w^2_{21} + (y-y*)w^3_{21}z^2_2(1-z^2_2)w^2_{22}\right) z^1_2(1-z^1_2)}_{cached} x_0 && (45)\\
&= \underbrace{(-0.1215\times(-3)+0.0911\times 3)\times 0.9975\times(1-0.9975)}_{cached} x_0 && (46)\\
&= 0.0016 && (47)\\
\frac{\partial L}{\partial w^1_{12}} &= 0.0016 x_1 = 0.0016 && (48)\\
\frac{\partial L}{\partial w^1_{22}} &= 0.0016 x_2 = 0.0016 && (49)
\end{aligned}$$

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,

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

- [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.
  Assume the samples are augmented with constant feature $x_0 = 1$. The MAP objective is
  $$\max_{\mathbf{w}} P(S|\mathbf{w})p(\mathbf{w})$$
  where under the i.i.d. assumption on both the dataset $S(m = 3, d = 4)$ and

entries of weight $\mathbf{w}$, we have

$$\begin{aligned}
\log P(S|\mathbf{w}) &= \log\left(\prod_{i=1}^{m} p(y_i|\mathbf{x}_i,\mathbf{w})\right) = \log\left(\prod_{i=1}^{m}\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}\right) && (50)\\
&= -\sum_{i=1}^{m}\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}) && (51)\\
\log p(\mathbf{w}) &= \log\left(\prod_{i=1}^{d} p(w_i)\right) = \log\left(\prod_{i=1}^{d}\frac{1}{\sqrt{2\pi}}e^{-\frac{w_i^2}{2}}\right) && (52)\\
&= -\sum_{i=1}^{d}\frac{w_i^2}{2} - d\log(2\pi) && (53)\\
&= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + Constant && (54)
\end{aligned}$$

As a result, the MAP objective becomes

$$\min_{\mathbf{w}} L(\mathbf{w}) := \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{m}\log\left(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}\right)$$

which is an unconstrained convex optimization problem. The derivative is ($\sigma(.)$ is the sigmoid function)

$$\nabla L(\mathbf{w}) = \mathbf{w} - \sum_{i=1}^{m} y_i\mathbf{x}_i\left(1-\sigma(y_i\mathbf{w}^T\mathbf{x}_i)\right)$$

and the SGD loss gradient is

$$\nabla L^{\text{sgd}}(\mathbf{w},\mathbf{x}_i,y_i) = \mathbf{w} - my_i\mathbf{x}_i\left(1-\sigma(y_i\mathbf{w}^T\mathbf{x}_i)\right), i\in\{1,2,3\}$$

- [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

The stochastic gradient $g(\mathbf{x}_i)$ is $g(\mathbf{x}_i) = \mathbf{w} - m y_i \mathbf{x}_i \left(1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)\right)$ for some $i = 1, 2, 3$. When initialized with $\mathbf{w}_0 = [0, 0, 0, 0]^T$, the updates are as follows.

$$\begin{aligned}
y_1 \mathbf{w}_0^T \mathbf{x}_1 &= 0 && (55)\\
\nabla L(\mathbf{w}_0) &= \mathbf{w}_0 - 3(1 - \sigma(0))[1, 0.5, -1, 0.3]^T = [-1.5, -0.75, 1.5, -0.45]^T && (56)\\
\mathbf{w}_1 &= \mathbf{w}_0 - 0.01 \nabla L(\mathbf{w}_0) = 0.01 \times [1.5, 0.75, -1.5, 0.45]^T && (57)\\
y_2 \mathbf{w}_1^T \mathbf{x}_2 &= -0.0285 && (58)\\
\nabla L(\mathbf{w}_1) &= \mathbf{w}_1 - 3(1 - \sigma(-0.0285))[1, -1, -2, -2]^T = [-1.5064, 1.5289, 3.0277, 3.0472]^T &&\\
\mathbf{w}_2 &= \mathbf{w}_1 - 0.005 \nabla L(\mathbf{w}_1) && (59)\\
&= 0.01 \times [1.5, 0.75, -1.5, 0.45]^T - 0.005 \times [-1.5064, 1.5289, 3.0277, 3.0472]^T && (60)\\
&= [0.0225, -0.0001, -0.0301, -0.0107]^T && (61)\\
y_3 \mathbf{w}_2^T \mathbf{x}_3 &= 0.0431 && (62)\\
\nabla L(\mathbf{w}_2) &= \mathbf{w}_2 - 3(1 - \sigma(0.0431))[1, 1.5, 0.2, -2.5]^T && (63)\\
&= [-1.4452, -2.2016, -0.3236, 3.6585]^T && (64)\\
\mathbf{w}_3 &= \mathbf{w}_2 - 0.0025 \nabla L(\mathbf{w}_2) && (65)\\
&= [0.0225, -0.0001, -0.0301, -0.0107]^T - 0.0025 \times [-1.4452, -2.2016, -0.3236, 3.6585]^T &&\\
&= [0.0261, 0.0054, -0.0293, -0.0198]^T && (66)
\end{aligned}$$

2 Practice [62 points + 50 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.*
   Check out: `https://github.com/xiangzhang-122/Machine_Learning`

2. [18 points] We will implement the logistic regression model with stochastic gradient descent. We will reuse 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. 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.
The objective is

$$\min_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{2v}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{m} \log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right)$$

and the derivative is

$$\nabla L(\mathbf{w}) = \frac{1}{v}\mathbf{w} - \sum_{i=1}^{m} y_i[1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)]\mathbf{x}_i \quad (67)$$

$$\nabla L^{\text{sgd}}(\mathbf{w}, \mathbf{x}_i, y_i) = \frac{1}{v}\mathbf{w} - m y_i[1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)]\mathbf{x}_i \quad (68)$$

We choose $\gamma_0 = 1, d = 2$ and Fig. 2. shows the convergence for various combination of variance and epochs. Training and testing errors for different variances are shown in Table 3.

variance	training error(%)	test error(%)
0.01	2.4	2.2
0.1	1.6	1.2
0.5	1.1	1.2
1	0.9	1.0
3	1.1	1.2
5	0.9	1.0
10	0.9	1.2
100	2.6	3.6

Table 3: MAP training and test errors.

(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.
Since MLE does not assume prior distribution of the weight, variance is not useful in this setting. Figure 3 shows evidence of convergence with $\gamma_0 = d = 2$. For $T = 100$, we obtain training error $= 3.6\%$ and test error $= 4.2\%$.

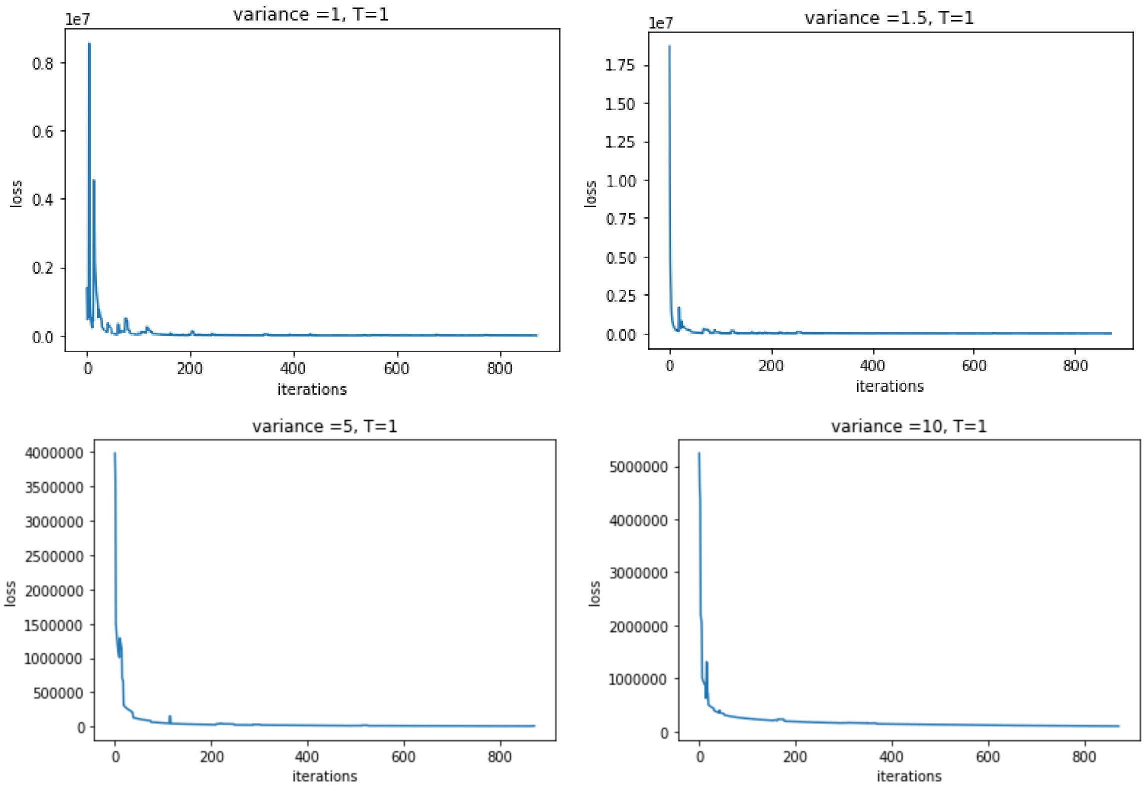

Figure 2: Evidence of convergence using $\gamma_0 = 1, d = 2$.

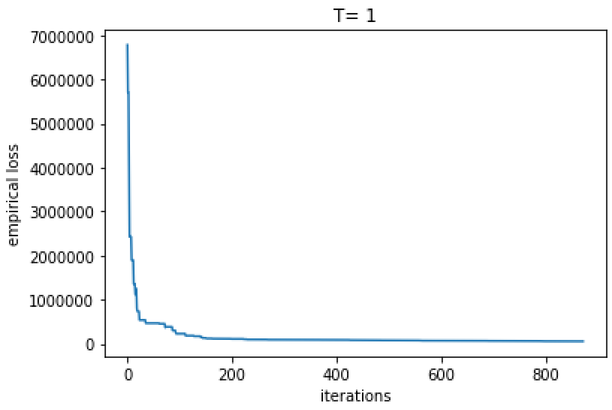

Figure 3: Evidence of convergence using $\gamma_0 = d = 2$.

(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?
From the result we can see that MLE has slightly larger errors (both training and test) than the MAP, which makes sense since prior knowledge of $\mathbf{w}$ distribution can only improve performance. The hyperparameter v implies a tradeoff between empirical loss and regularized term. More specifically, small v induces weights with smaller norms and large v induces larger weight norms to optimize the regularized term to push for better generalization performance.

3. [40 points] Now let us implement a three layer artificial neural network for classification. We will use the same dataset, "bank-note.zip". The architecture 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) [20 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.
   Denote $M \in \{5, 10, 25, 50, 100\}$ as the number of nodes in the hidden layers 1 and 2. The result of the code matches the derived partial derivatives of the weights for the paper problem. The partial derivative matrix are returned by the function 'NN.train()' with input argument $\mathbf{x} = [1, 1, 1]$ and is recorded in 'example-derivative.txt'.

   (b) [12 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. Report the training and test error for each setting of the width.
   The training and test errors and the tuned parameters are shown in Tbale 4. We noticed that the choice of γ_0 and d significantly affects the training and prediction accuracy.

width M	γ_0	d	training error(%)	test error(%)
5	1.5	2	1.3	1.2
10	0.5	2	0.6	0.8
25	0.01	2	1.1	0.8
50	0.02	1	1.6	2.6
100	0.03	2	1.7	1.4

Table 4: NN training and test errors with standard Gaussian weight initialization.

(c) [5 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? For the same set of combinations of γ_0 and d as in part (b), the training error is around 44.6% and test error is 44.2%. It seems that when initialized with all-zero weights, the neural network gets stuck and the updates is barely performed.

(d) [3 points]. As compared with the performance of the logistic regression and SVM, what do you conclude (empirically) about the neural network? For the same training and test dataset, it seems that neural network (NN) and logistic regression have similar error performance, with NN being slightly better. The SVM has significantly worse performance than NN and logistic, but kernel SVM can achieve near-perfect prediction on training and test datasets. Two things tend to be important for NN, 1) *Initialization*. From the experiment we see that improper initialization, like zero-weights, will lead to bad performance of NN; 2) *Stepsize scheduling*. The parameters γ_0, d also affects the performance. Moreover, the number of nodes in hidden layers also plays an important role. However, there is no general guideline for how to design those hyperparameters.

(e) [**Bonus**] [50 points] Please use tensor-flow (TF) 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", pelase use the "he" initialization. You can implement these initializations by yourselves or use TF 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. 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 document and web pages, find code snippets, and test and debug with TF to ensure the correct usage of TF. This is what all machine learning practitioners do in practice.
The file `'TensorFlow_NN_training.py'` (and `neural_network.ipynb`) implements the NN training using Keras. The training and test accuracy values are shown in Table 5 and Table 6 with training accuracy/test accuracy arrangement. The number of epochs is set to be 20 and all the hidden layers have the same number of nodes. At the output layer, we use `softmax` (or `sigmoid`) to determine the confidence of classification. It can be easily verified that all-zero initializer results in very low accuracy which is around 50% to 60%, which goes with our observation in part (c). From the result we can see that the expressiveness of NN is very powerful and achieves zero training and test errors for larger width of the hidden layers. For both settings of initializations, when the width is larger than 25, no matter how deep the NN is, the NN achieves perfect prediction on both training and test dataset.

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*

depth | width	5	10	25	50	100
3	0.9576/0.9600	0.9966/0.9960	1.00/1.00	1.00/1.00	1.00/1.00
5	0.9518/0.9500	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
9	0.9736/0.9760	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00

Table 5: Training accuracy with tanh activation and Xavier/glorot initializer.

depth | width	5	10	25	50	100
3	0.9587/0.9440	0.9977/0.9980	1.00/1.00	1.00/1.00	1.00/1.00
5	0.9839/0.9720	1.00/1.00	1.00/1.00	1.00/1.00	1.00/1.00
9	0.9759/0.9740	0.9908/0.9920	1.00/1.00	1.00/1.00	1.00/1.00

Table 6: Training accuracy with ReLU activation and 'He' initializer.

class! It is an excellent start of your journey to machine learning. Wish you further success in your future endeavours!
Check out `https://github.com/xiangzhang-122/Machine_Learning`