CS 5350/6350: Machine Learning Fall 2024

Homework 5

Handed out: 26 Nov, 2024
Due date: 11:59pm, 13 Dec, 2024

- 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 $x = 0$.

1 Paper Problems [40 points]

1.1 [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 $x = 0$.

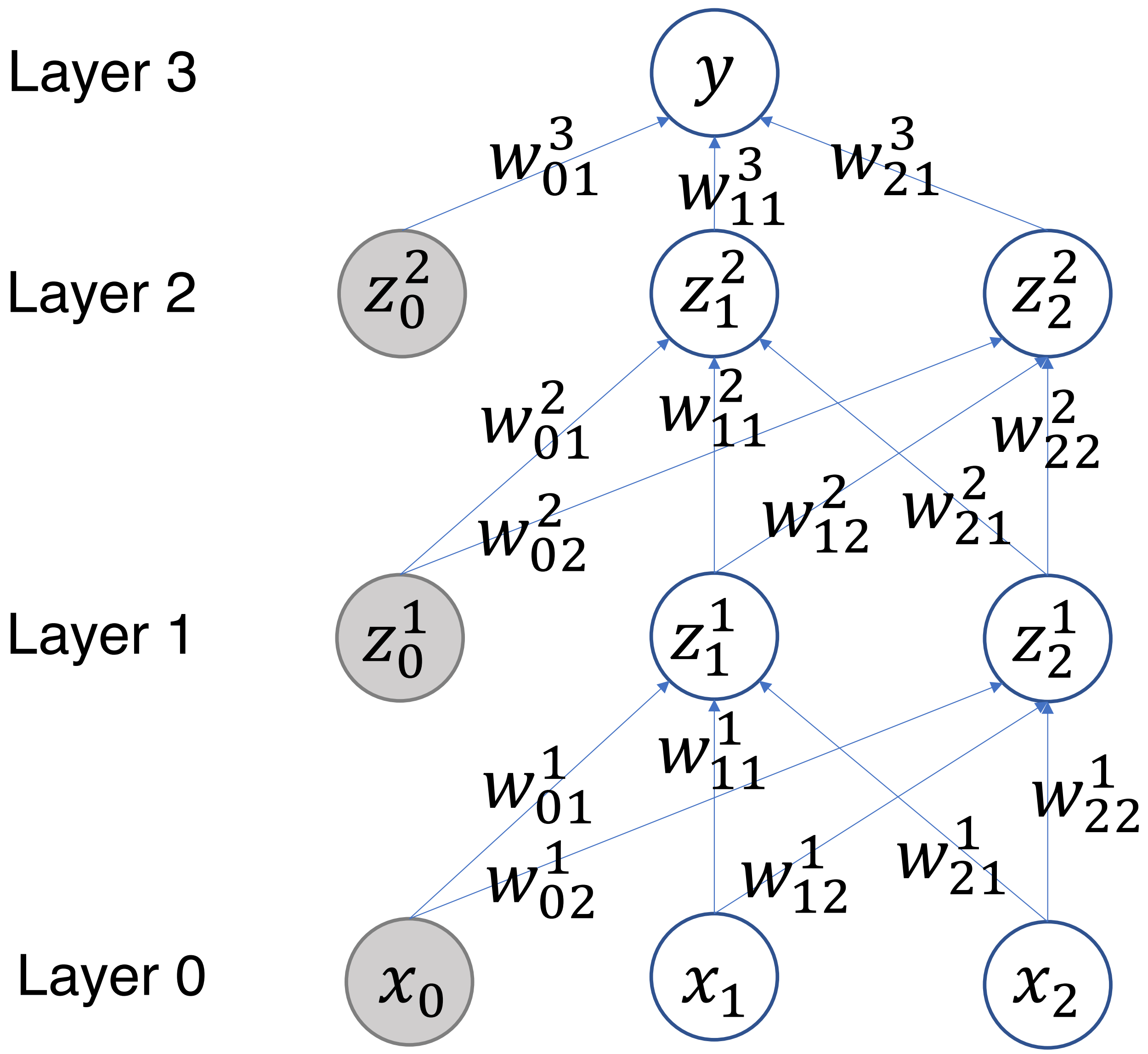

Figure 1: A three layer artificial neural network.

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.

Solution for 1.1 [5 points] Warm up

We are given the composite function:

$$z = \sigma(y_1^2 + y_2 y_3),$$

where:

$$y_1 = 3x, \quad y_2 = e^{-x}, \quad y_3 = \sin(x), \quad \text{and} \quad \sigma(t) = \frac{1}{1+e^{-t}}.$$

Step 1: Compute $\frac{\partial z}{\partial x}$ using the chain rule The derivative of z with respect to x can be written as:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \cdot \frac{\partial t}{\partial x},$$

where $t = y_1^2 + y_2 y_3$.

1. Compute $\frac{\partial z}{\partial t}$:

$$\frac{\partial z}{\partial t} = \sigma'(t) = \sigma(t)(1-\sigma(t)).$$

2. Compute $\frac{\partial t}{\partial x}$:

$$t = y_1^2 + y_2 y_3.$$

So,

$$\frac{\partial t}{\partial x} = \frac{\partial (y_1^2)}{\partial x} + \frac{\partial (y_2 y_3)}{\partial x}.$$

- For y_1^2:

$$\frac{\partial (y_1^2)}{\partial x} = 2y_1 \frac{\partial y_1}{\partial x} = 2y_1 \cdot 3 = 6y_1.$$

- For y_2y_3:

$$\frac{\partial(y_2y_3)}{\partial x} = \frac{\partial y_2}{\partial x} \cdot y_3 + y_2 \cdot \frac{\partial y_3}{\partial x}.$$

Where:

$$\frac{\partial y_2}{\partial x} = -e^{-x}, \quad \frac{\partial y_3}{\partial x} = \cos(x).$$

Thus:

$$\frac{\partial(y_2y_3)}{\partial x} = (-e^{-x}) \cdot y_3 + y_2 \cdot \cos(x).$$

Combining terms:

$$\frac{\partial t}{\partial x} = 6y_1 + (-e^{-x}) \cdot y_3 + y_2 \cdot \cos(x).$$

Step 2: Compute the derivative at $x = 0$

Substitute $x = 0$ into the equations:

$$y_1 = 3(0) = 0, \quad y_2 = e^0 = 1, \quad y_3 = \sin(0) = 0.$$

So:

$$t = y_1^2 + y_2y_3 = 0^2 + (1)(0) = 0.$$

The sigmoid function:

$$\sigma(0) = \frac{1}{1+e^0} = \frac{1}{2}.$$

And:

$$\sigma'(0) = \sigma(0)(1-\sigma(0)) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

For $\frac{\partial t}{\partial x}$ at $x = 0$:

$$\frac{\partial t}{\partial x} = 6(0) + (-e^0)(0) + (1)(\cos(0)) = 0 + 0 + 1 = 1.$$

Finally:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \cdot \frac{\partial t}{\partial x} = \frac{1}{4} \cdot 1 = \frac{1}{4}.$$

Final Answer:

$$\left.\frac{\partial z}{\partial x}\right|_{x=0} = \frac{1}{4}.$$

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 2. 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).

Solution: Given $x = [1, 1, 1]$ and biases $x_0 = z_0^1 = z_0^2 = 1$, the forward pass involves the following steps:

(a) **Compute activations at the first hidden layer:**

$$\begin{aligned}
z_1^1 &= \sigma(w_{10}^1 x_0 + w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3) \\
&= \sigma(1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + (-1) \cdot 1) \\
&= \sigma(1 + 1 + 0 - 1) = \sigma(1) = \frac{1}{1 + e^{-1}} \\
z_2^1 &= \sigma(w_{20}^1 x_0 + w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3) \\
&= \sigma(1 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + (-1) \cdot 1) \\
&= \sigma(1 + 0 + 1 - 1) = \sigma(1) = \frac{1}{1 + e^{-1}}.
\end{aligned}$$

(b) **Compute activations at the second hidden layer:**

$$\begin{aligned}
z_1^2 &= \sigma(w_{10}^2 z_0^1 + w_{11}^2 z_1^1 + w_{12}^2 z_2^1) \\
&= \sigma(1 \cdot 1 + (-1) \cdot \sigma(1) + 1 \cdot \sigma(1)) \\
&= \sigma(1 - \sigma(1) + \sigma(1)) \\
&= \sigma(1) = \frac{1}{1 + e^{-1}}.
\end{aligned}$$

(c) **Compute the output layer:**

$$\begin{aligned}
y &= \sigma(w_{10}^3 z_0^2 + w_{11}^3 z_1^2) \\
&= \sigma(1 \cdot 1 + (-1) \cdot \sigma(1)) \\
&= \sigma(1 - \sigma(1)) \\
&= \frac{1}{1 + e^{-(1-\sigma(1))}}.
\end{aligned}$$

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 2, 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.

Logistic Regression

4. [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 2, please use the backpropagation (BP) algorithm to compute the derivative of the loss L over all the weights, $\left\{\frac{\partial L}{\partial w_{ij}^m}\right\}$. 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.

Layer	Weights	Values
Input Layer to Hidden Layer 1	w_1, w_2, w_3	$w^{(1)} = [w_1, w_2, w_3]$
Hidden Layer 1 to Hidden Layer 2	w_4, w_5, w_6	$w^{(2)} = [w_4, w_5, w_6]$
Hidden Layer 2 to Output Layer	w_7, w_8, w_9	$w^{(3)} = [w_7, w_8, w_9]$

Table 2: Weights for the 3-layer neural network

The backpropagation steps are as follows:

Step 1: Forward Pass

1. Calculate the activation of the hidden layers using the sigmoid activation function:

$$a_i^{(1)} = \sigma(w_i^{(1)} \cdot \mathbf{x}), \quad \text{for each node in hidden layer 1}$$

$$a_i^{(2)} = \sigma(w_i^{(2)} \cdot a^{(1)}), \quad \text{for each node in hidden layer 2}$$

$$y = \sigma(w_i^{(3)} \cdot a^{(2)}), \quad \text{for output layer}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid activation function.

Step 2: Compute Loss

The loss function is given as:

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

where $y^* = 1$ is the true label.

Step 3: Backpropagation of Errors

1. Compute the error in the output layer:

$$\delta^{(3)} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z^{(3)}} = (y - y^*) \cdot \sigma'(z^{(3)})$$

where $z^{(3)}$ is the input to the output node, and $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ is the derivative of the sigmoid function.

2. Backpropagate the error to the second hidden layer:

$$\delta_i^{(2)} = \sum_j w_{ij}^{(3)} \delta_j^{(3)} \cdot \sigma'(z_i^{(2)})$$

3. Backpropagate the error to the first hidden layer:

$$\delta_i^{(1)} = \sum_j w_{ij}^{(2)} \delta_j^{(2)} \cdot \sigma'(z_i^{(1)})$$

Step 4: Compute Gradients

1. Compute the gradients of the weights in the output layer:

$$\frac{\partial L}{\partial w_i^{(3)}} = \delta_i^{(3)} \cdot a_i^{(2)}$$

2. Compute the gradients of the weights in the second hidden layer:

$$\frac{\partial L}{\partial w_i^{(2)}} = \delta_i^{(2)} \cdot a_i^{(1)}$$

3. Compute the gradients of the weights in the first hidden layer:

$$\frac{\partial L}{\partial w_i^{(1)}} = \delta_i^{(1)} \cdot \mathbf{x}_i$$

Step 5: Update Weights

Update the weights using gradient descent:

$$w_i^{(m)} \leftarrow w_i^{(m)} - \eta \frac{\partial L}{\partial w_i^{(m)}}$$

where η is the learning rate.

5. [10 points] Suppose we have the training dataset shown in Table 4. 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.
- [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.

article amsmath graphicx

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 3: Dataset

Logistic Regression with MAP Estimation

Objective and Gradient Calculation

6. [10 points] Suppose we have the training dataset shown in Table 4. 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,

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

The goal is to obtain the maximum a posteriori (MAP) estimation.

- [7 points] We want to obtain the MAP estimation. Please write down the objective function, namely, the log joint probability, and derive the gradient of the objective function.

  The objective function for MAP estimation combines the likelihood and the prior. Let $\mathbf{X}$ be the input features matrix and $\mathbf{y}$ the vector of outputs. The logistic regression model is:

$$p(y_i = 1|\mathbf{x}_i; \mathbf{w}) = \sigma(\mathbf{x}_i^T\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x}_i^T\mathbf{w}}}$$

  where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The likelihood is:

$$p(\mathbf{y}|\mathbf{X}; \mathbf{w}) = \prod_{i=1}^{N} \sigma(\mathbf{x}_i^T\mathbf{w})^{y_i} \left(1 - \sigma(\mathbf{x}_i^T\mathbf{w})\right)^{1-y_i}.$$

  The prior on the weights is given by a Gaussian distribution:

$$p(\mathbf{w}) = \prod_{i=0}^{3} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}w_i^2\right).$$

  The log joint probability (log of the posterior) is the sum of the log likelihood and the log prior:

$$\log p(\mathbf{y}, \mathbf{w}|\mathbf{X}) = \log p(\mathbf{y}|\mathbf{X}; \mathbf{w}) + \log p(\mathbf{w}).$$

The log-likelihood is:

$$\log p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \sum_{i=1}^{N} \left[y_i \log \sigma(\mathbf{x}_i^T \mathbf{w}) + (1-y_i) \log \left(1 - \sigma(\mathbf{x}_i^T \mathbf{w})\right) \right].$$

The log-prior is:

$$\log p(\mathbf{w}) = -\frac{1}{2} \sum_{i=0}^{3} w_i^2.$$

Therefore, the log joint probability is:

$$\log p(\mathbf{y}, \mathbf{w}|\mathbf{X}) = \sum_{i=1}^{N} \left[y_i \log \sigma(\mathbf{x}_i^T \mathbf{w}) + (1-y_i) \log \left(1 - \sigma(\mathbf{x}_i^T \mathbf{w})\right) \right] - \frac{1}{2} \sum_{i=0}^{3} w_i^2.$$

- [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.

  The gradient of the log joint probability with respect to the model parameters w_i is given by:

$$\frac{\partial}{\partial w_i} \log p(\mathbf{y}, \mathbf{w}|\mathbf{X}) = \sum_{i=1}^{N} \left[y_i \frac{1}{\sigma(\mathbf{x}_i^T \mathbf{w})} - (1-y_i) \frac{1}{1-\sigma(\mathbf{x}_i^T \mathbf{w})} \right] x_{i,j} - w_i.$$

  The gradient for each model parameter w_i can be computed by iterating over the training examples and updating weights using the stochastic gradient descent (SGD) rule:

$$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} \log p(\mathbf{y}, \mathbf{w}|\mathbf{X}),$$

  where η is the learning rate.

  For the first three steps, we will use the following dataset from Table 4:

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

  Now we calculate the stochastic gradients for each parameter in the first three steps:

  Step 1: Update parameters using the first training example $\mathbf{x}_1 = [0.5, -1, 0.3]$ and $y_1 = 1$.

  Step 2: Update parameters using the second training example $\mathbf{x}_2 = [-1, -2, -2]$ and $y_2 = -1$.

  Step 3: Update parameters using the third training example $\mathbf{x}_3 = [1.5, 0.2, -2.5]$ and $y_3 = 1$.

  After calculating the gradients for each step, the parameters w_0, w_1, w_2, w_3 will be updated according to the SGD rule using the corresponding learning rate from the set $\{0.01, 0.005, 0.0025\}$.

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 4: Dataset

2 Practice [62 points + 60 bonus]

(a) [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.*

(b) [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.

i. [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.

ii. [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.

iii. [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?

iv. [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?

v. [**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.

(c) [**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.

i. [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.

ii. [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.

iii. [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?

(d) [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!*