Assignment #3

Due: 7:59pm EST, March 5, 2021

Homework 3: Bayesian Methods and Neural Networks

Introduction

This homework is about Bayesian methods and Neural Networks. Section 2.9 in the textbook as well as reviewing MLE and MAP will be useful for Q1. Chapter 4 in the textbook will be useful for Q2.

Please type your solutions after the corresponding problems using this LaTeX template, and start each problem on a new page.

Please submit the **writeup PDF to the Gradescope assignment 'HW3'**. Remember to assign pages for each question. **All plots you submit must be included in your writeup PDF.** We will not be checking your code / source files except in special circumstances.

Please submit your **LaTeXfile and code files to the Gradescope assignment 'HW3 - Supplemental'**.

Problem 1 (Bayesian Methods)

This question helps to build your understanding of making predictions with a maximum-likelihood estimation (MLE), a maximum a posterior estimator (MAP), and a full posterior predictive.

Consider a scalar variable x with the following generative process: First, the mean μ is sampled from a prior $N(0, \tau^2)$. Next, each x_n is generated as $x_n = \mu + \epsilon_n$, where $\epsilon_n \sim N(0, \sigma^2)$. All ϵ_n's are independent of each other and of μ.

For this problem, use $\sigma^2 = 1$ and $\tau^2 = 5$.

Now, we see 14 independent samples of x to yield data

$$D = 3.3, 3.5, 3.1, 1.8, 3.0, 0.74, 2.5, 2.4, 1.6, 2.1, 2.4, 1.3, 1.7, 0.19$$

Make sure to include all required plots in your PDF.

1. Derive the expression for $p(\mu|D)$. Do *not* plug in numbers yet!

   Hint: Use properties of normal-normal conjugacy to simplify your derivation. You can also refer to this paper.

2. Now we get to our core interest: the predictive distribution of a new datapoint x^* given our observed data D, $p(x^*|D)$. Write down the expression for the full posterior predictive distribution:

   $$p(x^*|D) = \int p(x^*|\mu)p(\mu|D)d\mu$$

   Interpret your expression in a few words. Do *not* plug in numbers yet!

   Hint: To simplify your derivation, use the fact that $x = \mu + \epsilon$, and $\mu|D$ and ϵ are independent Gaussians whose distributions you know from above.

3. The full posterior predictive distribution had a nice analytic form in this case, but in many problems, it is often difficult to calculate because we need to marginalize out the parameters (here, the parameter is μ). We can mitigate this problem by plugging in a point estimate of μ^* rather than a distribution. Derive the estimates of $p(x^*|D) \approx p(x^*|\mu^*)$ for $\mu^* = \mu_{MLE}$ and $\mu^* = \mu_{MAP}$. How do these expressions compare to the expression for the full posterior above? Do *not* plug in numbers yet!

4. Plot how the above 3 distributions change after each data point is gathered. You will have a total of 15 plots, starting with the plot for the case with no data (they can be small, e.g. in a 3×5 grid). The x-axis of each plot will be the x value and the y-axis the density. You can make one plot for each estimator, or combine all three estimators onto one plot with a different colored line for each estimator.

5. How do the means of the predictive distributions vary with more data? How do the variances vary? Interpret the differences you see between the three different estimators.

6. Does the ordering of the data matter for the final predictive distributions?

7. Derive an expression for and then compute the marginal likelihood of the training data $p(D)$.

   Hint: You can rearrange the required integral such that it looks like an un-normalized Gaussian distribution in terms of μ, and take advantage of the fact that integrating over a normalized Gaussian distribution is equal to 1. You will need to complete the square.

8. Now consider an alternate model in which we were much more sure about the mean: $\mu \sim N(0, \tau^2)$, where $\tau^2 = 0.1$. Compute the marginal likelihood $p(D)$ for this model. Which of the two models has a higher marginal likelihood? Interpret this result.

Solution:

1. Here for D we have

$$p(D|\mu,\sigma^2) \sim N(\mu,\sigma^2).$$

$$\begin{aligned} p(\mu|D) &\propto p(D|\mu,\sigma)p(\mu|0,\tau^2) \\ &\propto \exp(-\frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2)\exp(-\frac{(\mu)^2}{2\tau^2}) \end{aligned}$$

Here we implicitly used the fact that $p(D|0,\tau^2)$ does not depend on μ. By assumption $p(\mu|D)$ also follows Gaussian distribution, we define it as $p(\mu|D) \approx N(\mu_n,\sigma_n)$. We can then get the parameters by comparing the Gaussian distribution with the equation above.

$$\begin{aligned} \mu_n &= \sigma_n(\frac{\sum_{i=1}^n x_i}{\sigma^2}), \\ \sigma_n^2 &= \frac{1}{\frac{n}{\sigma^2}+\frac{1}{\tau^2}}. \end{aligned}$$

2. We have

$$x^* \sim N(\mu,\theta^2).$$

$$\begin{aligned} p(x^*|D) &= \int p(x^*|\mu)p(\mu|D)d\mu \\ &= \int N(x^*|\mu,\sigma^2)N(\mu|\mu_n,\sigma_n^2)d\mu \\ &= N(x^*|\mu_n,\sigma_n^2+\sigma^2) \end{aligned}$$

It means the predicted x^* is expected to be a Gaussian distribution, with the mean value as μ_n, and standard deviation $\sigma_n^2+\sigma^2$. It can be understood as the distribution of μ, which is $N(\mu_n,\sigma_n^2)$ plus the distribution of the noise, which is $N(0,\sigma^2)$. The result is intuitive.

3. From what we learned in the class, the maximum likelihood estimation maximizes the distribution without the knowledge of prior, and we are familiar with the following equation

$$\mu_{MLE}^* = \bar{x} = \sum_i x/n$$

$$p(x^*|\mu_{MLE}^*) = N(x^*|\sum_i x/n,\sigma^2)$$

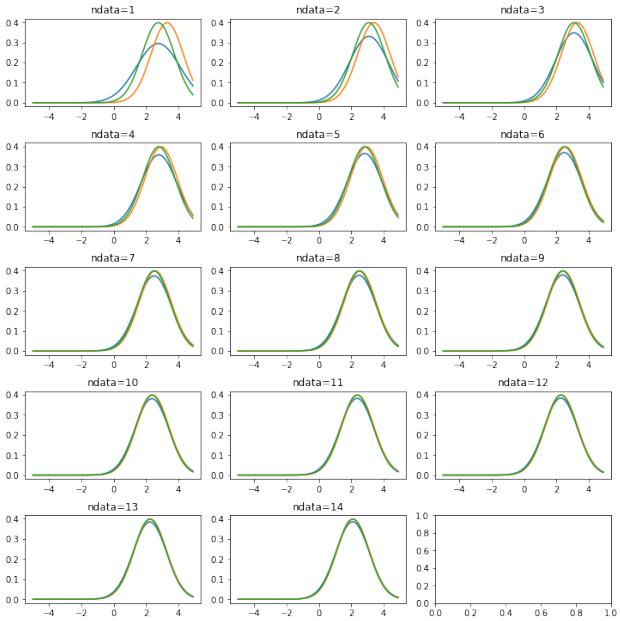
On the other hand, for MAP, we seek to maximize $p(\mu|D)$. Thus

$$\mu_{MAP}^* = \mu_n$$

$$p(x^*|\mu_{MAP}^*) = N(x^*|\mu_n,\sigma^2)$$

Compared to the result for the full posterior above, all three distributions are Gaussian distributions, however both $p(x^*|\mu_{MLE}^*)$ and $p(x^*|\mu_{MAP}^*)$ has a smaller standard deviation compared to the full posterior. That small standard deviation is caused by using the point estimate instead of full distribution, so the influence of the standard deviation of μ is ignored. For the mean value, we see the MAP result equals to the full posterior, however the mean value in MLE is larger than the full posterior. That is because the MLE ignored the information of the prior, i.e. the knowledge on distribution of μ.

4. The figure is shown here. The blue line is the full posterior, orange line the MLE, and green line the MAP.



5. For the mean, the MLE does not change, but the means for the other two measures grow larger as the number of data grow. For the variance, the MLE and the MAP does not change, but the full posterior decrease as number of data increase. What's more, as the size of data increase, finally the three measures get very close results.

6. No. We can see the answer from the equations in question 1.2 and 1.3. The result does not depend on the order.

7. $$p(D|\sigma^2,\tau^2) = \int (\prod_i N(x_i|\mu,\sigma^2))N(\mu|\tau^2)d\mu$$
$$= \frac{1}{(\sigma\sqrt{2\pi})^n(\tau\sqrt{2\pi})}\int \exp(-\frac{1}{2\sigma^2}\sum_i(x_i-\mu)^2 - \frac{1}{2\tau^2}\mu^2)d\mu$$

   Following the hint, we can calculate the integration for μ, making use of the property of the Gaussian distribution. The final expression is

$$p(D) = \frac{\sigma}{(\sqrt{2\pi}\sigma)^n\sqrt{n\tau^2+\sigma^2}}\exp(-\frac{\Sigma_i x_i^2}{2\sigma^2})\exp(\frac{\frac{\tau^2(\sum_i x)^2}{\sigma^2}}{2(n\tau^2+\sigma^2)})$$

   The value is 4.46×10^{-10}.

8. The value is 8.00×10^{-15}. We see the first model with larger τ^2 has a higher likelihood. Interpretation: Without any knowledge on the data we will get, we would expect data to follow the equation in the last question. We can see that when the τ^2 term is not too large, larger τ^2 will lead to larger possibility. That is intuitive because if the μ has a smaller variance, we would expect the final distribution of the data to be distributed symmetrically around zero, and vice versa. The current data are all positive, and a larger variance of μ will be give more possibility to allow that situation.

Problem 2 (Neural Net Optimization)

In this problem, we will take a closer look at how gradients are calculated for backprop with a simple multi-layer perceptron (MLP). The MLP will consist of a first fully connected layer with a sigmoid activation, followed by a one-dimensional, second fully connected layer with a sigmoid activation to get a prediction for a binary classification problem. Assume bias has not been merged. Let:

- $\mathbf{W}_1$ be the weights of the first layer, $\mathbf{b}_1$ be the bias of the first layer.
- $\mathbf{W}_2$ be the weights of the second layer, $\mathbf{b}_2$ be the bias of the second layer.

The described architecture can be written mathematically as:

$$\hat{y} = \sigma(\mathbf{W}_2 \left[\sigma \left(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1\right)\right] + \mathbf{b}_2)$$

where $\hat{y}$ is a scalar output of the net when passing in the single datapoint $\mathbf{x}$ (represented as a column vector), the additions are element-wise additions, and the sigmoid is an element-wise sigmoid.

1. Let:
   - N be the number of datapoints we have
   - M be the dimensionality of the data
   - H be the size of the hidden dimension of the first layer. Here, hidden dimension is used to describe the dimension of the resulting value after going through the layer. Based on the problem description, the hidden dimension of the second layer is 1.

   Write out the dimensionality of each of the parameters, and of the intermediate variables:

   $$\mathbf{a}_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1, \qquad \mathbf{z}_1 = \sigma(\mathbf{a}_1)$$
   $$a_2 = \mathbf{W}_2 \mathbf{z}_1 + \mathbf{b}_2, \qquad \hat{y} = z_2 = \sigma(a_2)$$

   and make sure they work with the mathematical operations described above.

2. We will derive the gradients for each of the parameters. The gradients can be used in gradient descent to find weights that improve our model's performance. For this question, assume there is only one datapoint $\mathbf{x}$, and that our loss is $L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$. For all questions, the chain rule will be useful.
   (a) Find $\frac{\partial L}{\partial b_2}$.
   (b) Find $\frac{\partial L}{\partial W_2^h}$, where W_2^h represents the hth element of $\mathbf{W}_2$.
   (c) Find $\frac{\partial L}{\partial b_1^h}$, where b_1^h represents the hth element of $\mathbf{b}_1$. (*Hint: Note that only the hth element of $\mathbf{a}_1$ and $\mathbf{z}_1$ depend on b_1^h - this should help you with how to use the chain rule.)
   (d) Find $\frac{\partial L}{\partial W_1^{h,m}}$, where $W_1^{h,m}$ represents the element in row h, column m in $\mathbf{W}_1$.

Solution:

1. The dimensions are shown below. Here the number in the parenthesis shows the dimensions when we only put in one data point at a time.
   x:MxN(1)
   W1:HxM
   $b1$:HxN(1)
   $a1$:HxN(1)
   z1:HxN(1)
   W2:1xH
   $b2$:1xN(1)
   a2:1xN(1)
   $\hat{y} = z_2$:1xN(1).

2. (a)
$$\begin{aligned}\frac{\partial L}{\partial b_2} &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial b_2} \\ &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\end{aligned}$$

   (b)
$$\begin{aligned}\frac{\partial L}{\partial W_2^h} &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial w_2^h} \\ &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2} z_1^h\end{aligned}$$

   (c)
$$\begin{aligned}\frac{\partial L}{\partial b_1^n} &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial \sigma(a_1^h)}\frac{\partial \sigma(a_1^h)}{\partial a_1^h}\frac{\partial a_1^h}{\partial b_1^h} \\ &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial \sigma(a_1^h)}\frac{\partial \sigma(a_1^h)}{\partial a_1^h}\end{aligned}$$

   (d)
$$\begin{aligned}\frac{\partial L}{\partial W_1^{h,m}} &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial \sigma(a_1^h)}\frac{\partial \sigma(a_1^h)}{\partial a_1^h} \\ &= -(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}})\frac{\partial \sigma(a_2)}{\partial a_2}\frac{\partial a_2}{\partial \sigma(a_1^h)}\frac{\partial \sigma(a_1^h)}{\partial a_1^h}\frac{\partial a_1^h}{\partial W_1^{h,m}}\end{aligned}$$

Problem 3 (Modern Deep Learning Tools: PyTorch)

As you might imagine from the previous problem, actually implementing backprop by hand can be frustrating! Fortunately, having modern automatic differentiation tools means that you will rarely have to do so. In this problem, you will learn how to use PyTorch (the ML library of choice in industry and academia) and implement your own neural networks for image classification.

To begin the assignment, **upload T3_P3.ipynb to Google Colab**. Write and run your code in Colab, and download your Colab notebook as an .ipynb file to submit as a supplemental file. Include your written responses, plots, and required code (as specified) in this LaTeX file.

If you have never used Google Colab before, see the 'HW 3' Addendum Post on Ed, where the staff have compiled resources to help you get started.

You can use the Listings package to display code, as shown below:

```
example_array = np.array([1, 2, 3])

```

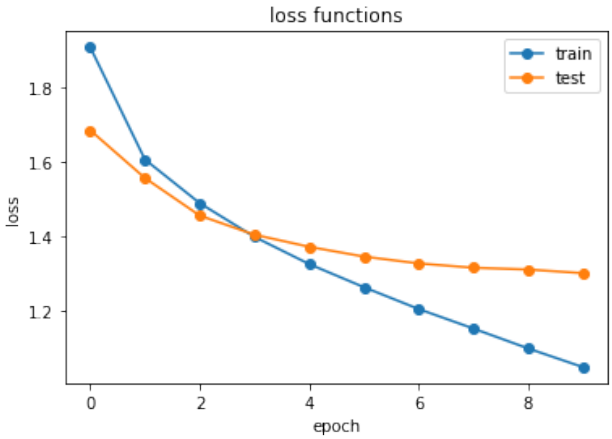
1. Please answer Problem 3.1 from the Colab here.
2. Please answer Problem 3.2 from the Colab here.
3. Please answer Problem 3.3 from the Colab here.
4. Please answer Problem 3.4 from the Colab here.
5. Please answer Problem 3.5 from the Colab here.
6. Please answer Problem 3.6 from the Colab here.
7. Please answer Problem 3.7 from the Colab here.

Solution:

1. The gradient is stored in a computation graph which logs every operation used to construct a certain output or mathematical expression. The gradient can be calculated (at least) through backward propagation. It can be accessed through .grad field in the model's model.parameters() field.
2. 3 weights, and 1 bias term. From the linear layer.
3. Fully connected layer is a layer where all the input from one layer are connected to every activation unit of the next layer. In pyTorch, it can be defined as a function inside torch.nn.Sequential() function. (e.g. torch.nn.Linear(3,1)) is a layer.
4. Code is shown below.

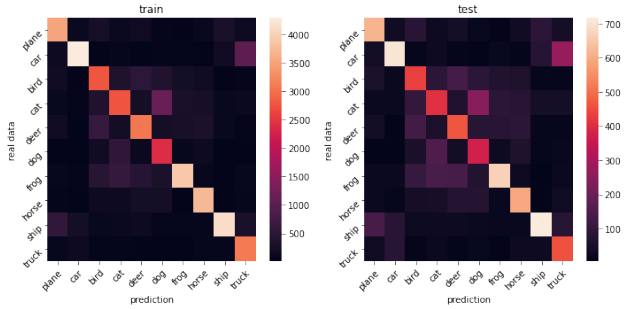
```
    class Part2NeuralNetwork(nn.Module):
    def __init__(self):
        super(Part2NeuralNetwork, self).__init__()
        ## TODO: Define your neural network layers here!
        ## Importantly, any modules which initialize weights should be initialized
        ## as member variables here.
        ## Note: Keep track of the shape of your input tensors in the training
        ## and test sets, because it affects how you define your layers!
        ## You might find this resource helpful:
        ## https://towardsdatascience.com/pytorch-layer-dimensions-what-sizes-should-
    they-be-and-why-4265a41e01fd
        datasize=32*3*32
        #datasize=1
        self.layer1=nn.Linear(datasize,1000)
        self.layer2=nn.Linear(1000,1000)
        self.layer3=nn.Linear(1000,10)

    def forward(self, x):
        ## TODO: This is where you should apply the layers defined in the __init__
        ## method and the ReLU activation functions to the input x.
        x=x.view(x.size(0),-1)
        x=F.relu(self.layer1(x))
        x=F.relu(self.layer2(x))
        x=F.relu(self.layer3(x))
        return x

```

5. The figure is shown here.



6. Accuracy: train:67%, test: 54%.
   Precision:

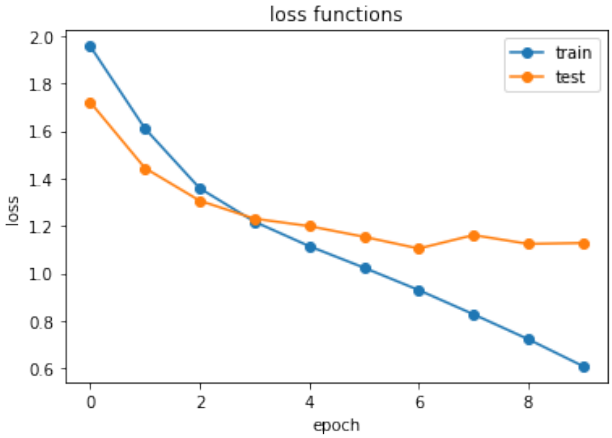
train, from class 1 to 10: 70.9%, 82.3%, 60.4%, 50.0% 59.2%, 55.2%, 66.8%, 81.3%, 76.9%, 77.4%.
test, from class 1 to 10: 60.5%, 68.5%, 46.6%, 35.8%, 46.0% 42.7%,54.5%, 67.6%, 64.8%, 60.7%.
Recall:
train, from class 1 to 10: 72.5%, 78.3%, 49.7%, 52.2%, 57.6%, 62.9%, 76.2%, 71.1%, 81.3%, 75.4%.
test, from class 1 to 10: 62.3%, 61.9%, 38.2%, 37.7%, 43.2%, 50.0%, 64.8%, 57.3%, 69.5%, 59.1%.
Type of errors: Here I creat a heatmap to show how data in different classes are classified. We can see the majority of data are classified correctly, and the wrong predictions goes relatively evenly to different classes. In other words, there is not very strong tendency that a certain class would be classifed as another. However, we can find some common mistakes, like cars tend to be classified as truck, bird tend to be classified as cat or deer, cat tend to be classified as dog, dear tend to be classified as bird, dog tend to be classified as cat, frog tend to be classified as bird, cat and dear, horse tend to be classified as dog, ship tend to be classified as plane or car, and truck tend to be classified as car. More generally, the network sometimes confuses different animals, or different vehicles. The tendency is more obvious for testing data.



7. I added an extra convolution layer and here is the code.

```
    class Part2NeuralNetwork_conv(nn.Module):
    def __init__(self):
        super(Part2NeuralNetwork_conv, self).__init__()
        ## TODO: Define your neural network layers here!
        ## Importantly, any modules which initialize weights should be initialized
        ## as member variables here.
        ## Note: Keep track of the shape of your input tensors in the training
        ## and test sets, because it affects how you define your layers!
        ## You might find this resource helpful:
        ## https://towardsdatascience.com/pytorch-layer-dimensions-what-sizes-should-
    they-be-and-why-4265a41e01fd
        self.layer0=nn.Conv2d(3,12,3)
        self.layer1=nn.Linear(12*30*30,1000)
        self.layer2=nn.Linear(1000,1000)
        self.layer3=nn.Linear(1000,10)

    def forward(self, x):
        ## TODO: This is where you should apply the layers defined in the __init__
        ## method and the ReLU activation functions to the input x.
        x=F.relu(self.layer0(x))
        #print(x.shape)
        x=x.view(x.size(0),-1)
        x=F.relu(self.layer1(x))
        x=F.relu(self.layer2(x))
        x=F.relu(self.layer3(x))
        return x

```

The loss function is shown below.



Accuracy: train:86%,test: 62%.
Precision:
train, from class 1 to 10: 83.7%, 94.5%, 78.2%, 79.6%, 80.2%, 83.7%, 85.1%, 91.3%, 94.7%, 93.4%.
test, from class 1 to 10: 64.6%, 71.9%, 50.9%, 45.8%, 53.4% 56,4%,63.4%, 67.5%, 77.5%, 69.8%.
Recall:
train, from class 1 to 10: 90.8%, 95.2%, 79.5%, 75.1%,83.5%, 77.1%, 92.4%, 91.6%, 87.9%, 91.3%.
test, from class 1 to 10: 74.8%, 72.7%, 50.7%, 41.5%, 53.0% 51.3%, 75.4%, 68.4%, 68.7%, 65.5%.
I expect this new architecture to perform better, because it has one extra convolution layer compared to the original one. The convolution layer is a filter which make the network more capable of learning certain patterns contained in the figure.

Name

Boer Zhang

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?
I checked my answer with Hanwen Zhang. I did not use extra resources.

Calibration

Approximately how long did this homework take you to complete (in hours)?
16 hours. (But 3 days in total without working on anything else)