Homework 3: Bayesian Methods and Neural Networks

Introduction

This homework is about Bayesian methods and neural networks.

1. You'll explore the Bayesian paradigm and compare it with the frequentist paradigm for the Beta-Binomial conjugate pair.
2. You'll derive the backpropagation algorithm for a single-hidden-layer neural network for the binary classification task.
3. You'll write some code using the PyTorch library for an image classification task.
4. You'll consider the opportunities and limitations of ML applications and learn to anticipate possible exploits of these systems.

As always, please start early and come to office hours with questions!

Resources and Submission Instructions

You may want to consider the lecture notes from Feb 13th to 22nd (weeks 4 and 5). Here's an outline of the questions:

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. **You must include your plots in your writeup PDF.** The supplemental files will only be checked in special cases, e.g. honor code issues, etc.

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

Problem 1 (Connecting Bayesian and Frequentist Approaches, 40 pts)

In this question, we will gain practice with Bayesian modeling and compare it with the frequentist paradigm.

In class, we discussed *Normal-Normal conjugacy.* Now we will turn to *Beta-Binomial conjugacy.* This model can be visualized in the following way.

You observe a fixed number N of coin flips (either heads or tails) of which Y (a random variable) are heads. You assume that these are drawn by flipping a coin with an unknown probability θ of landing heads. That is, we choose a **Binomial likelihood** $Y \sim \text{Bin}(N, \theta)$. The PMF of this distribution is given by

$$p(Y = y) = \binom{N}{y} \theta^y (1-\theta)^{N-y}.$$

1. **Frequentist paradigm and MLE.** The (log) likelihood is all we need for frequentist inference. Derive the MLE estimate for θ given the observations $Y = y$. That is, find

$$\arg\max_\theta \log p(Y = y \mid \theta).$$

2. **Beta-Binomial conjugacy.** Under the Bayesian paradigm, we must specify a prior distribution for the unknown parameter θ. We choose a **Beta prior** $\theta \sim \text{Beta}(\alpha, \beta)$. The PDF of this distribution is given by

$$p(\theta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}.$$

   When the prior and posterior belong to the same distribution family, we call the prior-and-likelihood pair a **conjugate pair.**

   For the rest of the problem, feel free to cite that the mean of the $\text{Beta}(\alpha, \beta)$ distribution is

$$\mathbb{E}[\theta] = \frac{\alpha}{\alpha + \beta},$$

   the mode is

$$\arg\max_\theta p(\theta) = \frac{\alpha - 1}{\alpha + \beta - 2},$$

   and the variance is

$$\text{Var}(\theta) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.$$

   (a) Qualitatively speaking, what does the $\text{Beta}(\alpha, \beta)$ distribution look like for different α and β? You can either plot this yourself or see its Wikipedia page after deriving the statistics above. What distribution does $\text{Beta}(1, 1)$ correspond to?

   (b) Show that the posterior $p(\theta \mid Y = y)$ is indeed Beta and derive its parameters. This proves that a Beta prior and a Binomial likelihood form a conjugate pair; in other words, the Beta distribution is a **conjugate prior** for the Binomial distribution.

   Hint (for convenience in calculation): Do you need to calculate the normalizing constant?

3. **Posterior mean and mode.** Often we wish to work with just a single point estimate of the posterior. Two commonly used point estimates are the *posterior mean* and the *posterior mode* (a.k.a. the maximum a posteriori (MAP) estimate).

   (a) Discuss the advantages and disadvantages of using posterior point estimates. Which of these are relevant for our Beta-Binomial conjugate pair? Consider the case when $\alpha, \beta < 1$.

   (b) Using your results from part 2, write down

      i. the posterior mean estimate $\theta_{\text{post mean}} = \mathbb{E}[\theta \mid Y = y]$,

      ii. the posterior MAP estimate $\theta_{\text{MAP}} = \arg\max_\theta p(\theta \mid Y = y)$,

      iii. and the posterior variance $\text{Var}(\theta \mid Y = y) = \mathbb{E}[\theta^2 \mid Y = y] - (\mathbb{E}[\theta \mid Y = y])^2$.

      Hint: Do you need to rederive anything? (This exercise should illustrate the power of conjugate priors!)

4. **Prior-posterior connections.**

   (a) Explain in your own words how α and β affect the MAP estimate. How would you set α and β to reflect a prior belief that the coin is fair (i.e. shows heads and tails with equal probability)? (Be careful that your answer does *not* give high probability to an "always heads" coin or "always tails" coin!)

   (b) Now let's analyze the variances of our prior and posterior distributions. Consider the case when $\alpha = \beta$. (If you'd enjoy it, consider the general case for better understanding.) Please write at most two sentences per point.

      i. How does the variance of the prior relate to the variance of the posterior?

      ii. How might you use the prior variance to encode a stronger or weaker prior belief?

      iii. How does the posterior variance change as we observe more samples n?

5. **Analysis and connection to frequentism.**

   (a) Write a loss function $\ell(\theta) \in \mathbb{R}$ in terms of $\theta, y, n, \alpha, \beta$ such that minimizing ℓ is equivalent to calculating the MAP estimate, i.e. $\theta_{\text{MAP}} = \arg\min_\theta \ell(\theta)$. Your function should be a sum of:

      i. a mean-squared-error term (which should loosely resemble $(y - \hat{y})^2$)

      ii. a regularization term $g(\theta) = -a\theta + b\theta^2$ for some a, b.

      Can you interpret the regularization term?

      Hint: Work backwards from part 1 to derive the MSE term and from the MAP in part 2 to get the regularization term. Watch out for the signs! For the interpretation, complete the square and then compare your expression with the prior mode you found in 2.a.ii.

   (b) What happens to both $\theta_{\text{post mean}}$ and θ_{MAP} as $n \to \infty$? Compare this to the MLE estimate. (Remember to account for the change in y.)

Solution

1. Answer:

$$\arg\max_\theta \log p(Y = y \mid \theta) =$$

Derivation:

2. (a)

   (b) Parameters:

$$\alpha' =$$

$$\beta' =$$

   Derivation:

3. (a)

   (b)

$$\theta_{\text{post mean}} = \mathbb{E}[\theta \mid Y = y] =$$
$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid Y = y) =$$
$$\text{Var}(\theta \mid Y = y) =$$

4. (a)

   (b) i.

   ii.

   iii.

5. (a) MSE term:

$$M =$$

   Regularization term:

$$R =$$

   Loss function, as a function of the MSE and regularization term:

$$\ell(\theta) =$$

   MSE term derivation:

   Regularization term derivation:

   Regularization term interpretation:

   (b)

Problem 2 (Neural Networks, 20 pts)

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. We use non-linear activation functions as the composition of linear functions is linear. 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 should be 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. Examining shapes is one of the key ways to debug your code, and can be done using .shape after any numpy array.

2. We will derive the gradients for each of the parameters, which can then be used along with 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 $\mathbf{W}_1$.

Problem 2 (cont.)

3. We now explore an example of forward-mode auto-differentiation. Consider the following equation:

$$f(x_1, x_2) = \ln(\sin(x_1)) + x_1 \exp\{x_2\}$$

This equation can be split up using intermediate variables $v_1, \ldots, v_7$ as follows:

$$\begin{aligned} v_1 &= x_1 \\ v_2 &= \sin(v_1) \\ v_3 &= \ln(v_2) \\ v_4 &= x_2 \\ v_5 &= \exp\{v_4\} \\ v_6 &= v_1 v_5 \\ v_7 &= v_3 + v_6 \\ f(x_1, x_2) &= v_7 \end{aligned}$$

Splitting up the equation like this is very similar to what an auto-differentiation library would do. From these equations we can construct a *computational graph* where each node of the graph corresponds to an input, an intermediate variable, or the output.

(a) Let $x_1 = \frac{\pi}{6}$ and $x_2 = 1$. Calculate the values of all the intermediate variables $v_1, \ldots v_7$ and $f(x_1, x_2)$.

(b) Calculate the derivative of all of the intermediate variables $v_1, \ldots, v_7$ and f with respect to x_1 evaluated at $x_1 = \frac{\pi}{6}$ and $x_2 = 1$. Remember to write out the equations before evaluating, e.g.,

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \frac{\partial g(x)}{\partial x}.$$

4. **Extra Credit (Hard):** Consider two neural networks f_1 and f_2 for binary classification. They each take in inputs $x \in \mathbb{R}^2$ and output a prediction $\hat{y} \in [0, 1]$. f_1 consists of a single hidden layer with 4 nodes, each with a ReLU activation function. These nodes are connected to a single sigmoid output node. Thus f_1 has the following form:

$$f_1(x) = \sigma\left(W_2[ReLU(W_1 x + b_1)] + b_2\right)$$

f_2 consists of 2 hidden layers, each with 2 ReLU activated nodes. Just as in f_1, the nodes of the final layer are connected to a single sigmoid output node. Thus f_2 has the following form:

$$f_2(x) = \sigma(W_3[ReLU(W_2[ReLU(W_1 x + b_1)] + b_2)] + b_3)$$

We leave finding the shapes of the weight and bias vectors up to you, noting that by convention x should be considered a column vector with 2 elements.

Draw a classification boundary that f_2 can express but f_1 cannot and argue why f_2 can express the boundary but f_1 cannot.

Solution

1.
   - $\mathbf{W}_1$:
   - $\mathbf{b}_1$:
   - $\mathbf{W}_2$:
   - $\mathbf{b}_2$:
   - $\mathbf{a}_1$:
   - $\mathbf{z}_1$:
   - a_2 :
   - $\hat{y}$:

2. (a)

$$\frac{\partial L}{\partial b_2} =$$

   (b)

$$\frac{\partial L}{\partial W_2^h} =$$

   (c)

$$\frac{\partial L}{\partial b_1^h} =$$

   (d)

$$\frac{\partial L}{\partial W_1^{h,j}} =$$

3. (a)

$$\begin{aligned} v_1 &= \\ v_2 &= \\ v_3 &= \\ v_4 &= \\ v_5 &= \\ v_6 &= \\ v_7 &= \\ f(x_1, x_2) &= \end{aligned}$$

(b)

$$\frac{\partial v_1}{\partial x_1} =$$
$$\frac{\partial v_2}{\partial x_1} =$$
$$\frac{\partial v_3}{\partial x_1} =$$
$$\frac{\partial v_4}{\partial x_1} =$$
$$\frac{\partial v_5}{\partial x_1} =$$
$$\frac{\partial v_6}{\partial x_1} =$$
$$\frac{\partial v_7}{\partial x_1} =$$
$$\frac{\partial f}{\partial x_1} =$$

Problem 3 (Modern Deep Learning Tools: PyTorch, 40 pts)

In this problem, you will learn how to use PyTorch. This machine learning library is massively popular and used heavily throughout industry and research.

1. In `T3_P3.ipynb` you will implement an MLP for image classification from scratch. Paste your code solutions below and include a final graph of your training progress. Also submit your completed `T3_P3.ipynb` file.
2. Discuss what trends you see with your plot (train/test loss and train/test accuracy).
3. **Out of Distribution (OOD) Analysis**: Now, let's evaluate the usefulness of the predictive uncertainties of our model for test data that are dissimilar to our training data. These test data points are called out of distribution (OOD) points. Report both the in and out distribution test accuracies of your model. In a couple of sentences, discuss what you notice about these accuracies.
4. Now let us consider the implications of what we have seen. First, just as in Homework 2, we want the predictive uncertainties from our models to help us distinguish in-distribution test data (test data that are similar to data on which we trained our model) and OOD test data. Look at some examples in which the model expresses uncertainty about an in-distribution output and in which the model expresses uncertainty about an out-of-distribution output. Characterize what you see. In what ways are the uncertainties of the model useful, and in what ways are they not? Do you think training multiple models and boostrapping, like you did in Homework 2, would help? (You do not have to code anything, just discuss.)
5. Suppose the postal service was going to use your model to help automatically sort mail by zipcodes (a real use of AI systems). They want to make sure that their system is safe against adversarial attacks. Let us suppose that the model is relatively safe against software attacks, that is, appropriate security is in place such that the adversary cannot simply change the weights of a deployed model without someone noticing (in practice, this would be an important element). Three scenarios, however, are of concern to them:
   (a) Hardware attack: Suppose the adversary has access to the scanner used to take pictures of the envelopes. How might they be able to change the outputs of the model to their desired ones? What safeguards might be possible, and what are their benefits and limitations/drawbacks?
   (b) Input attack: Suppose the adversary has access to the envelopes prior to scanning. How might they be able to change the outputs of the model to their desired ones? What safeguards might be possible, and what are their benefits and limitations/drawbacks?
   (c) Social Engineering attack: Suppose that the adversary has access to the teams that will be maintaining and retraining the model. How might they be able to change the outputs of the model to their desired ones? What safeguards might be possible, and what are their benefits and limitations/drawbacks?

   When you consider possible safeguards, think broadly: What might be done in software (relates to Part 4 above)? Regarding the system integration and work environment overall? You may find it useful to explore how you can manipulate your model by changing inputs, etc. but no coding is required for this question.

You will recieve no points for code not included below.

You will recieve no points for code using built-in APIs from the `torch.nn` library.

Solution

1. Plot:

   Code:

```
n_inputs = 'not implemented'
n_hiddens = 'not implemented'
n_outputs = 'not implemented'

W1 = 'not implemented'
b1 = 'not implemented'
W2 = 'not implemented'
b2 = 'not implemented'



def relu(x):
    'not implemented'



def softmax(x):
    'not implemented'



def net(X):
  'not implemented'



def cross_entropy(y_hat, y):
  'not implemented'



def sgd(params, lr=0.1):
  'not implemented'



def train(net, params, train_iter, loss_func=cross_entropy, updater=sgd):
  'not implemented'
```

2.

3. Test Accuracy (In Distribution):

   Test Accuracy (Out of Distribution):

   Discussion:

4.

5. (a)

   (b)

   (c)

Name

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?