Homework 2

1. Bayesian Interpretation of Regularization

Background: In Bayesian statistics, almost every quantity is a random variable, which can either be observed or unobserved. For instance, parameters θ are generally unobserved random variables, and data x and y are observed random variables. The joint distribution of all the random variables is also called the model (e.g., $p(x, y, \theta)$). Every unknown quantity can be estimated by conditioning the model on all the observed quantities. Such a conditional distribution over the unobserved random variables, conditioned on the observed random variables, is called the *posterior distribution*. For instance $p(\theta \mid x, y)$ is the posterior distribution in the machine learning context. A consequence of this approach is that we are required to endow our model parameters, i.e., $p(\theta)$, with a prior distribution. The prior probabilities are to be assigned *before* we see the data—they capture our prior beliefs of what the model parameters might be before observing any evidence.

In the purest Bayesian interpretation, we are required to keep the entire posterior distribution over the parameters all the way until prediction, to come up with the *posterior predictive* distribution, and the final prediction will be the expected value of the posterior predictive distribution. However in most situations, this is computationally very expensive, and we settle for a compromise that is less pure (in the Bayesian sense).

The compromise is to estimate a point value of the parameters (instead of the full distribution) which is the mode of the posterior distribution. Estimating the mode of the posterior distribution is also called *maximum a posteriori* estimation (MAP). That is,

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid x, y).$$

Compare this to the *maximum likelihood estimation* (MLE) we have seen previously:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(y \mid x, \theta).$$

In this problem, we explore the connection between MAP estimation, and common regularization techniques that are applied with MLE estimation. In particular, you will show how the choice of prior distribution over θ (e.g., Gaussian or Laplace prior) is equivalent to different kinds of regularization (e.g., L_2, or L_1 regularization). You will also explore how regularization strengths affect generalization in part (d).

(a)

Show that $\theta_{\text{MAP}} = \arg\max_{\theta} p(y \mid x, \theta)p(\theta)$ if we assume that $p(\theta) = p(\theta \mid x)$. The assumption that $p(\theta) = p(\theta \mid x)$ will be valid for models such as linear regression where the input x are not explicitly modeled by θ. (Note that this means x and θ are marginally independent, but not conditionally independent when y is given.)

(b)

Recall that L_2 regularization penalizes the L_2 norm of the parameters while minimizing the loss (i.e., negative log likelihood in case of probabilistic models). Now we will show that MAP estimation with a zero-mean Gaussian prior over θ, specifically $\theta \sim \mathcal{N}(0, \eta^2 I)$, is equivalent to applying L_2 regularization with MLE estimation. Specifically, show that for some scalar λ,

$$\theta_{\text{MAP}} = \arg\min_{\theta} \; -\log p(y \mid x, \theta) + \lambda \|\theta\|_2^2.$$

Also, what is the value of λ?

(c)

Now consider a specific instance, a linear regression model given by $y = \theta^T x + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Assume that the random noise $\epsilon^{(i)}$ is independent for every training example $x^{(i)}$. Like before, assume a Gaussian prior on this model such that $\theta \sim \mathcal{N}(0, \eta^2 I)$. For notation, let X be the design matrix of all the training example inputs where each row vector is one example input, and $\vec{y}$ be the column vector of all the example outputs. Come up with a closed form expression for θ_{MAP}.

(d)

Next, consider the Laplace distribution, whose density is given by

$$f_L(z \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|z - \mu|}{b}\right).$$

As before, consider a linear regression model given by $y = x^T\theta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Assume a Laplace prior on this model, where each parameter θ_i is marginally independent, and is distributed as $\theta_i \sim \mathcal{L}(0, b)$. Show that θ_{MAP} in this case is equivalent to the solution of linear regression with L_1 regularization, whose loss is specified as

$$J(\theta) = \|X\theta - \vec{y}\|_2^2 + \gamma\|\theta\|_1$$

Also, what is the value of γ?

Note: A closed form solution for linear regression problem with L_1 regularization does not exist. To optimize this, we use gradient descent with a random initialization and solve it numerically.

Remark: Linear regression with L_2 regularization is also commonly called *Ridge regression*, and when L_1 regularization is employed, is commonly called *Lasso regression*. These regularizations can be applied to any Generalized Linear models just as above (by replacing $\log p(y \mid x, \theta)$ with the appropriate family likelihood). Regularization techniques of the above type are also called *weight decay*, and *shrinkage*. The Gaussian and Laplace priors encourage the parameter values to be closer to their mean (i.e., zero), which results in the shrinkage effect.

Remark: Lasso regression (i.e., L_1 regularization) is known to result in sparse parameters, where most of the parameter values are zero, with only some of them non-zero.

2. Logistic Regression: Training Stability

In this problem, we will delve deeper into the workings of logistic regression. Consider a binary classification problem with data points $(x^{(i)}, y^{(i)})$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{0, 1\}$. We model the probability of the positive class using logistic regression:

$$P(y = 1|x^{(i)}) = \frac{1}{1 + e^{-\theta^T x^{(i)}}},$$

where $\theta \in \mathbb{R}^d$ is the parameter vector.

The logistic regression **loss function** (also known as the cross-entropy loss) over n data points is given by:

$$L(\theta) = -\sum_{i=1}^{n} \left(y^{(i)} \log(P(y = 1|x^{(i)})) + (1 - y^{(i)}) \log(1 - P(y = 1|x^{(i)}))\right).$$

Assume that John has implemented logistic regression and is using it to train on two labeled datasets:

- **Dataset A:** A non-separable dataset, where no hyperplane can perfectly separate the two classes. In other words, for any choice of parameter vector θ, some points from each class will always lie on opposite sides of the hyperplane defined by $\theta^T x = 0$.

- **Dataset B:** A separable dataset, where a hyperplane can perfectly separate the two classes, meaning there exists a parameter vector θ that places all positive examples ($y^{(i)} = 1$) on one side of the hyperplane and all negative examples ($y^{(i)} = 0$) on the other side.

After training, John observes that his model converges on Dataset A but fails to converge on Dataset B. He wants to understand why this is happening. Your task is to help him analyze this behavior.

(a)

Suppose we replace θ with a scaled version $c \cdot \theta$, where $c > 0$ is a constant. Show that as $c \to \infty$, the predicted probabilities for correctly classified examples approach 1 for positive examples and 0 for negative examples.

(b)

Using the result from (a), show that the loss function $L(\theta)$ continues to decrease as $c \to \infty$ in linearly separable data. Explain why this implies that gradient descent will fail to converge in this scenario.

(c)

Contrast this with the case of non-separable data. Explain why the minimum of the loss function is attained at a finite point in the non-separable case and why gradient descent would converge.

(d)

For each of the following modifications, state whether or not it would lead to the training algorithm converging on datasets like B. Justify your answers.

- Using a different constant learning rate.
- Decreasing the learning rate over time (e.g., scaling the initial learning rate by $1/t^2$, where t is the number of gradient descent iterations so far).
- Applying linear scaling to the input features.
- Adding a regularization term $\|\theta\|^2$ to the loss function.
- Adding zero-mean Gaussian noise to the training data or labels.

In the following questions, we consider neural networks with multiple layers. Each layer has multiple inputs and outputs, and can be broken down into two parts:

- A **linear** module that implements a linear transformation: $z_j = (\sum_{i=1}^{m} x_i w_{i,j}) + w_{0,j}$ specified by a weight matrix W and a bias vector W_0. The output is $[z_1, \ldots, z_n]^T$.
- An **activation** module that applies an activation function to the outputs of the linear module for some activation function f, such as Tanh or ReLU in the hidden layers or Softmax (see below) at the output layer. We write the output as: $[f(z_1), \ldots, f(z_m)]^T$, although technically, for some activation functions such as softmax, each output will depend on all the z_i, not just one.

We will use the following notation for quantities in a network:

- Inputs to the network are $x_1, \ldots, x_d$.

- Number of layers is L.
- There are m^l inputs to layer l.
- There are $n^l = m^l + 1$ outputs from layer l.
- The weight matrix for layer l is W^l, an $m^l \times n^l$ matrix, and the bias vector (offset) is W_0^l, an $n^l \times 1$ vector.
- The outputs of the linear module for layer l are known as **pre-activation** values and denoted z^l.
- The activation function at layer l is $f^l(\cdot)$.
- Layer l activations are $a^l = \left[f^l(z_1^l), \ldots, f^l(z_{n_l}^l)\right]^T$.
- The output of the network is the values $a^L = \left[f^L(z_1^L), \ldots, f^L(z_{n_L}^L)\right]^T$.
- Loss function $Loss(a, y)$ measures the loss of output values a when the target is y.

3. Multiclass classification

What if we needed to classify homework problems into three categories: enlightening, boring, impossible? We can do this by using a *one-hot* encoding on the output, and using three output units with what is called a *softmax* (SM) activation module. It's not a typical activation module, since it takes in all n_L pre-activation values z_j^L in $\mathbb{R}$ and returns n_L output values $a_j^L \in [0, 1]$ such that $\sum_j a_j^L = 1$. This can be interpreted as representing a probability distribution over the possible categories.

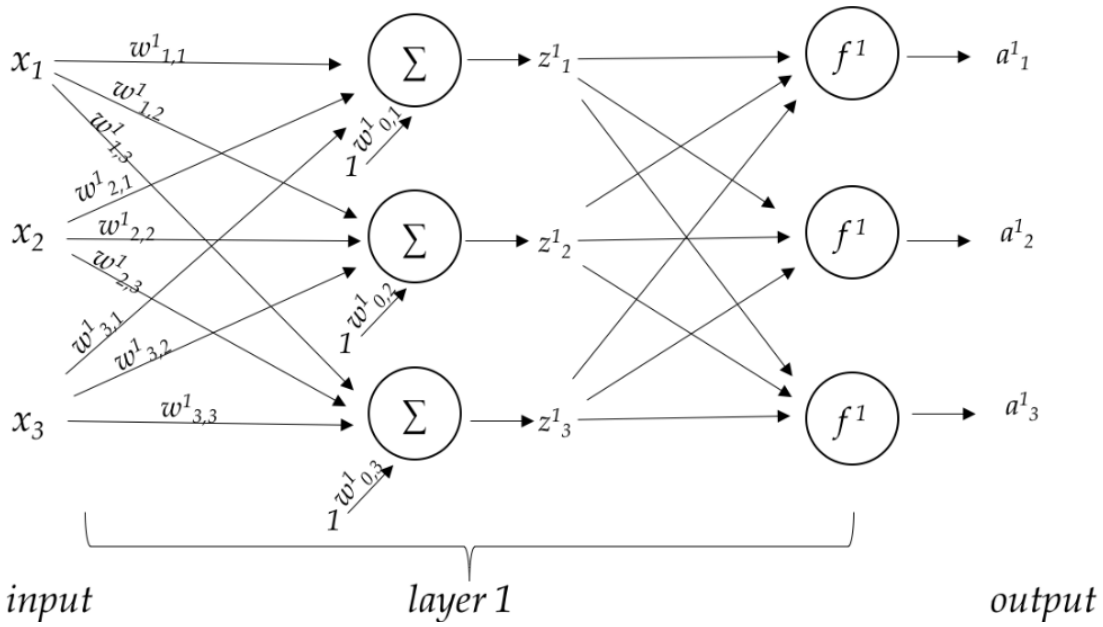
The individual entries are computed as

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{n_L} e^{z_k}}$$

We'll describe the relationship of the vector $\mathbf{a}$ on the vector $\mathbf{z}$ as

$$\mathbf{a} = \text{SM}(\mathbf{z})$$

The network below shows a one-layer network with a linear module followed by a softmax activation module.



(a)

What probability distribution over the categories is represented by $z_L = [-1, 0, 1]^T$?

(b)

Now, we need a loss function $Loss(\mathbf{a}, \mathbf{y})$ where $\mathbf{a}$ is a discrete probability distribution and $\mathbf{y}$ is a one-hot vector encoding of a single output value. It makes sense to use negative log likelihood as a loss function for the same reasons as before. So, we'll just extend our definition of NLL from earlier:

$$NLL(\mathbf{a}, \mathbf{y}) = -\sum_{j=1}^{n_L} y_j \ln a_j^L.$$

Note that the above expression is for multi-classes (number of class > 2).
If $\mathbf{a} = [0.3, 0.5, 0.2]^T$ and $\mathbf{y} = [0, 0, 1]^T$, what is $NLL(\mathbf{a}, \mathbf{y})$?

(c)

Now, we can think about a single layer with a softmax activation module, trained to minimize NLL. The pre-activation values (the output of the linear module) are:

$$z_j^L = \sum_k w_{k,j}^L x_k + w_{0,j}^L$$

and $a^L = SM(z^L)$.
To do gradient descent, we need to know $\frac{\partial}{\partial w_{k,j}^L} NLL(a^L, \mathbf{y})$. Try to prove

$$\frac{\partial}{\partial w_{k,j}^L} NLL(a^L, \mathbf{y}) = x_k(a_j^L - y_j)$$

And of course, it's easy to compute the whole matrix of these derivatives, $\nabla_{W^L} NLL(a^L, \mathbf{y})$, in one quick matrix computation.
Suppose we have two input units and three possible output values, and the weight matrix W^L is

$$W^L = \begin{bmatrix} 1 & -1 & -2 \\ -1 & 2 & 1 \end{bmatrix}$$

Assume the biases are zero, the input $\mathbf{x} = [1, 1]^T$, and the target output $\mathbf{y} = [0, 1, 0]^T$. What is the matrix $\nabla_{W^L} NLL(a^L, \mathbf{y})$?

(d)

What is the predicted probability that x is in class 1, before any gradient updates? (Assume we have classes 0, 1, and 2.)

(e)

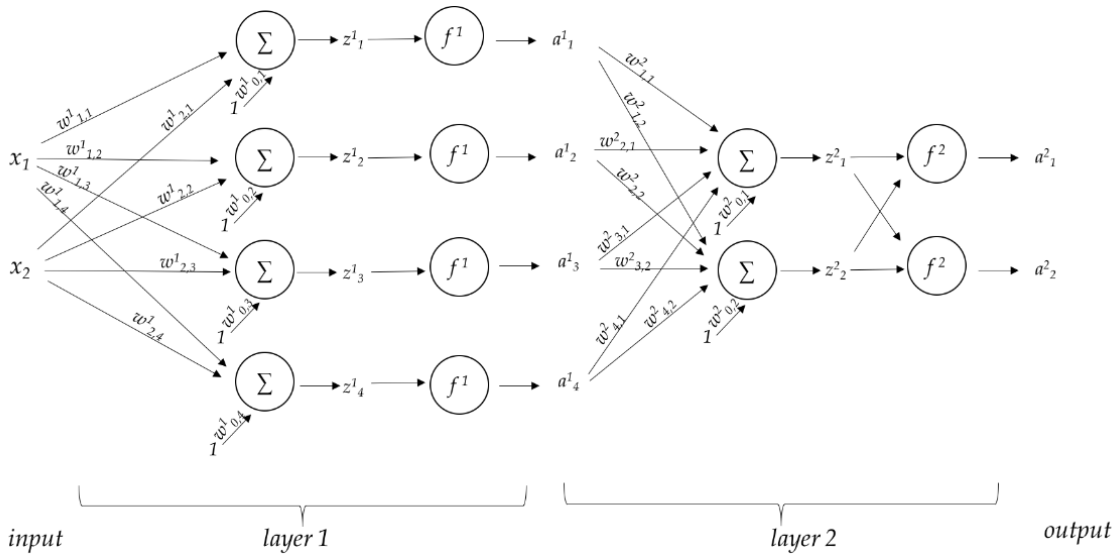
Using step size 0.5, what is W^L after one gradient update step?

(f)

What is the predicted probability that x is in class 1, given the new weight matrix?

4. Neural Networks

Consider the neural network given in the figure below, with ReLU activation functions (f^1 in the figure) on all hidden neurons, and softmax activation (f^2 in the figure) for the output layer, resulting in softmax outputs (a_1^2 and a_2^2 in the figure).



Given an input $\mathbf{x} = [x_1, x_2]^T$, the hidden units in the network are activated in stages as described by the following equations:

$$
\begin{aligned}
z_1^1 &= x_1 w_{1,1}^1 + x_2 w_{2,1}^1 + w_{0,1}^1 & a_1^1 &= \max\{z_1^1, 0\} \\
z_2^1 &= x_1 w_{1,2}^1 + x_2 w_{2,2}^1 + w_{0,2}^1 & a_2^1 &= \max\{z_2^1, 0\} \\
z_3^1 &= x_1 w_{1,3}^1 + x_2 w_{2,3}^1 + w_{0,3}^1 & a_3^1 &= \max\{z_3^1, 0\} \\
z_4^1 &= x_1 w_{1,4}^1 + x_2 w_{2,4}^1 + w_{0,4}^1 & a_4^1 &= \max\{z_4^1, 0\}
\end{aligned}
$$

$$
\begin{aligned}
z_1^2 &= a_1^1 w_{1,1}^2 + a_2^1 w_{2,1}^2 + a_3^1 w_{3,1}^2 + a_4^1 w_{4,1}^2 + w_{0,1}^2 \\
z_2^2 &= a_1^1 w_{1,2}^2 + a_2^1 w_{2,2}^2 + a_3^1 w_{3,2}^2 + a_4^1 w_{4,2}^2 + w_{0,2}^2
\end{aligned}
$$

The final output of the network is obtained by applying the softmax function to the last hidden layer,

$$
\begin{aligned}
a_1^2 &= \frac{e^{z_1^2}}{e^{z_1^2} + e^{z_2^2}} \\
a_2^2 &= \frac{e^{z_2^2}}{e^{z_1^2} + e^{z_2^2}}.
\end{aligned}
$$

In this problem, we will consider the following setting of parameters:

$$
\begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & w_{1,4}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & w_{2,4}^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \qquad \begin{bmatrix} w_{0,1}^1 \\ w_{0,2}^1 \\ w_{0,3}^1 \\ w_{0,4}^1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix},
$$

$$
\begin{bmatrix} w_{1,1}^2 & w_{1,2}^2 \\ w_{2,1}^2 & w_{2,2}^2 \\ w_{3,1}^2 & w_{3,2}^2 \\ w_{4,1}^2 & w_{4,2}^2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}, \qquad \begin{bmatrix} w_{0,1}^2 \\ w_{0,2}^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}.
$$

(a)

Consider the input $x_1 = 3, x_2 = 14$. What are the outputs of the hidden units, $(f^1(z_1^1), f^1(z_2^1), f^1(z_3^1), f^1(z_4^1))$ and the final output (a_1^2, a_2^2) of the network?

(b)

Consider the following input vectors: $x^{(1)} = [0.5, 0.5]^T$, $x^{(2)} = [0, 2]^T$, $x^{(3)} = [-3, 0.5]^T$. Enter a matrix where each column represents the outputs of the hidden units $(f(z_1^1), \ldots, f(z_4^1))$ for each of the input vectors.

In our network above, the output layer with two softmax units is used to classify into one of two classes. For class 1, the first unit's output should be larger than the other unit's output, and for class 2, the second unit's output should be larger. This generalizes nicely to k classes by using k output units.

Let's characterize the region in x-space where this network's output indicates the first class (that is, a_1^2 is larger) or indicates the second class (that is, a_2^2 is larger).

(c)

What is the output value of the neural network in each of the following cases?
Case 1) For $f(z_1^1) + f(z_2^1) + f(z_3^1) + f(z_4^1) = 0$
Case 2) For $f(z_1^1) + f(z_2^1) + f(z_3^1) + f(z_4^1) = 1$
Case 3) For $f(z_1^1) + f(z_2^1) + f(z_3^1) + f(z_4^1) = 3$

(d)

Using the cross-entropy loss function, and given the input sample $(x_1 = 3, x_2 = 14)$ with target vector $(y_1, y_2) = (0, 1)$, perform one backpropagation step with a learning rate $\eta = 0.1$ to update each weight in the network.