Homework 2

Deadline: Sunday, March 29th, at 11:59pm.

Submission: You need to submit your homework files on Microsoft Teams.
DO NOT submit zipped files.

- Put all your solutions and answers to all questions as a PDF file titled `hw2_writeup.pdf`. You can produce the file however you like (e.g. LATEX, Microsoft Word, scanner), as long as it is readable.
- Your final codes for Questions 2, 4 and 5 should be submitted as python notebook `.ipynb` files.

Late Submission: 50% of the marks will be deducted for any submission beyond the deadline. No submissions will be accepted after two days past the deadline.

Collaboration: Homeworks are individual work. Please refrain from copying others work (including previous year's solutions). In case you draw ideas from other's work, you must cite their work.

1. **[8 marks] Bias-Variance**: Assume a linear regression model

$$\underline{y} = \underline{\underline{X}}\,\underline{w}_* + \underline{\epsilon}$$

where $\underline{w}_* \in \mathbb{R}^p$ is the true parameter we are trying to estimate, $\underline{\epsilon} = [\epsilon_1, \ldots, \epsilon_N]^T \sim \mathcal{N}(\underline{0}, \sigma^2 \underline{\underline{I}}_N)$ is the measurement noise, and $\underline{y} = [y_1, \ldots, y_N]^T$ are the outputs.

Throughout the problem, assume $\underline{\underline{X}}^T \underline{\underline{X}}$ to be invertible. Given a realization of the outputs, the training set becomes $\mathcal{T} = \left\{\underline{\underline{X}}, \underline{y}\right\}$, two estimates of the parameter $\underline{w}_*$ are obtained as follows:

$$\underline{w}_{\text{ols}} = \arg\min \left\|\underline{y} - \underline{\underline{X}}\,\underline{w}\right\|_2^2$$
$$\underline{w}_{\text{ridge}} = \arg\min \left\|\underline{y} - \underline{\underline{X}}\,\underline{w}\right\|_2^2 + \lambda \left\|\underline{w}\right\|_2^2$$

(a) **[1.5 marks]** Let $\underline{\hat{w}} \in \mathbb{R}^p$ denote an estimate of $\underline{w}_*$. Note an estimate is a random variable as it depends upon the realization $\underline{y}$.
Define the mean squared error (MSE) of the estimate $\underline{\hat{w}}$ as

$$\text{MSE}(\underline{\hat{w}}) := \mathbb{E}_{\mathcal{T}}\left[\|\underline{\hat{w}} - \underline{w}_*\|_2^2\right]$$

Note that this is multivariate generalization of the mean squared error.
Define $\underline{\hat{\mu}} := \mathbb{E}_{\mathcal{T}}\left[\underline{\hat{w}}\right]$. Show that the MSE decomposes as

$$\text{MSE}(\underline{\hat{w}}) = \left\|\underline{\hat{\mu}} - \underline{w}_*\right\|_2^2 + \text{tr}\left(\text{Cov}(\underline{\hat{w}})\right)$$

Note that this is multivariate generalization of the bias-variance decomposition we have seen in class previously, except that these terms appear without the expectation $\mathbb{E}_*\left[\cdot\right]$ that we considered in lectures.

(b) **[2 marks]** Show that

$$\mathbb{E}_{\mathcal{T}}\left[\underline{w}_{\text{ols}}\right] = \underline{w}^*, \qquad \mathbb{E}_{\mathcal{T}}\left[\underline{w}_{\text{ridge}}\right] = \left(\underline{\underline{X}}^T\underline{\underline{X}} + \lambda\underline{\underline{I}}_p\right)^{-1}\underline{\underline{X}}^T\underline{\underline{X}}\underline{w}^*$$

That is, $\underline{w}_{\text{ols}}$ is an *unbiased* estimate of $\underline{w}_*$, whereas, $\underline{w}_{\text{ridge}}$ is a *biased* estimate of $\underline{w}_*$.

(c) **[4.5 marks]** If $\gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_p$ denote the p eigenvalues of the matrix $\underline{\underline{X}}^T\underline{\underline{X}}$ arranged in non-decreasing order, then show that

$$\operatorname{tr}\left(\operatorname{Cov}(\underline{w}_{\text{ols}})\right) = \sigma^2\sum_{i=1}^{p}\frac{1}{\gamma_i}, \qquad \operatorname{tr}\left(\operatorname{Cov}(\underline{w}_{\text{ridge}})\right) = \sigma^2\sum_{i=1}^{p}\frac{\gamma_i}{\left(\gamma_i+\lambda\right)^2}$$

Finally use the above two expressions to conclude that

$$\operatorname{tr}\left(\operatorname{Cov}(\underline{w}_{\text{ridge}})\right) < \operatorname{tr}\left(\operatorname{Cov}(\underline{w}_{\text{ols}})\right)$$

Hint: Use the relationship between the trace and the eigenvalues of a matrix. For ridge regression variance, consider writing $\underline{\underline{X}}^T\underline{\underline{X}}$ in terms of its singular value decomposition $\underline{\underline{U\Sigma V}}^T$.

2. **[8 marks] Coding Logistic Regression**: Consider a mechanical failure dataset, which consists of a total of examples 4601 machine components data, from which 57 features have been extracted as follows:

- 48 features denote the proportion (ranging from 0 to 1) of specific characteristics occurring in a given mechanical component such as range of vibration frequency, temperature, pressure, etc.
- 6 features represent the proportion (ranging from 0 to 1) of range of different vibration noise levels observed during the occurrence of mechanical failures.
- Feature 55 represents the average size of the machine component
- Feature 56 indicates the maximum length of time (in hours) before the first maintenance operation
- Feature 57 signifies the total number of working hours completed by the machine component

The dataset consists of a training set size 3450 and a test set of size 1151. One can imagine performing several kinds of preprocessing to this data matrix. Try each of the following separately:

(i) Standardize each column so they each have mean 0 and unit variance.

(ii) Transform the features using $x_{i,j} \leftarrow \log(x_{i,j} + 0.1)$.

(iii) Binarize the features using $x_{i,j} \leftarrow \mathbb{I}(x_{i,j} > 0)$. $\mathbb{I}$ denotes an indicator variable.

Note: You will need to tune the step size carefully to avoid numerical issues and to avoid a diverging training loss.

(a) [**2 marks**] Implement logistic regression to classify the spam data. Derive the gradient of the cross-entropy loss with respect to $\underline{\theta}$; your answer should be a matrix-vector expression. Do NOT write your answer in terms of the individual components of the gradient.

Use *batch gradient descent* for implementing logistic regression.

Plot the training loss (the cross-entropy loss of the training set) vs. the number of epochs. You should have one plot for each preprocessing method.

Note: One epoch amounts to scanning through the whole training data.

(b) [**2 marks**] Derive stochastic gradient descent equations for logistic regression and show your steps. Plot the training loss vs. number of iterations. You should have one plot for each preprocessing method. How are the plots different from part (a)?

Note: One stochastic gradient descent iteration amounts to computing the gradient using one data point.

(c) [**1 mark**] Instead of a constant learning rate (η), repeat (b) where the learning rate decreases as $\eta \propto 1/t$ for the t^{th} iteration. Plot the training loss vs. number of iterations. Is this strategy better than having a constant η? You should have one plot for each preprocessing method.

(d) [**3 marks**] Use logistic ridge regression with a suitable regularization parameter λ. Adjust the value of λ based on a validation set (using a 70-30% train-validate set).

Show that you need the following update equations:

$$\begin{aligned}\theta_i^{(t+1)} &\leftarrow \theta_i^{(t)} - \gamma\theta_i^{(t)} + \eta S(-z_i) y_i,\\ \theta_h^{(t+1)} &\leftarrow \theta_h^{(t)} - \gamma\theta_h^{(t)} \text{ for } h \neq i\end{aligned}$$

where $z_i = y_i f(\underline{x}_i)$ and $S(z_i) = \frac{1}{1+\exp(-z_i)}$.

Generate a plot of training loss vs. number of SGD iterations and another plot of validation loss vs. number of SGD iterations. Report the test set prediction you get.

Hint: If you get numerical issues, feel free to clamp the values of z.

NOTE: You are NOT supposed to use any kind of software package for logistic regression.

3. [**3 marks**] **Computational graph for backpropagation**: Consider a two-layered neural network consisting of N inputs, K hidden nodes and N output nodes. The forward computation of the output is done in the following way:

$$\begin{aligned}\underline{z} &= \underline{\underline{W}}^{(1)}\underline{x} + \underline{b}^{(1)}\\ \underline{h} &= \sigma(\underline{z})\\ \underline{\hat{y}} &= \underline{x} + \underline{\underline{W}}^{(2)}\underline{h} + \underline{b}^{(2)}\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid (logistic) function, applied element wise. The specially designed loss function L has both $\underline{h}$ and $\underline{\hat{y}}$:

$$\begin{aligned}L &= S + R\\ S &= \frac{1}{2}\left\|\underline{\hat{y}} - \underline{s}\right\|_2^2\\ R &= \underline{r}^T\underline{h}\end{aligned}$$

where $\underline{r}$ and $\underline{s}$ are given vectors.

(a) Draw the computation graph relating $\underline{x}$, $\underline{z}$, $\underline{h}$, $\underline{y}$, R, S and L.

(b) Derive the backpropagation equations for computing $\bar{\underline{x}} = \frac{\partial \mathcal{L}}{\partial \underline{x}}$. You may use σ' to denote the derivation of the sigmoid function, so you do not have to write it out explicitly.

4. **[7 marks] Coding FNN for MNIST**
Put all your plots and answers in the `hw2_writeup.pdf` file.

In class, we had talked about a two-layer feed-forward neural network for classifying MNIST handwritten digits. You are now given a starter code here [1] . Run the code as it is. You will notice that after training is done, three figures will appear.

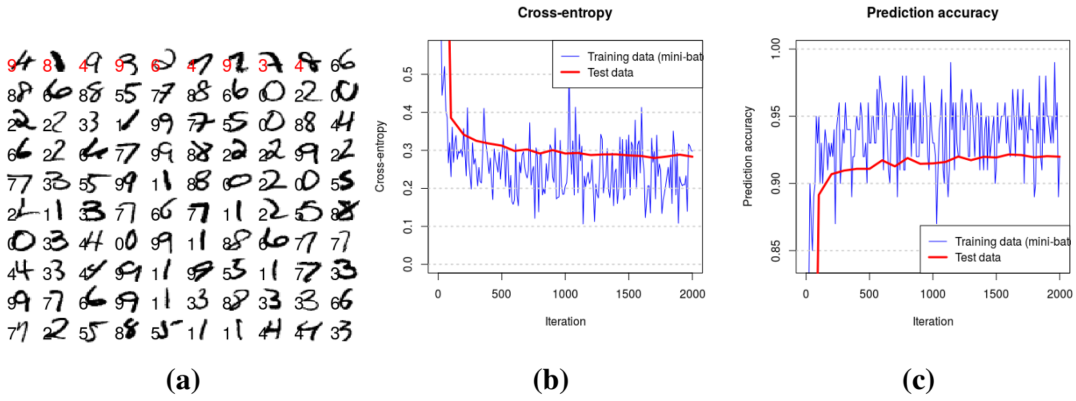

Figure 1: Three figures generated by the code: (a) Prediction performance on 100 randomly selected test points, (b) Cost function (cross-entropy) on test/training data, (c) Prediction accuracy on test/training data

100 randomly selected test images (out of the total 10 000 test images) are displayed together with their predicted label. For those images that have been incorrectly classified, the label is colored red. We use a cross-entropy loss function

(i) **[0.25 mark]** Why is the training accuracy so "noisy"?

(ii) **[0.25 mark]** What classification accuracy do you get on the test data?

(iii) **[0.5 mark]** How many parameters does this single-layer neural network model have?

(iv) **[1 mark]** Recall *epoch* vs *iteration* from the lecture notes. How many iterations does the code take until it has seen all training data points, i.e., how many iterations are included in each epoch? How many epochs do you think is being completed in the given code?

Next, add a hidden layer to make a **two-layer neural network**. Choose 200 hidden neurons. Keep softmax activation on the last layer, but use the sigmoid (or logistic) activation function for the hidden neurons. You can use `torch.sigmoid` command.

[1] Please go through Practical 7 for an introduction to PyTorch

Save the new file as **mnist_twolayers.ipynb**. Try using the following weights and biases

```
U = 200 # number of hidden units
self.W1 = nn.Parameter(0.1 * torch.randn(784, U))
self.b1 = nn.Parameter(torch.zeros(U))
self.W2 = nn.Parameter(0.1 * torch.randn(U, 10))
self.b2 = nn.Parameter(torch.zeros(10))
```

The forward model will have:

```
Q = torch.sigmoid(X.mm(self.W1) + self.b1)
G = F.softmax(Q.mm(self.W2) + self.b2, dim=1)
```

(v) [**0.5 mark**] What classification accuracy do you get on the test data now? Does the accuracy improve over a single-layer neural net? Plot the prediction accuracy for training and test data.

(vi) [**1 mark**] Try some other numbers of hidden neurons (ranging from 10 to 750). How does it affect the performance?

(vii) [**0.5 mark**] What happens if you initialize the weights with zeros as we did for the single layer neural network?

Continue to create even deeper neural nets. Save **mnist_twolayers.ipynb** as a new file **mnist_fivelayers.ipynb**. Add in total of five layers with 200, 100, 60 and 30 hidden neurons between each layer.

(viii) [**0.5 mark**] What classification accuracy do you get on test data? Does the accuracy improve over the two-layers neural net? Plot the prediction accuracy for training and test data.

(ix) [**0.5 mark**] Instead of using sigmoid activation function, use *Rectified Linear Unit* (ReLU) activation. To use ReLU, simply replace all **torch.sigmoid** in the code with **F.relu**. What classification accuracy on the test data do you get? Plot the prediction accuracy for training and test data.

(x) [**0.5 mark**] What happens when the weights and biases are all initialized to zero values?

(xi) [**0.5 mark**] Did you find any **NaN** values cropping up while training? If so, what do you think might be the reason? Instead of using **F.softmax** in the forward model, return only the logit

```
Z = Q4.mm(self.W5) + self.b5
return Z
```

and replace **crossentropy** with **F.cross_entropy**. Report if **NaN** values go away.

(xii) [**0.25 mark**] How many parameters does this five-layer neural network have?

(xiii) [**0.5 mark**] Use the Adam optimizer. Replace the **optim.SGD** with **optim.Adam**. Change the learning rate from 0.5 to 0.003. Use ReLU activations and random initialization of weights and small-positive (= 0.1) initialization of biases. What classification accuracy on the test data do you get? Is there any improvement in the classification accuracy on the test data? Justify why is there an improvement or deterioration. Plot the prediction accuracy for training and test data.

(xiv) **[0.75 mark]** Train for 10000 iterations. Plot the prediction accuracy for training and test data. Do you think you can get better classification accuracy on test data if you train longer?

5. **[5 marks] Coding CNN for MNIST**

Put all your plots and answers in the `hw2_writeup.pdf` file.

The previous question considered an FNN model where $28 \times 28 \times 1$[2] pixels in each image was *flattened* into a long vector with 784 elements. This destroys spatial information present in the images. CNN can exploit spatial information better compared to FNN. In this question, you will implement a CNN.

The CNN you create will have three convolutional layers and two final dense layers. The settings for the three convolutional layers are given in Table 1

Table 1: Architechture of the three convolutional layers

	Layer 1	**Layer 2**	**Layer 3**
Number of kernels/filters	4	8	12
Filter size	5×5	5×5	4×4
Stride (equal in both directions)	1	2	2
Zero Padding (equal in both directions)	2	2	1

You have to use PyTorch for constructing the CNN model. PyTorch requires weight tensors of the size: (`output channels` × `input channels` × `filter rows` × `filter columns`). Hence the weight tensor and offset vector for the first convolutional layer.

```
U1 = 4
self.W1 = nn.Parameter(0.1 * torch.randn(U1, 1, 5, 5))
self.b1 = nn.Parameter(torch.ones(U1)/10)
```

The corresponding model implementation for that convolutional layer is

```
Q1 = F.relu(F.conv2d(X, self.W1, bias=self.b1, stride=1,
    padding=2))
```

After the three convolutional layers, we use two dense layers. Before we can apply the first dense layer, all hidden units in the third convolutional layer needs to be flattened into a long vector. This can be done with the command

```
Q3flat = Q3.view(-1, U3flat)
```

(i) **[0.5 mark]** How many hidden units are there in total in the third convolutional layer, i.e., what is `U3flat` supposed to be in the code above?

[2] Note a grayscale image has depth of one, therefore the number of input channel is 1.

(ii) [**1.5 marks**] Save `mnist_fivelayers.ipynb` as a new file `mnist_CNN.ipynb`. Replace the first three dense layers in the previous code with three convolutional layers using the settings in Table 1 for each of these three layers. Add the reshaping command according to what is stated above. Update the fourth and fifth layer according to the instructions above. Train for at least 4000 iterations.

What prediction accuracy do you achieve on test data? Plot the prediction accuracy of test and train data.

(iii) [**1.5 marks**] Towards the end of training phase, the learning rate $\eta = 0.003$ may be really too large. The prediction accuracy and/or the cross-entropy on test data would oscillate and we may not get down to the best minimum of the cost function. You would therefore prefer having a lot smaller η, but with a very small η from the very beginning of the training will increase the training time. One solution is to start learning fast (to get approximately close to the minimum) and then slow down.

Reduce the learning rate `lr` slowly. Adjust the learning rate as the following function

$$\eta^{(t)} = \eta_{\text{min}} + (\eta_{\text{max}} - \eta_{\text{min}})\, e^{-t/2000}$$

with $\eta_{\text{min}} = 0.0001$ and $\eta_{\text{max}} = 0.003$. Incorporate this adjustment in the code, and train with atleast 6000 iterations. What prediction accuracy on the test data did you achieve? Plot the prediction accuracy of test and train data.

(iv) [**0.5 mark**] At this point, the network can start overfitting on training data. How can you see that?

(v) [**1 mark**] If you manage to get 99% prediction accuracy on test data, plot the prediction accuracy of test and train data.