Machine Learning for Mechanical Engineering, ME 343
Hojat Ghorbanidehno, Eric Darve
Winter 2019

Stanford University

ME 343: Homework Assignment 2

Total number of points = 100.

Submission instructions: submit your homework using gradescope. We expect you to upload your answers as a PDF file along with a zip file containing your code. See `Homework 2` and `Homework 2 Code` on gradescope.

You can create the PDF using any software you want. It is possible to write your homework paper using pen and paper, and take pictures using your phone. Make sure the lighting is sufficient. Create a single PDF with all the pages.

You will find the starter code and required files at this URL:
`https://drive.google.com/file/d/1vkhhMIN0cTG6nYONZrxDAAme16qWYFGv/view?usp=sharing`
The size of the file is 76 MB.[1]

Problem 1: Building a simple two-layer neural network

In this problem, you will build your first neural network for a 2-class classification problem. Your network will have an input layer, one hidden layer, and an output layer. You will compare the performance of your neural network with that of logistic regression and you will also learn how to select the hyper-parameters for your network.

1. **Logistic regression** 5 points

   Logistic regression is one of the most popular machine learning algorithms for binary classification. It is a simple algorithm that performs very well on a wide range of problems. Please refer to the following tutorial on logistic regression

   `https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python`

   Use the helper function provided to you to load the training data and train a logistic regression model. Plot the decision boundary and discuss the results.

2. **NN: initializing parameters** 5 points

   Complete the "model-parameter-init()" function to initialize the weighting parameters. How do you initialize W_1 and W_2 parameters? Why?

3. **Forward and Backward passes** 10 points

   Complete the "feed-forward()" and "backprop()" functions for a two-layer neural network with one hidden layer. You have the freedom to select the activation function (nonlinear function) for the hidden layer but you have to select the right type of activation function for the output layer based on the objective of the problem. The type of the activation function will affect your back-propagation function as the gradient formula will change. Note that we used cross-entropy as the cost function here.

[1] Please email `darve@stanford.edu` if the link does not work.

4. **Training the network** 10 points

   Now that you have all the required function, you can train the network. Complete the "train()" function first, then load the data and train the network for 1000 epochs. Use the default values given in the train function for training the network. Use the "plot-decision-boundary()" function to plot the separating line and discuss the results.

5. **Comparing neural network with logistic regression** 5 points

   Plot the results for your neural network for the cases with 2, 5, 10, and 20 hidden units. Compare the results with the result of logistic regression and discuss your findings.

Problem 2: Inverse modeling

In this problem, you are going to perform inverse modeling to estimate the depth of water bodies from noisy flow velocity measurements using deep learning techniques. Given the depth of a river, the surface flow velocity can be computed by using physical models of the form $v = f(h) + \epsilon$ where v is the flow velocity profile, h is the depth of river, and ϵ is the noise in the system due to inaccuracy of measurement devices and the simplification assumptions made to obtain the physical model. However, in most real-world applications, we are given the flow velocity profiles and the goal is to estimate the depth of the river using these velocity measurements. There are several inverse modeling techniques such as geostatistical approach or Kalman filtering techniques to solve this problem. However, these methods can be computationally expensive for online application. Here, you will learn to apply fully connected deep neural network to estimate the depth of a river using its corresponding velocity measurements.

In order to simplify the problem, you only need to estimate the depth profile for a small segment of the river profile. Also, in order to reduce the dimension of the problem, the riverbed profiles are projected to their first k important eigenvectors. Note that the first k important eigenvectors of a matrix, are the eigenvectors that correspond to the k largest eigenvalues of the matrix. Therefore, the input vector (x) of your network is the flow velocity measurements for each segment and the output of your network (y) would be a vector with k elements that corresponds to the k largest eigenvalues. Note that the data are preprocessed and the matrix X in the dataset corresponds to the flow velocity measurements for each segment and the matrix Y corresponds to the first k eigenvalues for each riverbed profile.

1. **Constructing the model** 10 points

   Complete the "generate-model()" function in the starter code. For simplicity, the number of layers and number of hidden units are given as a list to the function. Use the same parameters (activation function, regularization coefficient, . . .) for all the layers except the last layer. What type of activation function do you use for the last layer? Why? Explain what type of loss function and evaluation metrics you selected and provide the reasoning. Consider adding batch normalization layers to your network structure.

2. **Training the network** 10 points

   Complete the "train()" function and then set up your network parameters and train your network. You need to select the network parameters such as the number of layers, the number of hidden units, regularization parameters, etc. Use 80% of the data for the training set and 20% for the validation set. What is the number of epochs that you selected? Why? Plot the loss function for the training set and the validation set.

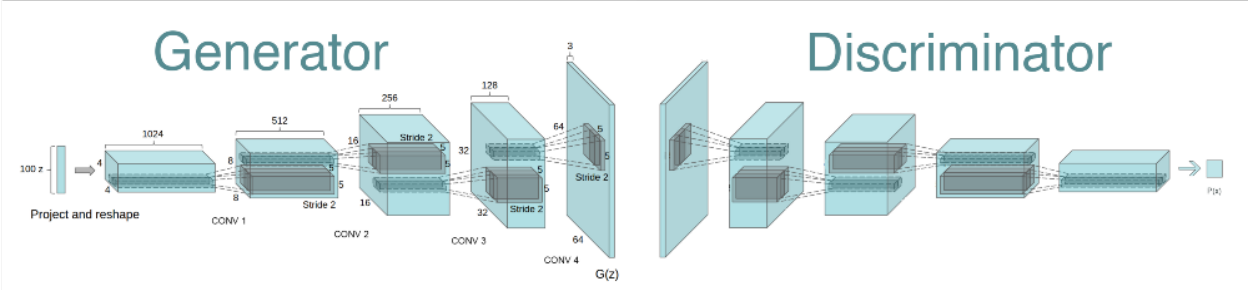

Figure 1: DCGAN network structure

3. **Hyper-parameter selection** 10 points

   In this section, you will learn how to select hyper-parameters. For simplicity, we try to find only two hyper-parameters which are regularization coefficient and learning rate decay. Here, we only run the network for 10 epochs to get results faster. Complete the "hyp-tuning()" function and call the function. Report the optimal values for the regularization coefficient and learning rate decay.

4. **Plotting results** 5 points

   Train your network for the optimal hyper-parameters that you found in the previous section and use the "plot-result()" function that is provided to you to load the test dataset and plot the results.

Problem 3: Deep Convolutional GAN (DCGAN)

In this problem, you'll get hands-on experience coding and training GANs. Here, we will implement a specific type of GAN designed to process images, called a Deep Convolutional GAN (DCGAN). We'll train the DCGAN using the MNIST dataset to generate numbers from samples of random noise. You'll gain experience implementing GANs by writing code for the generator, discriminator, and training loop, for each model.

DCGAN is a GAN method that uses a convolutional neural network as the discriminator and a network composed of transposed convolutions as the generator. The transposed convolution layer acts in the "opposite" direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input. You can refer to Keras documentation to learn more about transposed convolution layer and its implementation in keras.

To implement the DCGAN, we need to specify three things: (1) the generator, (2) the discriminator, and (3) the training procedure. We will develop each of these three components in the following subsections.

1. **Discriminator** 10 points

   In the code given to you, at each convolution layer of the discriminator, we downsample the spatial dimension of the input volume by a factor of two and we use a kernel size $k = 5$. Follow the instruction provided in the discriminator function to implement the discriminator part of the network. Discuss how the discriminator network works.

2. **Generator** 10 points

   Now, we will implement the generator of the DCGAN, which consists of a sequence of the transpose convolutional layers that progressively up-sample the input noise sample to generate a fake image.

   Follow the instructions provided in the generator function to implement the generator part of the network. Discuss why we need to use transpose convolution layers in the generator.

3. **Training** 10 points

   In order to simplify the problem for you, the training function is provided for you. Your task in this section is to explain this function in the write-up. How does the network learn to generate meaningful images? What happens at each iterations step of the `for` loop?

   Load the MNIST dataset and train your DCGAN network. Run the "gen-result()" function to generate images. Discuss the results and suggest how we can improve the network performance.