Assignment 1

Due date: Feb 15 (5.00 PM CT)

Turn in solutions as pdf(s) (see template) and ipynb file(s) on the course website (for instance, as a single zip file).

Note: Answer the following questions in complete sentences and with full clarity. Since this is a graduate class, please use any assumptions necessary to answer the questions satisfactorily. As in real life, use your judgment to figure out what conditions to assume while answering. Answering too little or too much will garner point deduction. The answer to 'should I include this in my answer' is almost always a yes, especially if it completes your answer to the question asked. If in doubt, use the discussion board on the course website. Across group collaboration is not allowed. Cite all your sources (see the Syllabus sheet for these and other pointers).

1 Backpropagation (5pt)

1. Draw the computation graph for the following function: $f(a, b, c, d, e) = \frac{1}{(1+(a^b+c^d)*e)^2}$. Compute the gradient of the function with respect it to its inputs at $(a, b, c, d, e) = (1, 1, 1, 1, 1)$ (see Lecture on Jan 11).

2 Gradient Descent (20pt)

1. Write a function to compute the mean squared error between a prediction and ground truth assuming both are *numpy* arrays (see python module *numpy*).
2. Consider a model: $y = mx + c$, where the model parameter $m = 1$ and parameter $c = 0$ and $x \in (0, 1)$. Plot the function.
3. Generate example data by drawing $N = 100$ uniform values from the range in which x lies and compute the corresponding y to get $\{x_i, y_i\}_{i=1}^N$.
4. Assuming that you do not know the model parameters, use backpropagation and gradient descent updates to find the model parameters (choose an appropriate learning rate). The loss function will be the mean squared error.
5. Plot the error in the estimates as a function of the number of iterations of gradient update (use python package called *matplotlib*). Change the learning rate and plot another curve on the previous plot.
6. Do steps 3-5 when the model is $y = m_1x + m_2x^2 + c$ and the true parameters are $m_1 = 0.5$, $m_2 = 1$ and $c = 1$. And $x \in (0, 1)$. Also, plot the ground truth function.
7. Do steps 3-5 when the model is $y = \tanh(m * x + c)$ and the true parameters are $m = 1$ and $c = 2$. And $x \in (0, 2)$. Also, plot the ground truth function.

3 ML Basics (5pt)

1. Write a function to compute the (multiclass) logistic loss (also called the cross-entropy loss) given the parameters (W, b) of a linear model (as *numpy* arrays) and an example (x, y).

2. Write a function to compute the multiclass SVM loss given the parameters (W, b) of a linear model (as *numpy* arrays) and an example (x, y).

3. Add an ℓ_1 regularization and an ℓ_2 regularization to the loss function.

4 Classification pipeline (20pt)

1. Generate data from *data_generator_2d.py*.

2. Split the data into test and train (20%:80%).

3. Build a linear classifier assuming the multiclass logistic loss and an ℓ_2 regularization for the weights only. Report the prediction accuracy on the training data and the test data.

4. Introduce a cross validation scheme and justify your choice. What is the validation accuracy compare to the test accuracy.

5. What is the sensitivity of the model's performance to different learning rates and the number of gradient descent iterations. Describe via suitable plots.

6. What is the sensitivity of the model's performance to different regularization parameter values. Find the best regularization parameter using an exhaustive search procedure. Describe your choice via suitable plots. What is the performance difference between using regularization and no regularization?

7. What is the sensitivity of the model's performance with respect to a different test train split (50%:50%).

5 Feedforward Neural Network (40pt)

1. Repeat Question 4 when the model is a 2-layer feedforward neural network (i.e., 1 hidden layer with $f(x, W_1, b_1, W_2, b_2) = W_2 \max(0, W_1 x + b_1) + b_2$). Use hand-computed backpropagation equations.

2. Repeat item 1 when the non-linearity is changed to leaky ReLU ($f(x) = x$ if $x > 0$, else $f(x) = 0.01 * x$).

3. Repeat Question 4 when the model is $f(x, W_1, b_1, W_2, b_2, W_3, b_3) = W_3 \max(W_2 x + b_2, W_1 x + b_1) + b_3$ (this is called the maxout non-linearity).

4. In items 1-3, what happens when the number of hidden units chosen is much smaller. Similarly, what happens when the number of hidden units chosen is much higher?

6 Real datasets and Keras (30pt)

1. Repeat Questions 4 and 5 with the MNIST dataset.

2. Repeat Questions 4 and 5 with the CIFAR-10 dataset.

3. Build the linear classifier and the feedforward neural networks in Questions 4 and 5 using the python module *Keras* (with backend *Tensorflow* for automatic backpropagation) and check the performances differences when the optimizer is SGD. What could be the cause of the differences, if any?

7 CNNs and finetuning (30pt)

1. Run the VGG pretrained model in Keras for sample images from the Cats&Dogs dataset and display the top 5 predictions for a random sample of 4 images (show the images and the prediction labels).

2. Use the VGG output (1000 dimensional) as the feature vector instead of the image itself and repeat questions 4 and 5 for either the MNIST or the CIFAR-10 or the Cats&Dogs dataset. Use either Keras or your custom backpropagation code from Question 4.

Datasets

- The 2D simulated data is available on the course website (see *data_generator_2d.py*).

- MNIST image data: Orignal data can be found here. Processed dataset can be obtained from Tensorflow directly (see this or this which hosts mnist.pkl.gz).

- CIFAR-10 dataset: Original data can be found here (see the python version). Another example way to obtained the processed dataset can be found here.

- Cats&Dogs dataset: Original dataset can be obtained from Kaggle (you may have to sign up).