Homework #5

ECE 661: Introduction to Machine Learning
Prof. Carlee Joe-Wong and Prof. Virginia Smith
Due: Tuesday November 6th, 2018 at 8:30AM PT / 11:30AM ET

October 27, 2018

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. The full collaboration and grading policies are available on the course website: https://18661.github.io/.

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) in PDF format by the deadline. We will not accept hardcopies. If you choose to hand-write your solutions, please make sure the uploaded copies are legible. Gradescope will ask you to identify which page(s) contain your solutions to which problems, so make sure you leave enough time to finish this before the deadline. We will give you a 30-minute grace period to upload your solutions in case of technical problems.

1 Entropy and Information Gain *[20 points]*

When discussing decision trees, we introduced the concepts of entropy $H(X)$ and information gain $H(X) - H(X|Y)$ for discrete random variables X, Y that take on values in a set $\mathcal{X}$ and $\mathcal{Y}$. In information theory, the information gain is also referred to as mutual information $I(X;Y)$. In particular, we can define:

$$H(X) = -\sum_{x\in\mathcal{X}} P(X=x)\log_2 P(X=x) \tag{1}$$

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(X=x,Y=y)\log_2 P(X=x,Y=y) \tag{2}$$

$$H(Y|X) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(X=x,Y=y)\log_2 P(Y=y|X=x) \tag{3}$$

In this question, we will derive some useful properties of mutual information and relate it to the relative entropy. Using the above definitions, answer the following questions:

a. *[5 points]* Show that for two random variables X,Y: $H(X,Y) = H(X) + H(Y|X)$.

b. *[5 points]* Starting from the definition of information gain, i.e., $I(X;Y) = H(X) - H(X|Y)$, show that

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(X=x,Y=y)\log_2\left(\frac{P(X=x,Y=y)}{P(Y=y)P(X=x)}\right)$$

Further, show that $I(X;Y) = I(Y;X)$ and $I(X;X) = H(X)$.

c. *[5 points]* Show that if two variables X and Y are independent, then their mutual information or information gain is $I(X;Y) = 0$.

d. *[5 points]* The relative entropy $D(P(X)||Q(X))$ between two probability distributions $P(X)$ and $Q(X)$ over a random variable X taking values in $\mathcal{X}$ is

$$D(P(X)||Q(X)) = \sum_{x \in \mathcal{X}} P(X = x) \log_2 \left(\frac{P(X = x)}{Q(X = x)} \right)$$

Jensen's inequality is a useful result that states that for any convex function $f(x)$, $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$. Use Jensen's inequality to show that $D(P(X)||Q(X)) \geq 0$.

It is evident that the mutual information or information gain is simply the relative entropy between $P(X, Y)$ and $P(X)P(Y)$. We can then directly conclude that the information gain must be non-negative. Note that this is a very intuitive result: information gain cannot be negative, that is, conditioning should never increase your entropy!

2 Decision Trees *[10 points]*

Consider the following set of data with features X,Y, and Z and output variable G

X	Y	Z	G
1	0	1	1
0	0	0	0
1	0	1	1
1	1	0	0
0	0	1	0
0	1	1	1
1	1	1	0
0	1	1	1

Build a decision tree for classifying G based on the binary features X,Y, and Z by greedily choosing the threshold splits that maximize the information gain. Break any ties by favoring splits in alphabetical order. Show your work for each split.

3 Boosting *[25 points]*

We learned about boosting in lecture, and the topic is covered in Murphy 16.4. On page 555 Murphy claims that "it was proved that one could boost the performance (on the training set) of any weak learner arbitrarily high, provided the weak learner could always perform slightly better than chance." We will now verify this statement in the AdaBoost framework.

a. *[5 points]* Given a set of N observations (x^j, y^j) where y^j is the label $y^j \in \{-1, 1\}$, let $h_t(x)$ be the weak classifier at step t and let α_t be its weight. First we note that the final classifier after T steps is defined as:

$$H(x) = \text{sgn} \left\{ \sum_{t=1}^{T} \alpha_t h_t(x) \right\} = \text{sgn}\{f(x)\}$$

Where

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Show that:

$$\epsilon_{\text{Training}} = \frac{1}{N} \sum_{j=1}^{N} 1_{\{H(x^j) \neq y^j\}} \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-f(x^j) y^j)$$

Where $1_{\{H(x^j)\neq y^j\}}$ is 1 if $H(x^j) \neq y^j$ and 0 otherwise.

b. ***[8 points]*** The weight for each data point j at step $t+1$ can be defined recursively by:

$$w_j^{(t+1)} = \frac{w_j^{(t)} \exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

Where Z_t is a normalizing constant ensuring the weights sum to 1:

$$Z_t = \sum_{j=1}^{N} w_j^{(t)} \exp(-\alpha_t y^j h_t(x^j))$$

Show that:

$$\frac{1}{N} \sum_{j=1}^{N} \exp(-f(x^j)y^j) = \prod_{t=1}^{T} Z_t$$

c. We showed above that training error is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values $Z_1, Z_2, \ldots, Z_{t-1}$ are already fixed. Therefore, at step t we can choose α_t to minimize Z_t. Let

$$\epsilon_t = \sum_{j=1}^{m} w_j^t 1_{\{h_t(x^j)\neq y^j\}}$$

be the weighted training error for weak classifier $h_t(x)$. Then we can re-write the formula for Z_t as:

$$Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

(a) ***[4 points]*** First, find the value of α_t that minimizes Z_t. Then show that

$$Z_t^{opt} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

(b) ***[4 points]*** Assume we choose Z_t in this way. Then re-write $\epsilon_t = \frac{1}{2} - \gamma_t$ where $\gamma_t > 0$ implies better than random and $\gamma_t < 0$ implies worse than random. Then show that:

$$Z_t \leq \exp(-2\gamma_t^2)$$

You may want to use the fact that $\log(1-x) \leq -x$ for $0 \leq x < 1$

Thus we have:

$$\epsilon_{\text{training}} \leq \prod_{t=1}^{T} Z_t \leq \exp(-2\sum_{t=1}^{T} \gamma_t^2)$$

(c) ***[4 points]*** Finally, show that if each classifier is better than random (e.g. $\gamma_t \geq \gamma$ for all t and $\gamma > 0$) then:

$$\epsilon_{\text{training}} \leq \exp(-2T\gamma^2)$$

which shows that the training error can be made arbitrarily small with enough steps.

4 Perceptrons ***[10 points]***

A perceptron learns a binary linear classifier with weight vector $\mathbf{w}$ (we omit the bias weight $\mathbf{b}$ for simplicity. That is, it learns a classifier that predicts $\hat{y} \in \{+1, -1\}$ as

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x}_t)$$

The perceptron algorithm we present is based on an *online learning* model, that is, it processes the data sequentially (one sample at a time), updating the weight vector at each step. The perceptron algorithm is as follows:

1 Set $t = 0$ and initialize with all-zeroes weight vector $\mathbf{w}_0 = \mathbf{0}$

2 Given example $\boldsymbol{x}_t$, predict $\hat{y}_t = \text{sign}(\mathbf{w}_t^T \mathbf{x}_t)$

3 Let y_t be the true label of x_t. If we make a mistake, i.e, $y_t \neq \hat{y}_t$ we update the weights as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_t \mathbf{x}_t$$

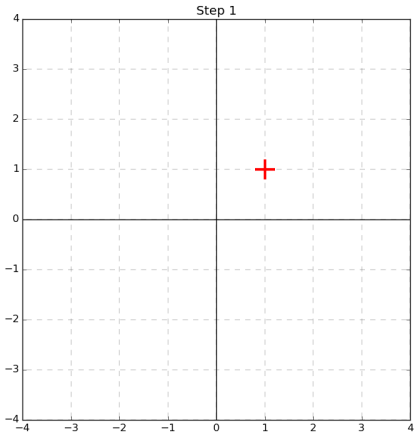
f If we don't make a mistake, then we don't update the weight.

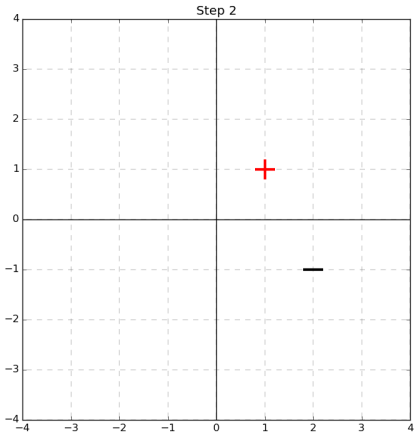
4 If there are more samples to train on, then $t \leftarrow t + 1$, go back to step 2.

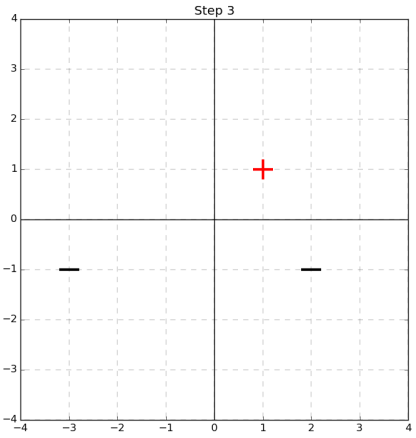
In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function. As a linear classifier, the single-layer perceptron is the simplest feedforward neural network. In this problem, we go through the perceptron algorithm step by step for a toy problem. Imagine that we have $\mathbf{x}_t \in \mathbb{R}^2$, and we encounter the following data points in order:

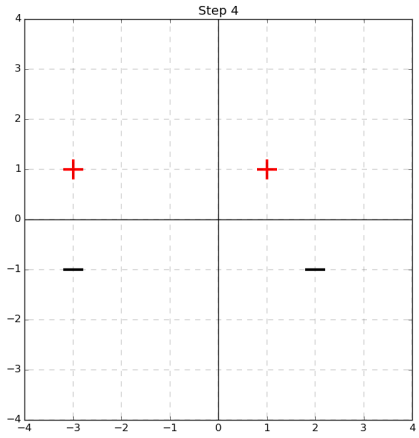
$\mathbf{x}[1]$	$\mathbf{x}[2]$	y
1	1	1
2	-1	-1
-3	-1	-1
-3	1	1

a. ***[8 points]*** Starting with $\mathbf{w} = [0 \quad 0]^T$, use the perceptron algorithm to learn on the data points in the order from top to bottom. Show the perceptron's linear decision boundary after observing each data point in the graphs below. Be sure to show which side is classified as positive.

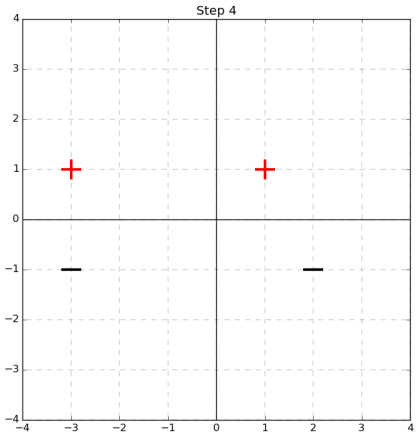








b. *[2 points]* Does our learned perceptron maximize the margin between the training data and the decision boundary (i.e., the minimum distance between a training point and the boundary)? If not, draw the linear decision boundary that maximizes the margin on the graph below.



5 Neural Networks for MNIST Digit Recognition *[35 points]*

In this problem, you will implement a neural network to classify handwritten digits using raw pixels as features. You will be using the classic MNIST digits dataset. The state-of-the-art error rate on this dataset using various learning methods is around 0.5% (see this leaderboard for details). However, you will be implementing a simple neutral network and should, with appropriate hyperparameter settings, get a test error of approximately 6% or better. Specifically, you will implement a neural network with a total of three layers: an input layer, a hidden layer, and an output layer. Please pay careful attention to the following implementation details.

5.1 Details

- You will be using a hidden layer of size 200. Let $n_{in} = 784$, the number of raw pixel features. Let $n_{hid} = 200$, the size of the hidden layer. Finally, let $n_{out} = 10$, the number of output labels or classes.

Then, you will have $n_{in}+1$ units in the input layer, $n_{hid}+1$ units in the hidden layer, and n_{out} units in the output layer. The input and hidden layers have one additional unit which always takes a value of 1 to represent bias. The output layer size is set to the number of classes. We will refer to the last layer as layer $L = 2$, and the input layer as layer 0.

- Each label will have to be transformed to a one-hot-encoding representation, i.e., a vector of length 10 that has a single 1 in the position of the true class and 0 everywhere else.
- Each layer is fully connected to the previous layer. The parameters of the model are the weights w_{ij}^{l}. For $l = 1$, the weights from the input to the hidden layer, there are n_{hid}-by-$(n_{in}+1)$ weights, which represent the weights from the ith input node to the jth hidden node. For $l = L = 2$, the weights from the hidden layer to the output, there are n_{out}-by-$(n_{hid}+1)$ weights, which represent the weights from the ith hidden node to the jth output node.
- You will be using the cross entropy error, given as

$$L = -\sum_{j=1}^{n_{out}} [y_j ln(x_j^{(L)}) + (1-y_j) ln(1-x_j^{(L)})]$$

  where $x_j^{(L)}$ is the output of the jth node at layer L.
- Hidden units should use $h(z) = tanh(z)$ as its activation function, and output units should use the sigmoid function $g(z) = \frac{1}{1+e^{-z}}$ as its activation function.
- Make sure you initialize your weights with random (small) values. This allows us to break symmetry that occurs when all weights are initialized to 0.
- It may be helpful to make your step size inversely proportional to the current training iteration.
- Datasets are provided as inside the `official_data` folder. We also provide prototype data which may be helpful in debugging and checking your code. Please report values for the official training set.
- Function signatures for training and testing your neural network , `trainNN()` and `testNN()`, are also provided for in `neural_net.py`. Please submit all your code.

5.2 Problems

a. Derive and write the equations for the stochastic gradient descent updates for all parameters w_{ij}^{l}.

b. Divide the full training dataset into training and validation sets (use an 80/20 split). Train this multi-layer neural network on the training data using stochastic gradient descent, and evaluate its accuracy on the validation set. As you see fit, tune various hyperparameters of the model, e.g., learning rate, when you stopped training, and how you initialized the weights. Report your hyperparameter setup and describe what you observe while performing this hyperparameter optimization.

c. After deciding on a good choice of hyperparameters, train the neural network on the *full training dataset*, and report the following results for this model:

- Training accuracy and test accuracy.
- Plots of training accuracy vs. iteration.