COMS 4771 HW2

Due: Fri Oct 27, 2017 at 11:59pm (Extended!)

You are allowed to work in groups of (at max) three students. These group members don't necessarily have to be the same from previous homeworks. Only one submission per group is required by the due date. Name and UNI of all group members must be clearly specified on the homework. You must cite all the references you used to do this homework. You must show your work to receive full credit.

1 **[Classification with 'unsure' option]** Often it is beneficial to construct classifiers which predict a label only when they are confident, and have an option to output 'unsure' if they are less confident in exchange to incurring a penalty. If the penalty for outputting 'unsure' is not too high, it may be a desirable action. Consider the penalty function

$$\varrho(\hat{y}; Y = y) = \begin{cases} 0 & \hat{y} = y \\ \lambda_u & \hat{y} = \text{'unsure'} \\ \lambda_s & \text{otherwise} \end{cases}$$

where λ_u is the penalty incurred for outputting 'unsure' and λ_s is the penalty incurred for mispredicting. Consider a joint distribution over the data X and labels Y.

(i) For a given test example x, what is the minimum possible penalty a classifier can yield? (Hint: there are multiple cases depending on the value of $P(Y|X = x)$.)

(ii) What happens if $\lambda_u = 0$? What happens if $\lambda_u \geq \lambda_s$?

2 **[Constrained optimization and duality]**

(i) Show that the distance from the hyperplane $g(x) = w \cdot x + w_0 = 0$ to a point x_a is $|g(x_a)|/\|w\|$ by minimizing the squared distance $\|x - x_a\|^2$ subject to the constraint $g(x) = 0$.

Consider the optimization problem

$$\min_{x \in \mathbb{R}^d} \|x\|$$
$$\text{such that} \sum_{i=1}^{d} x_i \geq 5$$

(ii) Is this optimization problem a convex optimization problem? why or why not?

(iii) What is the Lagrange dual of this problem?

(iv) Does strong duality hold? why or why not?

3 **[Making data linearly separable by feature space mapping]** Consider the infinite dimensional feature space mapping

$$\Phi_\sigma : \mathbb{R} \to \mathbb{R}^\infty$$
$$x \mapsto \left(\max\left\{0, 1 - |\frac{\alpha - x}{\sigma}|\right\}\right)_{\alpha \in \mathbf{R}}.$$

(It may be helpful to sketch the function $f(\alpha) := \max\{0, 1 - |\alpha|\}$ for understanding the mapping and answering the questions below)

(i) Show that for any n distinct points $x_1, \ldots, x_n$, there exists a $\sigma > 0$ such that the mapping Φ_σ can linearly separate *any* binary labeling of the n points.

(ii) Show that one can efficiently compute the dot products in this feature space, by giving an analytical formula for $\Phi_\sigma(x) \cdot \Phi_\sigma(x')$ for arbitrary points x and x'.

4 **[Perceptron case study]** We shall study the relative performance of different variations of the Perceptron algorithm on the handwritten digits dataset from HW1.

Consider a sequence of training data $(x_1, y_1), \ldots, (x_n, y_n)$ in an arbitrary but fixed order (the labels y_i are assumed to be binary in $\{-1, +1\}$).

Perceptron V0
learning:
- Initialize $w_0 := 0$
- for $t = 1, \ldots, T$
- pick example (x_i, y_i), where $i = (t \bmod n + 1)$
- if $y_i(w_{t-1} \cdot x_i) \leq 0$
- $w_t := w_{t-1} + y_i x_i$
- else
- $w_t := w_{t-1}$

classification:
$f(x) :=$ sign$(w_T \cdot x)$

Perceptron V1
learning:
- Initialize $w_0 := 0$
- for $t = 1, \ldots, T$
- pick example (x_i, y_i), such that $i := \arg\min_j \left(y_j w_{t-1} \cdot x_j\right)$
- if $y_i(w_{t-1} \cdot x_i) \leq 0$
- $w_t := w_{t-1} + y_i x_i$
- else
- $w_T := w_{t-1}$; terminate

classification:
$f(x) :=$ sign$(w_T \cdot x)$

Perceptron V2
learning:
- Initialize $w_1 := 0$, $c_0 := 0$, k := 1
- for $t = 1, \ldots, T$
- pick example (x_i, y_i), where $i = (t \bmod n + 1)$
- if $y_i(w_k \cdot x_i) \leq 0$
- $w_{k+1} := w_k + y_i x_i$
- $c_{k+1} := 1$
- $k := k + 1$
- else
- $c_k := c_k + 1$

classification:
$f(x) := \text{sign}\big(\sum_{i=1}^{k} c_i \ \text{sign}(w_i \cdot x)\big)$

(i) Implement the three variations of the Perceptron algorithm for the 10-way digit classification problem.
You must submit your code on Courseworks to receive full credit.

(ii) Which Perceptron version is better for classification? You must justify your answer with appropriate performance graphs demonstrating the superiority of one classifier over the other. Example things to consider: you should evaluate how the classifier behaves on a holdout test sample for various splits of the data; how does the training sample size and the number of passes affects the classification performance.

(iii) Implement the Kernel Perceptron as described in lecture with a high degree (say, 5 to 10) polynomial kernel. How does it affect the classification on test data?

5 **[Neural networks as universal function approximators]** Here we will experimentally verify that (feed-forward) neural networks can approximate any smooth function. Recall that a feed-forward neural network is simply a combination of multiple 'neurons' such that the output of one neuron is fed as the input to another neuron. More precisely, a neuron ν_i is a computational unit that takes in an input vector $\vec{x}$ and returns a wighted combination of the inputs (plus the bias associated with neuron ν_i) passed through an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, that is: $\nu_i(\vec{x}; \vec{w}_i, b_i) := \sigma(\vec{w}_i \cdot \vec{x} + b_i)$. With this notation, we can define a layer ℓ of a neural network $\mathcal{N}^\ell$ that takes in an I-dimensional input and returns a O-dimensional output as

$$\begin{aligned} \mathcal{N}^\ell(x) &:= \big(\nu_1^\ell(\vec{x}), \nu_2^\ell(\vec{x}), \cdots, \nu_O^\ell(\vec{x})\big) && x \in \mathbb{R}^I \\ &= \sigma(W_\ell^\mathsf{T} \vec{x} + \vec{b}_\ell) && W_\ell \in \mathbb{R}^{d_I \times d_O} \text{ defined as } [\vec{w}_1^\ell, \ldots, \vec{w}_O^\ell], \\ & && \text{and } \vec{b}_\ell \in \mathbb{R}^{d_O} \text{ defined as } [b_1^\ell, \ldots, b_O^\ell]^\mathsf{T}. \end{aligned}$$

Here $\vec{w}_i^\ell$ and b_i^ℓ refers to the weight and the bias associated with neuron ν_i^ℓ in layer ℓ, and the activation function σ is applied pointwise. An L-layer (feed-forward) neural network $\mathcal{F}_{L\text{-layer}}$ is then defined as a network consisting of network layers $\mathcal{N}^1, \ldots, \mathcal{N}^L$, where the input to

layer i is the output of layer $i-1$. By convention, input to the first layer (layer 1) is the actual input data.

(i) Consider a nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ defined as $x \mapsto 1/(1+e^{-x})$. Show that $\frac{\partial \sigma}{\partial x} = \sigma(x)(1-\sigma(x))$.

(ii) Consider a *single layer* feed forward neural network that takes a d_I-dimensional input and returns a d_O-dimensional output, defined as $\mathcal{F}_{\text{1-layer}}(x) := \sigma(W^T x + b)$ for some $d_I \times d_O$ weight matrix W and a $d_O \times 1$ vector b. Given a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, we can define the *average error* (with respect to the network parameters) of predicting y_i from input example x_i as:

$$E(W, b) := \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{F}_{\text{1-layer}}(x_i) - y_i\|^2.$$

What is $\frac{\partial E}{\partial W}$, and $\frac{\partial E}{\partial b}$?

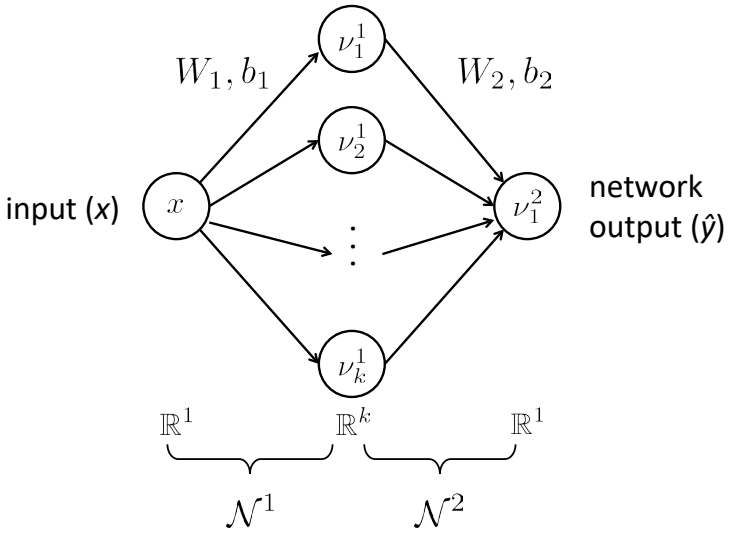
(note: we can use this gradient in a descent-type procedure to minimize this error and learn a good setting of the weight matrix that can predict y_i from x_i.)

(iii) Although reasonably expressive, single layer neural networks are not flexible enough to approximate arbitrary smooth functions. Here we will focus on approximating one-dimensional functions $f : \mathbb{R} \to [0, 1]$ (the range of the function is taken in the interval $[0, 1]$ only for convenience, one can transform any bounded function to this interval by simple scaling and translating) with a *multi-layer* neural network. Consider a L-layer neural network $\mathcal{F}_{L\text{-layer}}$ as described above. Given a sample of input-output pairs $(x_1, y_1), \ldots, (x_n, y_n)$ generated from an unknown fixed function f, one can approximate f from $\mathcal{F}_{L\text{-layer}}$ by learning the appropriate parameters by minimizing the following error function.

$$\begin{aligned} E(W_1, b_1, \ldots, W_L, b_L) &:= \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{F}_{L\text{-layer}}(x_i) - y_i\|^2 \\ &= \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{N}^L \circ \cdots \circ \mathcal{N}^1(x_i) - y_i\|^2. \end{aligned}$$

Assuming that our input data is one dimensional, that is each $x_i \in \mathbb{R}$ and $y_i \in [0, 1]$, implement a gradient descent procedure to learn the parameters of a two-layer (ie, $L = 2$) feed forward neural network. Note that since each y_i is 1-dimensional, layer $\mathcal{N}^2$ only contains one neuron. Layer $\mathcal{N}^1$ can contain an arbitrary number, say k, neurons.

Graphically the network you are implementing looks as follows.



Here is the pseudocode of your implementation.

Learn 2-layer neural network for 1-d functions
input: data $(x_1, y_1), \ldots, (x_n, y_n)$,
. size of the intermediate layer k
- Initialize weight parameters $(W_1, b_1), (W_2, b_2)$ randomly
- Repeat until convergence:
- for each training example (x_i, y_i)
- compute the network output $\hat{y}_i$ on x_i
- compute gradients $\frac{\partial E}{\partial W_2}$, $\frac{\partial E}{\partial b_2}$, $\frac{\partial E}{\partial W_1}$, $\frac{\partial E}{\partial b_1}$
- update weight parameters:
- $W_2^{\text{new}} := W_2 - \eta \frac{\partial E}{\partial W_2}, \quad b_2^{\text{new}} := b_2 - \eta \frac{\partial E}{\partial b_2}$
- $W_1^{\text{new}} := W_1 - \eta \frac{\partial E}{\partial W_1}, \quad b_1^{\text{new}} := b_1 - \eta \frac{\partial E}{\partial b_1}$

You must submit your code on Courseworks to receive full credit.

(iv) Download `hw2data.mat` from the course website. It contains two vector valued variables X and Y. Each row in Y is a noisy realization of applying an unknown smooth 1-dimensional function to the corresponding row in X. You can visualize the relationship between these variables by plotting X on the x-axis and Y on the y-axis.

Use your implementation from part (iii) to learn the unknown mapping function which can yield Y from X. Once the network parameters are learned, plot the network output on the y-axis (in red) along with the given Y values (in blue) for each input value X on the x-axis.