Homework Assignment 2

CSE 190: Neural Networks

Fall 2015

Instruction

Due on Sunday, Nov 1^{st} 11:59pm PST

1. You may have a group of up to 3 people to finish this homework assignment.
2. You are encouraged to make a LaTeX report for each assignment. You are free to choose the template you want.
3. You can submit the homework electronically to **both** TAs' Emails, or submit a hard-copy to the TA's office (EBU3B 4154).
4. You may use a language of your choice (Python and MATLAB are recommended) but all source code used in the assignment must be attached in the appendix. Please keep the code clean with explanatory comments, as they may be reused in the future.
5. Using the MATLAB toolbox for neural networks or any off-the-shelf code is strictly prohibited. You cannot use theano, mathematica or any other software to help you with the gradient.
6. Late policy: You will get deduction from the total score for: a) 0-1 day:10%; b) 1-2 days: 20%; c) 2-3 days: 50%; d) more than 3 days: 100%;
7. Any form of cheating, lying, or plagiarism will not be tolerated. Discussions on course materials and homework solutions are allowed, but you should write the final solutions alone. Books, notes, and Internet resources can be consulted, but not copied from. Working with people outside your team on homeworks must follow the (spirit of) Gilligan's Island Rule (Dymond, 1986): No notes can be made during a discussion, and you must watch one hour of Gilligan's Island or equally mindless TV before writing anything down. Suspected cheating will be reported to the Dean.

Multi-layer Neural Networks

In this problem, we will continue classifying handwritten digits from Yann LeCun's MNIST Database. In Homework Assignment 1, we classified the digits using a single-layer neural network with different output activation functions. (Logistic and Softmax regression). In this assignment, we are going to classify the full MNIST database using multi-layer neural networks.

Problem

1. In class we discussed two different error functions: sum-of-squared error (SSE) and cross-entropy error (Xent). We learned that SSE is appropriate for linear regression problems where we are trying to fit data generated from:

$$t = h(x) + \epsilon, \tag{1}$$

where x is a K-dimensional vector, $h(x)$ is a deterministic function of x and ϵ is random noise distributed as a Gaussian with zero mean and variance σ^2, i.e. $\epsilon \sim \mathcal{N}(\mu = 0, \sigma^2)$. Suppose we are trying to model this data with a linear function approximator with parameter vector θ:

$$f(x, \theta) = \sum_{k=0}^{K} \theta_k x_k \tag{2}$$

Proof. Please prove that finding the optimal parameter θ for the above linear regression problem on the dataset $D = \{(x^{(1)}, t^{(1)}), ..., (x^{(N)}, t^{(N)})\}$ is equal to finding the θ that minimizes the SSE:

$$\theta^* = \text{argmin}_\theta \sum_{i=1}^{N} (t^i - f((1, x^{(i)}), \theta))^2 \tag{3}$$

where $(1, x^{(i)})$ is our data supplemented with an additional 1 (representing x_0) to incorporate a bias term.

2. For multiclass classification on the MNIST database, we previously used softmax regression with cross-entropy error (Xent) as the objective function, and learned the weights of a single-layer network to classify these digits. In this assignment, we will add a hidden layer between the input and output consisting of J units with the sigmoid activation function, i.e., the network now has three layers: an input layer, a hidden layer and a softmax output layer.

   Notation: We use index k to represent a node in output layer and index j to represent a node in hidden layer and index i to represent a node in the input layer. Additionally, the weight from node i in the input layer to node j in the hidden layer is w_{ij}.

   Please complete the following tasks,

   (a) **Derivation** Derive the expression for δ for both the units of output layer (δ_k) and the hidden layer (δ_j). Recall that the definition of δ is $\delta_i = -\frac{\partial E}{\partial a_i}$, where a_i is the weighted sum of the inputs to unit i,

   (b) **Update rule.** Derive the update rule for w_{ij} and w_{jk} using learning rate α, starting with the gradient descent rule: $w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$.

   (c) **Vectorize computation.** The computation is much faster when you update all w_{ij}s and w_{jk}s at the same time, i.e. you update the weight matrices as matrices, rather than using **for** loops. Please show the update rule for the weight matrix from the hidden layer to output layer W_{HO} and the matrix from input layer to hidden layer W_{IH}.

   (d) **Classification.** Classification on MNIST datatbase.

      i. Read in the data from the MNIST database, preprocess the data by z-scoring the pixels across the 784 pixels of each data point, resulting in vectors of $x \in \mathbb{R}^{784}$ with 0 mean and unit standard deviation.

      ii. Check your code for computng the gradient using a small subset of data. You can compute the slope with respect to one weight using the numerical approximation:

      $$\frac{d}{dw} E(w) \approx \frac{E(w + \epsilon) - E(w - \epsilon)}{2\epsilon}$$

      where ϵ is a small constant around 10^{-5}. Compare the gradient using numerical differences with the one using backpropagation. Note that w here is *one* weight in the network, so this must be repeated for *every* weight and bias. Report your results.

      iii. Using the update rule you obtained from 2(c), perform gradient descent to learn a classifier to map the input data into the labels $y \in \{0, ..., 9\}$. Choose your own stopping criteria. You should use cross-validation to decide the stopping criteria. (*Hint*: You may choose to split the training set of 60000 images to two subsets: one training set with 50000 training images and one validation set with 10000 images.) Report your training procedure and plot your training and test accuracy vs. number of training iterations of gradient descent.

(e) **Experiment with Regularization.** Starting with the network of 2(d), add weight decay to the update rule. (You will have to decide the amount of regularization, i.e., λ, a weight we add to weight decay. Experiment with 0.001 and 0.0001) Report training and test accuracy vs. number of training iterations of gradient descent. Comment on change of performance.

(f) **Experiment with Momentum.** Start with the network of 2(d). Add a momentum term γ into the update rule, and set it to 0.9. Report training and test accuracy vs. number of training iterations of gradient descent. Comment on change of performance.

(g) **Experiment with Activations.** Start with the network of 2(d). Try using different activation functions for the hidden units. You are already using the sigmoid, try these other two. Note that the derivative changes!

   i. Sigmoid. $f(z) = \frac{1}{1+e^{-z}}$
   ii. Tanh. $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
   iii. ReLU. $f(z) = \max(0, z)$

   Show change of the update rule. Report training and test accuracy vs. number of training iterations of gradient descent. Comment on change of performance.

(h) **Experiment with Network Topology.** Start with the network of 2(d). Now, we will consider how the topology of the neural network changes the performance.

   i. Try halving and doubling the number of hidden units. What do you observe if the number of hidden units is too small? What if the number is too large?
   ii. Change the number of hidden layers. Use two (2) hidden layers instead of one. Create a new architecture that utilizes two hidden layers of equal size and has approximately the same number of parameters, i.e., roughly the same total number of weights and biases. Report training and test accuracy vs. number of training iterations of gradient descent.