Homework 4

Due: Nov 17, 2022 at 11:59pm EST

Instructions: This homework requires answering some open-ended questions, short proofs, and programming. This is an individual assignment, not group work. Though you may discuss the problems with your classmates, you must solve the problems and write the solutions independently. As stated in the syllabus, copying code from a classmate or the internet (even with minor changes) constitutes plagiarism. You are required to submit your answers in pdf form (use LaTeX) in a file called `hw3.pdf` to Gradescope under "HW4". Code should also be submitted to Gradescope in a file called `hw4.py` under "HW4 Programming". A LaTeX template for the pdf submission and a code skeleton for the code submission are available on Piazza. Please do **not** modify the base code in the coding skeleton; simply augment it with your solution. Late submissions do not receive full credit, except in extenuating circumstances such as medical or family emergency. Submissions submitted 0-24 hours late are eligible for only 90%, 24-48 hours late for 80%, 48-72 hours late for 70%, and later than 72 hours for 0% of the total credit for this assignment. Late days may be used (if available) to avoid these penalties.

The total assignment is worth 80 points.

Problem 1 (18 points)

This question pertains to the convergence of stochastic gradient descent. Let n be the total numer of samples in the data. The update performed by stochastic gradient descent at each step is then:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla L_i(\theta^{(t)}),$$

where $\nabla L_i(\theta^{(t)})$ is the gradient for a single sample i chosen uniformly at random, where $1 \leq i \leq n$.

As before, we will assume assume $L(\theta)$ is twice differentiable and its gradient is Lipschitz continuous with constant $K > 0$; we will not assume $L(\theta)$ is convex until part (iv).

(i) Starting from the second order Taylor approximation from HW1 shown below,

$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) + \nabla L(\theta^{(t)})^T(\theta^{(t+1)} - \theta^{(t)}) + \frac{1}{2}(\theta^{(t+1)} - \theta^{(t)})^T K(\theta^{(t+1)} - \theta^{(t)}),$$

prove that the following inequality holds (3 points):

$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) - \frac{\alpha}{2}\nabla L_i(\theta^{(t)})^T \nabla L(\theta^{(t)}) + \frac{\alpha^2 ||\nabla L_i(\theta)||^2 K}{2}$$

(ii) Argue, based on (i), that even if we set the learning rate as some positive constant $\alpha \leq 1/K$, this does not guarantee that SGD produces a lower loss at every step (even if we assume $||\nabla L_i(\theta)||^2$ is bounded.) (5 points)

(iii) We now prove that although the loss is not guaranteed to decrease for any single step in SGD, it is guaranteed to decrease in expectation. If we treat i as a random variable whose value in the range $1, \ldots, n$ is selected uniformly at random with probability $1/n$, our goal is to show that $\mathbb{E}[L(\theta^{(t+1)})] \leq L(\theta^{(t)})$, where the expectation on the left is taken over the distribution of i, the learning rate is $\alpha \leq 1/K$, and the variance of $\nabla L_i(\theta)$ is bounded by some relatively small positive constant σ^2. (7 points)

For full credit on this proof, briefly note why each expression simplifies, and write down any assumptions you made along the way to arrive at your final result.

Hint: It helps to start by taking the expectation of both sides of the inequality proved in (i).
Hint 2: Keep in mind the expectation and variance formulas for a random variable.

(iv) When $L(\theta)$ is also convex, if you continue the proof using the inequality you derived in (iii) and repeat steps very similar to the proof in Problem 1, HW2 (completing the square etc.), you get the following result:

$$\mathbb{E}[L(\theta^{(N)})] - L(\theta^*) \leq \frac{||\theta^{(0)} - \theta^*||^2}{2\alpha N} + \alpha\sigma^2,$$

where N is the total number of iterations of SGD and θ^* corresponds to the optimal parameter values.

Comment on how the guarantee of convergence of SGD differs from that of regular GD. (3 points)

Problem 2 (8 points)

(i) Draw a DAG representing the following sequence of operations (feel free to take a picture of a neat hand-drawn figure for your answer, i.e., it does not have to be computer generated.) (3 points)

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 0 \\ x_3 &= x_1 + x_2 \\ x_4 &= x_1 + x_3 \\ x_5 &= x_3 \times x_4 \\ L &= \log(x_5) \end{aligned}$$

Here, $\log(\cdot)$ refers to the natural logarithm.

(ii) Derive all partial derivatives $\frac{\partial L}{\partial L}$ and $\frac{\partial L}{\partial x_i}$ using backpropagation. You do not need to explicitly show all your work for full credit, but show the intermediate steps in the figure you drew for (i) if you'd like to receive partial credit in case of a mistake. (5 points)

Problem 3 (16 points, 4 points per question)

Justify why the following commonly applied tricks lead to more stable gradients when training neural networks with stochastic gradient descent.

Note: the answers to these questions should focus on improvements to the stability of the gradients and the descent procedure itself, rather than downstream training/test accuracy.

(i) Using a learning rate that decays over time according to some preset schedule/formula, e.g., dividing the learning rate at each iteration by the root of the sum of the squares of the gradients of all past iterations.[1]

[1] Side note: When we do this separately for each individual parameter, this results in an adaptive learning rate for each parameter in our model. This particular variation of the algorithm is called Adagrad.

(ii) Using the ReLU activation function in place of the sigmoid or tanh activation functions.

(iii) Gradient clipping, i.e., truncating the value of gradients to some maximum value for any examples where $||\nabla L_i(\theta)||^2 > G$ for some constant $G > 0$.

(iv) Applying dropout.

Extra credit: For up to 2 extra credit points: formally justify your answers, where possible, with reference to how the trick affects the Lipschitz constant of the gradients of $L(\theta)$.

Problem 4 (8 points)

(i) Describe 3 conceptually distinct ways of improving the bias-variance tradeoff of an overparametrized neural network (a network that has many more parameters than number of training samples.) (4 points)

(ii) Are there cases when training the same neural network (in terms of number of layers, width of each layer, activation functions etc.) multiple times on the same training data until convergence lead to (a) the exact same results and (b) different results? Why? (4 points)

Problem 5 (30 points)

In this problem you will implement and apply neural networks with dropout based on the code skeleton provided in `hw4.py` and helper functions provided in `data_loader.py`. There are two datasets provided to test your implementation: the first one is simulated and generated within the code itself, and the second is a song characteristics/popularity dataset from Spotify. The features in the Spotify dataset include features that may be helpful for building a machine learning model e.g., tempo of the song, valence, duration, as well as features that are probably not as useful, e.g., ID of the song and its name. The outcome of interest is whether the song is considered popular or not (1 corresponds to a song that crosses some pre-defined popularity threshold and 0 is a song that does not meet this threshold.)

There is less helper code provided for performing the real data analysis this time around. The idea is to help you get a feel for some of the data pre-processing and model building steps required to produce a good working model. This should help prepare you for the kinds of analysis you will undertake for your final project.

(i) Extend the `Value` class with `exp` and `log` methods. (4 points)

(ii) Implement the `sigmoid` and `negative_loglikelihood` functions. For the sigmoid function, we will be using the formula $\text{sig}(x) = 1/(1 + e^{-\text{scale}*x})$, where the scale parameter controls how quickly the sigmoid function approaches 1. (4 points)

(ii) Implement the `fit` function using the negative loglikelihood as the loss function. (6 points)

(iii) Implement regularization via dropout as discussed in lecture. (6 points)

(iv) Produce a model for the Spotify dataset that achieves a final test accuracy of 70% or higher using appropriate machine learning practices. A checklist of things to watch out for in your analysis. (6 points)

- Drop any variables that are not suitable for building an ML model.

- Standardize the data. Remember to compute the means and standard deviations only using the training set (to avoid information leak between training and testing, i.e., you should also treat the means and standard deviations as parameters you learn from your training data.)
- Try at least 3 different architectures, and select the best one using a validation set. Some ideas: Don't tune dropout probability and max epochs, set these to 0.5 and 25 or 50 respectively (depending on your learning rate.) Instead, try a shallow but relatively wide network, a narrow but deep network, and one that is somewhere inbetween. You probably don't need to go much higher than a total of 200 parameters in a network: this is intentional so that you don't spend too much time waiting for the model to train.
- Resist the temptation to optimize on test error – use the validation set to pick your best model

(v) What is the test accuracy and architecture (number of layers, width of each layer, dropout probability, and learning rate) of the model you have chosen? Would you consider this accuracy sufficient for deployment? (3 points)

(vi) What is the role of the scale parameter in the sigmoid function? (1 point)

Important submission note:
Report all answers to the above questions in the PDF that you turn in. Hand in your Python code separately on Gradescope under HW 4 Programming. Please make sure your variables and functions are aptly named, and your code is clear and readable, with comments where appropriate.