Homework III

STAT/CS 6685/5685 - Fall semester 2023

Due: Friday, October 13, 2023 - 5:00 PM

Please upload all relevant files & solutions into Canvas. Include your answers (minus code) in a single PDF titled `lastname_firstname_HW3.pdf`, and all your code in a single jupyter notebook titled `lastname_firstname_HW3Code.ipynb`. Any requested plots should be sufficiently labeled for full points.

Unless otherwise stated, programming assignments should use built-in functions in Python, Tensorflow, and PyTorch. In general, you may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Stater code is provided for all homeworks at `https://github.com/KevinMoonLab/STAT-6685`. You are not required to use this code, although it is highly recommended, especially if you are new to Python.

Problem 1 - 30 points

1. (3 pts) An extreme version of gradient descent is to use a mini-batch size of just 1. This procedure is known as online or incremental learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning compared to stochastic gradient descent with a mini-batch size of, say, 20. (Adapted from the Nielsen book, chapter 1)

2. (27 pts) Create a network that classifies the MNIST data set using only 4 layers: the input layer (784 neurons), two hidden layers (you choose the number of nodes), and the output layer (10 neurons). Train the network using stochastic gradient descent on the training data. For full credit, you will need to obtain a test accuracy of at least 97%. You will need to vary the number of nodes in the hidden layer and the learning rate. You may use Tensorflow or Pytorch for this problem. Starter code is available at `https://github.com/KevinMoonLab/STAT-6685/blob/master/HW-3/MNIST_LittleTuned.ipynb`.

Please include your code and state which library/framework you used. Report your final accuracy, the learning rate, the number of nodes in the hidden layer, and the mini-batch size you used. The code of downloading the MNIST data has already been provided in the starter code. However, if you don't using the start code, you may find the MNIST data at `https://www.kaggle.com/datasets/oddrationale/mnist-incsv/data`.

Problem 2 - (6685 - 16 pts) - (5685 - 26pts)

1. (**6685** - 5 pts, **5685** - 10 pts) Alternate presentation of the equations of backpropagation (Nielsen book, chapter 2)
   Show that $\delta^L = \nabla_a C \odot \sigma'(z^L)$ can be written as $\delta^L = \sum'(z^L)\nabla_a C$, where $\Sigma'(z^L)$ is a square matrix whose diagonal entries are the values $\sigma'(z^L_j)$ and whose off-diagonal entries are zero.

2. (**6685** - 5 pts, **5685** - 10pts) Show that $\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$ can be rewritten as $\delta^l = \Sigma'(z^l)(w^{l+1})^T\delta^{l+1}$.

3. (3 pts) By combining the results from problems 2.1 and 2.2, show that

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \dots \Sigma'(z^{L-1})(w^L)^T\Sigma'(z^L)\nabla_a C.$$

4. (3 pts) Backpropagation with linear neurons (Nielsen book, chapter 2)
   Suppose we replace the usual non-linear σ function (*sigmoid*) with $\sigma(z) = z$ throughout the network. Rewrite the backpropagation algorithm for this case. You will need to update the four fundamental equations of backpropagation to do this.

Problem 3 - (6685-24pts) - (5685-14pts)

1. (3 pts) It can be difficult at first to remember the respective roles of the ys and the as for cross-entropy. It's easy to get confused about whether the right form is $-[y \ln a + (1-y)\ln(1-a)]$ (the correct form) or $-[a \ln y + (1-a)\ln(1-y)]$ (the incorrect form). What happens to the second of these expressions when y=0 or 1? Does this problem afflict the first expression? Why or why not? (Nielsen book, chapter 3)

2. (**6685**-5pts) A common approach to derive a loss function is to use the negative log-likelihood of a model distribution, and then compute its expected value with respect to the empirical distribution ($\hat{P}_{data}$):

$$L(\theta) = -\mathrm{E}_{x,y\sim\hat{P}_{data}} \log P_{model}(y|x,\theta). \tag{1}$$

   In this problem, you will derive the binary cross entropy loss by using a Bernoulli distribution as a model for the conditional distribution of the label y given the inputs x.

Suppose we have a neural network with a single sigmoid activation in the final layer. We can write its output as $\hat{y}(\boldsymbol{x}, \boldsymbol{\theta})$, making explicit that it depends on the input vector of the network $\boldsymbol{x}$, as well as on the weights and biases of the network, which are our parameters $\boldsymbol{\theta}$. Now, using equation (1) and a Bernoulli distribution for $P_{model}(y|\hat{y})$, derive the binary cross entropy loss. **Hint:** $\hat{y} = P(y = 1|\boldsymbol{x}, \boldsymbol{\theta})$.

3. (**6685**-5pts) We can similarly derive the cross entropy loss function for multiple classes by defining a multinoulli distribution for $P_{model}(y|x, \theta)$. With this model, y can take multiple values (i.e. classes) and not just two. If we compute a softmax activation in the final layer of our network, we will get $\hat{y_k}(\boldsymbol{x}, \boldsymbol{\theta}) = P(y_k = 1|\boldsymbol{x}, \boldsymbol{\theta})$, where $k \in \{1, 2, \ldots, K\}$, K being the total number of classes. $y \in \mathrm{R}^K$ is a one-hot vector. Thus $y_k = 1$ means the kth entry of y is equal to one, and the rest are zeros. Using equation (1) and a multinoulli distribution for $P_{model}(y|\hat{y})$, derive the cross entropy loss.

4. (3 pts) Show that the cross-entropy cost function is still minimized when $\sigma(z) = y$ for all training inputs (i.e. even when $y \in (0, 1)$), where $\sigma(z)$ is the output of the neural network when $\mathbf{x}$ is the input. When this is the case the cross-entropy has the value: $C = -\frac{1}{n}\sum_{\mathbf{x}}[ylny + (1-y)ln(1-y)]$ (Nielsen book, chapter 3). This implies that cross-entropy can also be used for regression as long as the expected outputs are all between 0 and 1.

5. (3 pts) Where does the softmax name come from? (Nielsen book, chapter 3)

6. (5 pts) Backpropagation with softmax and the log-likelihood cost (Nielsen book, chapter 3). To apply the backpropagation algorithm for a network containing sigmoid layers to a network with a softmax layer, we need to figure out an expression for the error $\delta_j^L = \partial C/\partial z_j^L$ in the final layer. Show that a suitable expression is: $\delta_j^L = a_j^L - y_j$.

Problem 4 - 30 pts

In this problem, you will implement gradient descent to perform regularized logistic regression. Download the file `mnist_49_3000.mat` from Canvas. This is a Matlab data file that contains a subset of the MNIST dataset. Specifically, it contains examples of the digits 4 and 9. The data file contains variables x and y, with the former containing the image of the digit (reshaped into column vector form) and the latter containing the corresponding label ($y \in \{-1, 1\}$). To visualize an image, you will need to reshape the column vector into a square image. You will also need to transform the label vector y so that all values of -1 become 0 (based on the way our loss function is defined below).

Define the point $\tilde{\mathbf{x}}_i = [\mathbf{x}_i^T, 1]^T$, that is the d-dimensional data point $\mathbf{x}_i$ with a 1 appended to it. Logistic regression is a linear classifier. That is, it has the form $f(\mathbf{x}) = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$ where $\tilde{\mathbf{w}} = [\mathbf{w}^T, w_0]^T$ where w_0 corresponds to the offset and $\mathbf{w}$ is a d-dimensional weight vector. The classifier assigns a data point $\mathbf{x}$ to class 1 if $f(\mathbf{x}) \geq 0$ and to class 0 otherwise. The loss function for regularized logistic regression can be written as

$$\ell(\mathbf{w}, w_0) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\frac{1}{1+\exp(-\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i)}\right) + (1-y_i)\log\left(\frac{\exp(-\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i)}{1+\exp(-\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i)}\right)\right] + \lambda||\mathbf{w}||^2,$$

where λ is the regularization parameter that is set by the user.

1. (**6685 - 5 pts**) Derive the gradient of this loss function with respect to the weights and offset, i.e. $\nabla \ell(\mathbf{w}, w_0)$.

2. (20 pts) Implement and fit a logistic regression model using PyTorch (or Tensorflow). That is, define the parameters of your model, your loss function, your optimizer and your training procedure using PyTorch (or Tensorflow). Try setting $\lambda = 0.01$ and $\lambda = 1$. Vary your learning rate until you get good convergence results. Use the first 2000 examples as training data and the last 1000 as test data. Report the test error for both λ values, your termination criterion (you may choose), your learning rate, how you initialized the weights and offset, and the value of the objective function (i.e. the loss function) at the optimum.

   Hint: To do this, you can modify the architecture you used for Problem 1.3. Think about what the input and output dimensions are, how many hidden layers you need, what loss function you will use, and what activation function you will use.

3. (**6685 - 5 pts / 5685 - 10pts**) Generate a figure displaying 20 images in a 4×5 array. These images should be the 20 misclassified images for which the logistic regression classifier was most confident about its prediction (i.e., the classifier was confident but wrong). To define a notion of confidence, recall that the logistic regression classifier models the posterior probability as $p(y = 1|\mathbf{x}) = \frac{1}{1+\exp(-\tilde{\mathbf{w}}^T \tilde{\mathbf{x}})}$. Thus the logistic regression classifier is most confident for data points that have either a very large (close to 1) or very small (close to zero) value of $p(y = 1|\mathbf{x})$. In the title of each subplot of the misclassified images, indicate the true label of the image. What you should expect to see is a bunch of 4s that look like 9s and vice versa.

References

[1] "The scipy stack specification." [Online]. Available: https://www.scipy.org/stackspec.html