S&DS 355 / 555

Introductory Machine Learning

Assignment 7

Out: Thursday, November 14, 2019
Due: Tuesday, December 3, 2019

This assignment involves two problems on neural networks. The first problem involves training a neural network for classification of a toy dataset, using a "bare bones" implementation that only uses `numpy`. The second problem asks you to experiment with deep autoencoder networks for a dataset of synthetic images of clothing.

We are providing starter code that is essentially all you need—you will only be required to modify this in limited ways, and not write a lot of new code.

Submission instructions

For this assignment you should submit five files to Canvas—a file `gradients.pdf` with solutions to Problem 1(a) and Problem 2(a), together with the Python notebooks and their pdf versions for `minimal_neural_network.ipynb` and `autoencoder.ipynb`. As usual, these pdfs are obtained by first saving the notebook as html to preserve the cell structure and then exporting as pdf. Please include your NetID (and not your name) in the first cell of the notebooks.

1. ***Neural networks for classification*** (30 points)

In this problem you are asked to work with a simple and direct `numpy` implementation of back-propagation for training a two-layer neural network. Your job will be to understand how this code works and then extend it to train a three-layer network.

Let H_1 denote the number of hidden units, let D be the dimension of the input X, and let K be the number of classes. For the spiral data implemented in the code, we have $D = 2$ and $K = 3$. The two-layer network has the following form,

$$\begin{aligned} h_1 &= \text{ReLU}(W_1 X + b_1) \\ p &= \text{Softmax}(W_2 h_1 + b_2) \end{aligned}$$

where the parameters of the network are $W_1 \in \mathbb{R}^{H_1 \times D}$, $b_1 \in \mathbb{R}^{H_1}$, $W_2 \in \mathbb{R}^{K \times H_1}$, $b_2 \in \mathbb{R}^K$. Recall that ReLU is the "rectified linear unit" $\text{ReLU}(x) = \max\{0, x\}$ applied componentwise, and Softmax maps a d-vector to the probability simplex according to

$$\text{Softmax}(v)_k = \frac{\exp(v_k)}{\sum_{j=1}^K \exp(v_j)}.$$

Note that this can be thought of as a nonlinear logistic regression model. For a three-layer network, we add another hidden layer of dimension H_2, and change the model to

$$\begin{aligned} h_1 &= \text{ReLU}(W_1 X + b_1) \\ h_2 &= \text{ReLU}(W_2 h_1 + b_2) \\ p &= \text{Softmax}(W_3 h_2 + b_3) \end{aligned}$$

where the matrices W_j and vectors b_j are of the appropriate dimensions; specifically, $W_1 \in \mathbb{R}^{H_1 \times D}$, $b_1 \in \mathbb{R}^{H_1}$, $W_2 \in \mathbb{R}^{H_2 \times H_1}$, $b_2 \in \mathbb{R}^{H_2}$, $W_3 \in \mathbb{R}^{K \times H_2}$, $b_3 \in \mathbb{R}^K$. For a given input X, this specifies how to calculate the probabilities $p(Y = k \mid X)$ for $k = 1, 2, \ldots, K$.

(a) ***Gradients for a 3-layer network*** (5 points). For a given training set $\{(X_i, Y_i)\}_{i=1}^n$, consider the loss function

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n -\log p(Y = Y_i \mid X_i).$$

Give formulas for the derivatives

$$\frac{\partial \mathcal{L}}{\partial W_j}, \ \frac{\partial \mathcal{L}}{\partial b_j}$$

for each layer of the network $j = 1, 2, 3$. Include your solutions in the file `gradients.pdf`.

Hints:

i. You should ignore the fact that $\text{ReLU}(x)$ is not differentiable at $x_i = 0$.
ii. $\frac{\partial \mathcal{L}}{\partial W_j}$ should be a matrix of the appropriate dimension, and $\frac{\partial \mathcal{L}}{\partial b_j}$ should be a vector.
iii. You might try to first do the calculations with $\text{ReLU}()$ replaced by the identity function.
iv. The derivatives for a two-layer network are implemented in the code provided, and in the notes referenced in lecture. This essentially gives you the answers! But beware that the notation above may not exactly match how things are set up in the code.

(b) ***Extend the code from two layers to three layers*** (15 points). Run the code provided in the notebook `minimal_neural_network.ipynb` and inspect it to be sure you understand how it works. (We started this in class!) Then, after working out the derivatives in part (a) above, extend the code by writing a function that implements a 3-layer version. Your function declaration should look like this:

```
def train_3_layer_network(H1=100, H2=100)
```

where `H1` is the number of hidden units in the first layer, and `H2` is the number of hidden units in the second layer. Then train a 3-layer network and display the classification results in your notebook, showing the decision boundaries as is done for the 2-layer network in the starter code.

(c) ***Experiment with different parameter settings*** (10 points). Now experiment with different network configurations and training parameters. For example, you can train models with different numbers of hidden nodes `H1` and `H2`. Train at least three and at most five networks. For each network, display the decision boundaries on the training data, and include a Markdown cell that describes its behavior relative to the other networks you train. Specifically, comment on how the different settings of the parameters change the bias and variance of the fitted model. How does the 2-layer network compare to the 3-layer network?

Your code for parts (b) and (c) above should be contained in a single notebook `minimal_neural_network.ipynb` that you convert to HTML and then print as `minimal_neural_network.pdf`, submitting both files to Canvas.

2. ***Autoencoders*** (30 points)

This problem concerns various forms of autoencoder networks. Recall from class that autoencoder networks are unsupervised learning algorithms that recover representations of data by "squeezing" a high dimensional input through a bottleck of hidden units, and then generating an output from those hidden units. The network is trained to try to make the reconstruction close to the original input.

The first problem asks you to do a gradient calculation for a simple type of autoencoder network—this is similar to the first part of the previous problem. The remaining parts ask you to experiment with an implementation of autoencoders that uses Keras, a popular interface to the TensorFlow deep learning toolkit. We do not require that you write any substantial Python code for this problem.

(a) ***Gradients for a simple autoencoder*** (5 points). In a simple autoencoder, an input $x \in \mathbb{R}^D$ is reconstructed as $\widehat{x} \in \mathbb{R}^D$ by first encoding x to a hidden representation h, and then decoding h to $\widehat{x}$. The least squares objective is to minimize the squared error

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \widehat{x}_i\|^2$$

over a training sample $\{x_i\}_{i=1}^n$. Suppose that the encoder network is of the form

$$h = \text{ReLU}(W_1 x + b_1)$$

where $W_1 \in \mathbb{R}^{H\times D}$ and $b_1 \in \mathbb{R}^H$, and the decoder network is of the form

$$\widehat{x} = \text{ReLU}(W_2 h + b_2)$$

where $W_2 \in \mathbb{R}^{D\times H}$ and $b_2 \in \mathbb{R}^D$. Then the objective function becomes

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \text{ReLU}(W_2\text{ReLU}(W_1 x_i + b_1) + b_2)\|^2.$$

Give formulas for the gradients

$$\frac{\partial \mathcal{L}}{\partial W_j}, \; \frac{\partial \mathcal{L}}{\partial b_j}$$

for both the encoder and decoder networks, $j = 1, 2$. Include your solutions in the pdf file `gradients.pdf` used for Problem 1(a). Note that this architecture would not be a good one to use in a real application. Why? Give two reasons.

Hints: The same hints offered for Problem 1 apply here! (Except that it might be hard to look into the Keras code to find gradients.)

(b) ***Training various autoencoders on Fashion MNIST*** (15 points). The Jupyter notebook `autoencoder.ipynb` comes preloaded with the training of a series of autoencoders on the MNIST data. In this problem, your task is to re-run this after swapping the MNIST data with the "Fashion MNIST" data. Then, you should write documentation to explain what is actually being done in the code, and to describe the results obtained with the trained autoencoders, including the squared error on the test set. Specifically:

- Replace the line

  `(x_train, y_train), (x_test, y_test) = mnist.load_data()`

  with the line

  `(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()`

  Then re-run all of the cells. This will train a series of autoencoders on the data described here: `https://github.com/zalandoresearch/fashion-mnist`.

- The code trains a series of autoencoders: shallow, deep, and denoising. Each has a placeholder markdown cell for you to fill in. For each of the cells marked "*your documentation goes here*" give a text description of what the following code is doing. What size networks are being trained? How are they being trained? What is the purpose of this particular variant of the autoencoder, and how does it differ from a vanilla shallow encoder? Give a sufficient description that it is clear you understand what the code is doing, and could explain it to another person who is familiar with some of the basics of neural networks.

- After each model is trained, several sets of weights and reconstructed images are shown. You should modify the code to print out more sample images and network weights, to capture more of the variation in the data and the weights.

- After the weights are shown, there is a placeholder cell marked "Discussion of above results *[your description goes here]*". Fill in this cell with a description of the results. Is the model fully trained? How can you tell? How does it seem to perform compared to some of the other models? How do the reconstructions differ qualitatively from the original images? Is the model overfitting or underfitting the data? How could the training be improved? Include any other aspects you wish, thinking critically about the behavior of the models.

(c) ***Training a series of networks of varying complexity*** (10 points). Finally, fit a series of autoencoders of the "shallow" variety, varying the dimension of the bottleneck, which is the `code_size` parameter. Plot the training error and test error as a function of the code size. Describing your findings. Interpret them in terms of the bias/variance tradeoff.