TDT4265: Computer Vision and Deep Learning

Assignment 1

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

January 28, 2019

- **Delivery deadline: January 31, 2018 by 2359.**
- **This project count towards 6% of your final grade.**
- You can work on your own or in groups of up to 2 people.
- Upload your code as a single ZIP file.
- Upload your report as a PDF file to blackboard.
- You can choose what language and frameworks you use. Numpy is allowed to use with python, but any deep learning framework is not allowed in this assignment (Pytorch/tensorflow, matlab DL kit etc). If you're in doubt if a framework is allowed, ask a TA!
- We recommend you to use Python with numpy in this assignment. Matlab works as well.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

Introduction

In this assignment we will explore the power of gradient descent to perform binary classification with Logistic Regression. Then, we will explore multi-class classification with Softmax Regression on the MNIST dataset. Further, you will experiment with weight regularization through L2 norm, simple visualization of weights and basic visualization to present your result.

1 Logistic and Softmax Regression (1.5 points)

1.1 Logistic Regression

Logistic regression is a simple tool to perform binary classification. Logistic regression can be modeled as using a single neuron reading a input vector $x \in \mathbb{R}^{d+1}$ and parameterized by a weight vector $w \in \mathbb{R}^{d+1}$. d is the dimensionality of the input, and we add a 1 at the beginning for a bias parameter, known as the "bias trick". The neuron outputs the probibility that x is a member of class C_1.

$$P(x \in C_1|x) = g_w(x) = \frac{1}{1 + e^{-w^T x}} \tag{1}$$

$$P(x \in C_2|x) = 1 - P(x \in C_1|x) = 1 - g_w(x) \tag{2}$$

where $g_w(x)$ simply notes that g_w is parameterized by w. $g_w(x)$ returns the probability of x being a member of class C_1; $g_w \in [0, 1]$. By defining our output of our network as y^n, we have $y^n = g_w(x)$. With this hypothesis function, we use the cross entropy loss function (Equation 3) for two categories to measure how well our function performs over our dataset. This loss function measures how well our hypothesis function g_w does over the N data points.

$$E(w) = -\frac{1}{N} \sum_{n=1}^{N} t^n \ln(y^n) + (1 - t^n) \ln(1 - y^n) \tag{3}$$

Here, t^n is the target value for example n. Note that we are taking the mean over all training samples N; this is to ensure that our cost function is not dependent on number of training examples. Our goal is to minimize this cost function through gradient descent, and it is reaches a minimum of 0 when $t^n = y^n$ for all n.

Task 1.1: Derive the gradient for Logistic Regression (0.75 points)

To minimize the cost function with gradient descent, we require the gradient of the cost function in Equation 3. Show that for the logistic regression cost function, the gradient is:

$$-\frac{\partial E^n(w)}{\partial w_j} = (t^n - y^n) x_j^n \tag{4}$$

Show thorough work such that your approach is clear.
Hint: You can use the fact that:

$$\frac{\partial g_w^n}{\partial w_j} = x_j^n g_w^n (1 - g_w^n) \tag{5}$$

1.2 Softmax Regression

Softmax regression is simply a generalization of logistic regression to multi-class classification. Given a input x^n, softmax regression will output a vector y^n, where each element y_k^n represents the probability that x^n is a member of class k.

$$y_k^n = \frac{e^{a_k^n}}{\sum_{k'} e^{a_{k'}^n}} \tag{6}$$

where

$$a_k^n = w_k^T x^n \tag{7}$$

Here a_k^n is called the net input to unit y_k. Note each output has its own weight vector w_k. Equation 6 is known as the Softmax function and it has the attribute that $\sum_k y_k^n = 1$.

With our model defined, we now define the cross-entropy cost function for multiple classes in Equation 8.

$$E = -\sum_n^N \sum_{k=1}^C t_k^n \ln(y_k^n) \tag{8}$$

Again, taking the average of this number over training samples, N, makes the function independent over different training set sizes. Also, averaging this over number of categories, C, makes it independent over the number of categories.

Task 1.2: Derive the gradient for Softmax Regression (0.75 points)

For the softmax function in Equation 8, show that the gradient is:

$$-\frac{\partial E^n(w)}{\partial w_{kj}} = x_j^n(t_k^n - y_k^n) \tag{9}$$

w_{kj} is the weight from unit j to unit k. A few hints if you get stuck:

- Derivation of the softmax is the hardest part. Break it down into two cases.
- $\sum_{k=1}^C t_k^n = 1$
- $\ln(\frac{a}{b}) = \ln a - \ln b$

2 Logistic Regression through Gradient Descent (2.5 points)

Read in data

In this assignment you are going to start classifying digits in the well-known dataset MNIST. The MNIST dataset consists of 70,000 handwritten digits, split into 10 object classes (the numbers 0-9). The images are 28x28 grayscale images, and every image is perfectly labeled. The images are split into two datasets, a training set consisting of 60,000 images, and a testing set consisting of 10,000 images.

Each image is 28x28, so the unraveled vector will be $x \in \mathbb{R}^{784}$. For each image, append a '1' to it, giving us $x \in \mathbb{R}^{785}$ (This is the bias trick). To download and read the dataset, we have provided code in the file "mnist.py". To load the data, look at the code snippet in Listing 1. We recommend you to use the first 20,000 images in the training set and the **last** 2,000 images in the test set (these are the hardest images in the test set). If you use anything more, please note this in your report. Note that the first half of these images are written by postal workers, representing "clean" images, while the second half is written by high school students. *Hint:* it is recommended to use a much smaller size while developing your code(like 100) for debugging purposes.

```
import mnist
mnist.init()
X_train, Y_train, X_test, Y_test = mnist.load()
```

Listing 1: How to load the MNIST dataset with the mnist.py file (given in assignment files). (Works with Python 3!)

Task 2.1: Logistic Regression through gradient descent (1.5 points)

Now, using the gradient derived from logistic regression cross entropy loss, use the gradient to classify $x \in \mathbb{R}^{785}$ (there is one extra term for the bias term) for the two categories 2's and 3's. The target is 1 if the the input is from the "2" category, and 0 else. Remove all numbers that are not a 2 or 3. Then implement the mini-batch gradient descent algorithm (shown in listing 2). We recommend you to vectorize the code with numpy, which will make the runtime of your code significantly faster! (This is not required, but highly recommended).

```
1. Shuffle all training examples
2. w_0 ← 0
3. for t = 0, ..., m do
4.       w_{t+1} = w_t − α Σ_{n=1}^{N} ∂E^n(w)/∂w_kj
5. return w
```

Listing 2: The mini-batch gradient descent algorithm for m batches and a single epoch. N is the batch size and α is the learning rate.

Annealing learning rate: Experiment with different learning rates and a annealing learning rate by reducing it over time. A possible annealing learning rate schedule might be: $\alpha(t) = \frac{\alpha(0)}{1+t/T}$, where $\alpha(0)$ is the initial learning rate and T is a new hyperparameter. The initial learning rate should be small(e.g: 0.001). Experiment with different learning rates and settle for one that works for you.

Early Stopping: Early stopping is a tool to decide how many training iterations is required. Setting fixed number of training iterations is a difficult task; therefore, split your training set into a train set and validation set(the validation set is usually 10% of the previous training set). This is to prevent us from "data-snooping" on the test set. As you train your network on, regularly test your network on the validation set after each epoch (an "epoch" is one pass through the whole training data). If the cost function (error) on the validation set begin to go up consistently (lets say 3 epochs), we stop the training and return the weights at the minimum validation loss. Use the weights for your minimal validation loss for your answer for the following tasks.

For your report, please:

(a) *(1.2pt)* Decide on a good learning rate. Once this is found, plot the loss function (in Equation 3) for the training set and holdout set. The loss function should be computed over the whole training set and validation set after each epoch. Hold the weight fixed when computing this.

The test set is usually not available in the "real world"; however, since we are not in the real world, but stuck in academica, compute the loss on the test set as well. Plot all three losses on the same graph. (We recommend matplotlib for plotting). Plot the loss on the y-axis and number of training iterations of gradient descent on the x-axis.

If your algorithm learns this task over a few epochs (less than 5), compute the losses every 1/10th epoch, or even more frequently. This is to give us a gradual progress of our network. We are looking for a smooth curve.

Does the validation set work as a good stand-in for the test set? Discuss shortly.

(b) *(0.3 pt)* Plot the percent classified correctly over training for the training set, validation set and test set. Again, plot this on the same graph. This can be done simultanously as computing the loss. Count a classification as right if the input is a "2" pattern and the output is ≤ 0.5, and vice-versa for the other pattern.

Task 2.2: Regularization (1 point)

One way to improve generalization is to use regularization. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Regularization is the idea that we should penalize the model for being too complex. In this assignment, we will carry this out by introducing a new term in our objective function to make the model "smaller" by minimizing the weights.

$$J(w) = E(w) + \lambda C(w), \tag{10}$$

where $C(w)$ is the complexity penalty and λ is the strength of regularization (constant). For large λ the model ignores the error function and reduces only the complexity of our model. There are several forms for C, such as L_2 regularization

$$C(w) = ||w||^2 = \sum_{i,j} w_{i,j}^2, \tag{11}$$

where w is the weight vector of our model. Or L_1 regularization

$$C(w) = |w| = \sum_{i,j} |w_{i,j}| \tag{12}$$

For your report, please:

(a) *(0.2pt)* Derive the update term for logistic regression for gradient descent in J with respect to w, for L_2 regularization. All you have to do is figure out $\partial C/\partial W$

(b) *(0.4pt)* Implement L_2 regularization, and train logistic regression for 2 vs 3. Do this for different values of λ: 0.01, 0.001, 0.0001. Plot the validation accuracy for different values of λ on the same graph (during training). What do you observe? Which λ value works best for you?

(c) *(0.2pt)* For the same points you plot the accuracy, plot the length of the weight vector for the different λ values during training. What do you observe?

(d) *(0.2pt)* Plot the weights in each value of λ as an image (ignoring the bias). What do you observe?

3 Softmax Regression through Gradient Descent (2.0 points)

Use the gradient derived for softmax regression loss, use gradient descent to perform 10-way classification. Use a validation set and use L_2 regularization with a λ value that works best for you (The λ value that works best for you is the one performing best on the validation set).

One-hot encoding: With multi-class classification tasks it is required to one-hot encode the target values. Convert the target values from integer to one-hot vectors. (E.g: $3 \rightarrow [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$). The length of the vector should be equal to the number of classes (10 for MNIST).

For your report, please:

(a) *(1.2pt)* Plot the loss function over the number of training iterations for the train, validation and test set. You don't need to make different plots for different λ values, but state in your report what λ value you use and why you decided for this value. Plot this in the same graph.

(b) *(0.3pt)* Plot the percent correct over the number of training iterations for the train, validation and test set for the same λ. Plot this in the same graph.

(c) *(0.5pt)* Create an image of the weights for each digit. Compare this to an image of the average of all examples of this digit in the portion of training set you used for changing the weights (plot them side-by-side). Discuss shortly. This should all be in one figure.

4 Bonus - Nesterov Momentum(0.5 points)

The maximum number of points in this assignment is 6 points and you can not exceed this score. The bonus is a chance for you to get max score on the assignment even though some of the previous tasks are incorrect.

The method of momentum is a modification of the standard gradient descent algorithm. It is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

Implement the Nesterov Momentum algorithm. The algorithm is explained in Chapter 8.3.3 in Goodfellow et al. (2016). Also, this stackoverflow post has a good visualization of how it works in practice.

For your report, please:

(a) *(0.35pt)* Plot the loss for the train, validation and test set in the same graph.

(b) *(0.15pt)* Compare the convergence of your model with and without nesterov momentum. Plot the loss of the validation set over training for a model trained with and for a model trained without nesterov momentum. Plot this in the same graph. What do you observe? Discuss shortly

5 Delivery

Write a report detailing the work you have done. Include all tasks in the report, and mark it clearly with the task you are answering (Task 1.1, Task1.2, Task 2.1a etc). The report should be in a single PDF file.

For the plots in the report, ensure that they are large and easily readable. You might want to use the "ylim" function in the matplotlib package to "zoom" in on your plots. Label the different graphs such that it is easy for us to see which graphs corresponds to the train, validation and test set.

Include all the necessary code in a single ZIP file. Make sure that your code is well documented and easily readable. Do not include any unnecessary files or folders(do not include the MNIST dataset in your ZIP file).

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.