EECS 6327 (Fall 2016) Project One

Matthew Tesfaldet

Nov 3, 2016

Part I

Feature Extraction and Data Virtualization

In this exercise we studied the effects of various dimensionality reduction methods on the MNIST dataset. Namely, Principle Component Analysis and Linear Discriminant Analysis. We will be analyzing the digits 3, 5, and 9 from the training set. The dimensionality of this dataset is 28x28 $\rightarrow$ 784.

Our first task was to estimate the PCA projection matrices (details and explanation in **pca.ipynb**) and plot the total distortion error of these images as a function of some used PCA dimensions, as shown in Figure 1. Naturally, the fewer PCA dimensions we use for projecting the features onto, the higher the feature reconstruction error. Total distortion error was measured as the MSE (mean-squared error) between the reconstructed features and the original features. As shown, if we use all PCA dimensions (784) for projection, then the reconstruction error is zero.

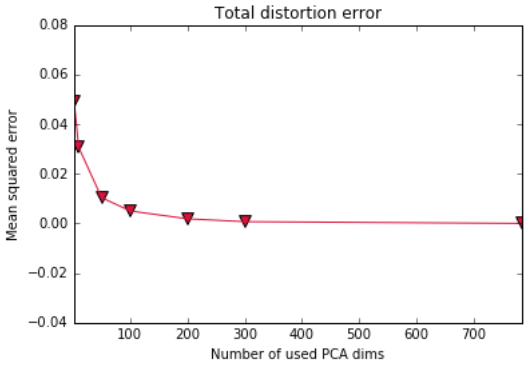

Figure 1: PCA Total Distortion Error

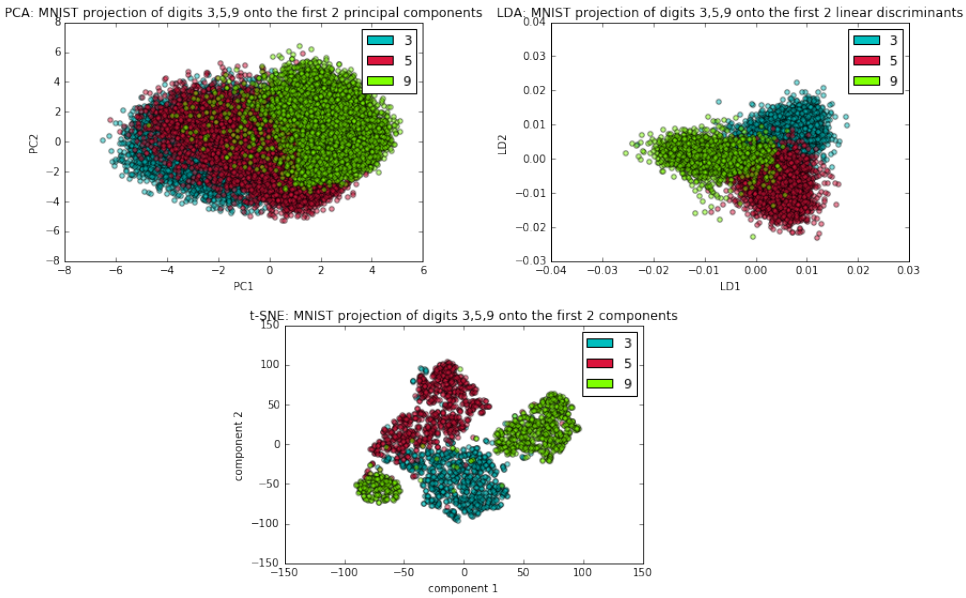

Figure 2: Projection into 2D Space

Our second task was to use PCA, Linear Discriminant Analysis (LDA), and t-SNE (t-Distributed Stochastic Neighbor Embedding) to project these images into 2D space and plot them with three different colors for data virtualization. For the sake of easing computational complexity, the first 2500 digits for each of 3, 5, and 9 were used for t-SNE. The results are shown in Figure 2. As we can see for PCA, the data has been mean centered and there's a good amount of energy captured using these principle components, but there's plenty of overlap between classes. LDA allowed for a better class separation while maintaining high variance within each class, however, there is still some overlap. t-SNE provides a projection that is almost completely devoid of class overlap. This is due to it learning a high-dimensional manifold that fits the data, and because of this it takes time to train a t-SNE projection. t-SNE is useful for visualizing high-dimensional data and thus is useful for interpreting the classification performance of a high-dimensional classifier (e.g. plotting decision boundaries).

Part II
Linear and Logistic Regression

In this exercise we implemented two types of linear classifiers that classified between digits in the MNIST dataset. The dataset consists of 60000 training images and 10000 test images. In order to validate our classifier performance,

we held out 10000 validation images from the training set, resulting in 50000 training images and 10000 validation images. We used the training data to learn two 10-class classifiers using linear regression and logistic regression, and reported the best possible classification performance in the held-out test images. For simplicity, we made use of the *bias trick* to incorporate the bias in the weights as an extra column. Also, all images were mean-centred. As for all classifiers used in this project, the input is X with size $n - samples \times d - dimensions$ and the labels y have size $n - samples \times 1$. Due to page limitations, confusion matrices and precision-recall summaries for all classifiers were not included in this report, however, they are included in the Python notebooks (read the included README).

Linear Regression

Here we built a multiclass linear regression model to classify our data. Specifically, we used Ordinary Least Squares (OLS) as our classifier. Our model is in the form $y = X\beta$, where the least squares solution solves the problem $\hat{\beta} = \arg\min \|X\beta - y\|_2$. The closed form solution is $\hat{\beta} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$.

Since the closed form solution assumes a binary classification, and the problem set is separated between 10 classes, we created multiple one vs. all classifiers where the vector y was separated to 10 one vs. all label vectors $y^{(j)}$ where $j \in \{0, 1, \ldots, 9\}$ and $y_i^{(j)} = 1$ if x_i is digit j and $y_i^{(j)} = -1$ otherwise. Thus, a separate classifier $\hat{\beta}^{(j)}$ was solved for each digit j. At test time, inference consisted of a matrix multiplication between the test image and each classifier; the image is labeled correspondingly to the classifier with the highest score. We arrived at a classification accuracy of 86.06% on the test set, not far off from the 88% accuracy reported on the MNIST site for a 1-layer neural net (the closest comparison).

Logistic Regression

Here we built a multi-class logistic regression model to classify our data. Specifically, we used the softmax activation and cross-entropy loss:

$$\text{Softmax:} \qquad P(y_i|x_i;W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

$$\text{Cross-entropy:} \qquad H(\hat{y_i}, P(y_i|x_i;W)) = -\sum_i \hat{y_i} \log P(y_i|x_i;W)$$

Where $f_{y_i} = f(x_i;W) = x_iW$ is our prediction for data point x_i, W is our weights with size $d - dimensions \times c - classes$, y_i is the label of data point x_i, and $\hat{y_i}$ is the one-hot representation of y_i. Weights and biases were initialized to zero. Since the exponentials can produce very large values, a normalization trick is performed to prevent numerical instability; we multiply the top and bottom

by a constant C and push it into the sum; we choose $\log C = -\max_j f_j$. This states we should shift our predictions so that the highest value is zero. As we can see, only a single classifier is needed as it is capable of k-class classification. For optimization, we used an iterative approach: mini-batch stochastic gradient descent. We used a batch size of 600 and trained for 2000 iterations (24 epochs). In order to test out different combinations of learning rates and regularization strengths, we implemented cross-validation with learning rates $l \in \{1, 7\text{e}{-1}, 5\text{e}{-1}\}$ and regularization strengths $r \in \{0, 0.25, 0.5\}$. We arrived at a classification accuracy of 92.28% on the test set with $l = 1$ and $r = 0$. The MNIST site does not report results using logistic regression, but in theory it should perform as well as linear SVM (which we report next). Interestingly, we arrived at similar test accuracies with the other learning rates, as long as $r = 0$. Increasing regularization decreased overall accuracy. When $r = 0.25$, changing the learning rates didn't change validation or training accuracy.

Part III
Support Vector Machines (SVMs)

In this exercise we implemented a linear and non-linear classifier that classified between digits in the MNIST dataset. Specifically, we used the training data to train a linear SVM and non-linear SVM using a Gaussian RBF kernel.

Linear SVM

Here we trained a multi-class linear SVM model to classify our data. The objective was a modified hinge loss function [5] that was minimized:

$$L_i = \sum_{j \neq y_i} \max(0, x_i w_j - x_i w_{y_i} + \Delta)$$

where w_j is the j-th row of W reshaped as a column and $\Delta = 1$. Figure 3 gives more detail. Weights and biases were initialized to zero.

We used the same cross-validation settings as for training our logistic regression classifier and arrived at a classification accuracy of 92.2% on the test set with $l = 7\text{e} - 1$ and $r = 0$. The MNIST site does not report results using linear SVM, so the closest comparison was with a 1-layer neural net, in which we achieve comparable results to their pairwise linear classifier accuracy of 92.4%. However, they de-skewed their dataset whereas we did not. Although we mean-centered ours. We also made use of the same *bias trick* as was used for the logistic regression classifier. A similar phenomenon to logistic regression was noticed where we arrived at similar test accuracies with all learning rates as long as $r = 0$. Also, increasing regularization decreased overall accuracy and lowering the learning rate mitigates this decrease.

Figure 3: The Multiclass Support Vector Machine "wants" the score of the correct class to be higher than all other scores by at least a margin of delta. If any class has a score inside the red region (or higher), then there will be accumulated loss. Otherwise the loss will be zero. Our objective will be to find the weights that will simultaneously satisfy this constraint for all examples in the training data and give a total loss that is as low as possible.

Non-Linear SVM

Here we trained a multiclass non-linear SVM model to classify our data. Here we trained a multiclass non-linear SVM model to classify our data. We maximized the dual objective $\sum_i \alpha_i + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ with respect to the Lagrange multipliers α_i (initialized to zero) using batch dual coordinate ascent. After each learning step, the constraints $\sum_i \alpha_i y_i = 0$ and $0 \leq a_i \leq C$ (where $C = 10$) were applied. We used the Gaussian RBF as our kernel, where $K(x_i, x_j) = \exp\left\{-\gamma \|x_i - x_j\|^2\right\}$. We trained multiple one vs. all classifiers, one for each digit. For each classifier, we balanced the dataset it was trained on. For example, if we were training on the digit "0", we would obtain 5000 positive examples ("0") and 5000 negative examples (all other digits), resulting in a perfectly balanced dataset. We arrived at a classification accuracy of 95.52% on the test set with a learning rate $l = 7\text{e} - 3$ and $\gamma = 0.02682696$ for classifiers trained on digits "0", "3", "5", "7", "8", and "9", $\gamma = 0.01$ for the classifier trained on digit "1", and $\gamma = 0.03727594$ for the classifiers trained on digits "2", "4", and "6". We trained for 48 epochs. The highest reported test set accuracy on the MNIST site using a non-linear svm with the Gaussian RBF kernel is 98.6%. We're not too close, but due to the difficulty in choosing the right parameter γ and that we implemented our own quadratic optimizer from scratch, our 95.52% result is not bad. It beats our linear svm by 3.32%.

Part IV
Deep Neural Networks (DNNs)

In this exercise we implemented two different two-layer neural nets that classified between digits in the MNIST dataset. Our first architecture consisted of the following layers: input → fully connected layer → PReLU → fully connected layer → softmax activation → cross-entropy loss. Our second architecture consisted of the following layers: input → fully connected layer → sigmoid activation → fully connected layer → sigmoid activation → cross-entropy loss.

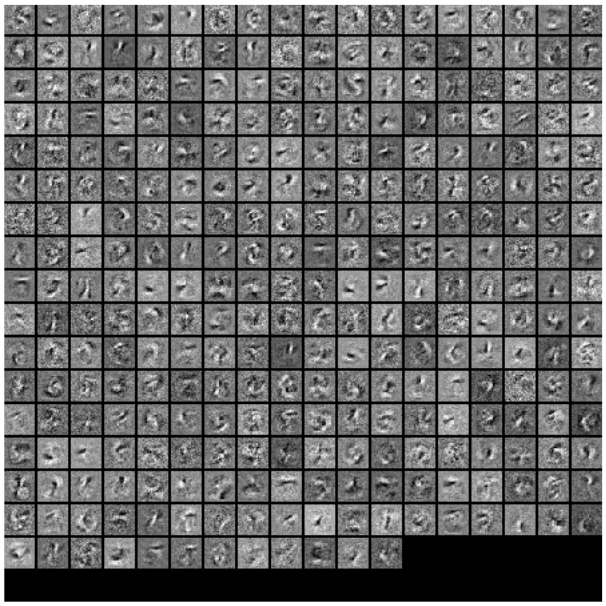

Figure 4: Visualization of Weights (PReLU net) in the First Layer

Weights were initialized to small random values and biases were initialized to zero. The dataset was mean-centred for the PReLU neural net (1% increase in test accuracy) but not for the sigmoid neural net (1% decrease in test accuracy).

We experimented with *Xavier* and *MSRA/Kaiming* [1, 2] weight initialization, but the difference in accuracy was negligble. We cross-validated with learning rates $l \in \{7\text{e}{-1}, 5\text{e}{-1}, 1\text{e}{-1}\}$, regularization strengths $r \in \{0, 1\text{e}{-3}, 5\text{e}{-4}\}$, and hidden-layer sizes $h \in \{30, 100, 300\}$. We used a batch size of 200 and trained for 5000 iterations (20 epochs). Learning rate decay was set at 0.90 and decayed every two epochs. The PReLU leakiness was set to 0. We optimized using mini-batch stochastic gradient descent SGD and experimented with gradient updates using classical momentum [4] and gradient updates using *Nesterov* momentum [3]. Classical and *Nesterov* momentum performed similarly and increased test accuracy by about 0.5% over using no momentum.

We arrived at a classification accuracy of 98.35% for our PReLU neural net on the test set with $l = 5\text{e} - 1$, $r = 0$, and $h = 300$. Our sigmoid neural net performed with a test set classification accuracy of 96.82% with $l = 7\text{e}{-1}$, $r = 0$, and $h = 300$. Not as good as the PReLU net. The closest comparison listed on the MNIST site is 98.4% accuracy with a two-layer NN using 800 hidden units. Our results with the PReLU neural net are favourable since it achieved a comparable accuracy and used fewer hidden units (300 as opposed to 800).

As we can see in Figure 4, the nueral net is learning filters that resemble a series of Gabor filters in the shape of the mean-centered digits.

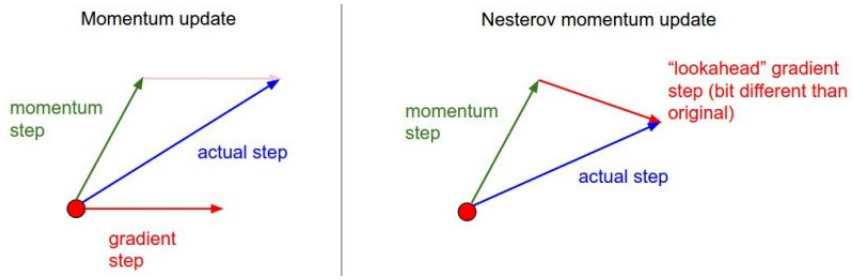

Figure 5: Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

References

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[4] Boris Teodorovich Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[5] Jason Weston, Chris Watkins, et al. Support vector machines for multi-class pattern recognition. In *ESANN*, volume 99, pages 219–224, 1999.