Machine Learning with Insufficient Data Amount

Phan Minh Nguyen

***Abstract*—The big data era posits the basic assumption of data abundance. What should we do when we have a lot of data, but a comparable number of features or parameters to learn? In this project, we explore various supervised learning algorithms for the digit recognition task in this setting. By testing their performances with varying training data size, we draw empirical observations and discuss (unexpected) effects of having too many parameters.**

***Index Terms*—High-dimensional, supervised classification, SVM, CNN, MNIST, digit recognition.**

I. Introduction

How do learning algorithms perform when the number of training examples m is comparable to the number of trainable parameters n? Consider a simple example in regression: least squares. The solution to this is well known: $\theta = \left(X^T X\right)^{-1} X^T y$, with $X \in \mathbb{R}^{m \times n}$. When $m < n$, it is a linear algebra fact that $X^T X$ is non-invertible! In such case, the solution is not even unique, and overfitting becomes a major concern.

This shows that the case where $m \approx n$ (or even $m \ll n$) exhibits certain phenomena that are not encountered in the common scenario $m \gg n$. One may naively think that this is the extreme of data shortage, e.g. when m is as low as a few tens. Here we specifically concern with the *high-dimensional*[1] setting: $m \approx n$, and both m and n are large. The second requirement ensures some sort of laws of large numbers would work, eliminating the uninteresting high variance due to low dimensions, and hence focusing on the effects of $m \approx n$.

The setting is not entirely new. It is frequently the case in genomics (see e.g. [HTF09, Example 4, p.5]), and can potentially happen when a learning system is scaled up. For example, the VGGNet neural network system was trained with 138 million parameters on the ImageNet data set, which has about 14 million images (and in its associated 2014 ImageNet ILSVRC challenge where VGGNet set the record, just 1.2 million training images) . See [CS16] for a summary. As another example, the classic text The Elements of Statistical Learning [HTF09] gives a whole chapter on this setting in its second edition in 2009. While it is not new, it is also not completely understood.

In this project, we explore this setting for supervised classification, in particular, the digit recognition task with the well-known MNIST data set. This data set contains 28×28 images (and hence 784 unprocessed features) of hand-written digits from 0 to 9. We opt for $m = 100, 250, ..., 10000$. In doing so, we randomly choose m images from the MNIST training set, such that the number of images per class is roughly $m/10$, i.e. the same for each class. The aim is to classify a new test image with as high accuracy as possible. Note that we easily (and unavoidably) get a training error of 0 for most of the times, and so we only focus on the test error as our performance metric.

[1]Not to be confused with the setting where n is very large, yet $m \gg n$.

The algorithms to be explored here include: k-nearest neighbor (k-NN) with Euclidean distance, multi-class linear regression (MLR), SVM with linear kernel (linear SVM), SVM with degree-2 polynomial kernel (2-poly SVM), degree-4 polynomial kernel (4-poly SVM), SVM with radial / Gaussian kernel (radial SVM), and convolutional neural network (CNN). The LIBSVM library [LIBb] is used for SVM. To speed up simulations, in certain experiments, we opt for the squared-loss SVM (2-SVM), instead of the traditional linear SVM, and use the LIBLINEAR library [LIBa]. We also use LIBLINEAR for MLR. To simulate CNN, we use the Tiny DNN library [DNN]. To tune hyper-parameters, we do cross-validation or at least training/validation splitting with the ratio 70:30.

In the rest of the paper, we discuss our experimental results and key observations in each section. Some experimental results are not shown due to space constraints, but are briefly mentioned wherever applicable.

II. Overall Comparison of All Algorithms

Fig. 1 presents the performances of the algorithms without regularization. Unsurprisingly CNN works consistently best among all. MLR does not perform so well. Notably certain algorithms, such as k-NN and 4-poly SVM, are severely hurt when m is small. For k-NN, this is somewhat expectable. Intuitively the performance of k-NN depends on how well-spread and dense the training data is in the space. Since when $m \ll n$, the data points can only sparsely occupy the n-dimensional space, which can lead to high bias. This curse of dimensionality is well-known (see [HTF09, Section 2.5]).

It is surprising however that a strongly kernelized SVM like 4-poly SVM performs much worse, whereas the radial SVM, another strongly kernelized SVM, is consistently the runner-up and approaches closely the performance of CNN. In light of that fact, it is also surprising that 4-poly SVM is worse than 2-poly SVM. While one may argue that overfitting makes 2-poly SVM a better choice than 4-poly SVM, how could one possibly extend such argument for radial SVM?

At this point, it seems unreasonable to depend entirely on how strong a learning algorithm is (or is perceived to be), as we could be inclined to when $m \gg n$. It also seems that there is some special structure in the MNIST data set that favors the radial kernel. In other words, in this setting, should we pay more attention to the key (i.e. the algorithm) or the lock (i.e. the data)? This insight can be found in common arguments for CNN in image classification tasks, that CNN is a suitable

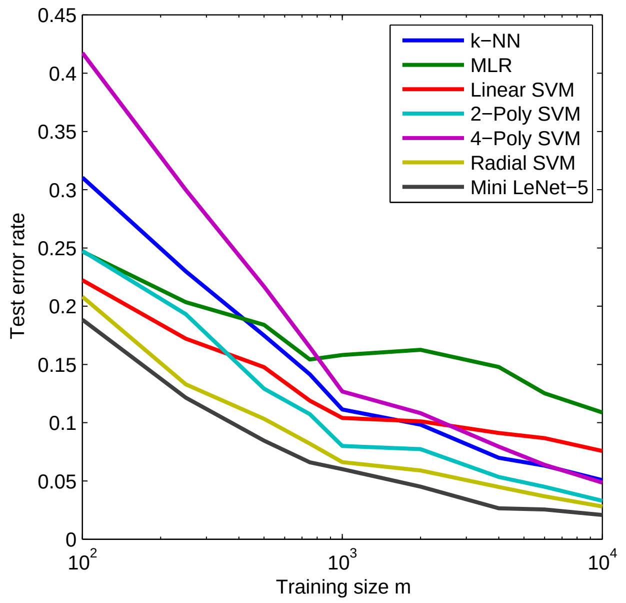

Figure 1. Test error vs training size m. Here Mini LeNet-5 refers to a CNN architecture described in Section VI and is taken as the representative for CNNs.

choice due to layering interpretation of images. We pause here and do not attempt to resolve the issue further.

III. Regularization is (Still) Very Helpful

We investigate the use of ℓ_2^2 regularization on 2-SVM (henceforth in this section, SVM for brevity) and MLR. The regularization simply adds the term $\|\theta\|_2^2$ to the objective function. For example, in binary 2-SVM, we solve the following[2]:

$$\min_{\theta}\left[C\sum_{i=1}^{m}\left(\max\left\{0,1-y^{(i)}\theta^{T}x^{(i)}\right\}\right)^{2}+\frac{1}{2}\|\theta\|_2^2\right]$$

Fig. 2 and 3 show the results. Here in both figures, C is a regularization hyper-parameter: the smaller C is, the heavier the regularization. It can be observed that regularization is very helpful in both cases.

At a first glance, one may argue that regularization limits overfitting. Yet as noted in [HTF09, Section 18.3],

> *"... when [$m \ll n$] the unregularized [SVM] often works about as well as the best regularized version. Overfitting often does not seem to be a problem..."*

It was proven that, in layman terms, for binary SVM, if the training set is separable, then the test error is $O\left(\frac{1}{m}\right)$ independent of n [Vap95, Theorem 5.2], and so having $n \gg m$ does not worsen the performance. In our context, when m is much lower than 784, the training error without regularization is 0, meaning the training set is "separable." While our problem is a multi-class one, we may hope to see a similar conclusion, that overfitting is not really the issue.

It is notable that MLR exhibits a similar behavior to SVM under ℓ_2^2 regularization, and that regularization helps even

[2]SVM is more complicated for multi-class classification. For this, LIBLINEAR employs one-versus-all strategy.

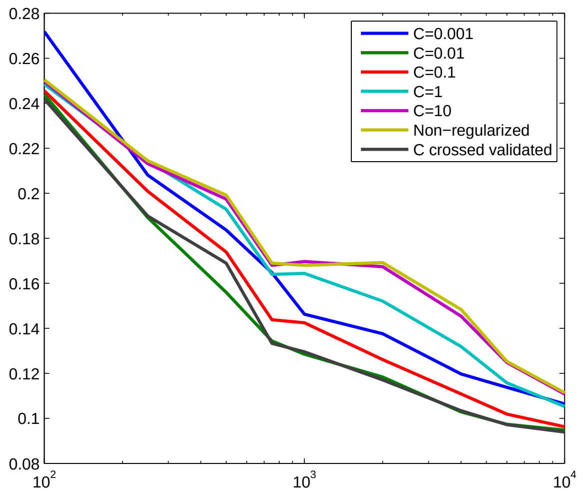

Figure 2. Test error vs m, for ℓ_2^2-regularized 2-SVM.

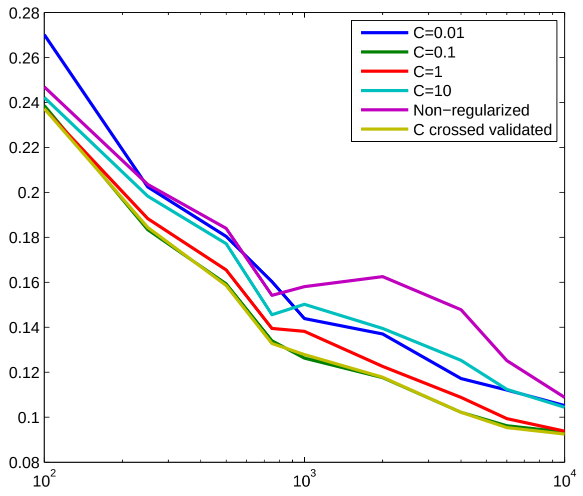

Figure 3. Test error vs m, for ℓ_2^2-regularized MLR.

when $m \gg n$. One remark is that the employed library LIBLINEAR can cause a performance gap from its predecessor LIBSVM in multi-class classification, and so the results here may not necessarily reflect the true behavior of the algorithms. If we assume the library works well in that aspect, all these perhaps call into question the notion of overfitting. Suppose there is structure in the data, characterized by a single number $s < n$. Is it fair to mark $m \approx n$ as a sign of overfitting, or should we use $m \approx s$? If so, should we see regularization as simply a way to combat overfitting? Despite leaving many questions unanswered, we end the section restating the fact that empirically regularization helps and is a should-do.

IV. Dimensionality Reduction with PCA is Not Great

Since the high-dimensional setting $m \approx n$ or $m \ll n$ exhibits certain peculiarities, it is an idea to convert back to the common setting $m \gg n$, in particular, by using dimensionality reduction via the principal component analysis (PCA). Note that this is unsupervised data preprocessing; we do not consider supervised PCA here. Fig. 4 and 5 show the results for 2-SVM, in the cases $m = 750$ and $m = 4000$. A similar behavior can be observed when applying PCA to MLR, but the results are omitted due to space constraints.

We first discuss the unregularized 2-SVM with PCA (blue curves in the plots). The two cases $m = 750$ and $m = 4000$ display a dichotomy: when $m \gg n$, PCA helps a lot, but when $m \approx n$, PCA not only is unhelpful but also degrades the performance. A common argument of why PCA could help is that it extracts structure out of noise. This argument does not seem to involve how large m and n are. Another attempt is to argue that SVM is designed to work with high dimensions (recall the discussion in Section III), and as such, preprocessing the data should not help. There are 2 problems with such argument: firstly, it does not explain the similar behavior observed for MLR, and secondly, it does not explain why PCA dimensionality reduction degrades the performance.

While we remind the reader of the potential pitfall in using LIBLINEAR as in Section III, we offer here another angle from random matrix theory (RMT). A particularly quick reference is [Dob16]. See also [YZB15]. In layman terms, PCA computes the eigenvalues $\lambda(\hat{\Sigma})$ of the sample covariance matrix $\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)T}$, where the data $x^{(i)}$ is drawn from a zero-mean distribution with true covariance matrix Σ. When $m \gg n$, $\hat{\Sigma} \approx \Sigma$. In high-dimensional settings, however, RMT tells us that the spectrum of $\lambda(\hat{\Sigma})$ is drastically different from the spectrum of $\lambda(\Sigma)$. In fact, it spreads out from $\lambda(\Sigma)$. As such, even if $\lambda(\Sigma)$ is concentrated so that information is well separated from noise, it may not be the case in $\lambda(\hat{\Sigma})$. By discarding some principal components, even if their eigenvalues are small, we could be discarding information.

We now turn attention to regularized 2-SVM with PCA (red curves in the plots). It is remarkable that, firstly, when $m \approx n$, PCA does not severely degrade the performance, and when $m \gg n$, it does not help nor hurt much. A practical implication is that dimensionality reduction with PCA is not too bad, as long as we use regularization and tolerate some small loss. Another hypothesis that one can come up with, in light of the discussion in Section III, is that regularization seeks a particular structure in the MNIST data set, and hence alleviates the loss incurred by PCA.

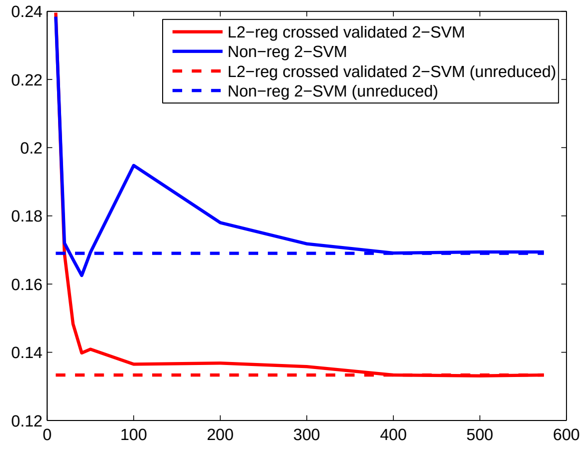

Figure 4. Test error vs reduced dimension through PCA, in the case $m = 750$, for 2-SVM.

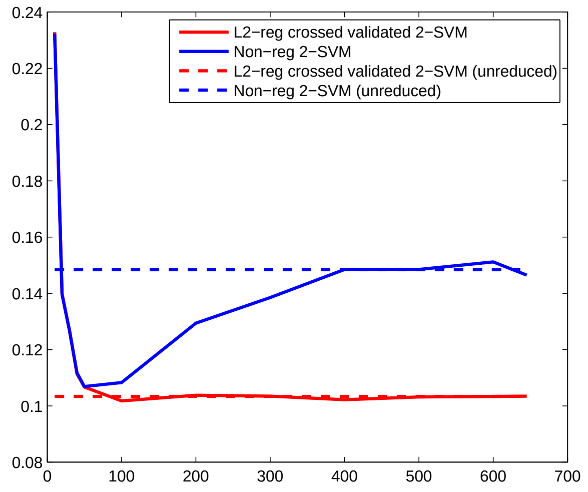

Figure 5. Test error vs reduced dimension through PCA, in the case $m = 4000$, for 2-SVM.

V. More on Structure & the Haar Transform

This section offers another look at the existence of structure, as mentioned in Section III, using the Haar transform to extract features, instead of using raw image pixels. In essence, the Haar transform linearly converts an image vector $x \in \mathbb{R}^n$ to another vector $z \in \mathbb{R}^n$ via an orthonormal matrix $\Phi \in \mathbb{R}^{n\times n}$, i.e. $z = \Phi x$. This transform is known to induce sparsity in z, especially for images, although it is usually the case that x is not sparse. Hence the Haar transform can be viewed as a way to seek structure.

As a word of caution, since Φ is orthonormal, we are effectively not doing anything if the objective function in the learning algorithm is invariant to orthogonal transformations. For example, in linear classification with ℓ_2^2 regularization, since the decision is based on $\theta^T x$, we can always effectively get the same decision with $\Phi^T\theta$ replacing θ, since $\left(\Phi^T\theta\right)^T z = \theta x$, and still retain the same penalization since $\left\|\Phi^T\theta\right\|_2^2 = \|\theta\|_2^2$. As such, to draw meaningful conclusions from the Haar transform, we opt for ℓ_1 regularization instead. This is potentially useful in feature selection, the task where ℓ_1 regularization is a common choice.

The result with 2-SVM is shown in Fig. 6. In Fig. 7, we also plot the size of the selected feature set, averaged over the 10 classes, along with the standard deviation among the classes. Similar results are observed for MLR, but are omitted. While the Haar transform may improve the performance, it is marginal and not as good as with ℓ_2^2 regularization. What is interesting here is that the size of the selected feature set is consistently smaller when using Haar features. Using the Haar

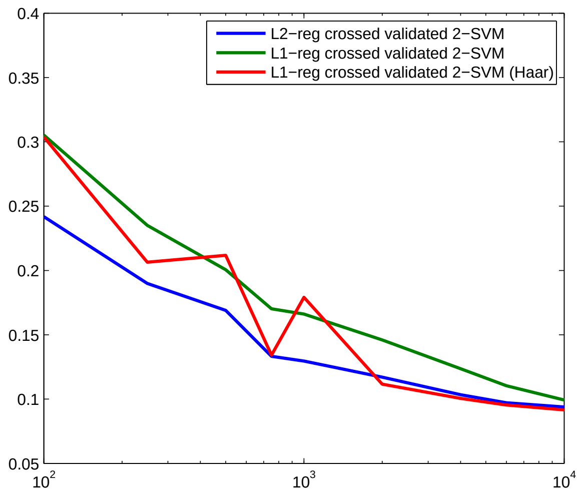

Figure 6. Test error vs m, for 2-SVM. We classify (with ℓ_1 regularization) with the original image, and with the Haar-transformed image.

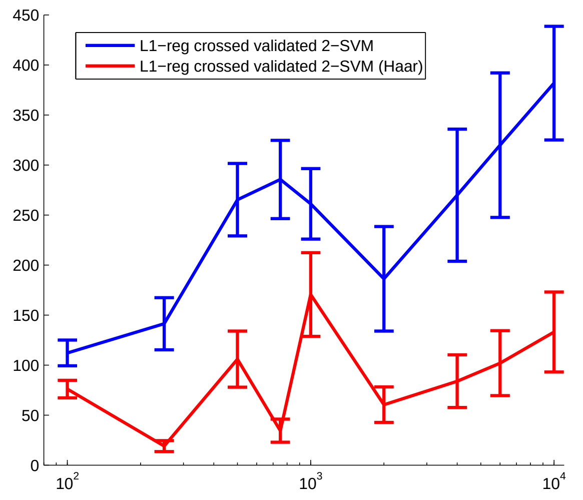

Figure 7. The size of selected feature set vs m, for 2-SVM. We classify (with ℓ_1 regularization) with the original image, and with the Haar-transformed image.

transform therefore yields a more selective algorithm in term of feature selection, for the MNIST data set. This observation holds for all values of m considered. This seems to consolidate the understanding that, even though high-dimensional settings can be very different from the case $m \gg n$, there is some universal structure in the data that influences performances, howsoever the training size is.

VI. Convolutional Neural Network

We spend this section on CNN. It deserves separate attention for several reasons. Firstly, CNN holds the record for image classification tasks. Secondly, as we shall see, we can easily get into the high-dimensional setting with CNN although m can be as large as the values that are considered to be $m \gg n$ previously for e.g. linear classifications. Even if we use the full MNIST training set, we still have $m \approx n$ or $m \ll n$. Thirdly, it is a mechanism designed to automatically select features, and hence extract structure; any preprocessing steps are unnecessary. Somewhat intriguing, for this third reason, CNN usually requires a high number of layers and a large number of trainable parameters, and hence approaches itself to the high-dimensional regime.

A. *CNN Architectures*

We describe the CNN architectures being explored in the project. First we recall key details of the architecture of the celebrated LeNet-5 [LBBH98]. It includes a convolutional layer with 6 feature mappings (C1), then a convolutional layer with 16 mappings (C3), then a convolutional layer with 120 mappings (C5), then two fully connected layers, in which the latter one employs the Gaussian kernel. Between the convolutional layers, there are subsampling layers with trainable parameters (S2 between C1 and C3, and S4 between C3 and C5).

Our CNN architectures are based on LeNet-5. In particular, we preserve the C1-S2-C3 layers, which have a special structure to break symmetry, although this somewhat limits the design of CNN with higher number of layers. The architectures are shown in Table I. Here the depth refers to the number of layers with trainable parameters. The number of convolutional weights refer to the total number of weights in the convolutional layers, which is approximately the total number of trainable parameters. C (i, j) refers to a convolutional layer with j feature mappings, each of which is a $i \times i$ kernel, with stride 1. S (i, j) refers to a subsampling layer with stride j and feature mapping size $i \times i$. All architectures end with one fully connected layer with 120 inputs and 10 outputs (i.e. we discard the layer with Gaussian kernel in LeNet-5, for simplicity). We see that some architectures can be relatively deep, but all are of small sizes.

To train the CNN, we use AdaGrad with mini-batch size of 10. We run the optimization for a varying number of iterations. For $m = 250$, 750, 4000, 10000 and 60000 respectively, the number of iterations is set to be 1000, 500, 500, 200 and 200 respectively. Admitted these choices of iteration number are for convenience and time interest.

We note that these CNNs are run without any forms of regularization. We also have not tried architectures where convolutional layers are stacked up on each other, instead of being followed immediately by a subsampling layers. Hence the observations to be made here may not be immediately generalizable, at least to the aforementioned cases.

B. *Results and Discussion*

Fig. 8 shows the results. We note that even when $m = 60000$ (full MNIST training set), the numbers of convolutional weights of certain CNNs are still comparable, and as such, we do not completely escape the $m \approx n$ regime.

While the results show a general trend that a CNN architecture which works well with large m still works well with small m, there are several interesting observations. By varying the training size m, one can alter the relative performance of

	Convolutional and subsampling layers	Depth	# Conv. weights
Mini LeNet-5	C1 - S2 - C3 - S4 - C5	6	50.8k
CNN-2	C1 - S2 - C3 - S4 - C(2,20) - S(2,2) - C(2,120)	8	13.7k
CNN-3	C1 - S2 - C3 - S4 - C(2,20) - S(2,1) - C(2,40) - S(2,2) - C(1,120)	10	12.1k
CNN-4	C1 - S2 - C3 - S(2,1) - C(2,20) - S(2,1) - C(2,40) - S(2,2) - C(3,120)	10	50.5k
CNN-5	C1 - S2 - C3 - S(2,1) - C(2,20) - S(2,1) - C(2,40) - S(2,1) - C(2,60) - S(2,2) - C(2,120)	12	45.7k
CNN-6	C1 - S2 - C3 - S(2,1) - C(2,20) - S(2,1) - C(2,40) - S(2,1) - C(2,60) - S(2,1) - C(2,60) - S(2,2) - C(1,120)	14	38.5k

Table I
STRUCTURE OF CNN ARCHITECTURES.

the architectures. For example, CNN-4 outperforms CNN-3 for large m, but losses to it for small m. Similarly, compared to CNN-3, Mini LeNet-5 is better for large m and worse for small m. The same behavior with CNN-5 and CNN-6 can be observed. It should be noted that both CNN-4 and Mini LeNet-5 have the same or lower depth than CNN-3 but much more parameters. A similar observation can be made for CNN-5 and CNN-6, although not as noticeable.

From this observation, it is tempting to conclude that overfitting is a more severe problem for CNN architectures with lower depth but higher number of parameters. However there does not seem to be a monotonic formula for the relative performance as a function of m. For example, a racing behavior can be observed between Mini LeNet-5 and CNN-4.

Furthermore, departing from pairwise comparisons, we see that overall a deeper CNN or a CNN with more weights may not perform better. For example, CNN-2 is consistently best, although it is not as deep, and has much less parameters. Note that only in the case $m = 60000$ does CNN-2 depart from the high-dimensional regime. A similar observation can be made for Mini LeNet-5, which works consistently better than CNN-5 and CNN-6, although it is shallower and has a comparable number of parameters.

Overall we see that CNN is still a (charming) mystery. To end this section, we make some practical remarks on the use of CNN based on the above discussion. Note that these conclusions are not well verified, since the experiments are small-scaled and limited.

- In predicting how well a CNN architecture performs, it should not be naively compressed down to two metrics: the number of layers and the number of trainable parameters. There seem to be certain architectures that are suitable with the MNIST data set and will perform well for small and large m.
- If we are just to look at those two metrics when working with a small training set, it may be preferable to have a deeper CNN with less parameters.
- One should take caution when taking a CNN architecture that was trained and shown to work well on a large data set and applying it (with retraining) to a smaller data set.

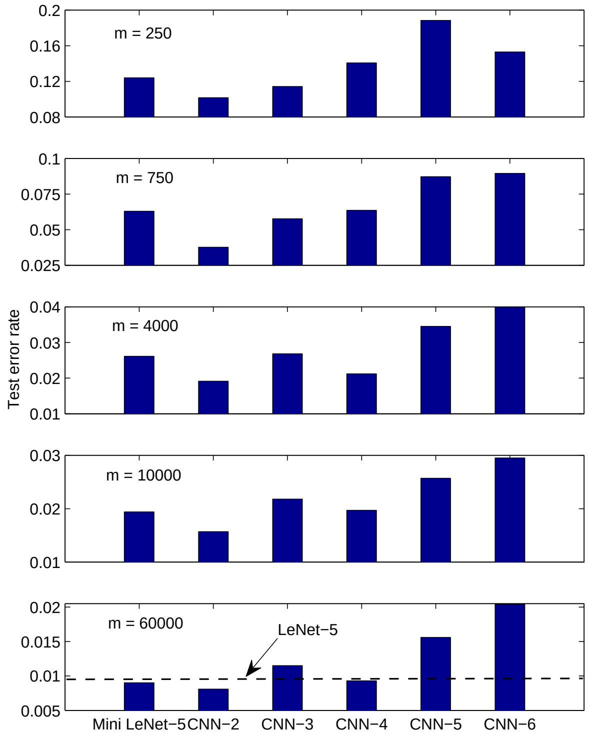

Figure 8. Test error of CNN architectures, for different m. Here LeNet-5 refers to the original LeNet-5 architecture [LBBH98]. Its test error rate reported in [LBBH98] is included for reference. Since LeNet-5 was not trained using AdaGrad, any comparisons do not seem fair.

VII. CONCLUSION: GOOD, BAD, AND UGLY

We summarize the good, the bad and the ugly with the digit classification task on the MNIST data set in the high-dimensional setting $m \approx n$:

- The good: We still have CNN and radial SVM holding the record.
- The bad: One should apply regularization, and may not depend entirely on how strong a learning algorithm is commonly perceived to be. Dimensionality reduction with PCA can fail.
- The ugly: Many mysteries.

For future work, more carefully designed experiments could possibly give more insights. One can also further extend the experiments to other tasks at larger scales. We expect to see quite different behaviors for other tasks.

ACKNOWLEDGMENT

We thank Yin Zi for helpful and thoughtful comments on the project. We also thank the CS 229 course staff for providing the opportunity to embark on this project.

References

[CS16] *CS 231N notes*, http://cs231n.github.io/convolutional-networks/, (accessed December 13, 2016).

[DNN] *Tiny DNN*, https://github.com/tiny-dnn.

[Dob16] Edgar Dobriban, *EigenEdge package for MATLAB, readme.pdf*, https://github.com/dobriban/EigenEdge, (accessed December 13, 2016).

[HTF09] Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman, *The elements of statistical learning : Data mining, inference, and prediction*, Springer, New York, 2009.

[LBBH98] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 1998, pp. 2278–2324.

[LIBa] *LIBLINEAR*, https://www.csie.ntu.edu.tw/~cjlin/liblinear/.

[LIBb] *LIBSVM*, https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[Vap95] Vladimir N. Vapnik, *The nature of statistical learning theory*, Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[YZB15] Jianfeng Yao, Shurong Zheng, and Zhidong Bai, *Large sample covariance matrices and high-dimensional data analysis*, Cambridge University Press, 2015.