Analyzing the Regret of Online Classification Algorithms Using Feature Selection from Stacked Denoising Autoencoders

Viswesh Periyasamy
Department of Computer Science
University of Wisconsin-Madison
Madison, WI 53706
`vperiyasamy@wisc.edu`

Abstract

Feature learning (or representation) is a widely-known challenge in machine learning algorithms. In this report we attempt to tackle this issue using a well known method known as stacked denoising autoencoders - neural networks trained to reconstruct the original output in such a fashion that reduces the dimensionality of the input space. We then transfer this smaller dimensionality input to an online setting where the feature reduction is constantly tuned by incoming examples. We show the performance when compared to a traditional online classifier, and further analyze the regret and mistake bound of such algorithms with respect to the number of training instances. Finally, we see how these measures scale and whether they agree with their respective theoretical bounds.

1 Introduction

1.1 Stacked denoising autoencoders

Dimensionality reduction is a key piece in solving many problems in machine learning. There is often a surplus of data saturated with noisy features that make it difficult for certain algorithms to learn useful weights/parameters. Additionally, it can become computationally infeasible to process large amounts of data as the number of features is grows. Therefore, a number of methods have been attempted to condense a full set of features to a subset to ease computation or remove unnecessary and noisy features. One method to tackle this issue is using autoencoders (i.e. neural networks trained in an unsupervised fashion to reconstruct the input as the output). In doing so, these neural networks can learn useful "meta-features" which describe inherent and complex relationships between the individual features [1] [2]. Such meta-features accomplish both a dimensionality reduction as they condense several features into relationships and a denoising effect as individual feature noise holds less weight when grouped with others. An example in the domain of image classification would be training an autoencoder to learn the straight, round, and sharp edges that outline a cat's figure by learning to reproduce a given image of a cat.

Formally, we can define A as the class of functions from $\mathbb{R}^p$ to $\mathbb{R}^n$ where $p < n$. Similarly, we define B as the class of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$. Let $X = \{x_1, ..., x_m\}$ is our set of training instances which are vectors in $\mathbb{R}^n$ and $Y = \{y_1, ..., y_m\}$ be our target vectors. Finally, we define a distortion function (e.g. Euclidean distance) as Δ [3].

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Then the *autoencoder problem* is to find weights (i.e. A and B) to minimize the overall distortion function:

$$\min E(A, B) = \min_{A,B} \sum_{t=1}^{m} E(x_t) = \min_{A,B} \sum_{t=1}^{m} \Delta(A \circ B(x_t), x_t) \tag{1}$$

In the case of a deep neural network, A represents the "encoder" to transform the input into a lower-dimension representation, and B represents the "decoder" in which we rescale the transformed representation back to the dimensionality of the input. This is not a lossless reconstruction; however, the overarching goal is to learn these meta-features and not to achieve perfect reconstruction.

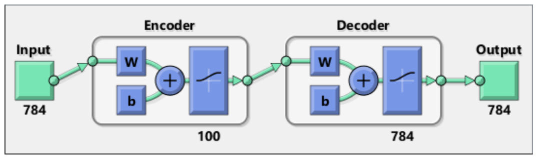
We augment this idea by introducing the design of stacked denoising autoencoders: training several layers of autoencoders such that each layer learns a more condensed set of meta features. These layers become denoising by training on input instances which have been slightly perturbed in order to account for noise on unseen data.

1.2 Classification using autoencoders

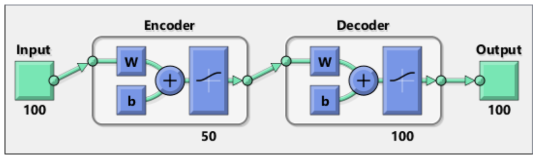
Once a stacked denoising autoencoder (hereby referred to as SDA) has been trained, adapting it for classification is quite simple. The steps are as follows:

1. Separate the SDA into the encoder and decoder portions (freeze the weights of the encoder)
2. Add regular network layers which scale and flatten the encoded values
3. Add a classification layer (such as a dense layer using a softmax activation function)
4. Using the frozen encoder weights and fresh classifying layer weights, retrain the new model in a supervised fashion

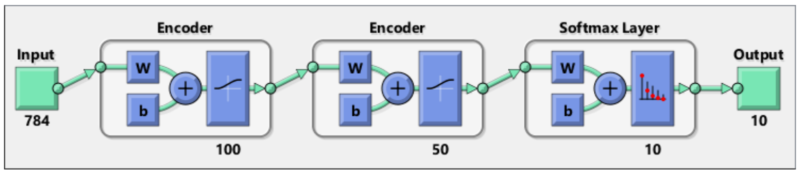
Our final hypothesis space becomes the full classification network represented below by Figure 1



(a) A single autoencoder reconstruction layer at full scale.



(b) A single autoencoder reconstruction layer on a condensed feature set.



(c) Full stacked denoising autoencoder classifier.

Figure 1: An example of an SDA classifier comprised of encoder "slices" trained with different input sizes. (adopted from [4])

1.3 Mistake bound and Regret

With stacked denoising autoencoders established, we move to an online setting where data does not come in a training and test batch, but rather sequentially for our models to incrementally classify and adjust (thus moving through the hypothesis space towards an optimal h^*). Here we recall the notion of a mistake bound - the maximal number of mistakes an algorithm might make over a sequence of examples $x_1, ..., x_T$ that are labeled by h^*. Let our SDA algorithm be called A. Then the mistake bound of our hypothesis space (SDA classifiers in this case) is defined as:

$$\sup_{x_{1:T} \in \underline{\bar{X}}^T} M(A, x_{1:T}) \tag{2}$$

Unfortunately, our hypothesis space is not finite (else we could use the halving algorithm to upper-bound $M(A)$.

We also introduce the notion of regret - here we operate in the unrealizable case where there is no h^* generating our example labels. Now it becomes more difficult to consider an absolute mistake bound, but we can discuss our mistakes with respect to h^* (the best SDA classifier we can possibly achieve). We define regret with respect to a certain hypothesis h as:

$$\text{Reg}_T(h) = \sum_{t=1}^{T} \mathbb{1}[p_t \neq y_t] - \sum_{t=1}^{T} \mathbb{1}[h(x_t) \neq y_t] \tag{3}$$

Where p_t denotes our prediction for example x_t. Note that the latter part of the equation was previously 0 in the realizable case but must be considered now. We then define regret with respect to a hypothesis class $\mathcal{H}$ as:

$$\text{Reg}_T(\mathcal{H}) = \max_{h \in \mathcal{H}} \text{Reg}_T(h) \tag{4}$$

We can bound such regret at $\frac{T}{2}$, but unfortunately in the case of SDA we cannot achieve a sub-linear bound to achieve 0 regret asymptotically. Still, we analyze the empirical mistake and regret to see if an SDA classifier is better than a regular neural network.

2 Method

2.1 Overview

To tackle the discrepancy between a standard neural network classifier and an SDA classifier, we turn to the image classification domain and use the famous MNIST data set of handwritten digits. Specifically, we train an SDA completely, freeze the weights as mentioned above, and then create an SDA classifier to compare side-by-side with a standard classifier. As is common practice in the domain of image classification, we use deep convolutional neural networks for both the SDA and standard classifiers [5]. Once created, we introduce these networks to the online setting and have them incrementally predict and then train in a supervised fashion. Additionally, we take instances of each method and also pre-train them on a batch of images before introducing them to the online setting. All networks and functions are built using Python 3.6 and the Keras machine learning framework using a TensorFlow GPU backend (base code adopted from [6]).

2.2 Building the SDA classifier

The SDA for this experiment is comprised of several deep convolutional 2-dimensional layers along with several max-pooling and up-sampling layers in between. Each convolutional layer uses a Rectified Linear Unit (ReLU) activation function except the final decoder layer which uses a sigmoid activation function. The full network uses the ADAM optimization method and a binary cross entropy loss function for standard training.

Once the SDA has been trained on reconstruction of the input images, the weights of the encoder layers are frozen and the SDA is sliced in half. We then attach a flattening layer and a dense classification layer using the softmax activation function. This final SDA classifier is then used for classification and incremental training in the pre-training phase (when appropriate) and finally the online setting. Figure 2 depicts the full network of the SDA classifier.

2.3 Building the standard classifier

The standard deep convolutional network is created in almost exactly the same fashion as the SDA except all of the layers are fully connected and it is never trained for reconstruction nor sliced in half.

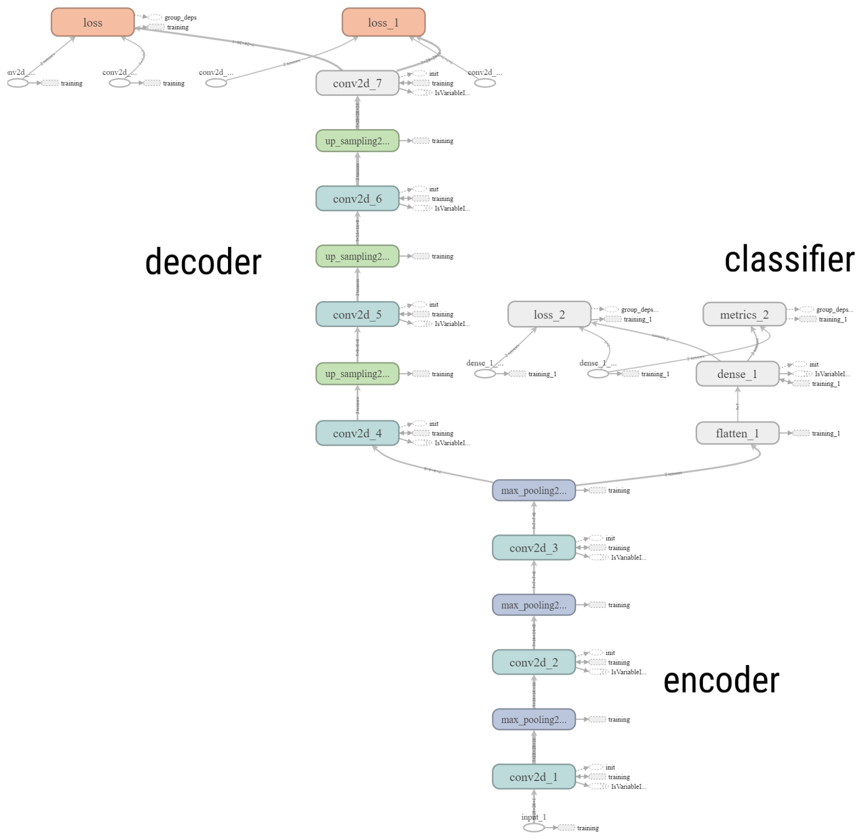


Figure 2: A deep convolutional SDA classifier along with the discarded decoder.

Again, each convolutional layer uses the ReLU activation function and the final dense layer uses a softmax activation function for classification. It follows the same structure as the SDA without the decoder portion of the network.

2.4 Training in an online setting

We compare four methods in the online setting: (a) SDA classifier, (b) CNN classifier, (c) SDA classifier pre-trained on a batch of examples, and (d) CNN classifier pre-trained on a batch of examples. The MNIST data set has 70,000 images total, so pre-training steps use a batch of 60,000 images and all four methods use a final test set of 10,000 images fed incrementally. For each example, the method's prediction is recorded and then the label is exposed for a round of supervised back-propogation. Methods employing the autoencoders do **not** have their encoder layer weights changed, as they remain frozen to keep the autoencoder's meta-features.

3 Results

3.1 SDA reconstruction

The first step to running the SDA classifier is to train the SDA for reconstruction of the handwritten digits. The SDA uses the ADAM method of optimization as well as a binary cross entropy loss function. Using a batch size of 256 and training length of 50 epochs, Figure 3 shows the reconstruction on a sample of 10 test images after passing through the encoder and decoder.

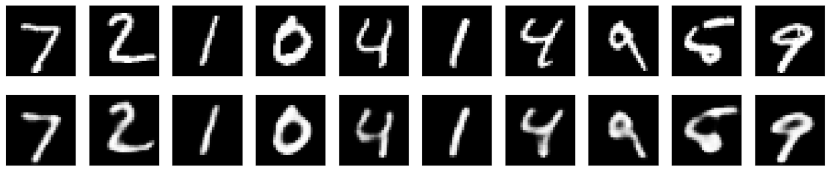


Figure 3: Reconstruction of the handwritten digits after passing through the SDA.

3.2 Training

Training each classifier in an online setting is done incrementally. Since there is only one example per fit, there are no epochs nor batches, and each classifier is tuned with only one round of a forward pass and back-propogation. For the models which are pre-trained, they use a standard batch size of 256 and 50 epochs on a total training set of 60,000 images.

3.3 Testing

Table 1 shows the results of each method when processing 10,000 examples sequentially. As the mistakes and accuracy show, without pre-training the SDA classifier actually outperforms the standard CNN classifier. The MSE remains about the same, but we can see a significant rise in accuracy with the SDA method. However, when looking at the pre-trained models it seems the opposite shows (a significant drop in accuracy with the SDA method). This indicates that pre-training on a large batch of data before exposing to the online setting may actually harm the performance.

Table 1: Results on running each classifier in online setting of 10,000 examples

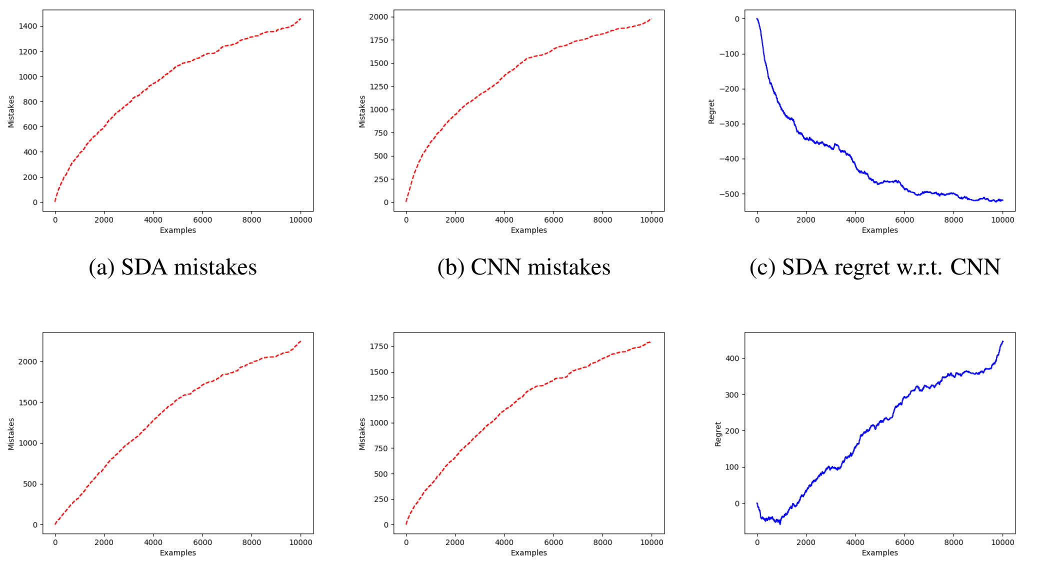
Method	Number of Mistakes	Accuracy	Average Mean squared error
SDA Classifier	1459	85.4%	0.023
CNN Classifier	1978	80.2%	0.032
SDA Classifier (pre-trained)	2247	77.5%	0.038
CNN Classifier (pre-trained)	1801	82.0%	0.030

3.4 Regret analysis

While the SDA classifier seemed to outperform the CNN classifier without any pre-training in the online setting, a further analysis of the regret proves insightful. Recall (3) which defines the regret of an algorithm with respect to another hypothesis. In this case, we calculate the regret of the SDA classifier with respect to the standard CNN classifier. Figure 4 shows the mistakes of each method: Figure 4a and Figure 4b show the mistakes of the SDA and CNN classifiers, respectively, as they scale with the number of training instances. Figure 4c shows the regret of this SDA classifier - as depicted in the plot, it highly outperformed CNN and achieved a steady, decreasing negative regret. Similarly, Figure 4d and Figure 4e show the mistakes of the SDA and CNN classifiers with pre-training, respectively, as they scale with the number of training instances. Figure 4f shows the regret of this pre-trained SDA classifier which has its regret steadily increasing upwards. However, in both cases, we achieve at least our regret bound of $\frac{T}{2}$. Furthermore, we can see that the regret tapers off towards the latter half of the examples - this is indicative that the regret for either classifier will not continue its momentum too far.

4 Conclusions and future work

Autoencoders are fairly popular and have been used in practice for dimensionality and noise reduction, however they are rarely introduced into an online learning setting. This work serves as an example of the performance of using stacked denoising autoencoders for continually-tuned feature representation. Additionally, we showed that without pre-training a model, using an SDA classifier can actually



(a) SDA mistakes (b) CNN mistakes (c) SDA regret w.r.t. CNN

(d) SDA (pretrain) mistakes (e) CNN (pretrain) mistakes (f) SDA (pretrain) regret w.r.t. CNN

Figure 4: Analyzing the mistakes and regret scaled by the number of training examples

provide more accurate results given that it's able to doubly tune its internal representation of each example and the output simultaneously. It's possible that pre-training from the start introduces too much weight bias such that the autoencoder either doesn't have much effect for online training instances or it may actually harm them as well. Further studies are necessary across several domains (other image types, signal processing, etc.) to prove whether SDA classifiers are indeed better or worse than standard practices.

In terms of regret analysis, we showed that the regret can vary widely between algorithms depending on the size of the hypothesis class (in this case infinite). Further work is necessary, such as applying algorithms like Follow the Regularized Leader or variations which have been adapted for neural networks [7]. There is little to no literature about the regret analysis of autoencoders in an online setting, and while autoencoders have fallen out of favor recently, this might be a setting where they are more appropriate.

References

[1] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research 11*, pages 3371–3408, 2010.

[2] H. C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach. Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1930–1943, Aug 2013.

[3] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. *Journal of Machine Learning Research: Workshop and Conference Proceedings 27*, pages 37–50, 2012.

[4] Train stacked autoencoders for image classification (https://www.mathworks.com/examples/neural-network/mw/nnet-ex33663708-train-stacked-autoencoders-for-image-classification).

[5] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao. Stacked convolutional denoising auto-encoders for feature representation. *IEEE Transactions on Cybernetics*, 47(4):1017–1027, April 2017.

[6] The keras blog (https://blog.keras.io/building-autoencoders-in-keras.html).

[7] Shuai Zheng and James T. Kwok. Follow the moving leader in deep learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4110–4119, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.