Deep Learning via Semi-Supervised Embedding

Jason Weston[*] JASONW@NEC-LABS.COM
Frédéric Ratle[†] FREDERIC.RATLE@GMAIL.COM
Ronan Collobert[*] COLLOBER@NEC-LABS.COM
(∗) NEC Labs America, 4 Independence Way, Princeton, NJ 08540 USA
(†) IGAR, University of Lausanne, Amphipôle, 1015 Lausanne, Switzerland

Abstract

We show how nonlinear embedding algorithms popular for use with *shallow* semi-supervised learning techniques such as kernel methods can be applied to deep multilayer architectures, either as a regularizer at the output layer, or on each layer of the architecture. This provides a simple alternative to existing approaches to *deep* learning whilst yielding competitive error rates compared to those methods, and existing *shallow* semi-supervised techniques.

1. Introduction

Embedding data into a lower dimensional space or the related task of clustering are unsupervised dimensionality reduction techniques that have been intensively studied. Most algorithms are developed with the motivation of producing a useful analysis and visualization tool.

Recently, the field of semi-supervised learning (Chapelle et al., 2006), which has the goal of improving generalization on supervised tasks using unlabeled data, has made use of many of these techniques. For example, researchers have used nonlinear embedding or cluster representations as features for a supervised classifier, with improved results.

Most of these architectures are *disjoint* and *shallow*, by which we mean the unsupervised dimensionality reduction algorithm is trained on unlabeled data separately as a first step, and then its results are fed to a supervised classifier which has a shallow architecture such as a (kernelized) linear model. For example, several methods learn a clustering or a distance measure based on a nonlinear manifold embedding as a first step (Chapelle et al., 2003; Chapelle & Zien, 2005). Transductive Support Vector Machines (TSVMs) (Vapnik, 1998) (which employs a kind of clustering) and LapSVM (Belkin et al., 2006) (which employs a kind of embedding) are examples of methods that are *joint* in their use of unlabeled data and labeled data, but their architecture is still *shallow*.

Deep architectures seem a natural choice in hard AI tasks which involve several *sub-tasks* which can be coded into the layers of the architecture. As argued by several researchers (Hinton et al., 2006; Bengio et al., 2007) semi-supervised learning is also natural in such a setting as otherwise one is not likely to ever have enough labeled data to perform well.

Several authors have recently proposed methods for using unlabeled data in deep neural network-based architectures. These methods either perform a greedy layer-wise pre-training of weights using unlabeled data alone followed by supervised fine-tuning (which can be compared to the *disjoint* shallow techniques for semi-supervised learning described before), or learn unsupervised encodings at multiple levels of the architecture jointly with a supervised signal. Only considering the latter, the basic setup we advocate is simple:

1. Choose an unsupervised learning algorithm.
2. Choose a model with a deep architecture.
3. The unsupervised learning is plugged into any (or all) layers of the architecture as an *auxiliary task*.
4. Train supervised and unsupervised tasks using the same architecture *simultaneously*.

The aim is that the unsupervised method will improve accuracy on the task at hand. However, the unsupervised methods so far proposed for deep architectures are in our opinion somewhat complicated and

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

restricted. They include a particular kind of generative model (a restricted Boltzmann machine) (Hinton et al., 2006), autoassociators (Bengio et al., 2007), and a method of sparse encoding (Ranzato et al., 2007). Moreover, in all cases these methods are not compared with, and appear on the surface to be completely different to, algorithms developed by researchers in the field of semi-supervised learning.

In this article we advocate simpler ways of performing deep learning by leveraging *existing* ideas from semi-supervised algorithms so far developed in *shallow* architectures. In particular, we focus on the idea of combining an *embedding*-based regularizer with a supervised learner to perform semi-supervised learning, such as is used in Laplacian SVMs (Belkin et al., 2006). We show that this method can be: (i) generalized to multi-layer networks and trained by stochastic gradient descent; and (ii) is valid in the *deep* learning framework given above.

Our experimental evaluation is then split into three parts: (i) stochastic training of semi-supervised multi-layered architectures is compared with existing semi-supervised approaches on several benchmarks, with positive results; (ii) a demonstration of how to use semi-supervised regularizers in *deep* architectures by plugging them into any layer of the architecture is shown on the well-known MNIST dataset; and (iii) a case-study is presented using these techniques for deep-learning of semantic role labeling of English sentences.

The rest of the article is as follows. In Section 2 we describe existing techniques for semi-supervised embedding. In Section 3 we describe how to generalize these techniques to the task of *deep learning*. Section 4 reviews existing techniques for deep learning, Section 5 gives an experimental comparison between all these approaches, and Section 6 concludes.

2. Semi-Supervised Embedding

A key assumption in many semi-supervised algorithms is the structure assumption[1]: points within the same structure (such as a cluster or a manifold) are likely to have the same label. Given this assumption, the aim is to use unlabeled data to uncover this structure. In order to do this many algorithms such as cluster kernels (Chapelle et al., 2003), LDS (Chapelle & Zien, 2005), label propagation (Zhu & Ghahramani, 2002) and LapSVM (Belkin et al., 2006), to name a few, make use of regularizers that are directly related to unsupervised embedding algorithms. To understand these methods we will first review some relevant approaches to linear and nonlinear embedding.

[1]This is often referred to as the cluster assumption or the manifold assumption (Chapelle et al., 2006).

2.1. Embedding Algorithms

We will focus on a rather general class of embedding algorithms that can be described by the following type of optimization problem: given the data $x_1, \ldots, x_U$ find an embedding $f(x_i)$ of each point x_i by minimizing

$$\sum_{i,j=1}^{U} L(f(x_i, \alpha), f(x_j, \alpha), W_{ij})$$

w.r.t. α, subject to

Balancing constraint.

This type of optimization problem has the following main ingredients:

- $f(x) \in \mathbb{R}^n$ is the embedding one is trying to learn for a given example $x \in \mathbb{R}^d$. It is parametrized by α. In many techniques $f(x_i) = f_i$ is a lookup table where each example i is assigned an independent vector f_i.
- L is a loss function between pairs of examples.
- The matrix W of weights W_{ij} specifying the similarity or dissimilarity between examples x_i and x_j. This is supplied in advance and serves as a kind of label for the loss function.
- A balancing constraint is often required for certain objective functions so that a trivial solution is not reached.

Many well known algorithms fit into this framework.

Multidimensional scaling (MDS) is a classical algorithm that attempts to preserve the distance between points, whilst embedding them in a lower dimensional space, e.g. by using the loss function

$$L(f_i, f_j, W_{ij}) = (||f_i - f_j|| - W_{ij})^2$$

MDS is equivalent to PCA if the metric is Euclidean (Williams, 2001).

ISOMAP (Tenenbaum et al., 2000) is a nonlinear embedding technique that attempts to capture manifold structure in the original data. It works by defining a similarity metric that measures distances along the manifold, e.g. W_{ij} is defined by the shortest path on the neighborhood graph. One then uses those distances to embed using conventional MDS.

Laplacian Eigenmaps (Belkin & Niyogi, 2003) learn manifold structure by emphasizing the preservation of *local distances*. One defines the distance metric between the examples by encoding them in the Laplacian $L = W - D$, where $D_{ii} = \sum_j W_{ij}$ is diagonal. Then, the following optimization is used:

$$\sum_{ij} L(f_i, f_j, W_{ij}) = \sum_{ij} W_{ij} ||f_i - f_j||^2 = f^\top L f \quad (1)$$

subject to the balancing constraint:

$$f^\top D f = I \text{ and } f^\top D 1 = 0. \quad (2)$$

Siamese Networks (Bromley et al., 1993) are also a classical method for nonlinear embedding. Neural networks researchers think of such models as a network with two identical copies of the same function, with the same weights, fed into a "distance measuring" layer to compute whether the two examples are similar or not, given labeled data. In fact, this is exactly the same as the formulation given at the beginning of this Section.

Several loss functions have been proposed for *siamese networks*, here we describe a margin-based loss proposed by the authors of (Hadsell et al., 2006):

$$L(f_i, f_j, W_{ij}) = \begin{cases} ||f_i - f_j||^2 & \text{if } W_{ij} = 1, \\ \max(0, m - ||f_i - f_j||^2) & \text{if } W_{ij} = 0 \end{cases} \quad (3)$$

which encourages similar examples to be close, and dissimilar ones to have a distance of at least m from each other. Note that no balancing constraint is needed with such a choice of loss as the margin constraint inhibits a trivial solution. Compared to using constraints like (2) this is much easier to optimize by gradient descent.

2.2. Semi-Supervised Algorithms

Several *semi-supervised* classification algorithms have been proposed which take advantage of the algorithms described in the last section. Here we assume the setting where one is given $L + U$ examples x_i, but only the first L have a known label y_i.

Label Propagation (Zhu & Ghahramani, 2002) adds a Laplacian Eigenmap type regularization to a nearest-neighbor type classifier:

$$\min_f \sum_{i=1}^{L} ||f_i - y_i||^2 + \lambda \sum_{i,j=1}^{L+U} W_{ij} ||f_i - f_j||^2 \quad (4)$$

The algorithm tries to give two examples with large weighted edge W_{ij} the same label. The neighbors of neighbors tend to also get the same label as each other by transitivity, hence the name *label propagation*.

LapSVM (Belkin et al., 2006) uses the Laplacian Eigenmaps type regularizer with an SVM: minimize

$$||w||^2 + \gamma \sum_{i=1}^{L} H(y_i f(x_i)) + \lambda \sum_{i,j=1}^{L+U} W_{ij} ||f(x_i) - f(x_j)||^2 \quad (5)$$

where $H(x) = \max(0, 1 - x)$ is the hinge loss.

Other Methods In (Chapelle & Zien, 2005) a method called *graph* is suggested which combines a modified version of ISOMAP with an SVM. The authors also suggest to combine modified ISOMAP with TSVMs rather than SVMs, and call it *Low Density Separation* (LDS).

3. Semi-supervised Embedding for Deep Learning

We would like to use the ideas developed in semi-supervised learning for *deep learning*. Deep learning consists of learning a model with several layers of non-linear mapping. In this article we will consider multi-layer networks with M layers of hidden units that give a C-dimensional output vector:

$$f_i(x) = \sum_{j=1}^{d} w_j^{O,i} \; h_j^M(x) + b^{O,i}, \quad i = 1, \ldots, C \quad (6)$$

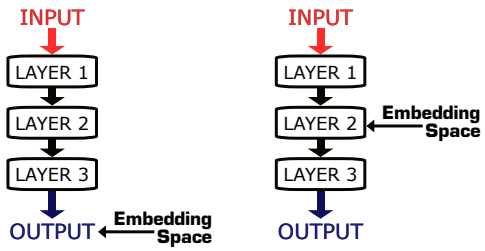
where w^O are the weights for the output layer, and typically the k^{th} layer is defined as

$$h_i^k(x) = S\left(\sum_j w_j^{k,i} \; h_j^{k-1}(x) + b^{k,i} \right), \; k > 1 \quad (7)$$

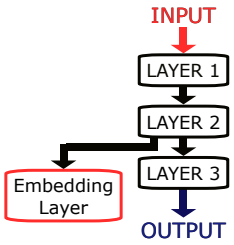
$$h_i^1(x) = S\left(\sum_j w_j^{1,i} \; x_j + b^{1,i} \right) \quad (8)$$

and S is a non-linear squashing function such as tanh. Here, we describe a standard *fully connected* multi-layer network but prior knowledge about a particular problem could lead one to other network designs. For example in sequence and image recognition time delay and convolutional networks (TDNNs and CNNs) (LeCun et al., 1998) have been very successful. In those approaches one introduces layers that apply convolutions on their input which take into account locality information in the data, i.e. they learn features from image patches or windows within a sequence.

The general method we propose for *semi-supervised deep learning* is to add a semi-supervised regularizer in deep architectures in one of three different modes, as shown in Figure 1:



(a) Output (b) Internal



(c) Auxiliary

Figure 1. Three modes of embedding in deep architectures.

(a) Add a semi-supervised loss (regularizer) to the supervised loss on the entire network's output (6):

$$\sum_{i=1}^{L} \ell(f(x_i), y_i) + \lambda \sum_{i,j=1}^{L+U} L(f(x_i), f(x_j), W_{ij}) \quad (9)$$

This is most similar to the *shallow* techniques described before, e.g. equation (5).

(b) Regularize the k^{th} hidden layer (7) directly:

$$\sum_{i=1}^{L} \ell(f(x_i), y_i) + \lambda \sum_{i,j=1}^{L+U} L(f^k(x_i), f^k(x_j), W_{ij}) \quad (10)$$

where $f^k(x) = (h_1^k(x), \ldots, h_{N_k}^k(x))$ is the output of the network up to the k^{th} hidden layer.

(c) Create an auxiliary network which shares the first k layers of the original network but has a new final set of weights:

$$g_i(x) = \sum_j w_j^{AUX,i} \; h_j^k(x) + b^{AUX,i} \quad (11)$$

We train this network to *embed* unlabeled data simultaneously as we train the original network on *labeled* data.

In our experiments we use the loss function (3) for embedding, and the hinge loss

$$\ell(f(x), y) = \sum_{c=1}^{C} H(y(c) f_c(x)),$$

Algorithm 1 *Embed*NN

Input: labeled data (x_i, y_i), $i = 1, \ldots, L$, unlabeled data x_i, $i = L+1, \ldots, U$, set of functions $f(\cdot)$, and embedding functions $g^k(\cdot)$, see Figure 1 and equations (9), (10) and (11).
repeat
  Pick a random *labeled* example (x_i, y_i)
  Make a gradient step to optimize $\ell(f(x_i), y_i)$
  for each embedding function $g^k(\cdot)$ **do**
    Pick a random pair of neighbors x_i, x_j.
    Make a gradient step for $\lambda L(g^k(x_i), g^k(x_j), 1)$
    Pick a random unlabeled example x_n.
    Make a gradient step for $\lambda L(g^k(x_i), g^k(x_n), 0)$
  end for
until stopping criteria is met.

for labeled examples, where $y(c) = 1$ if $y = c$ and -1 otherwise. For neighboring points, this is the same regularizer as used in LapSVM and Laplacian Eigenmaps. For non-neighbors, where $W_{ij} = 0$, this loss "pulls" points apart, thus inhibiting trivial solutions without requiring difficult constraints such as (2). To achieve an embedding *without* labeled data the latter is necessary or all examples would collapse to a single point in the embedding space. We therefore prefer this regularizer to using (1) alone. Pseudocode of our approach is given in Algorithm 1.

Labeling unlabeled data as neighbors Training neural networks online using stochastic gradient descent is fast and can scale to millions of examples. A possible bottleneck with our approach is computation of the matrix W, that is, computing which unlabeled examples are neighbors and have value $W_{ij} = 1$. Embedding algorithms often use k-nearest neighbor for this task, and although many methods for its fast computation do exist, this could still be slower than we would like. One possibility is to approximate it with sampling techniques.

However, there are also many other ways of collecting neighboring unlabeled data, notably if one is given *sequence* data such as in audio, video or text problems. For example, one can take images from two consecutive frames of video as a neighboring pair with $W_{ij} = 1$. Such pairs are likely to have the same label, and are collected cheaply. In Section 5 we apply this kind of idea to text and train a semi-supervised semantic role labeler using an unlabeled set of 631 million words.

When do we expect this approach to work? One can see our approach as an instance of multi-task learning (Caruana, 1997) using unsupervised auxiliary

tasks. In common with other semi-supervised learning approaches, and indeed other deep learning approaches, we only expect this to work if $p(x)$ is useful for the supervised task $p(y|x)$, i.e. if the structure assumption is true. We believe many natural tasks have this property.

We note that an alternative multi-task learning scheme is presented in (Ando & Zhang, 2005) and applied to neural networks in (Ahmed et al., 2008) which instead constructs auxiliary *supervised* tasks from unlabeled data by constructing tasks with labels y^*. This is useful when $p(y^*|x)$ is correlated to $p(y|x)$, however an expert must engineer a useful target y^*.

4. Existing Approaches to Deep Learning

Hinton and coworkers (2006) proposed the Deep Belief Net (DBN) which is a multi-layered network first trained as a generative model with unlabeled data before being subsequently trained in supervised mode. It is based around iteratively training Restricted Boltzmann machines (RBMs) for each layer. An RBM is a two-layer network in which visible, binary stochastic pixels v are connected to hidden binary stochastic feature detectors h. The probability assigned to an example x is:

$$P(x) = \sum_{h \in \mathcal{H}} P(x,h) = \sum_{h \in \mathcal{H}} \frac{e^{-E(x,h)}}{Z}$$

$$E(x,h) = -\sum_{i \in \text{pixels}} w_i^P v_i - \sum_{j \in \text{features}} w_j^F h_j - \sum_{i,j} v_j h_j w_{ij}$$

The idea is to obtain large values for the training examples, and small values elsewhere just as in any maximum likelihood density estimator. This is trained with a procedure called contrastive divergence whereby one pushes up the energy on training data and pushes down the energy on samples generated by the model. The authors used this method to pretrain a deep neighborhood component analysis model (DBN-NCA) and a regularized version that simultaneously trains an autoencoder (DBN-rNCA) (Salakhutdinov & Hinton, 2007).

The authors of (Bengio et al., 2007) suggested a simpler scheme: define an autoencoder that given an input x tries to encode it in a low dimensional space $z = f_{enc}(x)$, and then decode it again to reproduce it as well as possible, e.g. so that

$$||x - f_{dec}(f_{enc}(x))||^2$$

is small. (Actually you can also view RBMs in this way, see (Ranzato et al., 2007).) The idea is to use an autoencoder as a regularizer which is trained on unlabeled data. If the autoencoder is linear it corresponds to PCA (Japkowicz et al., 2000) and hence also MDS, making a clear link to the embedding algorithms we discussed in Section 2.1. The authors claim that autoassociators have the advantage "that almost any parametrizations of the layers are possible, as long as the training criterion is continuous in the parameters [...] the class of probabilistic models for which [DBNs] can be applied is currently more limited."

Finally, recently the authors of (Ranzato et al., 2007) introduced another method of deep learning which also amounts to a kind of encoder/decoder architecture, called SESM. In this case they choose to learn large, sparse codes as they believe these are good for classification. They choose an encoder $f_{enc}(x) = w^\top x + b_{enc}$ and a decoder with shared weights $f_{dec}(z) = wS(z) + b_{dec}$. They then optimize the following loss:

$$\alpha_e ||z - f_{enc}(x)||_2^2 + ||x - f_{dec}(z)||_2^2 + \alpha_s h(z) + \alpha_r ||w||_1$$

where the first term makes the output of the encoder close to the code z (which is also learnt), the second term makes the decoder try to reproduce the input, and the third and fourth terms sparsify the codes z and the weights of the encoder and decoder w. α_e, α_s and α_r are all hyperparameters. The training requires an online coordinate descent scheme because both z and w are being optimized.

We believe all of the methods just described are significantly more complicated than our approach. Our embedding approach can also be seen as an encoder $f_{enc}(x)$ that embeds data into a low dimensional space. However we do not need to decode during training (or indeed at all). Further, if the data is high dimensional and sparse there is a significant speedup from not having to decode.

Finally, existing approaches advocate greedy layer-wise training, followed by a "fine-tuning" step using the supervised signal. The intention is that the unsupervised learning provides a better initialization for supervised learning, and hence a better final local minimum. Our approach does not use a pre-training step, but instead *directly* optimizes our new objective function. We advocate that it is the new choice of objective that can provide improved results.

5. Experimental Evaluation

We test our approach on several datasets summarized in Table 1.

Small-scale experiments g50c, Text and Uspst are small-scale datasets often used for semi-supervised

Table 1. Datasets used in our experiments. The first three are small scale datasets used in the same experimental setup as found in (Chapelle & Zien, 2005; Sindhwani et al., 2005; Collobert et al., 2006). The following six datasets are large scale. The Mnist 1h,6h,1k,3k and 60k variants are MNIST with a labeled subset of data, following the experimental setup in (Collobert et al., 2006). SRL is a Semantic Role Labeling task (Pradhan et al., 2004) with one million labeled training examples and 631 million unlabeled examples.

data set	classes	dims	points	labeled
g50c	2	50	500	50
Text	2	7511	1946	50
Uspst	10	256	2007	50
Mnist1h	10	784	70k	100
Mnist6h	10	784	70k	600
Mnist1k	10	784	70k	1000
Mnist3k	10	784	70k	3000
Mnist60k	10	784	70k	60000
SRL	16	-	631M	1M

learning experiments (Chapelle & Zien, 2005; Sindhwani et al., 2005; Collobert et al., 2006). We followed the same experimental setup, averaging results of ten splits of 50 labeled examples where the rest of the data is unlabeled. In these experiments we test the embedding regularizer on the output of a neural network (see equation (9) and Figure 1(a)). We define a two-layer neural network (NN) with hu hidden units. We define W so that the 10 nearest neighbors of i have $W_{ij} = 1$, and $W_{ij} = 0$ otherwise. We train for 50 epochs of stochastic gradient descent and fixed $\lambda = 1$, but for the first 5 we optimized the supervised target alone (without the embedding regularizer). This gives two free hyperparameters: the number of hidden units $hu = \{0, 5, 10, 20, 30, 40, 50\}$ and the learning rate $lr = \{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$.

We report the optimum choices of these values optimized by 5-fold cross validation and by optimizing on the test set in Table 2. Note the datasets are very small, so cross validation is unreliable. Several methods from the literature optimized their hyperparameters using the test set (those that are not marked with (cv)). Our *Embed*NN is competitive with state-of-the-art semi-supervised methods based on SVMs, even outperforming them in some cases.

MNIST experiments We compare our method in all three different modes (Figure 1) with conventional semi-supervised learning (TSVM) using the same data split and validation set as in (Collobert et al., 2006). We also compare to several deep learning methods: RBMs, SESM and DBN-NCA and DBN-rNCA (however, they are trained on a different data split). In

Table 2. Results on Small-Scale Datasets. We report the best test error over the hyperparameters of our method, *Embed*NN, as in the methodology of (Chapelle & Zien, 2005) as well as the error when optimizing the parameters by cross-validation, *Embed*NN$^{(cv)}$. LDS$^{(cv)}$ and LapSVM$^{(cv)}$ also use cross-validation.

	g50c	Text	Uspst
SVM	8.32	18.86	23.18
TSVM	5.80	5.71	17.61
LapSVM$^{(cv)}$	5.4	10.4	12.7
LDS$^{(cv)}$	5.4	5.1	15.8
Label propagation	17.30	11.71	21.30
Graph SVM	8.32	10.48	16.92
NN	10.62	15.74	25.13
*Embed*NN	5.66	5.82	15.49
*Embed*NN$^{(cv)}$	6.78	6.19	15.84

Table 3. Results on MNIST with 100, 600, 1000 and 3000 labels. A two-layer Neural Network (NN) is compared to an NN with Embedding regularizer (*Embed*NN) on the output (O), i^{th} layer (Ii) or auxiliary embedding from the i^{th} layer (Ai) (see Figure 1). A convolutional network (CNN) is also tested in the same way. We compare to SVMs and TSVMs. RBM, SESM, DBN-NCA and DBN-rNCA (marked with $(*)$) taken from (Ranzato et al., 2007; Salakhutdinov & Hinton, 2007) are trained on a different data split.

	Mnst1h	Mnst6h	Mnst1k	Mnst3k
SVM	23.44	8.85	7.77	4.21
TSVM	16.81	6.16	5.38	3.45
RBM$^{(*)}$	21.5	-	8.8	-
SESM$^{(*)}$	20.6	-	9.6	-
DBN-NCA$^{(*)}$	-	10.0	-	3.8
DBN-rNCA$^{(*)}$	-	8.7	-	3.3
NN	25.81	11.44	10.70	6.04
Embed^{O}NN	17.05	5.97	5.73	3.59
Embed^{I1}NN	16.86	9.44	8.52	6.02
Embed^{A1}NN	17.17	7.56	7.89	4.93
CNN	22.98	7.68	6.45	3.35
Embed^{O}CNN	11.73	3.42	3.34	2.28
Embed^{I5}CNN	7.75	3.82	2.73	1.83
Embed^{A5}CNN	7.87	3.82	2.76	2.07

Table 4. Mnist1h dataset with deep networks of 2, 6, 8, 10 and 15 layers; each hidden layer has 50 hidden units. We compare classical NN training with *Embed*NN where we either learn an embedding at the output layer (^{O}) or an auxiliary embedding on all layers at the same time (^{ALL}). .

	2	4	6	8	10	15
NN	26.0	26.1	27.2	28.3	34.2	47.7
Embed^{O}NN	19.7	15.1	15.1	15.0	13.7	11.8
Embed^{ALL}NN	18.2	12.6	7.9	8.5	6.3	9.3

Table 5. Full Mnist60k dataset with deep networks of 2, 6, 8, 10 and 15 layers, using either 50 or 100 hidden units. We compare classical NN training with $Embed^{ALL}$NN where we learn an auxiliary embedding on all layers at the same time.

	2	4	6	8	10	15
NN (HUs=50)	2.9	2.6	2.8	3.1	3.1	4.2
$Embed^{ALL}$NN	2.8	1.9	2.0	2.2	2.4	2.6
NN (HUs=100)	2.0	1.9	2.0	2.2	2.3	3.0
$Embed^{ALL}$NN	1.9	1.5	1.6	1.7	1.8	2.4

these experiments we consider 2-layer networks (NN) and 6-layer convolutional neural nets (CNN) for embedding. We optimize the parameters of NN ($hu = \{50, 100, 150, 200, 400\}$ hidden units and learning rates as before) on the validation set. The CNN architecture is fixed: 5 layers of image patch-type convolutions, followed by a linear layer of 50 hidden units, similar to (LeCun et al., 1998). The results given in Table 3 show the effectiveness of embedding in all three modes, with both NNs and CNNs.

Deeper MNIST experiments We then conducted a similar set of experiments but with very deep architectures – up to 15 layers, where each hidden layer has 50 hidden units. Using Mnist1h, we first compare conventional NNs to $Embed^{ALL}$NN where we learn an auxiliary nonlinear embedding (50 hidden units and a 10 dimensional embedding space) on each layer, as well as $Embed^{O}$NN where we only embed the outputs. Results are given in Table 4. When we increase the number of layers, NNs trained with conventional backpropagation overfit and yield steadily *worse* test error (although they are easily capable of achieving zero training error). In contrast, $Embed^{ALL}$NN *improves* with increasing depth due to the semi-supervised "regularization". Embedding on *all* layers of the network has made *deep learning* possible. $Embed^{O}$NN (embedding on the outputs) also helps, but not as much.

We also conducted some experiments using the full MNIST dataset, Mnist60k. Again using deep networks of up to 15 layers using either 50 or 100 hidden units $Embed^{ALL}$NN outperforms standard NN. Results are given in Table 5. Increasing the number of hidden units is likely to improve these results further, e.g. using 4 layers and 500 hidden units on each layer, one obtains 1.27% using $Embed^{ALL}$NN.

Semantic Role Labeling The goal of semantic role labeling (SRL) is, given a sentence and a relation of interest, to label each word with one of 16 tags that indicate that word's semantic role with respect to the action of the relation. For example the sentence *"The cat eats the fish in the pond"* is labeled in the following way: *"The*$_{ARG0}$ *cat*$_{ARG0}$ *eats*$_{REL}$ *the*$_{ARG1}$ *fish*$_{ARG1}$ *in*$_{ARGM-LOC}$ *the*$_{ARGM-LOC}$ *pond*$_{ARGM-LOC}$*"* where ARG0 and ARG1 effectively indicate the subject and object of the relation "eats" and ARGM-LOC indicates a locational modifier. The PropBank dataset includes around 1 million labeled words from the Wall Street Journal. We follow the experimental setup of (Collobert & Weston, 2007) and train a 5-layer convolutional neural network for this task, where the first layer represents the input sentence words as 50-dimensional vectors. Unlike (Collobert & Weston, 2007), we do not give any prior knowledge to our classifier. In that work words were stemmed and clustered using their parts-of-speech. Our classifier is trained using only the original input words.

Table 6. A deep architecture for Semantic Role Labeling with no prior knowledge outperforms state-of-the-art systems ASSERT and SENNA that incorporate knowledge about parts-of-speech and parse trees. A convolutional network (CNN) is improved by learning an auxiliary embedding ($Embed^{A1}$CNN) for words represented as 100-dimensional vectors using the entire Wikipedia website as unlabeled data.

Method	Test Error
ASSERT (Pradhan et al., 2004)	16.54%
SENNA (Collobert & Weston, 2007)	16.36%
CNN [no prior knowledge]	18.40%
$Embed^{A1}$CNN [no prior knowledge]	14.55%

We attempt to improve this system by, as before, learning an *auxiliary embedding* task. Our embedding is learnt using unlabeled sentences from the Wikipedia web site, consisting of 631 million words in total using the scheme described in Section 3. The same lookup table of word vectors as in the supervised task is used as input to an 11 word window around a given word, yielding 550 features. Then a linear layer projects these features into a 100 dimensional embedding space. All windows of text from Wikipedia are considered neighbors, and non-neighbors are constructed by replacing the middle word in a sentence window with a random word. Our lookup table indexes the most frequently used 30,000 words, and all other words are assigned index 30,001.

The results in Table 6 indicate a clear improvement when learning an auxiliary embedding. ASSERT (Pradhan et al., 2004) is an SVM parser-based system with many hand-coded features, and SENNA is a NN which uses part-of-speech information to build its word vectors. In contrast, our system is the only state-

of-the-art method that does not use prior knowledge in the form of features derived from parts-of-speech or parse tree data. This application will be described in more detail in a forthcoming paper.

6. Conclusion

In this work, we showed how one can improve supervised learning for deep architectures if one jointly learns an embedding task using unlabeled data. Our results both confirm previous findings and generalize them. Researchers using *shallow* architectures already showed two ways of using embedding to improve generalization: (i) embedding unlabeled data as a *separate* pre-processing step (i.e., first layer training) and; (ii) using embedding as a regularizer (i.e., at the output layer). More importantly, we generalized these approaches to the case where we train a semi-supervised embedding *jointly* with a supervised *deep* multi-layer architecture on any (or all) layers of the network, and showed this can bring real benefits in complex tasks.

References

Ahmed, A., Yu, K., Xu, W., & Gong, Y. (2008). Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. *ECCV*. Submitted.

Ando, R., & Zhang, T. (2005). A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *The Journal of Machine Learning Research*, *6*, 1817–1853.

Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, *15*, 1373–1396.

Belkin, M., Niyogi, P., & Sindhwani, V. (2006). Manifold regularization: a geometric framework for learning from Labeled and Unlabeled Examples. *Journal of Machine Learning Research*, *7*, 2399–2434.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems, NIPS 19*.

Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Sackinger, E., & Shah, R. (1993). Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, *7*.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, *28*, 41–75.

Chapelle, O., Schölkopf, B., & Zien, A. (2006). *Semi-supervised learning*. Adaptive computation and machine learning. Cambridge, Mass., USA: MIT Press.

Chapelle, O., Weston, J., & Schölkopf, B. (2003). Cluster kernels for semi-supervised learning. *NIPS 15* (pp. 585–592). Cambridge, MA, USA: MIT Press.

Chapelle, O., & Zien, A. (2005). Semi-supervised classification by low density separation. *AISTATS* (pp. 57–64).

Collobert, R., Sinz, F., Weston, J., & Bottou, L. (2006). Large scale transductive svms. *Journal of Machine Learning Research*, *7*, 1687–1712.

Collobert, R., & Weston, J. (2007). Fast semantic extraction using a novel neural network architecture. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 25–32.

Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comp.*, *18*, 1527–1554.

Japkowicz, N., Hanson, S., & Gluck, M. (2000). Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, *12*, 531–545.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J., & Jurafsky, D. (2004). Shallow semantic parsing using support vector machines. *Proceedings of HLT/NAACL-2004*.

Ranzato, M., Huang, F., Boureau, Y., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press.

Salakhutdinov, R., & Hinton, G. (2007). Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure. *AISTATS*.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). Beyond the point cloud: from transductive to semi-supervised learning. *International Conference on Machine Learning, ICML*.

Tenenbaum, J., de Silva, V., & Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*, 2319–2323.

Vapnik, V. N. (1998). *Statistical learning theory*. John Wiley and Sons, New York.

Williams, C. (2001). On a connection between kernel PCA and metric multidimensional scaling. *Advances in Neural Information Processing Systems, NIPS 13*.

Zhu, X., & Ghahramani, Z. (2002). *Learning from labeled and unlabeled data with label propagation* (Technical Report CMU-CALD-02-107). Carnegie Mellon University.

Acknowledgements

Thanks to Sandra Cordero for producing the figures. Frédéric Ratle is funded by the SNF, Grant no. 105211-107862.