Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks

Caglar Gulcehre, Kyunghyun Cho, Razvan Pascanu, and Yoshua Bengio*

Département d'Informatique et de Recherche Opérationelle
Université de Montréal
(*) CIFAR Fellow

Abstract. In this paper we propose and investigate a novel nonlinear unit, called L_p unit, for deep neural networks. The proposed L_p unit receives signals from several projections of a subset of units in the layer below and computes a normalized L_p norm. We notice two interesting interpretations of the L_p unit. First, the proposed unit can be understood as a generalization of a number of conventional pooling operators such as average, root-mean-square and max pooling widely used in, for instance, convolutional neural networks (CNN), HMAX models and neocognitrons. Furthermore, the L_p unit is, to a certain degree, similar to the recently proposed maxout unit [13] which achieved the state-of-the-art object recognition results on a number of benchmark datasets. Secondly, we provide a geometrical interpretation of the activation function based on which we argue that the L_p unit is more efficient at representing complex, nonlinear separating boundaries. Each L_p unit defines a superelliptic boundary, with its exact shape defined by the order p. We claim that this makes it possible to model arbitrarily shaped, curved boundaries more efficiently by combining a few L_p units of different orders. This insight justifies the need for learning different orders for each unit in the model. We empirically evaluate the proposed L_p units on a number of datasets and show that multilayer perceptrons (MLP) consisting of the L_p units achieve the state-of-the-art results on a number of benchmark datasets. Furthermore, we evaluate the proposed L_p unit on the recently proposed deep recurrent neural networks (RNN).

Keywords: deep learning, L_p unit, multilayer perceptron.

1 Introduction

The importance of well-designed nonlinear activation functions when building a deep neural network has become more apparent recently. Novel nonlinear activation functions that are unbounded and often piecewise linear but not continuous such as rectified linear units (ReLU) [22,11], or rectifier, and maxout units [13] have been found to be particularly well suited for deep neural networks on many object recognition tasks.

A pooling operator, an idea which dates back to the work in [17], has been adopted in many object recognizers. Convolutional neural networks which often

T. Calders et al. (Eds.): ECML PKDD 2014, Part I, LNCS 8724, pp. 530–546, 2014.

employ max pooling have achieved state-of-the-art recognition performances on various benchmark datasets [20,9]. Also, biologically inspired models such as HMAX have employed max pooling [27]. A pooling operator, in this context, is understood as a way to summarize a high-dimensional collection of neural responses and produce features that are invariant to some variations in the input (across the filter outputs that are being pooled).

Recently, the authors of [13] proposed to understand a pooling operator itself as a nonlinear activation function. The proposed maxout unit *pools* a group of linear responses, or outputs, of neurons, which overall acts as a piecewise linear activation function. This approach has achieved many state-of-the-art results on various benchmark datasets.

In this paper, we attempt to generalize this approach by noticing that most pooling operators including max pooling as well as maxout units can be understood as special cases of computing a normalized L_p norm over the outputs of a set of filter outputs. Unlike those conventional pooling operators, however, we claim here that it is beneficial to estimate the order p of the L_p norm instead of fixing it to a certain predefined value such as ∞, as in max pooling.

The benefit of learning the order p, and thereby a neural network with L_p units of different orders, can be understood from geometrical perspective. As each L_p unit defines a spherical shape in a non-Euclidean space whose metric is defined by the L_p norm, the combination of multiple such units leads to a non-trivial separating boundary in the input space. In particular, an MLP may learn a highly curved boundary efficiently by taking advantage of different values of p. In contrast, using a more conventional nonlinear activation function, such as the rectifier, results in boundaries that are piece-wise linear. Approximating a curved separation of classes would be more expensive in this case, in terms of the number of hidden units or piece-wise linear segments.

In Sec. 2 a basic description of a multi-layer perceptron (MLP) is given followed by an explanation of how a pooling operator may be considered a nonlinear activation function in an MLP. We propose a novel L_p unit for an MLP by generalizing pooling operators as L_p norms in Sec. 3. In Sec. 4 the proposed L_p unit is further analyzed from the geometrical perspective. We describe how the proposed L_p unit may be used by recurrent neural networks in Sec. 5. Sec. 6 provides empirical evaluation of the L_p unit on a number of object recognition tasks.

2 Background

2.1 Multi-layer Perceptron

A multi-layer perceptron (MLP) is a feedforward neural network consisting of multiple layers of nonlinear neurons [29]. Each neuron u_j of an MLP typically receives a weighted sum of the incoming signals $\{a_1, \ldots, a_N\}$ and applies a nonlinear activation function ϕ to generate a scalar output such that

$$u_j\left(\{a_1,\ldots,a_N\}\right) = \phi\left(\sum_{i=1}^{N} w_{ij}a_i\right). \tag{1}$$

With this definition of each neuron[1], we define the output of an MLP having L hidden layers and q output neurons given an input $\mathbf{x}$ by

$$\mathbf{u}(\mathbf{x} \mid \boldsymbol{\theta}) = \phi\left(\mathbf{U}^\top \phi_{[L]}\left(\mathbf{W}_{[L]}^\top \cdots \phi_{[1]}\left(\mathbf{W}_{[1]}^\top \mathbf{x}\right)\cdots\right)\right), \tag{2}$$

where $\mathbf{W}_{[l]}$ and $\phi_{[l]}$ are the weights and the nonlinear activation function of the l-th hidden layer, and $\mathbf{W}_{[1]}$ and $\mathbf{U}$ are the weights associated with the input and output, respectively.

2.2 Pooling as a Nonlinear Unit in MLP

Pooling operators have been widely used in convolutional neural networks (CNN) [21,10,27] to reduce the dimensionality of a high-dimensional output of a convolutional layer. When used to group spatially neighboring neurons, this operator which summarizes a group of neurons in a lower layer is able to achieve the property of (local) translation invariance. Various types of pooling operator have been proposed and used successfully, such as average pooling, root-of-mean-squared (RMS) pooling and max pooling [19,33].

A pooling operator may be viewed instead as a nonlinear activation function. It receives input signal from the layer below, and it returns a scalar value. The output is the result of applying some nonlinear function such as max (max pooling). The difference from traditional nonlinearities is that the pooling operator is not applied element-wise on the lower layer, but rather on groups of hidden units. A *maxout* nonlinear activation function proposed recently in [13] is a representative example of *max* pooling in this respect.

3 L_p Unit

The recent success of maxout has motivated us to consider a more general nonlinear activation function that is rooted in a pooling operator. In this section, we propose and discuss a new nonlinear activation function called an L_p unit which replaces the *max* operator in a *maxout* unit by an L_p norm.

3.1 Normalized L_p-Norm

Given a finite vector/set of input signals $[a_1,\ldots,a_N]$ a normalized L_p norm is defined as

$$u_j\left([a_1,\ldots,a_N]\right) = \left(\frac{1}{N}\sum_{i=1}^{N}|a_i - c_i|^{p_j}\right)^{\frac{1}{p_j}}, \tag{3}$$

[1] We omit a bias to make equations less cluttered.

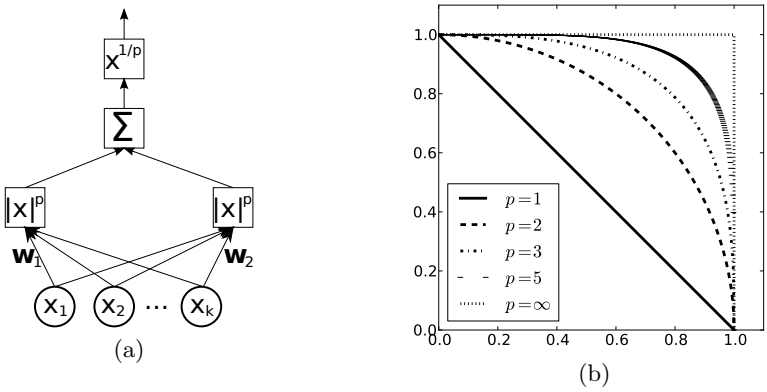


Fig. 1. (a) An illustration of a single L_p unit with two sets of incoming signals. For clarity, biases and the division by the number of filters are omitted. The symbol x in each block (square) represents an input signal to that specific block (square) only. (b) An illustration of the effect of p on the shape of an ellipsoid. Only the first quadrant is shown.

where p_j indicates that the order of the norm may differ for each neuron. It should be noticed that when $0 < p_j < 1$ this definition is not a norm anymore due to the violation of triangle inequality. In practice, we re-parameterize p_j by $1 + \log(1 + e^{\rho_j})$ to satisfy this constraint.

The input signals (also called filter outputs) a_i are defined by

$$a_i = \mathbf{w}_i^\top \mathbf{x},$$

where $\mathbf{x}$ is a vector of activations from the lower layer. c_i is a center, or bias, of the i-th input signal a_i. Both p_j and c_i are model parameters that are learned.

We call a neuron with this nonlinear activation function an L_p unit. An illustration of a single L_p unit is presented in Fig. 1 (a).

Each L_p unit in a single layer receives input signal from a subset of linear projections of the activations of the layer immediately below. In other words, we project the activations of the layer immediately below linearly to $A = \{a_1, \ldots, a_N\}$. We then divide A into equal-sized, non-overlapping groups of which each is fed into a single L_p unit. Equivalently, each L_p unit has its private set of filters.

The parameters of an MLP having one or more layers of L_p units can be estimated by using backpropagation [30], and in particular we adapt the order p of the norm.[2] In our experiments, we use Theano [5] to compute these partial derivatives and update the orders p_j (through the parametrization of p_j in terms of ρ_j), as usual with any other parameters.

[2] The activation function is continuous everywhere except a finite set of points, namely when $a_i - c_i$ is 0 and the absolute value function becomes discontinuous. We ignore these discontinuities, as it is done, for instance, in maxouts and rectifiers.

3.2 Related Approaches

Thanks to the definition of the proposed L_p unit based on the L_p norm, it is straightforward to see that many previously described nonlinear activation functions or pooling operators are closely related to or special cases of the L_p unit. Here we discuss some of them.

When $p_j = 1$, Eq. (3) becomes

$$u_j\left([a_1, \ldots, a_N]\right) = \frac{1}{N} \sum_{i=1}^{N} |a_i| .$$

If we further assume $a_i \geq 0$, for instance, by using a logistic sigmoid activation function on the projection of the lower layer, the activation is reduced to computing the average of these projections. This is a form of average pooling, where the non-linear projections represent the pooled layer. With a single filter, this is equivalent to the absolute value rectification proposed in [19]. If p_j is 2 instead of 1, the root-of-mean-squared pooling from [33] is recovered.

As p_j grows and ultimately approaches ∞, the L_p norm becomes

$$\lim_{p_j \to \infty} u_j\left([a_1, \ldots, a_N]\right) = \max\left\{|a_1|, \ldots, |a_N|\right\} .$$

When $N = 2$, this is a generalization of a rectified linear unit (ReLU) as well as the absolute value unit [19]. If each a_i is constrained to be non-negative, this corresponds exactly to the maxout unit.

In short, the proposed L_p unit interpolates among different pooling operators by the choice of its order p_j. This was noticed earlier in [8] as well as [33]. However, both of them stopped at analyzing the L_p norm as a pooling operator with a fixed order and comparing those conventional pooling operators against each other. The authors of [4] investigated a similar nonlinear activation function that was inspired by the cells in the primary visual cortex. In [18], the possibility of learning p has been investigated in a probabilistic setting in computer vision.

On the other hand, in this paper, we claim that the order p_j needs to, and can be *learned*, just like all other parameters of a deep neural network. Furthermore, we conjecture that (1) an optimal distribution of the orders of L_p units differs from one dataset to another, and (2) each L_p unit in a MLP requires a different order from the other L_p. These properties also distinguish the proposed L_p unit from the conventional radial-basis function network (see, e.g., [15])

4 Geometrical Interpretation

We analyze the proposed L_p unit from a geometrical perspective in order to motivate our conjecture regarding the order of the L_p units. Let the value of an L_p unit u be given by:

$$u(\mathbf{x}) = \left(\frac{1}{N} \sum_{i=1}^{N} \left| \mathbf{w}_i^\top \mathbf{x} - c_i \right|^p \right)^{\frac{1}{p}} , \tag{4}$$

where $\mathbf{w}_i$ represents the i-th column of the matrix $\mathbf{W}$. The equation above effectively says that the L_p unit computes the p-th norm of the projection of the input $\mathbf{x}$ on the subspace spanned by N vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_N\}$. Let us further assume that $\mathbf{x} \in \mathbb{R}^d$ is a vector in an Euclidean space.

The space onto which $\mathbf{x}$ is projected may be spanned by linearly dependent vectors $\mathbf{w}_i$'s. Due to the possible lack of the linear independence among these vectors, they span a subspace $\mathcal{S}$ of dimensionality $k \leq N$. The subspace $\mathcal{S}$ has its origin at $\mathbf{c} = [c_1, \ldots, c_N]$.

We impose a non-Euclidean geometry on this subspace by defining a norm in the space to be L_p with p potentially not 2, as in Eq. (4). The geometrical object to which a particular value of the L_p unit corresponds forms a superellipse when projected back into the original input space. [3] The superellipse is centered at the inverse projection of $\mathbf{c}$ in the Euclidean input space. Its shape varies according to the order p of the L_p unit and due to the potentially linearly-dependent bases. As long as $p \geq 1$ the shape remains convex. Fig. 1 (b) draws some of the superellipses one can get with different orders of p, as a function of a_1 (with a single filter).

In this way each L_p unit partitions the input space into two regions – inside and outside the superellipse. Each L_p unit uses a curved boundary of learned curvature to divide the space. This is in contrast to, for instance, a maxout unit which uses piecewise linear hyperplanes and might require more linear pieces to approximate the same curved segment.

4.1 Qualitative Analysis in Low Dimension

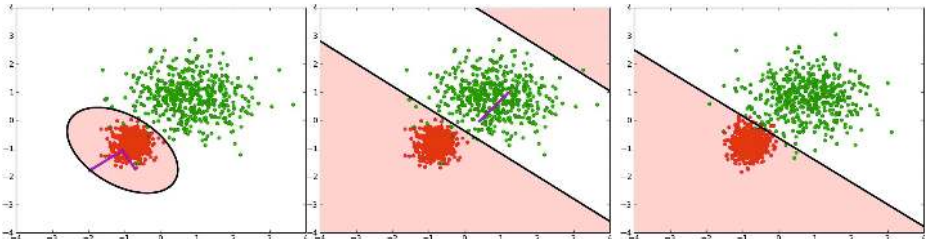
When the dimensionality of the input space is 2 and each L_p receives 2 input signals, we can visualize the partitions of the input space obtained using L_p units as well as conventional nonlinear activation functions. Here, we examine some artificially generated cases in a 2-D space.

Two Classes, Single L_p Unit. Fig. 2 shows a case of having two classes (● and ●) of which each corresponds to a Gaussian distribution. We trained MLPs having a single hidden neuron. When the MLPs had an L_p unit, we fixed p to either 2 or ∞. We can see in Fig. 2 (a) that the MLP with the L_2 unit divides the input space into two regions – inside and outside a rotated superellipse.[4] The superellipse correctly identified one of the classes (red).

In the case of $p = \infty$, what we see is a degenerate rectangle which is an extreme form of a superellipse. The superellipse again spotted one of the classes and appropriately draws a separating curve between the two classes.

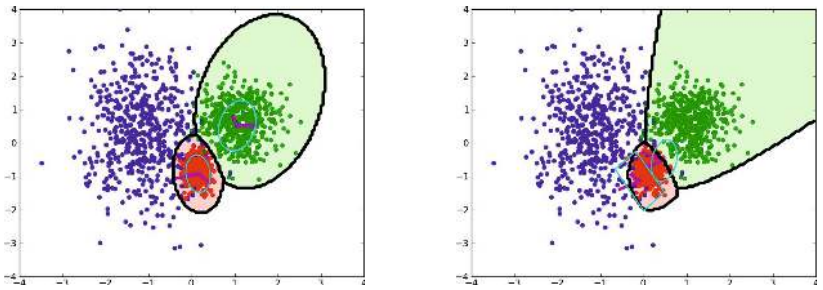
[3] Since $k \leq N$, the superellipse may be degenerate in the sense that in some of the $N-k$ axes the width may become infinitely large. However, as this does not invalidate our further argument, we continue to refer this kind of (degenerate) superellipse simply by an superellipse.

[4] Even though we use $p = 2$, which means an Euclidean space, we get a superellipse instead of a circle because of the linearly-dependent bases $\{\mathbf{w}_1, \ldots, \mathbf{w}_N\}$.



(a) L_p with $p = 2$ (b) L_p with $p = \infty$ (c) Rectifier

Fig. 2. Visualization of separating curves obtained using different activation functions. The underlying data distribution is a mixture of two Gaussian distributions. The red and green dots are the samples from the two classes, respectively, and the black curves are separating curves found by the MLPs. The purple lines indicate the axes of the subspace learned by each L_p unit. Best viewed in color.



(a) $p_1 = p_2 = 2$ (b) $p_1 = 2$ and $p_2 = \infty$

Fig. 3. Visualization of separating curves obtained using different orders of L_p units. The underlying data distribution is a mixture of three Gaussian distributions. The blue curves show the shape of the superellipse learned by each L_p unit. The red, green and blue dots are the samples. Otherwise, the same color convention as in Fig. 2 has been used.

In the case of rectifier units it could find a correct separating curve, but it is clear that a single rectifier unit can only partition the input space *linearly* unlike L_p units. A combination of several rectifier units can result in a nonlinear boundary, specifically a piecewise-linear one, though our claim is that you need more such rectifier units to get an arbitrarily shaped curve whose curvature changes in a highly nonlinear way.

Three Classes, Two L_p Units. Similarly to the previous experiment, we trained two MLPs having two L_p units on data generated from a mixture of three Gaussian distribution. Again, each mixture component corresponds to each class.

For one MLP we fixed the orders of the two L_p units to 2. In this case, see Fig. 3 (a), the separating curves are constructed by combining two translated

superellipses represented by the L_p units. These units were able to locate the two classes, which is sufficient for classifying the three classes (●, ● and ●).

The other MLP had two L_p units with p fixed to 2 and ∞, respectively. The L_2 unit defines, as usual, a superellipse, while the L_∞ unit defines a rectangle. The separating curves are constructed as a combination of the translated superellipse and rectangle and may have more non-trivial curvature as in Fig. 3 (b).

Furthermore, it is clear from the two plots in Fig. 3 that the curvature of the separating curves may change over the input space. It will be easier to model this non-stationary curvature using multiple L_p units with different p's.

Decision Boundary with Non-stationary Curvature: Representational Efficiency. In order to test the potential efficiency of the proposed L_p unit from its ability to learn the order p, we have designed a binary classification task that has a decision boundary with a non-stationary curvature. We use 5000 data points of which a subset is shown in Fig. 4 (a), where two classes are marked with blue dots (●) and red crosses (+), respectively.

On this dataset, we have trained MLPs with either L_p units, L_2 units (L_p units with fixed $p = 2$), maxout units, rectifiers or logistic sigmoid units. We varied the number of parameters, which correspond to the number of units in the case of rectifiers and logistic sigmoid units and to the number of inputs signals to the hidden layer in the case of L_p units, L_2 units and maxout units, from 2 to 16. For each setting, we trained ten randomly initialized MLPs. In order to reduce effects due to optimization difficulties, we used in all cases natural conjugate gradient [23].

From Fig. 4 (c), it is clear that the MLPs with L_p units outperform all others in terms of representing this specific curve. They were able to achieve the zero training error with only three units (i.e., 6 filters) on all ten random runs and achieved the lowest average training error even with less units. Importantly, the comparison to the performance of the MLPs with L_2 units shows that it is beneficial to learn the orders p of L_p units. For example, with only two L_2 units none of the ten random runs succeed while at least one succeeds with two L_p units. All the other MLPs, especially ones with rectifiers and maxout units which can only model the decision boundary with piecewise linear functions, were not able to achieve the similar efficiency of the MLPs with L_p units (see Fig 4 (b)).

Fig. 4 (a) also shows the decision boundary found by the MLP with two L_p units after training. As can be observed from the shapes of the L_p units (purple and cyan dashed curves), each L_p unit learned an appropriate order p that enables them to model the non-stationary decision boundary. Fig. 4 (b) shows the boundary obtained by a rectifier model with four units. We can see that it has to use linear segments to compose the boundary, resulting in not perfectly solving the task. The rectifier model represented here has 64 mistakes, versus 0 obtained by the L_p model.

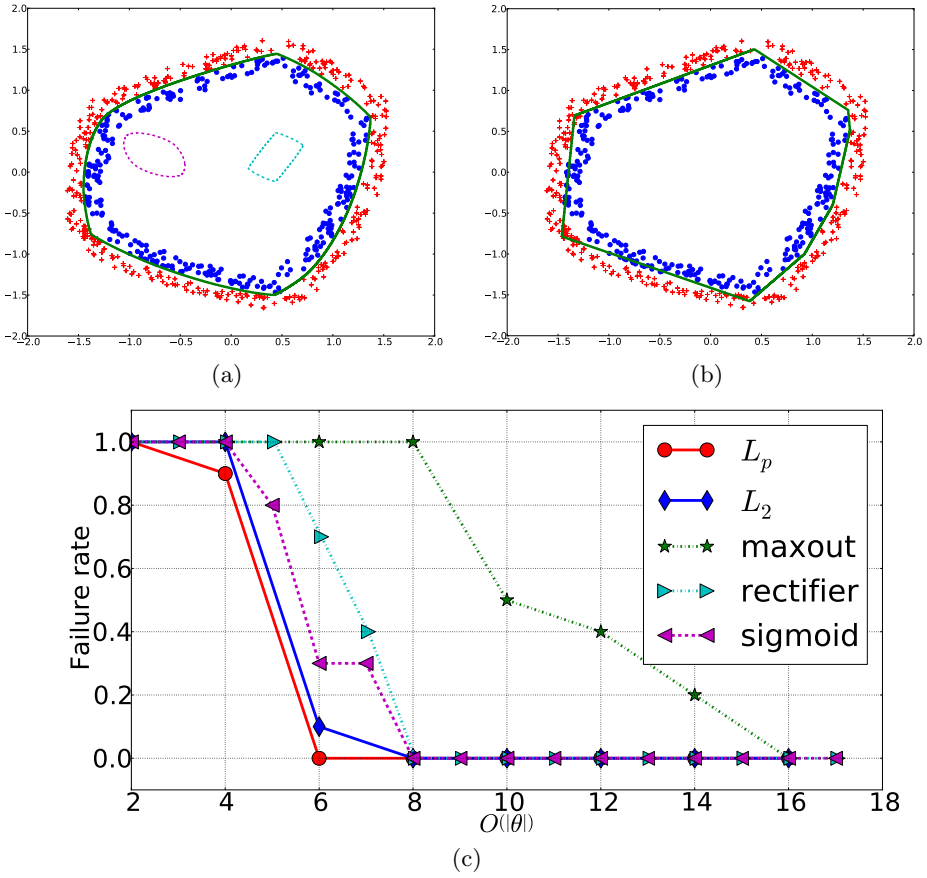
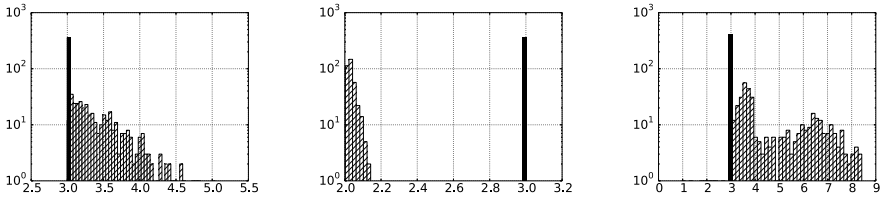


Fig. 4. (a) Visualization of data (two classes, + and •), a decision boundary learned by an MLP with two L_p units (green curve) and the shapes corresponding to the orders p's learned by the L_p units (purple and cyan dashed curves). (b) The same visualization done using four rectifiers. (c) The failure rates computed with MLPs using different numbers of different nonlinear activation functions (L_p: red solid curve with red •, L_2: blue solid curve with blue ♦, maxout: green dashed curve with green ⋆, rectifier: cyan dash-dot curve with cyan ▶ and sigmoid: purple dashed curve with purple ◀). The curves show the proportion of the failed attempts over ten random trials (y-axis) against either the number of units for sigmoid and rectifier model or the total number of linear projection going into the maxout units or L_p units (x-axis).



(a) MNIST (b) TFD (c) Pentomino

Fig. 5. Distributions of the initial (black bars ■) and learned (shaded bars ／) orders on MNIST, TFD and Pentomino. x-axis and y-axis show the order and the number of L_p units with the corresponding order. Note the difference in the scales of the x-axes and that the y-axes are in logarithmic scale.

Although this is a low-dimensional, artificially generated example, it demonstrates that the proposed L_p units are efficient at representing decision boundaries which have non-stationary curvatures.

5 Application to Recurrent Neural Networks

A conventional recurrent neural network (RNN) mostly uses saturating nonlinear activation functions such as tanh to compute the hidden state at each time step. This prevents the possible explosion of the activations of hidden states over time and in general results in more stable learning dynamics. However, at the same time, this does not allow us to build an RNN with recently proposed non-saturating activation functions such as rectifiers and maxout as well as the proposed L_p units.

The authors of [24] recently proposed three ways to extend the conventional, *shallow* RNN into a deep RNN. Among those three proposals, we notice that it is possible to use non-saturating activations functions for a deep RNN with deep transition without causing the instability of the model, because a saturating non-linearity (tanh) is applied in sandwich between the L_p MLP associated with each step.

The deep transition RNN (DT-RNN) has one or more intermediate layers between a pair of consecutive hidden states. The transition from a hidden state $\mathbf{h}_{t-1}$ at time $t-1$ to the next hidden state $\mathbf{h}_t$ is

$$\mathbf{h}_t = g\left(\mathbf{W}^\top f\left(\mathbf{U}^\top \mathbf{h}_{t-1} + \mathbf{V}^\top \mathbf{x}_t\right)\right),$$

not showing biases, as previously.

When a usual saturating nonlinear activation function is used for g, the activations of the hidden state $\mathbf{h}_t$ are bounded. This allows us to use any, potentially non-saturating nonlinear function for f. We can simply use a layer of the proposed L_p unit in the place of f.

As argued in [24], if the procedure of constructing a new summary which corresponds to the new hidden state $\mathbf{h}_t$ from the combination of the current

input $\mathbf{x}_t$ and the previous summary $\mathbf{h}_{t-1}$ is highly nonlinear, any benefit of the proposed L_p unit over the existing, conventional activation functions in feedforward neural networks should naturally translate to these deep RNNs as well. We show this effect empirically later by training a deep output, deep transition RNN (DOT-RNN) with the proposed L_p units.

6 Experiments

In this section, we provide empirical evidences showing the advantages of utilizing the L_p units. In order to clearly distinguish the effect of employing L_p units from introducing data-specific model architectures, all the experiments in this section are performed by neural networks having densely connected hidden layers.

6.1 Claims to Verify

Let us first list our claims about the proposed L_p units that need to be verified through a set of experiments. We expect the following from adopting L_p units in an MLP:

1. The optimal orders of L_p units vary across datasets
2. An optimal distribution of the orders of L_p units is not close to a (shifted) Dirac delta distribution

The first claim states that there is no universally optimal order p_j. We train MLPs on a number of benchmark datasets to see the resulting distribution of p_j's. If the distributions had been similar between tasks, claim 1 would be rejected.

This naturally connects to the second claim. As the orders are estimated via learning, it is unlikely that the orders of all L_p units will convergence to a single value such as ∞ (maxout or max pooling), 1 (average pooling) or 2 (RMS pooling). We expect that the response of each L_p unit will specialize by using a distinct order. The inspection of the trained MLPs to confirm the first claim will validate this claim as well.

On top of these claims, we expect that an MLP having L_p units, when the parameters including the orders of the L_p units are well estimated, will achieve highly competitive classification performance. In addition to classification tasks using feedforward neural networks, we anticipate that a recurrent neural network benefits from having L_p units in the intermediate layer between the consecutive hidden states, as well.

6.2 Datasets

For feedforward neural networks or MLPs, we have used four datasets; MNIST [21], Pentomino [14], the Toronto Face Database (TFD) [31] and Forest Covertype[5] (data split DS2-581) [32]. MNIST, TFD and Forest Covertype are three

[5] We use the first 16 principal components only.

representative benchmark datasets, and Pentomino is a relatively recently proposed dataset that is known to induce a difficult optimization challenge for a deep neural network. We have used three music datasets from [7] for evaluating the effect of L_p units on deep recurrent neural networks.

6.3 Distributions of the Orders of L_p Units

To understand how the estimated orders p of the proposed L_p unit are distributed we trained MLPs with a single L_p layer on MNIST, TFD and Pentomino. We measured validation error to search for good hyperparameters, including the number of L_p units and number of filters (input signals) per L_p unit. However, for Pentomino, we simply fixed the size of the L_p layer to 400, and each L_p unit received signals from six hidden units below.

Table 1. The means and standard deviations of the estimated orders of L_p units

Dataset	Mean	Std. Dev.
MNIST	3.44	0.38
TFD	2.04	0.22
Pentomino	5.81	1.56

In Table 1, the averages and standard deviations of the estimated orders of the L_p units in the single-layer MLPs are listed for MNIST, TFD and Pentomino. It is clear that the distribution of the orders depend heavily on the dataset, which confirms our first claim described earlier. From Fig. 5 we can clearly see that even in a single model the estimated orders vary quite a lot, which confirms our second claim. Interestingly, in the case of Pentomino, the distribution of the orders consists of two distinct modes.

The plots in Fig. 5 clearly show that the orders of the L_p units change significantly from their initial values over training. Although we initialized the orders of the L_p units around 3 for all the datasets, the resulting distributions of the orders are significantly different among those three datasets. This further confirms both of our claims. As a simple empirical confirmation we tried the same experiment with the fixed $p = 2$ on TFD and achieved a worse test error of 0.21.

6.4 Generalization Performance

The ultimate goal of any novel nonlinear activation function for an MLP is to achieve better generalization performance. We conjectured that by learning the orders of L_p units an MLP with L_p layers will achieve highly competitive classification performance.

For MNIST we trained an MLP having two L_p layers followed by a softmax output layer. We used a recently introduced regularization technique called dropout [16]. With this MLP we were able to achieve 99.03% accuracy on the test set, which is comparable to the state-of-the-art accuracy of 99.06% obtained by the MLP with maxout units [13].

On TFD we used the same MLP from the previous experiment to evaluate generalization performance. We achieved a recognition rate of 79.25%. Although we use neither pretraining nor unlabeled samples, our result is close to the current

Table 2. The generalization errors on three datasets obtained by MLPs using the proposed L_p units. The previous state-of-the-art results obtained by others are also presented for comparison.

Data	MNIST	TFD	Pentomino	Forest Covertype
L_p	0.97 %	20.75 %	31.85 %	2.83 %
Previous	0.94 %[1]	21.29 %[2]	44.6 %[3]	2.78 %[4]

state-of-the-art rate of 82.4% on the permutation-invariant version of the task reported by [26] who pretrained their models with a large amount of unlabeled samples.

As we have used the five-fold cross validation to find the optimal hyperparameters, we were able to use this to investigate the variance of the estimations of the p values. Table 3 shows the averages and standard deviations of the estimated orders for MLPs trained on the five folds using the best hyperparameters. It is clear that in all the cases the orders ended up in a similar region near two without too much difference in the variance.

Similarly, we have trained five randomly initialized MLPs on MNIST and observed the similar phenomenon of all the resulting MLPs having similar distributions of the orders. The standard deviation of the averages of the learned orders was only 0.028, while its mean is 2.16.

The MLP having a single L_p layer was able to classify the test samples of Pentomino with 31.38% error rate. This is the best result reported so far on Pentomino dataset [14] without using any kind of prior information about the task (the best previous result was 44.6% error).

On Forest Covertype an MLP having three L_p layers was trained. The MLP was able to classify the test samples with only 2.83% error. The improvement is large compared to the previous state-of-the-art rate of 3.13% achieved by the manifold tangent classifier having four hidden layers of logistic sigmoid units [28]. The result obtained with the L_p is comparable to that obtained with the MLP having maxout units.

These results as well as previous best results for all datasets are summarized in Table 2.

In all experiments, we optimized hyperparameters such as an initial learning rate and its scheduling to minimize validation error, using random search [3], which is generally more efficient than grid search when the number of hyperparameters is not tiny. Each MLP was trained by stochastic gradient descent. All the experiments in this paper were done using the Pylearn2 library [12].

[1] Reported in [13].
[2] This result was obtained by training an MLP with rectified linear units which outperformed an MLP with maxout units.
[3] Reported in [14].
[4] This result was obtained by training an MLP with maxout units which outperformed an MLP with rectified linear units.

6.5 Deep Recurrent Neural Networks

We tried the polyphonic music prediction tasks with three music datasets; Nottingam, JSB and MuseData [7]. The DOT-RNNs we trained had deep transition with L_p units and tanh units and deep output function with maxout in the intermediate layer (see Fig. 6 for the illustration). We coarsely optimized the size of the models and the initial leaning rate as well as its schedule to maximize the performance on validation sets. Also, we chose whether to threshold the norm of the gradient based on the validation performance [25]. All the models were trained with dropout [16].

Table 3. The means and standard deviations of the estimated orders of L_p units obtained during the hyperparameter search using the 5-fold cross-validation.

Fold	Mean	Std. Dev.
1	2.00	0.24×10^{-4}
2	2.00	0.24×10^{-4}
3	2.01	0.77×10^{-4}
4	2.02	1.50×10^{-4}
5	2.00	0.24×10^{-4}

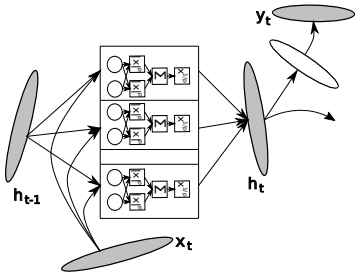


Fig. 6. The illustration of the DOT-RNN using L_p units

As shown in Table 4, we were able to achieve the state-of-the-art results (RNN-only case) on all the three datasets. These results are much better than those achieved by the same DOT-RNNs using logistic sigmoid units in both deep transition and deep output, which suggests the superiority of the proposed L_p units over the conventional saturating activation functions. This suggests that the proposed L_p units are well suited not only to feedforward neural networks, but also to recurrent neural networks. However, we acknowledge that more investigation into applying L_p units is needed in the future to draw more concrete conclusion on the benefits of the L_p units in recurrent neural networks.

Dataset	DOT-RNN L_p	DOT-RNN sigmoid*	RNN *
Nottingam	2.95	3.22	3.09
JSB	7.92	8.44	8.01
Muse	6.59	6.97	6.75

Table 4. The negative log-probability of the test sets computed by the trained DOT-RNNs. ($\star$) These are the results achieved using DOT-RNNs having logistic sigmoid units, which we reported in [24]. (*) These are the previous best results achieved using conventional RNNs obtained in [2].

7 Conclusion

In this paper, we have proposed a novel nonlinear activation function based on the generalization of widely used pooling operators. The proposed nonlinear

activation function computes the L_p norm of several projections of the lower layer. Max-, average- and root-of-mean-squared pooling operators are special cases of the proposed activation function, and naturally the recently proposed maxout unit is closely related under an assumption of non-negative input signals.

An important difference of the L_p unit from conventional pooling operators is that the order of the unit is *learned* rather than pre-defined. We claimed that this estimation of the orders is important and that the optimal model should have L_p units with various orders.

Our analysis has shown that an L_p unit defines a non-Euclidean subspace whose metric is defined by the L_p norm. When projected back into the input space, the L_p unit defines an ellipsoidal boundary. We conjectured and showed in a small scale experiment that the combination of these curved boundaries may more efficiently model separating curves of data with non-stationary curvature.

These claims were empirically verified via training both deep feedforward neural networks and deep recurrent neural networks. We tested the feedforward neural network on on four benchmark datasets; MNIST, Toronto Face Database, Pentomino and Forest Covertype, and tested the recurrent neural networks on the task of polyphonic music prediction. The experiments revealed that the distribution of the estimated orders of L_p units indeed depends highly on dataset and is far away from a Dirac delta distribution. Additionally, our conjecture that deep neural networks with L_p units will be able to achieve competitive generalization performance was empirically confirmed.

Acknowledgments. We would like to thank the developers of Pylearn2 [12] and Theano [6,1]. We would also like to thank CIFAR, and Canada Research Chairs for funding, and Compute Canada, and Calcul Québec for providing computational resources.

References

1. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N., Bengio, Y.: Theano: new features and speed improvements. In: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop (2012)
2. Bayer, J., Osendorfer, C., Korhammer, D., Chen, N., Urban, S., van der Smagt, P.: On fast dropout and its applicability to recurrent networks. arXiv:1311.0701 (cs.NE) (2013)
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Machine Learning Res. 13, 281–305 (2012)
4. Bergstra, J., Bengio, Y., Louradour, J.: Suitability of V1 energy models for object classification. Neural Computation 23(3), 774–790 (2011)
5. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy) (2010)
6. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). Oral Presentation (June 2010)

7. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In: ICML 2012 (2012)
8. Boureau, Y., Ponce, J., LeCun, Y.: A theoretical analysis of feature pooling in vision algorithms. In: Proc. International Conference on Machine learning, ICML 2010 (2010)
9. Ciresan, D., Meier, U., Masci, J., Schmidhuber, J.: Multi column deep neural network for traffic sign classification. Neural Networks 32, 333–338 (2012)
10. Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics 36, 193–202 (1980)
11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS 2011 (2011)
12. Goodfellow, I.J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., Bengio, Y.: Pylearn2: a machine learning research library. arXiv preprint arXiv:1308.4214 (2013)
13. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: ICML 2013 (2013)
14. Gulcehre, C., Bengio, Y.: Knowledge matters: Importance of prior information for optimization. In: International Conference on Learning Representations, ICLR 2013 (2013)
15. Haykin, S.: Neural Networks and Learning Machines, 3rd edn. Prentice Hall (November 2008)
16. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580 (2012)
17. Hubel, D., Wiesel, T.: Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London) 195, 215–243 (1968)
18. Hyvärinen, A., Köster, U.: Complex cell pooling and the statistics of natural images. Network: Computation in Neural Systems 18(2), 81–100 (2007)
19. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: Proc. International Conference on Computer Vision (ICCV 2009), pp. 2146–2153. IEEE (2009)
20. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25, NIPS 2012 (2012)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
22. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Bottou, L., Littman, M. (eds.) Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010), pp. 807–814. ACM (2010)
23. Pascanu, R., Bengio, Y.: Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584 (2013)
24. Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y.: How to construct deep recurrent neural networks. arXiv:1312.6026 (cs.NE) (December 2013)
25. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML 2013 (2013)
26. Ranzato, M., Mnih, V., Susskind, J.M., Hinton, G.E.: Modeling natural images using gated mrfs. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(9), 2206–2222 (2013)

27. Riesenhuber, M., Poggio, T.: Hierarchical models of object recognition in cortex. Nature Neuroscience (1999)
28. Rifai, S., Dauphin, Y., Vincent, P., Bengio, Y., Muller, X.: The manifold tangent classifier. In: NIPS 2011 (2011)
29. Rosenblatt, F.: Principles of neurodynamics: perceptrons and the theory of brain mechanisms. Report (Cornell Aeronautical Laboratory). Spartan Books (1962)
30. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature 323, 533–536 (1986)
31. Susskind, J., Anderson, A., Hinton, G.E.: The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto (2010)
32. Trebar, M., Steele, N.: Application of distributed svm architectures in classifying forest data cover types. Computers and Electronics in Agriculture 63(2), 119–130 (2008)
33. Yang, J., Yu, K., Gong, Y., Huang, T.: Linear spatial pyramid matching using sparse coding for image classification. In: Proc. Conference on Computer Vision and Pattern Recognition (CVPR 2010) (2010)