Maxout Networks

Ian J. Goodfellow GOODFELI@IRO.UMONTREAL.CA
David Warde-Farley WARDEFAR@IRO.UMONTREAL.CA
Mehdi Mirza MIRZAMOM@IRO.UMONTREAL.CA
Aaron Courville AARON.COURVILLE@UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Département d'Informatique et de Recherche Opérationelle, Université de Montréal
2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Abstract

We consider the problem of designing models to leverage a recently introduced approximate model averaging technique called *dropout*. We define a simple new model called *maxout* (so named because its *out*put is the *max* of a set of inputs, and because it is a natural companion to dropout) designed to both facilitate optimization by dropout and improve the accuracy of dropout's fast approximate model averaging technique. We empirically verify that the model successfully accomplishes both of these tasks. We use maxout and dropout to demonstrate state of the art classification performance on four benchmark datasets: MNIST, CIFAR-10, CIFAR-100, and SVHN.

1. Introduction

Dropout (Hinton et al., 2012) provides an inexpensive and simple means of both training a large ensemble of models that share parameters and approximately averaging together these models' predictions. Dropout applied to multilayer perceptrons and deep convolutional networks has improved the state of the art on tasks ranging from audio classification to very large scale object recognition (Hinton et al., 2012; Krizhevsky et al., 2012). While dropout is known to work well in practice, it has not previously been demonstrated to actually perform model averaging for deep architectures [1] .

Dropout is generally viewed as an indiscriminately applicable tool that reliably yields a modest improvement in performance when applied to almost any model.

We argue that rather than using dropout as a slight performance enhancement applied to arbitrary models, the best performance may be obtained by directly designing a model that enhances dropout's abilities as a model averaging technique. Training using dropout differs significantly from previous approaches such as ordinary stochastic gradient descent. Dropout is most effective when taking relatively large steps in parameter space. In this regime, each update can be seen as making a significant update to a different model on a different subset of the training set. The ideal operating regime for dropout is when the overall training procedure resembles training an ensemble with bagging under parameter sharing constraints. This differs radically from the ideal stochastic gradient operating regime in which a single model makes steady progress via small steps. Another consideration is that dropout model averaging is only an approximation when applied to deep models. Explicitly designing models to minimize this approximation error may thus enhance dropout's performance as well.

We propose a simple model that we call *maxout* that has beneficial characteristics both for optimization and model averaging with dropout. We use this model in conjunction with dropout to set the state of the art on four benchmark datasets [2] .

2. Review of dropout

Dropout is a technique that can be applied to deterministic feedforward architectures that predict an output y given input vector v. These architectures contain

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1] Between submission and publication of this paper, we have learned that Srivastava (2013) performed experiments on this subject similar to ours.

[2] Code and hyperparameters available at http://www-etud.iro.umontreal.ca/~goodfeli/maxout.html

a series of hidden layers $\mathbf{h} = \{h^{(1)}, \ldots, h^{(L)}\}$. Dropout trains an ensemble of models consisting of the set of all models that contain a subset of the variables in both v and $\mathbf{h}$. The same set of parameters θ is used to parameterize a family of distributions $p(y \mid v; \theta, \mu)$ where $\mu \in \mathcal{M}$ is a binary mask determining which variables to include in the model. On each presentation of a training example, we train a different sub-model by following the gradient of $\log p(y \mid v; \theta, \mu)$ for a different randomly sampled μ. For many parameterizations of p (such as most multilayer perceptrons) the instantiation of different sub-models $p(y \mid v; \theta, \mu)$ can be obtained by elementwise multiplication of v and $\mathbf{h}$ with the mask μ. Dropout training is similar to bagging (Breiman, 1994), where many different models are trained on different subsets of the data. Dropout training differs from bagging in that each model is trained for only one step and all of the models share parameters. For this training procedure to behave as if it is training an ensemble rather than a single model, each update must have a large effect, so that it makes the sub-model induced by that μ fit the current input v well.

The functional form becomes important when it comes time for the ensemble to make a prediction by averaging together all the sub-models' predictions. Most prior work on bagging averages with the arithmetic mean, but it is not obvious how to do so with the exponentially many models trained by dropout. Fortunately, some model families yield an inexpensive *geometric* mean. When $p(y \mid v; \theta) = \text{softmax}(v^T W + b)$, the predictive distribution defined by renormalizing the geometric mean of $p(y \mid v; \theta, \mu)$ over $\mathcal{M}$ is simply given by $\text{softmax}(v^T W/2 + b)$. In other words, the average prediction of exponentially many sub-models can be computed simply by running the full model with the weights divided by 2. This result holds exactly in the case of a single layer softmax model. Previous work on dropout applies the same scheme in deeper architectures, such as multilayer perceptrons, where the $W/2$ method is only an approximation to the geometric mean. The approximation has not been characterized mathematically, but performs well in practice.

3. Description of maxout

The maxout model is simply a feed-forward achitecture, such as a multilayer perceptron or deep convolutional neural network, that uses a new type of activation function: the maxout unit. Given an input $x \in \mathbb{R}^d$ (x may be v, or may be a hidden layer's state), a maxout hidden layer implements the function

$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

where $z_{ij} = x^T W_{\ldots ij} + b_{ij}$, and $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are learned parameters. In a convolutional network, a maxout feature map can be constructed by taking the maximum across k affine feature maps (i.e., pool across channels, in addition spatial locations). When training with dropout, we perform the elementwise multiplication with the dropout mask immediately prior to the multiplication by the weights in all cases–we do not drop inputs to the max operator. A single maxout unit can be interpreted as making a piecewise linear approximation to an arbitrary convex function. Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit. See Fig. 1 for a graphical depiction of how this works.

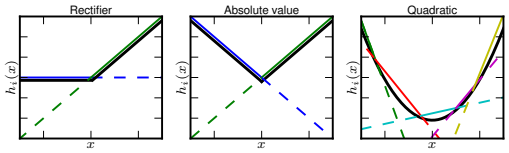

Figure 1. Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

Maxout abandons many of the mainstays of traditional activation function design. The representation it produces is not sparse at all (see Fig. 2), though the gradient is highly sparse and dropout will artificially sparsify the effective representation during training. While maxout may learn to saturate on one side or the other this is a measure zero event (so it is almost never bounded from above). While a significant proportion of parameter space corresponds to the function being bounded from below, maxout is not constrained to learn to be bounded at all. Maxout is locally linear almost everywhere, while many popular activation functions have signficant curvature. Given all of these departures from standard practice, it may seem surprising that maxout activation functions work at all, but we find that they are very robust and easy to train with dropout, and achieve excellent performance.

4. Maxout is a universal approximator

A standard MLP with enough hidden units is a universal approximator. Similarly, maxout networks are universal approximators. Provided that each individual maxout unit may have arbitrarily many affine components, we show that a maxout model with just two hidden units can approximate, arbitrarily well, any

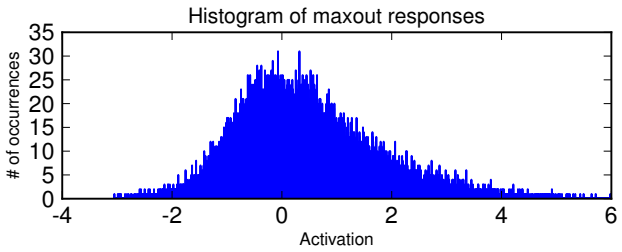

Figure 2. The activations of maxout units are not sparse.

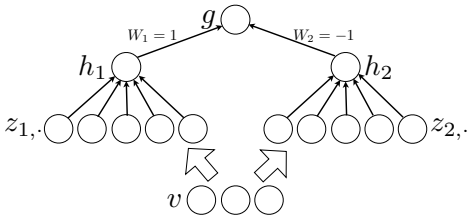

Figure 3. An MLP containing two maxout units can arbitrarily approximate any continuous function. The weights in the final layer can set g to be the difference of h_1 and h_2. If z_1 and z_2 are allowed to have arbitrarily high cardinality, h_1 and h_2 can approximate any convex function. g can thus approximate any continuous function due to being a difference of approximations of arbitrary convex functions.

continuous function of $v \in \mathbb{R}^n$. A diagram illustrating the basic idea of the proof is presented in Fig. 3.

Consider the continuous piecewise linear (PWL) function $g(v)$ consisting of k locally affine regions on $\mathbb{R}^n$.

Proposition 4.1 *(From Theorem 2.1 in (Wang, 2004)) For any positive integers m and n, there exist two groups of $n+1$-dimensional real-valued parameter vectors $[W_{1j}, b_{1j}], j \in [1, k]$ and $[W_{2j}, b_{2j}], j \in [1, k]$ such that:*

$$g(v) = h_1(v) - h_2(v) \tag{1}$$

That is, any continuous PWL function can be expressed as a difference of two convex PWL functions. The proof is given in (Wang, 2004).

Proposition 4.2 *From the* Stone-Weierstrass approximation theorem, *let C be a compact domain $C \subset \mathbb{R}^n$, $f : C \to \mathbb{R}$ be a continuous function, and $\epsilon > 0$ be any positive real number. Then there exists a continuous PWL function g, (depending upon ϵ), such that for all $v \in C$, $|f(v) - g(v)| < \epsilon$.*

Theorem 4.3 Universal approximator theorem. *Any continuous function f can be approximated arbitrarily well on a compact domain $C \subset \mathbb{R}^n$ by a maxout network with two maxout hidden units.*

Table 1. Test set misclassification rates for the best methods on the permutation invariant MNIST dataset. Only methods that are regularized by modeling the input distribution outperform the maxout MLP.

Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov & Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.91%
Deep Convex Network (Yu & Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et al., 2012)	0.79%

Sketch of Proof By Proposition 4.2, any continuous function can be approximated arbitrarily well (up to ϵ), by a piecewise linear function. We now note that the representation of piecewise linear functions given in Proposition 4.1 exactly matches a maxout network with two hidden units $h_1(v)$ and $h_2(v)$, with sufficiently large k to achieve the desired degree of approximation ϵ. Combining these, we conclude that a two hidden unit maxout network can approximate any continuous function $f(v)$ arbitrarily well on the compact domain C. In general as $\epsilon \to 0$, we have $k \to \infty$.

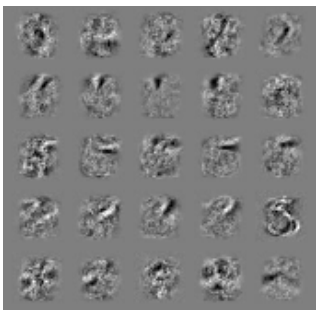

Figure 4. Example filters learned by a maxout MLP trained with dropout on MNIST. Each row contains the filters whose responses are pooled to form a maxout unit.

5. Benchmark results

We evaluated the maxout model on four benchmark datasets and set the state of the art on all of them.

Table 2. Test set misclassification rates for the best methods on the general MNIST dataset, excluding methods that augment the training data.

METHOD	TEST ERROR
2-LAYER CNN+2-LAYER NN (JARRETT ET AL., 2009)	0.53%
STOCHASTIC POOLING (ZEILER & FERGUS, 2013)	0.47%
Conv. maxout + dropout	**0.45%**

Table 3. Test set misclassification rates for the best methods on the CIFAR-10 dataset.

METHOD	TEST ERROR
STOCHASTIC POOLING (ZEILER & FERGUS, 2013)	15.13%
CNN + SPEARMINT (SNOEK ET AL., 2012)	14.98%
Conv. maxout + dropout	**11.68 %**
CNN + SPEARMINT + DATA AUGMENTATION (SNOEK ET AL., 2012)	9.50 %
Conv. maxout + dropout + data augmentation	**9.38 %**

5.1. MNIST

The MNIST (LeCun et al., 1998) dataset consists of 28 × 28 pixel greyscale images of handwritten digits 0-9, with 60,000 training and 10,000 test examples. For the *permutation invariant* version of the MNIST task, only methods unaware of the 2D structure of the data are permitted. For this task, we trained a model consisting of two densely connected maxout layers followed by a softmax layer. We regularized the model with dropout and by imposing a constraint on the norm of each weight vector, as in (Srebro & Shraibman, 2005). Apart from the maxout units, this is the same architecture used by Hinton et al. (2012). We selected the hyperparameters by minimizing the error on a validation set consisting of the last 10,000 training examples. To make use of the full training set, we recorded the value of the log likelihood on the first 50,000 examples at the point of minimal validation error. We then continued training on the full 60,000 example training set until the validation set log likelihood matched this number. We obtained a test set error of 0.94%, which is the best result we are aware of that does not use unsupervised pretraining. We summarize the best published results on permutation invariant MNIST in Table 1.

We also considered the MNIST dataset without the permutation invariance restriction. In this case, we used three convolutional maxout hidden layers (with spatial max pooling on top of the maxout layers) followed by a densely connected softmax layer. We were able to rapidly explore hyperparameter space thanks to the extremely fast GPU convolution library developed by Krizhevsky et al. (2012). We obtained a test set error rate of 0.45%, which sets a new state of the art in this category. (It is possible to get better results on MNIST by augmenting the dataset with transformations of the standard set of images (Ciresan et al., 2010)) A summary of the best methods on the general MNIST dataset is provided in Table 2.

5.2. CIFAR-10

The CIFAR-10 dataset (Krizhevsky & Hinton, 2009) consists of 32 × 32 color images drawn from 10 classes split into 50,000 train and 10,000 test images. We preprocess the data using global contrast normalization and ZCA whitening.

We follow a similar procedure as with the MNIST dataset, with one change. On MNIST, we find the best number of training epochs in terms of validation set error, then record the training set log likelihood and continue training using the entire training set until the validation set log likelihood has reached this value. On CIFAR-10, continuing training in this fashion is infeasible because the final value of the learning rate is very small and the validation set error is very high. Training until the validation set likelihood matches the cross-validated value of the training likelihood would thus take prohibitively long. Instead, we retrain the model from scratch, and stop when the new likelihood matches the old one.

Our best model consists of three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. Using this approach we obtain a test set error of 11.68%, which improves upon the state of the art by over two percentage points. (If we do not train on the validation set, we obtain a test set error of 13.2%, which also improves over the previous state of the art). If we additionally augment the data with translations and horizontal reflections, we obtain the absolute state of the art on this task at 9.35% error. In this case, the likelihood during the retrain never reaches the likelihood from the validation run, so we retrain for the same number of epochs as the validation run. A summary of the best CIFAR-10 methods is provided in Table 3.

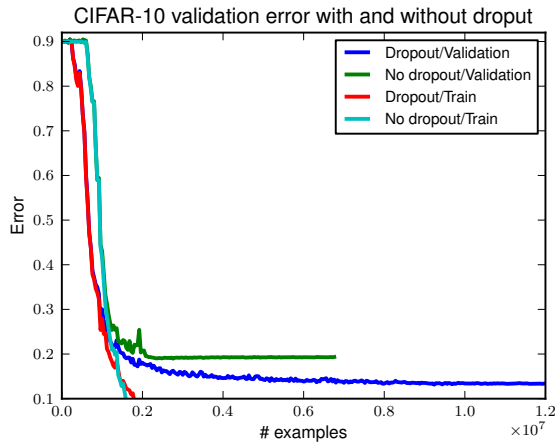

Figure 5. When training maxout, the improvement in validation set error that results from using dropout is dramatic. Here we find a greater than 25% reduction in our validation set error on CIFAR-10.

Table 4. Test set misclassification rates for the best methods on the CIFAR-100 dataset.

Method	Test error
Learned pooling (Malinowski & Fritz, 2013)	43.71%
Stochastic pooling(Zeiler & Fergus, 2013)	42.51%
Conv. maxout + dropout	**38.57%**

5.3. CIFAR-100

The CIFAR-100 (Krizhevsky & Hinton, 2009) dataset is the same size and format as the CIFAR-10 dataset, but contains 100 classes, with only one tenth as many labeled examples per class. Due to lack of time we did not extensively cross-validate hyperparameters on CIFAR-100 but simply applied hyperparameters we found to work well on CIFAR-10. We obtained a test set error of 38.57%, which is state of the art. If we do not retrain using the entire training set, we obtain a test set error of 41.48%, which also surpasses the current state of the art. A summary of the best methods on CIFAR-100 is provided in Table 4.

5.4. Street View House Numbers

The SVHN (Netzer et al., 2011) dataset consists of color images of house numbers collected by Google Street View. The dataset comes in two formats. We consider the second format, in which each image is of size 32 × 32 and the task is to classify the digit in the center of the image. Additional digits may appear beside it but must be ignored. There are 73,257 digits in the training set, 26,032 digits in the test set and 531,131 additional, somewhat less difficult examples,

Table 5. Test set misclassification rates for the best methods on the SVHN dataset.

Method	Test error
(Sermanet et al., 2012a)	4.90%
Stochastic pooling (Zeiler & Fergus, 2013)	2.80 %
Rectifiers + dropout (Srivastava, 2013)	2.78 %
Rectifiers + dropout + synthetic translation (Srivastava, 2013)	2.68 %
Conv. maxout + dropout	**2.47 %**

to use as an extra training set. Following Sermanet et al. (2012b), to build a validation set, we select 400 samples per class from the training set and 200 samples per class from the extra set. The remaining digits of the train and extra sets are used for training.

For SVHN, we did not train on the validation set at all. We used it only to find the best hyperparameters. We applied local contrast normalization preprocessing the same way as Zeiler & Fergus (2013). Otherwise, we followed the same approach as on MNIST. Our best model consists of three convolutional maxout hidden layers and a densely connected maxout layer followed by a densely connected softmax layer. We obtained a test set error rate of 2.47%, which sets the state of the art. A summary of comparable methods is provided in Table 5.

6. Comparison to rectifiers

One obvious question about our results is whether we obtained them by improved preprocessing or larger models, rather than by the use of maxout. For MNIST we used no preprocessing, and for SVHN, we use the same preprocessing as Zeiler & Fergus (2013). However on the CIFAR datasets we did use a new form of preprocessing. We therefore compare maxout to rectifiers run with the same processing and a variety of model sizes on this dataset.

By running a large cross-validation experiment (see Fig. 6) we found that maxout offers a clear improvement over rectifiers. We also found that our preprocessing and size of models improves rectifiers and dropout beyond the previous state of the art result. Cross-channel pooling is a method for reducing the size of state and number of parameters needed to have a given number of filters in the model. Performance seems to correlate well with the number of filters for maxout but with the number of output units for rectifiers–i.e, rectifier units do not benefit much

from cross-channel pooling. Rectifier units do best without cross-channel pooling but with the same number of filters, meaning that the size of the state and the number of parameters must be about k times higher for rectifiers to obtain generalization performance approaching that of maxout.

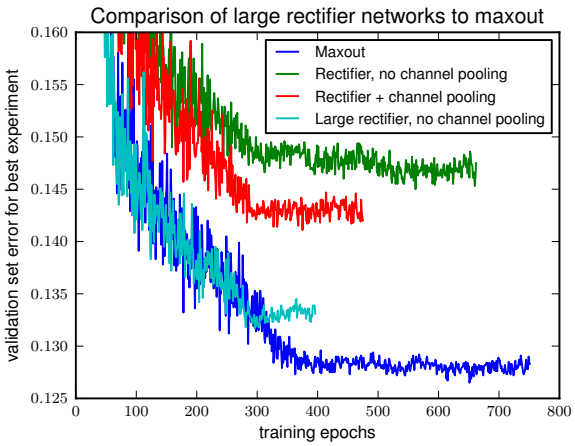

Figure 6. We cross-validated the momentum and learning rate for four architectures of model: 1) Medium-sized maxout network. 2) Rectifier network with cross-channel pooling, and exactly the same number of parameters and units as the maxout network. 3) Rectifier network without cross-channel pooling, and the same number of units as the maxout network (thus fewer parameters). 4) Rectifier network without cross-channel pooling, but with k times as many units as the maxout network. Because making layer i have k times more outputs increases the number of inputs to layer $i+1$, this network has roughly k times more parameters than the maxout network, and requires significantly more memory and runtime. We sampled 10 learning rate and momentum schedules and random seeds for dropout, then ran each configuration for all 4 architectures. Each curve terminates after failing to improve the validation error in the last 100 epochs.

7. Model averaging

Having demonstrated that maxout networks are effective models, we now analyze the reasons for their success. We first identify reasons that maxout is highly compatible with dropout's approximate model averaging technique.

The intuitive justification for averaging sub-models by dividing the weights by 2 given by (Hinton et al., 2012) is that this does exact model averaging for a single layer model, softmax regression. To this characterization, we add the observation that the model averaging remains exact if the model is extended to multiple linear layers. While this has the same representational power as a single layer, the expression of the weights as a product of several matrices could have a different inductive bias. More importantly, it indicates that dropout does exact model averaging in deeper architectures provided that they are locally linear among the space of inputs to each layer that are visited by applying different dropout masks.

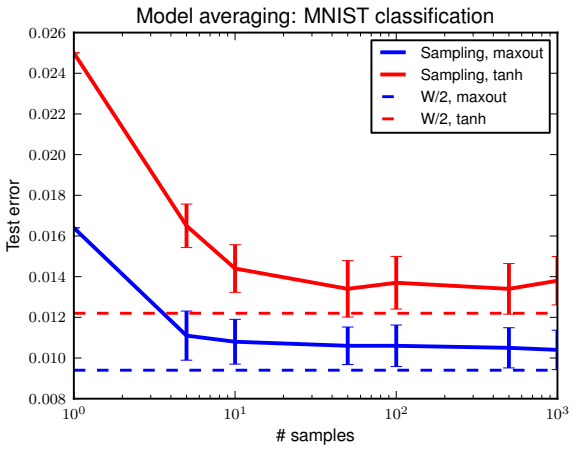

Figure 7. The error rate of the prediction obtained by sampling several sub-models and taking the geometric mean of their predictions approaches the error rate of the prediction made by dividing the weights by 2. However, the divided weights still obtain the best test error, suggesting that dropout is a good approximation to averaging over a very large number of models. Note that the correspondence is more clear in the case of maxout.

We argue that dropout training encourages maxout units to have large linear regions around inputs that appear in the training data. Because each sub-model must make a good prediction of the output, each unit should learn to have roughly the same activation regardless of which inputs are dropped. In a maxout network with arbitrarily selected parameters, varying the dropout mask will often move the effective inputs far enough to escape the local region surrounding the clean inputs in which the hidden units are linear, i.e., changing the dropout mask could frequently change which piece of the piecewise function an input is mapped to. Maxout *trained with dropout* may have the identity of the maximal filter in each unit change relatively rarely as the dropout mask changes. Networks of linear operations and max$(\cdot)$ may learn to exploit dropout's approximate model averaging technique well.

Many popular activation functions have significant curvature nearly everywhere. These observations suggest that the approximate model averaging of dropout will not be as accurate for networks incorporating such activation functions. To test this, we compared the best maxout model trained on MNIST with dropout to a hyperbolic tangent network trained on MNIST

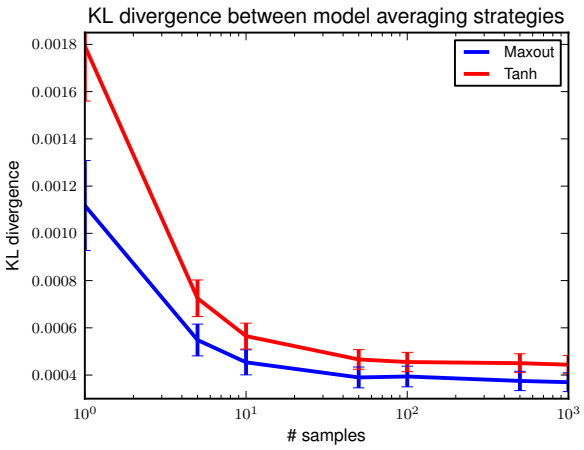

Figure 8. The KL divergence between the distribution predicted using the dropout technique of dividing the weights by 2 and the distribution obtained by taking the geometric mean of the predictions of several sampled models decreases as the number of samples increases. This suggests that dropout does indeed do model averaging, even for deep networks. The approximation is more accurate for maxout units than for tanh units.

with dropout. We sampled several subsets of each model and compared the geometric mean of these sampled models' predictions to the prediction made using the dropout technique of dividing the weights by 2. We found evidence that dropout is indeed performing model averaging, even in multilayer networks, and that it is more accurate in the case of maxout. See Fig. 7 and Fig. 8 for details.

8. Optimization

The second key reason that maxout performs well is that it improves the bagging style training phase of dropout. Note that the arguments in section 7 motivating the use of maxout also apply equally to rectified linear units (Salinas & Abbott, 1996; Hahnloser, 1998; Glorot et al., 2011). The only difference between maxout and max pooling over a set of rectified linear units is that maxout does not include a 0 in the max. Superficially, this seems to be a small difference, but we find that including this constant 0 is very harmful to optimization in the context of dropout. For instance, on MNIST our best validation set error with an MLP is 1.04%. If we include a 0 in the max, this rises to over 1.2%. We argue that, when trained with dropout, maxout is easier to optimize than rectified linear units with cross-channel pooling.

8.1. Optimization experiments

To verify that maxout yields better optimization performance than max pooled rectified linear units when training with dropout, we carried out two experiments. First, we stressed the optimization capabilities of the training algorithm by training a small (two hidden convolutional layers with $k = 2$ and sixteen kernels) model on the large (600,000 example) SVHN dataset. When training with rectifier units the training error gets stuck at 7.3%. If we train instead with maxout units, we obtain 5.1% training error. As another optimization stress test, we tried training very deep and narrow models on MNIST, and found that maxout copes better with increasing depth than pooled rectifiers. See Fig. 9 for details.

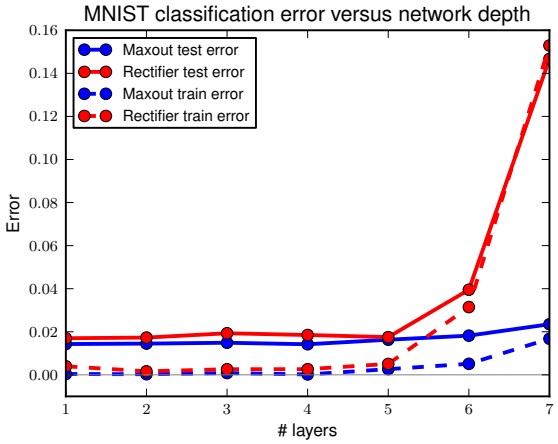

Figure 9. We trained a series of models with increasing depth on MNIST. Each layer contains only 80 units (k=5) to make fitting the training set difficult. Maxout optimization degrades gracefully with depth but pooled rectifier units worsen noticeably at 6 layers and dramatically at 7.

8.2. Saturation

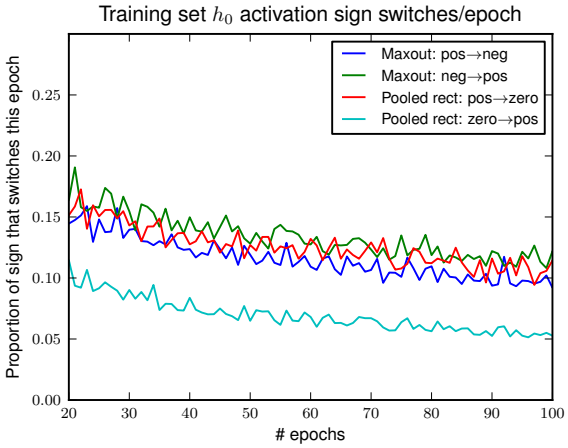

Figure 10. During dropout training, rectifier units transition from positive to 0 activation more frequently than they make the opposite transition, resulting a preponderence of 0 activations. Maxout units freely move between positive and negative signs at roughly equal rates.

Optimization proceeds very differently when using dropout than when using ordinary stochastic gradient descent. SGD usually works best with a small learning rate that results in a smoothly decreasing objective function, while dropout works best with a large learning rate, resulting in a constantly fluctuating objective function. Dropout rapidly explores many different directions and rejects the ones that worsen performance, while SGD moves slowly and steadily in the most promising direction. We find empirically that these different operating regimes result in different outcomes for rectifier units. When training with SGD, we find that the rectifier units saturate at 0 less than 5% of the time. When training with dropout, we initialize the units to sature rarely but training gradually increases their saturation rate to 60%. Because the 0 in the $\max(0, z)$ activation function is a constant, this blocks the gradient from flowing through the unit. In the absence of gradient through the unit, it is difficult for training to change this unit to become active again. *Maxout does not suffer from this problem because gradient always flows through every maxout unit*–even when a maxout unit is 0, this 0 is a function of the parameters and may be adjusted Units that take on negative activations may be steered to become positive again later. Fig. 10 illustrates how active rectifier units become inactive at a greater rate than inactive units become active when training with dropout, but maxout units, which are always active, transition between positive and negative activations at about equal rates in each direction. We hypothesize that the high proportion of zeros and the difficulty of escaping them impairs the optimization performance of rectifiers relative to maxout.

To test this hypothesis, we trained two MLPs on MNIST, both with two hidden layers and 1200 filters per layer pooled in groups of 5. When we include a constant 0 in the max pooling, the resulting trained model fails to make use of 17.6% of the filters in the second layer and 39.2% of the filters in the second layer. A small minority of the filters usually took on the maximal value in the pool, and the rest of the time the maximal value was a constant 0. Maxout, on the other hand, used all but 2 of the 2400 filters in the network. Each filter in each maxout unit in the network was maximal for some training example. All filters had been utilised and tuned.

8.3. Lower layer gradients and bagging

To behave differently from SGD, dropout requires the gradient to change noticeably as the choice of which units to drop changes. If the gradient is approximately constant with respect to the dropout mask, then dropout simplifies to SGD training. We tested the hypothesis that rectifier networks suffer from diminished gradient flow to the lower layers of the network by monitoring the variance with respect to dropout masks for fixed data during training of two different MLPs on MNIST. The variance of the gradient on the output weights was 1.4 times larger for maxout on an average training step, while the variance on the gradient of the first layer weights was 3.4 times larger for maxout than for rectifiers. Combined with our previous result showing that maxout allows training deeper networks, this greater variance suggests that maxout better propagates varying information downward to the lower layers and helps dropout training to better resemble bagging for the lower-layer parameters. Rectifier networks, with more of their gradient lost to saturation, presumably cause dropout training to resemble regular SGD toward the bottom of the network.

9. Conclusion

We have proposed a new activation function called maxout that is particularly well suited for training with dropout, and for which we have proven a universal approximation theorem. We have shown empirical evidence that dropout attains a good approximation to model averaging in deep models. We have shown that maxout exploits this model averaging behavior because the approximation is more accurate for maxout units than for tanh units. We have demonstrated that optimization behaves very differently in the context of dropout than in the pure SGD case. By designing the maxout gradient to avoid pitfalls such as failing to use many of a model's filters, we are able to train deeper networks than is possible using rectifier units. We have also shown that maxout propagates variations in the gradient due to different choices of dropout masks to the lowest layers of a network, ensuring that every parameter in the model can enjoy the full benefit of dropout and more faithfully emulate bagging training. The state of the art performance of our approach on five different benchmark tasks motivates the design of further models that are explicitly intended to perform well when combined with inexpensive approximations to model averaging.

Acknowledgements

The authors would like to thank the developers of Theano (Bergstra et al., 2010; Bastien et al., 2012), in particular Frédéric Bastien and Pascal Lamblin for their assistance with infrastructure development and performance optimization. We would also like to thank Yann Dauphin for helpful discussions.

References

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Breiman, Leo. Bagging predictors. *Machine Learning*, 24 (2):123–140, 1994.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 22:1–14, 2010.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011.

Goodfellow, Ian J., Courville, Aaron, and Bengio, Yoshua. Joint training of deep Boltzmann machines for classification. In *International Conference on Learning Representations: Workshops Track*, 2013.

Hahnloser, Richard H. R. On the piecewise analysis of networks of linear threshold neurons. *Neural Networks*, 11(4):691–697, 1998.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580, 2012.

Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, Marc'Aurelio, and LeCun, Yann. What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153. IEEE, 2009.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*. 2012.

LeCun, Yann, Bottou, Leon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Malinowski, Mateusz and Fritz, Mario. Learnable pooling regions for image classification. In *International Conference on Learning Representations: Workshop track*, 2013.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. Deep Learning and Unsupervised Feature Learning Workshop, NIPS, 2011.

Rifai, Salah, Dauphin, Yann, Vincent, Pascal, Bengio, Yoshua, and Muller, Xavier. The manifold tangent classifier. In *NIPS'2011*, 2011. Student paper award.

Salakhutdinov, R. and Hinton, G.E. Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 8, 2009.

Salinas, E. and Abbott, L. F. A model of multiplicative neural responses in parietal cortex. *Proc Natl Acad Sci U S A*, 93(21):11956–11961, October 1996.

Sermanet, Pierre, Chintala, Soumith, and LeCun, Yann. Convolutional neural networks applied to house numbers digit classification. *CoRR*, abs/1204.3968, 2012a.

Sermanet, Pierre, Chintala, Soumith, and LeCun, Yann. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*, 2012b.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan Prescott. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

Srebro, Nathan and Shraibman, Adi. Rank, trace-norm and max-norm. In *Proceedings of the 18th Annual Conference on Learning Theory*, pp. 545–560. Springer-Verlag, 2005.

Srivastava, Nitish. Improving neural networks with dropout. Master's thesis, U. Toronto, 2013.

Wang, Shuning. General constructive representations for continuous piecewise-linear functions. *IEEE Trans. Circuits Systems*, 51(9):1889–1896, 2004.

Yu, Dong and Deng, Li. Deep convex net: A scalable architecture for speech pattern classification. In *INTERSPEECH*, pp. 2285–2288, 2011.

Zeiler, Matthew D. and Fergus, Rob. Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations*, 2013.