Deep Neural Networks for Wind Energy Prediction. In: Rojas, I., Joya, G., Catala, A. (eds) Advances in Computational Intelligence. IWANN 2015. Lecture Notes in Computer Science 9094 (2015): 430-443

DOI: https://doi.org/10.1007/978-3-319-19258-1_36

Deep Neural Networks for Wind Energy Prediction

David Díaz(✉), Alberto Torres, and José Ramón Dorronsoro

Departamento de Ingeniería Informática e Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, Madrid, Spain
david.diazv@estudiante.uam.es, {alberto.torres,jose.dorronsoro}@uam.es

Abstract. In this work we will apply some of the Deep Learning models that are currently obtaining state of the art results in several machine learning problems to the prediction of wind energy production. In particular, we will consider both deep, fully connected multilayer perceptrons with appropriate weight initialization, and also convolutional neural networks that can take advantage of the spatial and feature structure of the numerical weather prediction patterns. We will also explore the effects of regularization techniques such as dropout or weight decay and consider how to select the final predictive deep models after analyzing their training evolution.

1 Introduction

Having had a big first impact around 1990, Multilayer Perceptrons (MLPs) started a mild decline after the second half of that decade. A particularly puzzling issue was the difficulty to build efficient MLPs with three or more layers, in spite of the fact that the backpropagation computation of the gradient of the MLP error function could be carried out in a rather straightforward fashion. The reason behind this was the vanishing gradient phenomenon [8] which in turn was in part a consequence of the inadequacy of weight initialization.

However, this changed radically with the seminal paper by G. Hinton and R. Salakhutdinov [12] that showed how an unsupervised, stacking scheme based on Boltzmann machines could yield a good initialization (or pretraining) of the weights of a many-layered MLP, that could be then efficiently fine-tuned by backpropagation. Shortly afterwards, Y. Bengio and his coworkers proposed a similar and somewhat simpler pretraining using stacked autoencoders [4]. This opened the way to the enormous attention that deep MLPs (i.e., MLPs with three or more layers) or, in general, deep learning, have received in the past years.

This attention has in turn resulted in a great simplification of the initial schemes of Hinton and Bengio and has brought many new procedures and ideas to the MLP field, such as new initializations, or the replacement of some of the initial MLP recipes, such as sigmoid activations or weight decay regularization, by new proposals like rectified linear unit (ReLU) activations [9] or dropout

regularization [18]. Moreover, the very large datasets and deep MLP parameters often rules out batch learning. This has resulted in a large emphasis on online learning, usually over minibatches of randomly selected patterns, with much work being devoted to the choice of learning rates (or how to avoid them) or momentum methods such as Nesterov's acceleration. This raises the issue of when to stop training, something rather straightforward in the batch training of classical MLPs if an adequate regularization and an efficient optimizer were used. Furthermore, once the previous ingredients are in place, the need to specialized (and costly) pretraining is less accute and several initialization methods have been proposed that result in the training of effective deep models. A good example of such a global approach is [19]. Another key ingredient in the successful applications of deep learning is the use of convolutional layers, that concatenate a purely convolutional sublayer that processes inputs using localized window filters, and a pooling sublayer that aggregates the outputs of the previous sublayer. Starting with the work of Y. LeCun in the late 1990's, this processing is particularly natural when inputs have a spatial structure, as it is the case with images, and it has led convolutional deep nets to achieve state-of-the-art results in problems such as MNIST [7] or ImageNet [14].

All these advances have made possible the effective training of very large deep networks with hundreds of thousands of weights which, in turn, makes imperative the use of software that can take advantage of high performance hardware endowed with parallelization (i.e., multicore machines) and vectorization (i.e., GPU units). Besides, the fast pace of change in the field and the fact that there is still not an accepted multipurpose architecture makes it quite difficult to work with self developed code; instead, it is well advised to rely on publicly available libraries and environments such as the Caffe [13] deep learning framework or the Pylearn2–Theano libraries [5] [3] [10] that we use here.

In any case, it seems that the bulk of deep learning research concentrates on computer vision, speech recognition and natural language processing problems. This is partially natural in view of the broad similarity between deep learning architectures and the processing hierarchies in the visual cortex [15] (although deep learning algorithms are quite different from the Spike-Timing-Dependent Plasticity learning rule most accepted in neurobiology). As such, deep learning algorithms are increasingly seen as representation learning procedures that yield at each layer increasingly more abstract representations in such a way that features in the higher layers capture possibly more powerful data features.

However, the successful exploitation of such a processing may also take place in simpler regression problems that, nevertheless, have input patterns with a spatial structure. The goal of this work is the prediction of wind energy production. Spain is among the world leaders in wind energy with a very high penetration that in some special days and hours can meet a very high percentage of Spain's electricity demand. Obviously, this high penetration makes it very important to provide accurate prediction of wind energy, with standard MLPs (usually at the farm level) and Support Vector Regression (SVR) (for large scale prediction) being the models of choice. The inputs for such models are the forecasts provided

by numerical weather prediction (NWP) systems such as the ECMWF [1] or the GFS [2]. These predictions are forecasts of several weather variables given at the points of a rectangular grid that covers the areas under study and that reflects some underlying orographic model. One may thus view an area wide NWP forecasts as a set of feature maps (the individual weather variables) having a spatial structure (that of the underlying geography) in much the same way that the RGB channels of an image correspond to feature maps with a two dimensional structure.

Under the previous scheme, the consideration of convolutional networks to derive wind energy forecast arises as a natural option and they will be one of the models considered in this paper. Our main purpose is to develop a methodology to build models that can provide accurate predictions from the original data with as little pre-processing and expert knowledge as possible. Given the very wide range of proposals in the literature, this implies we must make beforehand concrete choices of network initialization, online training procedure, activation function and regularization scheme. Of course, on top of this, a more or less general network architecture also has to be selected.

We will develop the choices we make in the next sections. Besides standard "small" MLPs and SVR models that we use as reference benchmarks, we will consider deep MLPs with a standard multilayer structure, general deep convolutional networks (CNNs) and also an adaptation of the well-known LeNet [16], one of the most successful architectures for character recognition. In all those deep nets we will use Glorot–Bengio weight initialization [8], ReLUs as activation functions [9], dropout regularization [18] complemented with standard weight decay in the final fully-connected layers, random mini-batch gradient descent over batches of moderate size and conjugate gradient as the training algorithm. This enables us to work with a fixed, fairly general learning rate, that is no longer a parameter to explore. Summing things up, our main contributions are:

- We review some of the latest proposals in DNNs and propose general guidelines to apply deep MLPs in regression problems.
- We thoroughly explore the application of the two main paradigms in DNNs to the problem of local and large scale wind energy prediction.
- We introduce a variant of the well known LeNet convolutional neural network adapted to wind energy prediction and show it to be very competitive with other DNN architectures or state of the art methods such as Support Vector Regression.

As mentioned before, we will use Pylearn2 [10]–Theano [3] [5] platform as it includes a wide variety of already tested neural networks and allows us to explore several of the latest and most effective proposals for deep network training. An important advantage of having Theano as the underlying numerical library is that we can exploit its capabilities for code execution on GPUs, something crucial given the network sizes and input dimensions we work with. We run our experiments on a machine equipped with a NVidia Tesla K40 GPU which makes possible reasonable execution times and, hence, the capability of exploring a fairly large number of deep model configurations.

The rest of the paper is organized as follows. In Section 2 we review our choices for deep network configuration and discuss some of its details. Section 3 contains a succinct description of the framework for wind energy prediction over NWP inputs, a description of our experimental setup and the prediction results for both the Sotavento wind farm and the entire wind energy prediction over peninsular Spain that is overseen by Red Eléctrica de España (REE). Finally, in Section 4 we briefly discuss our results and offer pointers to further work.

2 Deep Neural Networks

We briefly review here some of the key issues when configuring and training Deep Neural Networks.

2.1 Initialization

There have been several heuristic proposals for weight initialization in "classical" MLPs. For instance, a common choice is to take them from a uniform distribution $U\left[-\frac{1}{\sqrt{M}}, \frac{1}{\sqrt{M}}\right]$, with M the fan-in of the neuron, i.e., the number of weights feeding into it. However, it was found experimentally in [8] that in a deep MLP initialized in such a way, back-propagated gradients were progressively smaller when moving from the output layer towards the input layer and, in addition, their variances also decrease. In other words, backpropagating such an initialization may result in vanishing gradients in the first layers following the input and, thus, in a network which is insensitive to its inputs and unable to "learn" them.

The more detailed analysis in [17], also oriented to "classical" MLPs and where properly normalized hyperbolic tangents were used, pursued a goal of keeping the (linear) activations and (non linear) outputs of a neuron in the $[-1,\ 1]$ active range of the (normalized) hyperbolic tangent. Assuming inputs normalized to zero mean and unit variance component-wise, it is suggested in [17] to use an uniform distribution $U\left[-\frac{\sqrt{3}}{\sqrt{M}}, \frac{\sqrt{3}}{\sqrt{M}}\right]$. This analysis was extended in [8], where, assuming again the neuron outputs z_i of the i-th layer also to be in the $[-1,\ 1]$ active range, initialization should ensure first that $Var(z_i) \simeq Var(z_i')$ across the successive layers and also that $Var\left(\frac{\partial J}{\partial z_i}\right) \simeq Var\left(\frac{\partial J}{\partial z_i}\right)$, where J denotes the MLP cost function. This translates into the following equations for the initial weights W_i

$$M_i Var(W_i) = 1;\ M_{i+1} Var(W_i) = 1$$

where M_i and M_{i+1} are the fan-in and fan-out of the units in the i-th layer. An approximation to both is to take $Var(W_i) = \frac{2}{M_i + M_{i+1}}$, i.e., to initialize the W_i using an uniform distribution $U\left[-\frac{\sqrt{6}}{\sqrt{M_i+M_{i+1}}}, \frac{\sqrt{6}}{\sqrt{M_i+M_{i+1}}}\right]$. Note that when $M_i = M_{i+1}$, we get back the initialization proposed in [17].

We will use the initialization in [8] but working with Rectified Linear Unit (ReLU) activations, discussed next, instead of the hyperbolic tangent ones. While the rationale in [8] may not apply, the recent analysis in [11] of weight initialization for ReLU activation suggests to dilate the Glorot–Bengio uniform intervals by a factor of 1.5, and, in fact, we have observed that this usually yields better results.

2.2 Activation Function

As mentioned, we have used ReLUs for all hidden layer activations and linear units in the output layer. The ReLU transfer function is $r(x) = \max(0, x)$, that is, their response to the opposite of a positive excitatory input is just 0; in particular, ReLUs do not have a sign antisymmetry, as is the case with the hyperbolic tangent. On the other hand, ReLUs share some similarities with the functions relating neuronal input currents and firing rates that appear in the leaky integrate and fire models used in biological neuron models [9]. Besides, ReLUs induce sparsity in the representations of the successive layers; for instance, right after the uniform weight initialization, the outputs of about half the network neurons should be zero, as they would correspond to negative (inhibitory) inputs. This may partially explain the fact that ReLUs seem to be less affected than other activations by poor initializations. In any case, this point deserves further study.

2.3 Regularization

It is obvious that the extremely large number of weights in a deep MLP makes regularization mandatory to avoid overfitting. The standard regularization technique in classical MLPs is weight decay applied across all the layers; i.e., the square norm weight penalty considered for all layer weights is added to the MLP cost function. It has an obvious place in a last layer with linear outputs, as it performs ridge regression on the features induced in that last layer by the deep processing of the inputs.

However, for the other layers we will use dropout [18], that we briefly describe next. If a_i^l denotes the i-th activation of the l-layer and z_i^l the corresponding output, the standard feedforward processing would yield $z_i^l = f(a_i^l) = f(w_i^l z^{l-1} + b_i^l)$, where f is the activation function. However, with dropout, a 0–1 vector r^l is first generated applying a Bernoulli distribution componentwise. The feedforward process becomes

$$z_i^l = f(a_i^l) = f\left(w_i^l (z^{l-1} \odot r^l) + b_i^l\right) ,$$

where $\odot$ denotes the componentwise product. Each element in r^l has a probability p of being 1, so dropout can be seen as sub-sampling a larger network at each layer. The output errors are backpropagated as in standard MLPs for gradient computations and the final optimal weights w^* are downscaled as $w_f^* = pw^*$ to yield the final weights used for testing.

Dropout clearly induces a regularization of the network's weights. Moreover, it is reminiscent to the well known bagging technique for ensembles that repeatedly subsamples data to build specific models and then takes the average. However, in dropout all the "models" (i.e., the particular feedforward Bernoulli realizations) share weights and they are "trained" in a single step. Although we will not use it, in [18] it is also suggested that network performance improves when dropout is combined with a bound on the L_2 norm of the weights, i.e., when they are constrained as $\|w\|_2 \leq c$, with c a second tunable parameter on top of the Bernoulli probability p.

2.4 Convolutional Layers

Standard deep MLP architectures tend to favor layers with a high number of hidden units. This also leads to a high number of weights, $M \times M'$ if we fully connect an M unit layer with an M' one, a number that can become rather large if, for instance, inputs have a two dimensional structure, as it is the case with images. Convolutional layers arise in part as a way to avoid this by limiting the fan–in of a hidden unit to come from a localized subset of units in the previous layer. Of course, how to define such a restricted fan–in is, in general, problem–dependent, but when data have an intrinsic spatial structure a natural approach to localize the connections is to work over small patches.

More precisely, assume inputs or layer outputs to be one channel structured as an $M_1 \times M_2$ matrix, and consider in them $K \times K$ submatrix patches. They could be either disjoint or partially overlapping; we can parameterize this considering a stride value S that gives the displacement applied when we move horizontally and vertically from one patch to the next one. Assuming for simplicity a $S = 1$ stride, there are such $(M_1 - K + 1) \times (M_2 - K + 1)$ (overlapping) patches x_j. A first transform is to derive a patch feature $p_j = f(w * x_j + b)$ where f is the activation function, $*$ denotes the convolution operator between the $K \times K$ filter w and the patch x_j and b is the bias of the filter. This transforms an $M_1 \times M_2$ input X into an $(M_1 - K + 1) \times (M_2 - K + 1)$ convolutional output X' and usually a number L of filter pairs (w_l, b_l) (or of feature maps) have to be learned. Thus, the number of weights in a convolutional sub–layer is a rather modest $L \times K^2$ but, on the other hand, the output dimension would be $L \times (M_1 - K + 1) \times (M_2 - K + 1)$, which for $L > K$ might greatly increase the number of hidden units in the next layer. To curb this (and avoid a possible overfitting), a second pooling (or subsampling) sub–layer is applied in which an operation such as averaging or computing the max is applied on $P \times P$ patches of X' to derive the final output X_C of the convolution–pooling combined process; X_C has a $L \times (M_1 - K - P + 2) \times (M_2 - K - P + 2)$ dimension.

This combined convolution–pooling process is called a convolutional layer; it allows for a localized processing of the layer's input using a moderate number of weights (note that there are no weights in the pooling sub–layer) while arriving at a number of units in the next layer similar to that of the previous one. Of course, we stress again that, to be effective, a convolutional layer must act on inputs that have a spatial structure (such as images) and are naturally distributed in

feature channels (such as the RGB decomposition). This is also happens in our case, where weather prediction has an obvious spatial structure in which different meteorological features (pressure, temperature, wind components, etc.) can be seen as corresponding to different channels.

3 Experiments

In this section we will apply DNNs to the problem of predicting wind energy production, first on the Sotavento wind farm and then over peninsular Spain.

3.1 NWP and Production Data

We will work with the following eight meteorological variable forecasts given by the European Centre for Medium-Range Weather Forecasts (ECMWF) system for Numerical Weather Prediction (NWP):

- P, the pressure at surface level.
- T, the temperature at 2m.
- V_x, the x wind component at surface level.
- V_y, the y wind component at surface level.
- V, the wind norm at surface level.
- V_x^{100}, the x wind component at 100m.
- V_y^{100}, the y wind component at 100m.
- V^{100}, the wind norm at 100m.

In the Sotavento case they are taken on 15×9 rectangular grid centered on the Sotavento site (43.34°N, 7.86°W); input dimension in this case is thus $15 \times 9 \times 8 = 1,080$. For peninsular Spain we consider a 57×35 rectangular grid that covers entirely the Iberian peninsula; input dimension is now a very large $57 \times 35 \times 8 = 15,960$.

Wind energy data for Sotavento are publicly available; those for peninsular Spain were kindly provided by Red El?ctrica de Espa?a (REE). In both cases we normalize them to the $[0, 1]$ interval by dividing actual wind energy production by the maximum possible value in each case. We will work with data for the years 2011, 2012 and 2013, that we will use as training, validation and test subsets respectively. Since NWP forecasts are given every three hours, each subset will approximately have $(24/3) * 365 = 2,920$ patterns.

3.2 Deep Models

We will consider deep networks with either all their layers being fully connected, which we call deep MLPs, or with a number of initial convolutional layers followed by fully connected ones; we call these models deep convolutional neural networks, or deep CNNs. As reference models we will work with "standard" one hidden layer MLPs and also with Support Vector Regression (SVR) models,

Algorithm 1 Hyper-parameter search

procedure HYPER-PARAMETER SEARCH(n, m) ▷ $n \times m$ iterations
  randomly initialize an hyper–parameter vector p
  $p^* = p$ ▷ p^*: optimal hyper–parameter vector
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, m$ **do**
      $k \leftarrow$ random value in $\{1, \ldots, m\}$
      $p_k \leftarrow$ random value in $\{v_1^k, \ldots, v_{N_k}^k\}$
      evaluate the p–parameterized model and update p^* if needed
    end for
  end for
  return p^*
end procedure

among the most powerful modelling methods in wind energy prediction. The number of possible architectures and the many choices available for them would result in an unmanageable number of model hyperparameters to explore when looking for the best ones. To limit this, we have first fixed some of them to reasonable values that give good results in a first coarse model exploration.

A first such choice is that of the deep architectures to be considered. For deep MLPs we will consider two hidden layers with the same number of units. Our first choice for deep CNNs, which we call standard deep CNN or sdCNN, will have an initial convolutional layer followed by two fully connected layers again with the same number of units. Our second CNN choice, which we call LeNet CNN or lnCNN, will be an adaptation of the well known LeNet–5 architecture [16], that was specifically designed for the MNIST character recognition problem.

We will use the non–symmetric ReLUs at the hidden layers and, as discussed before, for network initialization we will apply the Glorot–Bengio heuristic proposed in [8] of using a 0–symmetric uniform distribution with a width adjusted to the layers' fan-in, scaling then up these initial weights by a factor of 1.5.

The training algorithm we are going to use for all the experiments is conjugate gradient descent (CGD) over random mini–batches. In other words, over each new mini–batch we apply CGD starting at the weights derived over the previous mini–batch; their size clearly affects the performance of the network and we have used sizes of either 200 or 250, i.e., about 6% and 9% of the training sample size. Our error measure is the mean absolute error (MAE)

$$MAE = \frac{1}{N} \sum_{n=1}^{N} |D(x_n; P) - y_n| \,,$$

where $D(x; P)$ denotes the value on pattern x of the current deep network D built using the hyperparameter set P. We use the MAE instead of the more often used squared error as it is the measure of choice in renewable energy, for it represents energy deviation and, thus, the energy to be shed or obtained from other generation sources to compensate errors in wind energy estimates.

As we shall see in the next subsection, the overall MAE evolution during training is decreasing but it often presents spikes due to the use of mini–batches, and this carries on to validation MAE values. In addition, validation MAE seems to stabilize even while training MAE keeps decreasing. Because of this, our model selection strategy is to train a deep NN while there is at least a 1% drop in MAE in the last 100 epochs, with a a maximum of $1,000$ epochs (i.e., goes throught the entire training set). For convolutional networks we will consider each weather variable to define an input feature map; there are thus 8 such features. The above choices leave us with the following hyperparameters to be selected:

- For deep MLPs (which we denote by MLP2) we have to decide on the number (one or two) of hidden layers, the number of hidden units per layer, the weight decay and dropout coefficients, and mini–batch size.
- For the standard deep CNN (which we denote by CNN) we add to the previous deep MLP parameters the convolutional filter and pooling sizes, and their strides.
- For the LeNet CNN (which we denote by LeNet) we also have to decide on the deep MLP parameters but we simplify the other choices by selecting filter and pooling sizes and strides as adequately scaled versions to our problem of the choices made for LeNet–5.

In any case, it is clear that even after the previous simplifications, the number of hyperparameters is too large for an exhaustive grid search. To alleviate this we have used a greedy approach in which we fix first the number of fully connected hidden layers as 2 and then apply Algorithm 1, in which models are evaluated in terms of the MAE over the validation subset. The algorithm performs $n = 50$ external iterations on each of which a concrete hyper–parameter vector p is evaluated. The hyper–parameters considered are the number of hidden units in fully connected layers, the weight decay multipliers used in them, the dropout fraction and the minibatch size. On each external iteration m random choices are made of hyper–parameter indices k and for each a possible updating value p_k is randomly selected from the list $\{v_1^k, \ldots, v_{N_k}^k\}$ of values of the k–th hyper–parameter to be explored. Both random selections are uniform. Actual tested values were

- Hidden unit numbers: 50, 100, 150, 200, 250, 300, 350, 400.
- Weight decay multipliers: 0.1, 0.2, 0.3, 0.4, 0.5.
- Dropout fractions: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8.
- Minibatch size: 50, 100, 150, 200, 250, 300.

For the deep CNNs we fixed the stride to 1 and adjusted filter and pooling sizes by a limited heuristic search; notice that these sizes imply at least four more parameters and a fully random search over the entire parameter set is nearly impossible. The same is true for the number of convolutional feature maps. The just described hyperparameter search results in the following deep NN definitions:

Table 1. Mean Absolute Errors for the Sotavento and REE problems

	MAE Sotavento			MAE REE		
	Test	Validation	Train	Test	Validation	Train
SVR	7.80	6.73	5.62	3.13	3.30	1.01
LeNet	7.63	6.25	5.82	3.13	3.01	2.48
CNN	7.76	6.26	5.39	3.31	3.05	1.96
MLP2	7.76	6.33	5.86	3.37	2.96	1.97
MLP1	8.25	6.41	5.51	3.70	3.10	1.81

- Deep MLPs (MLP2) for Sotavento will have two hidden layers of 250 units, a weight decay coefficient of 0.3 and dropout coefficient of 0.7; mini–batch size is 200. The REE ones will have the same weight decay and dropout coefficients, two hidden layers of 300 units and mini–batch size is 250.
- Standard deep CNNs (CNN) for Sotavento will have a first convolutional layer with 2×6 filters and max pooling is performed over 2×2 patches. This layer is followed by two fully connected layers of 200 units, no weight decay, dropout coefficient of 0.7 and mini–batch size 250. The REE CNN has the same structure; the first layer has now 3×3 filters, max pooling is done over 3×5 patches. This is followed by two fully connected layers of 400 units, weight decay and dropout coefficients are 0.3 and 0.7 respectively and mini–batch size is 200. We used 16 convolutional feature maps for Sotavento and 8 for REE.
- The adapted LeNet–5 (LeNet) network for Sotavento has a first convolutional layer with 2×2 filters and max pooling, and a second one with 4×2 filters and 2×2 max pooling. They are followed by two fully connected 200 unit layers, no weight decay, dropout coefficient of 0.7 and mini–batch size 250. For REE, the LeNet network has a first convolutional layer with 6×8 filters, 2×2 max pooling, and a second one with 6×6 filters and 2×2 max pooling. They are followed by two fully connected 200 unit layers, weight decay and dropout coefficients of 0.3 and 0.7 respectively, and mini–batch size is 200. For both problems we used 16 convolutional feature maps in the first layer and 32 in the second.

3.3 Results

For comparison purposes, we also consider a Gaussian SVR model and a "standard" one–hidden layer, 10–unit MLP. We have used the very well known LIBSVM library [6] and the SVR hyperparameters C, γ and ϵ have been established by a grid search; their optimal values were C=128.0, $\gamma = 3.0518 \times 10^{-5}$ and $\epsilon = 0.0625$ for Sotavento and C=128.0, $\gamma = 12.2078 \times 10^{-5}$ and $\epsilon = 0.01$ for REE. For the standard MLPs we used again Pylearn2–Theano and the optimal parameters were 0.001 weight decay coefficient and 200 mini–batch size in Sotavento and 0.1 weight decay coefficient and 250 mini–batch size for REE.

Table 2. Training complexity parameters and times in seconds for the Sotavento (top) and REE (bottom) deep models

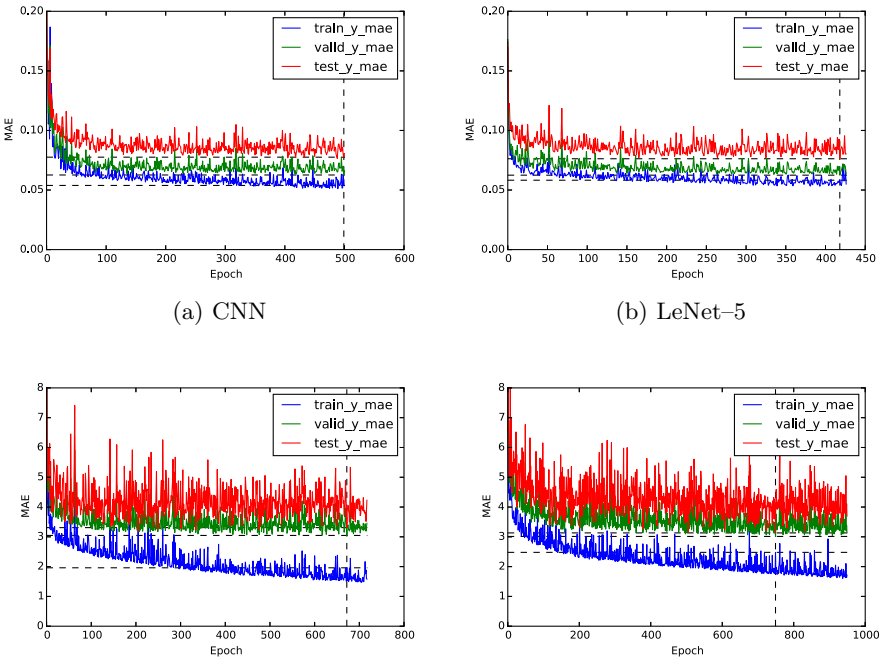
Model	#Params.	#Iters.	Time	Time/Iter.
LeNet	140808	426	1175	2.76
CNN	105736	500	705	1.41
MLP2	332750	259	276	1.07
LeNet	224776	949	19494	20.54
CNN	548176	717	6880	9.60
MLP2	4878300	258	1208	4.68

Table 1 gives training, validation and test errors for the optimal models and the two problems. As it can be seen, the SVR and LeNet–5 models have a similar in performance in the REE problem, followed by the other two deep models; the standard MLP is in a distant last place. However, in Sotavento the LeNet–5 model is clearly the best model while the SVR and the two deep models essentially tie for second place; again, the standard MLP comes in last place. We point out that although we follow a straightforward train–validation–test scheme for model evalution, a more accurate comparison should be made using an appropriate statistical test such as the well known Wilcoxon Rank Sum test, that takes into account not only MAE values but also standard deviations. This requires larger training periods and will be considered in further work.

Figure 1 shows the evolution of the train, test and validation errors for the optimal CNN and LeNet–5 networks for Sotavento (top) and REE (bottom). The large error variations are caused by mini–batch training; while at first sight validation and test error evolution appears smoother for Sotavento, this is partially due to a scale effect (about twice as large for Sotavento than for REE). For Sotavento the smallest errors seem to have essentially reached stable values; this is also the case for the validation and test errors in REE although training error would keep on decreasing, probably because the higher dimensionality of this problem. In both cases the vertical dotted line indicates the epoch with a lowest validation error and the horizontal dotted lines indicate the training, validation and test errors in that epoch. These are the values reported in Table 1.

Finally, in Table 2 we give the give the complexity parameters and training times in seconds for the deep models used in the Sotavento (top) and REE (bottom) problems. As it can be seen, all models are rather large, and more so those used for REE (remember that input dimensions are respectively 1,080 and 15,960). Besides, while the convolutional networks have less parameters than MLP2, their feedforward passes are much costlier due to the convolution operations and the same is true for the backpropagation of gradients. We observe that the smaller number of weights in LeNet for REE is due to the larger filters used.

It follows that deep training is rather costly and must take advantage of all possible hardware–based improvements available. In our case experiments



(a) CNN (b) LeNet–5

(c) CNN (d) LeNet–5

Fig. 1. Training, validation and test evolution of the optimal CNN and LeNet–5 networks for Sotavento (top) and REE (bottom)

have been run on a machine equipped with a NVidia Tesla K40 GPU and the Pylearn2–Theano framework. Working with Pylearn2 eases somewhat the development process, since most of it is written in Python.

There are other platform alternatives with the already mentioned Caffe being an interesting one, as its core is written in C++ and CUDA, which should result in a performance improvement. Another important improvement comes from the NVidia cuDNN library, that inter–operates perfectly with Pylearn2–Theano (more than doubling the performance of the previous version) and Caffe.

4 Conclusions

While undeniably very powerful, the optimal architectures and best hyper–parameters of deep neural networks are also quite hard to set up and select. However, when properly tuned, they can often produce better results than other classical models, as we have demonstrated here on two wind energy problems. The use of weather variables gives to both problems a bi–dimensional input structure; moreover, these variables can be naturally seen as input channels. This may suggest a reason why the best deep results were obtained using convolutional

layers. Deep network training is also very computationally demanding but, on the other hand, lends itself extremely well to the use of GPUs and the large speed-ups that they allow.

In any case, the work presented here has to be considered as a first step. A first line of further work is to consider other convolutional architectures, specially of the AlexNet type ([14]). Another natural option is to try to reduce variance by combining several deep models (notice that they have naturally a low bias). The usual choice in standard MLPs is to repeat training from different random initializations but given the high validation variability during training, a simpler, less costly possibility is to select a certain number M of the models with smallest validation that were obtained in a single training run as the ones followed here.

Furthermore, the tremendous activity in deep learning is producing a large number of proposals for network initialization and architectures as well as model training and regularization. We are also pursuing some of these options.

Acknowledgments. With partial support from Spain's grants TIN2013-42351-P (MINECO) and S2013/ICE-2845 CASI-CAM-CM (Comunidad de Madrid), and the UAM–ADIC Chair for Data Science and Machine Learning.

The second author is also kindly supported by the FPU-MEC grant AP-2012-5163. The authors gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM and thank Red Eléctrica de España for kindly supplying wind energy production data.

References

1. European center for medium-range weather forecasts. http://www.ecmwf.int/
2. Global forecast system. http://www.emc.ncep.noaa.gov/index.php?branch=gfs
3. 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)
4. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems 19 (NIPS 2006), pp. 153–160 (2007). http://www.iro.umontreal.ca/~lisa/pointeurs/BengioNips2006All.pdf
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), June 2010. Oral Presentation
6. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**(3), 27:1–27:27 (2011). software available at http://www.csie.ntu.edu.tw/cjlin/libsvm
7. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, July 16–22, 2011, pp. 1237–1242 (2011). http://ijcai.org/papers11/Papers/IJCAI11-210.pdf

8. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010), vol. 9, pp. 249–256, May 2010
9. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), April 2011
10. 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). http://arxiv.org/abs/1308.4214
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR abs/1502.01852 (2015). http://arxiv.org/abs/1502.01852
12. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006). http://www.sciencemag.org/content/313/5786/504.abstract
13. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R.B., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. CoRR abs/1408.5093 (2014). http://arxiv.org/abs/1408.5093
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012). http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
15. Kruger, N., Janssen, P., Kalkan, S., Lappe, M., Leonardis, A., Piater, J., Rodriguez-Sanchez, A., Wiskott, L.: Deep hierarchies in the primate visual cortex: What can we learn for computer vision? IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(8), 1847–1871 (2013)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
17. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient BackProp. In: Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 1524, pp. 9–50. Springer, Heidelberg (1998)
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**, 1929–1958 (2014). http://jmlr.org/papers/v15/srivastava14a.html
19. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML 2013), vol. 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings, May 2013. http://jmlr.org/proceedings/papers/v28/sutskever13.pdf