Journal of Machine Learning Research 12 (2011) 2563-2581 Submitted 7/10; Revised 2/11; Published 9/11

Kernel Analysis of Deep Networks

Grégoire Montavon GMONTAVON@CS.TU-BERLIN.DE
Mikio L. Braun MIKIO@CS.TU-BERLIN.DE
Klaus-Robert Müller[*] KLAUS-ROBERT.MUELLER@TU-BERLIN.DE
Machine Learning Group
Technische Unversität Berlin
Franklinstr. 28/29
10587 Berlin, Germany

Editor: Yoshua Bengio

Abstract

When training deep networks it is common knowledge that an efficient and well generalizing representation of the problem is formed. In this paper we aim to elucidate what makes the emerging representation successful. We analyze the layer-wise evolution of the representation in a deep network by building a sequence of deeper and deeper kernels that subsume the mapping performed by more and more layers of the deep network and measuring how these increasingly complex kernels fit the learning problem. We observe that deep networks create increasingly better representations of the learning problem and that the structure of the deep network controls how fast the representation of the task is formed layer after layer.

Keywords: deep networks, kernel principal component analysis, representations

1. Introduction

Finding an appropriate representation of data is a central problem in machine learning. The representation should ideally distill the relevant information about a learning problem in a compact manner, such that it becomes possible to learn the data from a small number of examples.

Deep networks (e.g., Rumelhart et al., 1986; Hinton et al., 2006) have shown promise by automatically extracting representations from raw data. Through their deep multi-layered architecture, simpler and more accurate representations of the learning problem can be built layer after layer. Their depth makes possible the creation of abstractions that are important in order to learn the desired well-generalizing representation. Also, their flexibility offers the possibility to systematically and structurally incorporate prior knowledge, for example, by constraining the connectivity of the deep network (e.g., LeCun, 1989; Lang et al., 1990), by learning multiple tasks at the same time (Caruana, 1997; Collobert and Weston, 2008) or by regularizing the solution with unlabeled samples (Salakhutdinov and Hinton, 2007; Weston et al., 2008). Such prior knowledge can significantly improve the generalization ability of deep networks, leading to state-of-the-art performance on several complex real-world data sets.

While a considerable amount of work has been dedicated to learning efficiently deep architectures (Orr and Müller, 1998; Hinton et al., 2006; Bengio et al., 2006), leading to simple and efficient training algorithms, these learning machines still lack of analytic understanding. Recently,

∗. Also at the Institute for Pure & Applied Mathematics, University of California, Los Angeles, Los Angeles, CA 90095.

a significant amount of research has focused on improving our theoretical understanding of deep networks, in particular, understanding the benefits of unsupervised pretraining (Erhan et al., 2010), understanding what are the main difficulties when training deep networks (Larochelle et al., 2009) and studying the invariance of representations built in deep networks (Goodfellow et al., 2009). However, quantifying how good hidden representations are and measuring how the representation evolves layer after layer are still open questions. Overall, deep networks are thus generally assumed to be powerful and flexible learning machines that are however not well understood theoretically (Bengio, 2009).

In parallel to the development of deep networks, kernel methods (Müller et al., 2001; Schölkopf and Smola, 2002) offer an elegant framework that decouples learning algorithms from data representations. The kernel operator $k(x,x')$—a central concept of the kernel framework—measures the similarity between two points x and x' of the input distribution, yielding an implicit kernel feature map $x \mapsto \phi(x)$ (Schölkopf et al., 1999) that ideally implements all the prior knowledge of the learning problem contained in the kernel operator. This decoupling between learning algorithms and data representations opens the door to a whole world of generic learning machines and data analysis tools such as support vector machines (Cortes and Vapnik, 1995), kernel discriminant analysis (Mika et al., 1999; Baudat and Anouar, 2000; Mika et al., 2003) and kernel principal component analysis (Schölkopf et al., 1998) that can be applied independently of the data set. The kernel framework has also been used as an abstraction tool for modeling complex real systems such as the visual cortex (Smale et al., 2010).

The goal of this paper is to study in the light of the kernel framework how exactly the representation is built in a deep network, in particular, how the representation evolves as we map the input through more and more layers of the deep network. Here, the kernel framework is not used as an effective learning machine, but as an abstraction tool for modeling the deep network. Our analysis takes a trained deep network $f(x) = f_L \circ \cdots \circ f_1(x)$ as input, defines a sequence of "deep kernels"

$$\begin{aligned} k_0(x,x') &= k_{RBF}(x,x'), \\ k_1(x,x') &= k_{RBF}(f_1(x), f_1(x')), \\ &\vdots \\ k_L(x,x') &= k_{RBF}(f_L \circ \cdots \circ f_1(x), f_L \circ \cdots \circ f_1(x')) \end{aligned}$$

that subsume the mapping performed by more and more layers of the deep network and outputs how good the representations yielded by these deeper and deeper kernels are. We quantify for each kernel how good the representation with respect to the learning problem is by measuring how much task-relevant information is contained in the leading principal components of the kernel feature space. This method is based on the theoretical results of Braun (2006) and Braun et al. (2008) which show that eigenvalues and projections to eigenspaces of the kernel matrix have small approximation errors, even for already a small number of samples.

This analysis allows us for the first time to observe and quantify the evolution of the representation in deep networks. We use our analysis to test two hypotheses on deep networks:

> Hypothesis 1: *as the input is propagated through more and more layers of the deep network, simpler and more accurate representations of the learning problem are obtained.*

Indeed, as the input is mapped through more and more layers, abstractions learned by the deep network are likely to change the perception of whether a task is simple or not. For example, in

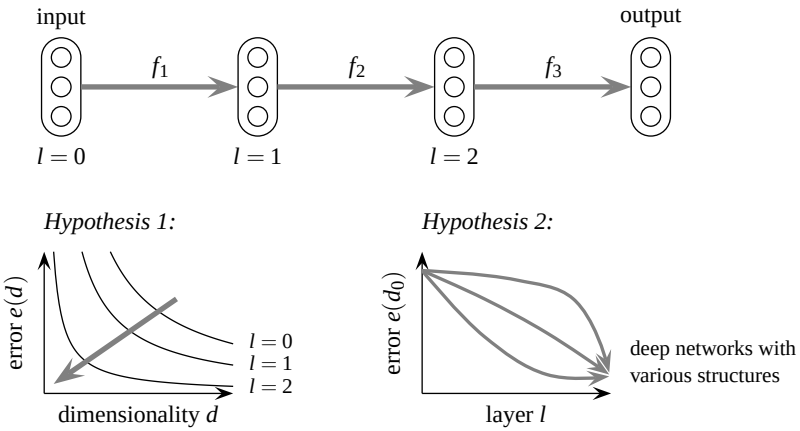


Figure 1: Illustration of our analysis. Curves on the left plot relate the simplicity (dimensionality) and accuracy (error) of the representation of the learning problem at each layer of the deep network. The dimensionality is measured as the number of kernel principal components on which the representation is projected. The thick gray arrows indicate the forward path of the deep network. Hypothesis 1 states that as deeper and deeper kernels are built, simpler and more accurate representations of the learning problem are obtained. Hypothesis 2 states that the structure of the deep network controls the way the solution is formed layer after layer.

the context of image classification, classifying between cat and dog would appear simpler in the last layers of the deep network than in the first layers since irrelevant factors of variation such as occlusion and orientation would be progressively filtered out by the hierarchy of abstractions built in the deep network.

> Hypothesis 2: *the structure of the deep network controls how fast the representation of the task is formed layer after layer.*

It has been empirically corroborated that carefully regularizing the training process by means of specific learning rates, weight penalties, initial weights, shared weights or restricted connectivity can greatly improve the generalization of deep networks (LeCun, 1989; Orr and Müller, 1998; Hinton et al., 2006). We hypothesize that a common aspect of these various regularization techniques is to control the layer-wise evolution of the representation through the deep network. On the other hand, a simple unregularized deep network may make inefficient use of the representational power of deep networks, distributing the discrimination steps across layers in a suboptimal way.

These two hypotheses are illustrated in Figure 1. Testing them are, to our opinion, of significant importance as they might shed light on the nature of deep learning and on the way complex problems are to be solved. This paper completes our conference paper (Montavon et al., 2010) by extending the discussion on the interest of analyzing deep networks within the kernel framework and by extending the empirical study to more data sets and larger deep networks.

1.1 Related Work

The concept of building kernels imitating the structure of deep architectures—or more simply, building "deep kernels"—is not new. Cho and Saul (2009) already expressed deep architectures as kernels in order to solve a convex optimization problem and achieve large margin discrimination in a deep network. This approach differs from our work in the sense that their deep kernel is not used as an analysis tool for trained deep networks but as part of an effective learning machine.

The concept was also developed in Smale et al. (2010) where the authors give a recursive definition of the neural response as hierarchy of simple kernels operating on subparts of the sensory input and in Wibisono et al. (2010) where a principal component analysis is performed on top of these deep kernels in order to measure invariance properties of deep networks. While the last authors focus mostly on the representation of data in static deep architectures made of predefined features, we are considering instead trainable deep architectures.

Although not directly using the kernel framework, Goodfellow et al. (2009) also analyze the layer-wise evolution of the representation in deep networks, showing that deep networks trained in an unsupervised fashion build increasing levels of invariance with respect to several engineered transformations of the input and to temporal transformations in video data.

2. Theory

Before being able to observe the layer-wise evolution of the representation in deep networks, we first need quantify how good a representation is with respect to the learning problem. The representation is said to be *good* if simple and accurate models of the learning problem can be built on top of it. We measure it by means of an analysis based on kernel principal component analysis that determines how much of the relevant problem subspace is contained in the leading kernel principal components, more precisely, how well the learning problem can be solved from the leading kernel principal components. The analysis extends naturally to deep networks by building a sequence of kernels that subsume the mapping performed by more and more layers of the deep network and repeating the analysis for these deeper and deeper kernels.

2.1 Quantifying How Good a Representation Is

In this section, we are interested in quantifying how good a representation is with respect to the learning problem. The representation is *good* when it is possible to build models of the learning problem on top of it that are both *simple* and *accurate*.

A first technical difficulty is to quantify how simple a model is. Indeed, the notion of simplicity is highly subjective (Bousquet et al., 2004) and typically depends on which prior knowledge on the learning problem is taken for granted. For example, visual recognition tasks are very simple for humans, but very complex for simple learning algorithms such as a local learning machine. In this example, humans possess a form of prior on how the image should look like (e.g., we know how to classify real images from artificial images) and a machinery to make sense more easily of this complex data.

We choose to model this prior by isolating it into a kernel operator that measures how similar two data points drawn from the input distribution are. For example, a local predictor could be modeled with a Gaussian kernel while a more intelligent human-like predictor should be modeled with a more complex kernel encoding translation invariance, rotation invariance, etc. Then, the

induced kernel feature map $x \mapsto \phi(x)$ encodes implicitly all the prior defined in the kernel with the advantage that linear models can be built on top of it (Schölkopf et al., 1999).

A second technical difficulty comes from the fact that accuracy and simplicity are not always measurable in practice: accuracy of a model can only be estimated up to a certain precision from the finite data set and estimating the simplicity depends on whether we consider, for example, the number of parameters of a model, its entropy or its algorithmic complexity. For these reasons, we need to restrict ourselves to a class of models whose simplicity and accuracy can be easily measured and that are expressive enough to solve the learning problem.

We choose to use the kernel principal component analysis (kernel PCA, Schölkopf et al., 1998) as a basis for building measurably simple models of the learning problem. Our method consists of projecting the input distribution on the d first components (in terms of variance) of the kernel feature space and fitting a linear model on this low-rank representation. The number of components d controls the simplicity of the model. When d is small, the model is simple. When d is large, the model is complex. The accuracy can in turn be obtained by measuring the prediction error $e(d)$ of a linear predictor on top of the d-component kernel representation. We refer to the parameter d as the dimensionality of the model and $e(d)$ as the prediction error obtained with the d-component model. The curve $e(d)$ gives a complete picture of how good a representation is with respect to the learning problem. Figure 2 gives some examples of curves $e(d)$ and explains how these curves can be interpreted.

An advantage of the kernel PCA method is that there exists a theoretical framework and convergence bounds for the estimation of spectral properties from a limited number of samples drawn from the input distribution. In the case of fixed kernels, Braun et al. (2008) show that the projections to kernel principal components obtained with a finite and typically small number of samples n are close with essentially multiplicative errors to those that would be obtained in the asymptotic case where $n \to \infty$. This result can be naturally extended to a finite set of kernels. These convergence properties are desirable since the data distribution is unknown and only a finite number of observations are available for our analysis. Appendix A gives some additional information on the convergence of kernel principal components.

A second advantage of the kernel PCA method is the high flexibility that it offers with respect to the nature of the learning problem. Kernel PCA is not only independent of the input representation due to the kernel embedding, but also independent of the output representation. Indeed, kernel PCA simply acts as a regularizer on the kernel feature space that limits the complexity of the subsequent learning machine. Therefore any discriminative model can be used on top of the regularized representation, allowing to treat various classes of problems such as binary classification, multi-class classification or regression within the same framework.

To summarize, the kernel framework combines the four requirements of our analysis: (1) the kernel operator expresses and isolates the subjective notion of simplicity, (2) the complexity of the model is controlled by projecting the input distribution on a limited number of kernel principal components, (3) convergence bounds allow to effectively measure the accuracy of the model and (4) various models can be built on top of the leading kernel principal components in order to express the various types of learning problems (regression, classification, ...) that arise in real applications.

We present below the computation steps required to estimate how good a kernel k and its associated feature map $x \mapsto \phi(x)$ are with respect to a learning problem $p(x,y)$. Let $\{(x_1,y_1),\ldots,(x_n,y_n)\}$ be a data set of n points drawn independently from $p(x,y)$. Let $X = (x_1,\ldots,x_n)$ and $Y = (y_1,\ldots,y_n)$ be the matrices associated to the inputs and labels of the data set. We compute the kernel matrix K

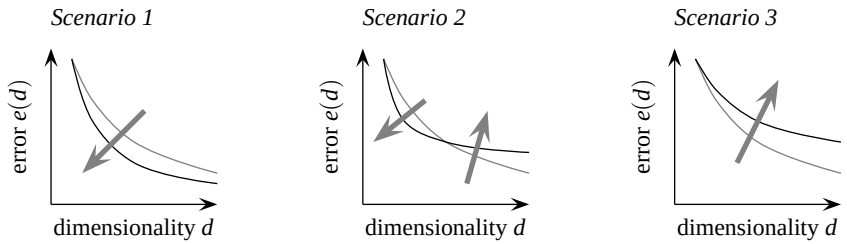


Figure 2: Effect of converting a representation of the learning problem $p(x,y)$ (gray curve) to a new representation of the learning problem $p(f(x),y)$ (black curve) where the input x is mapped to $f(x)$. We can distinguish three scenarios: (Scenario 1) the mapping produces a better representation from which more accurate models are obtained for every dimensionality—this is the desired behavior of deep networks,—(Scenario 2) the mapping concentrates the label information in the leading kernel principal components but also loses some information—lossy feature extractors typically fall into that category—and (Scenario 3) the mapping makes the learning problem more complex—this would be the result of introducing noise or throwing away label information.

associated to the data set:

$$K = \begin{pmatrix} k(x_1,x_1) & \dots & k(x_1,x_n) \\ \vdots & & \vdots \\ k(x_n,x_1) & \dots & k(x_n,x_n) \end{pmatrix}.$$

The kPCA components $u_1,\dots,u_n$ are obtained by performing an eigendecomposition of K where eigenvectors $u_1,\dots,u_n$ have unit length and eigenvalues $\lambda_1,\dots,\lambda_n$ are sorted by decreasing magnitude:

$$K = (u_1|\dots|u_n)\cdot \mathrm{diag}(\lambda_1,\dots,\lambda_n)\cdot (u_1|\dots|u_n)^\top.$$

Let $\hat{U} = (u_1|\dots|u_d)$ and $\hat{\Lambda} = \mathrm{diag}(\lambda_1,\dots,\lambda_d)$ be a d-dimensional approximation of the eigendecomposition. The space spanned by this basis approximates the space spanned by the d leading components of the infinite-dimensional kernel feature space associated to the probability distribution $p(x)$. In this space, the learning problem can be solved by a standard linear or logistic regression model. For regression problems, we fit a linear model $\beta^\star$ that maps the d leading components to the output:

$$\beta^\star = \mathrm{argmin}_\beta ||\hat{U}\beta - Y||_F^2 = \hat{U}^\top Y. \tag{1}$$

For classification problems, instead of fitting the model directly on the outputs, we fit the model on the log-likelihood of classes

$$\beta^\star = \mathrm{argmin}_\beta \prod_{i=1}^{n} \mathrm{softmax}([\hat{U}\beta]_i)_{y_i} \tag{2}$$

where $\mathrm{softmax}(z) = e^z / \sum_j e^{z_j}$ converts a vector z into a probability distribution over classes. Note that the optimization criterion only consists of the empirical risk minimization term and lacks a

regularization term. Indeed, the regularization is implicitly carried out by the projection on the d leading principal components. The problem is therefore well-posed only when $d \ll n$. Once the model $\beta^\star$ is computed, the estimated outputs can be calculated as

$$\hat{Y} = \hat{U}\beta^\star$$

for regression problems and as

$$\hat{y}_i = \text{argmax}([\hat{U}\beta^\star]_i) \qquad 1 \leq i \leq n$$

for classification problems. The training error is estimated as

$$e(d) = \frac{1}{n}\sum_{i=1}^{n} ||\hat{y}_i - y_i||^2 \tag{3}$$

for regression problems and as

$$e(d) = \frac{1}{n}\sum_{i=1}^{n} 1_{\hat{y}_i \neq y_i} \tag{4}$$

for classification problems. The test error can be obtained by cross-validating the linear model on random partitions of $(x_1, \ldots, x_n)$. Training and test error can be used as approximation bounds for the asymptotic case $n \to \infty$ where the model $\beta^\star$ would minimize the error on the real distribution $p(x,y)$. In the next sections, the upper and lower approximation bounds are respectively depicted as solid and dotted lines in Figure 5, 6 and 7.

2.2 Application to Deep Networks

In this section, we describe how the analysis of representations presented above can be used to measure the layer-wise forming of the representation in deep networks. Let $f(x) = f_L \circ \cdots \circ f_1(x)$ be a trained deep network of L layers. Our analysis consists of defining a sequence of "deep kernels"

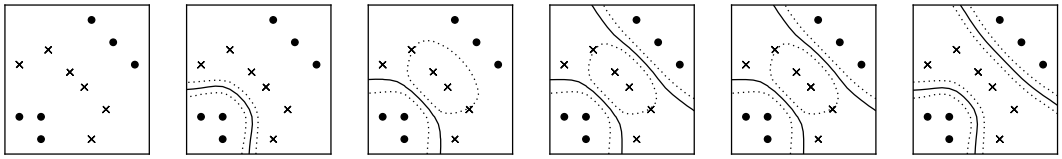
$$\begin{aligned} k_0(x,x') &= k_{RBF}(x,x'), \\ k_1(x,x') &= k_{RBF}(f_1(x), f_1(x')), \\ &\vdots \\ k_L(x,x') &= k_{RBF}(f_L \circ \cdots \circ f_1(x), f_L \circ \cdots \circ f_1(x')) \end{aligned}$$

that subsume the mapping performed by more and more layers of the deep network and repeating for each kernel the analysis presented in Section 2.1. Algorithm 1 summarizes the main computational steps of our analysis. The kernel k_{RBF} is the standard Gaussian kernel defined as $k_{RBF}(x,x') = \exp(-||x-x'||^2/2\sigma^2)$.

The main prior encoded by Gaussian kernels is the smoothness of the task of interest in the input space (Smola et al., 1998). Gaussian kernels are appropriate when two neighboring samples (in terms of Euclidean distance) are likely to have the same class. It remains to see how the concept of simplicity encoded by the Gaussian kernel can be understood from the perspective of the induced prediction model. Figure 3 shows that the simplicity of the model can be related to the number of allowed local variations in the input space. When d increases, more variations of the

Algorithm 1: Main computational steps of our layer-wise analysis of deep networks. At every layer of the deep network, the same analysis is performed, returning a list of curves $e(d)$ capturing the evolution of the representation in the deep network.

Input: A data set $\{(x_1,y_1),\ldots,(x_n,y_n)\}$
A deep network $f : x \mapsto f_L \circ \cdots \circ f_1(x)$
Output: The curves $e(d)$ for each layer l
for $l \in \{1,\ldots,L\}$ **do**
  for $\sigma \in \Sigma$ **do**
    $k(x,x') = k_{RBF(\sigma)}(f_l \circ \cdots \circ f_1(x), f_l \circ \cdots \circ f_1(x'))$
    compute the kernel matrix K associated to $k(x,x')$ and $(x_1,\ldots,x_n)$
    do the eigendecomposition $K = (u_1|\ldots|u_n) \cdot \mathrm{diag}(\lambda_1,\ldots,\lambda_n) \cdot (u_1|\ldots|u_n)^\top$
    for $d \in \{0,1,2,\ldots\}$ **do**
      build a low rank approximation of the input $\hat{U} \leftarrow (u_1|\ldots|u_d)$
      fit the model $\beta^\star$ that predicts $(y_1,\ldots,y_n)$ from $\hat{U}$ (cf. Equation 1 and 2)
      compute the error $e(d,\sigma)$ of the model $\beta^\star$ (cf. Equation 3 and 4)
  $e(d) = \min_\sigma e(d,\sigma)$
  plot the curve $e(d)$



$d = 1$, $e(d) = 0.5$ | $d = 2$, $e(d) = 0.25$ | $d = 3$, $e(d) = 0.25$ | $d = 4$, $e(d) = 0$ | $d = 5$, $e(d) = 0$ | $d = 6$, $e(d) = 0$

Figure 3: Interpretation of a prediction model based on the leading components of the Gaussian kernel on a toy data set. As we add more and more leading components of the kernel, the model becomes more flexible, creating a better decision boundary. Note that with four leading components, all the samples are already perfectly classified.

learning problem can be encoded and the prediction improves. Figure 4 shows that by making the problem increasingly complex—for example, by distorting it—the number of dimensions required to approach the error of the optimal classifier becomes larger and larger.

The notion of simplicity encoded by the Gaussian kernel is meaningful for a wide range of learning problems, however, it does not explain how simple more intelligent systems perceive problems such as vision and speech. Indeed, domain-specific regularities such as invariance to translation, scale or occlusion can not be modeled efficiently by a Gaussian kernel. Consequently, observing the learning problem become simpler as we build deeper and deeper kernels highlights the capacity of deep networks to model the regularities of the input distribution.

A last aspect that has not been discussed yet is how to choose the scale parameter σ of the Gaussian kernel. We decide to choose the parameter σ that minimizes the error $e(d)$, leading to a different scale for each dimensionality. The rationale for taking a different scale for each d is that the

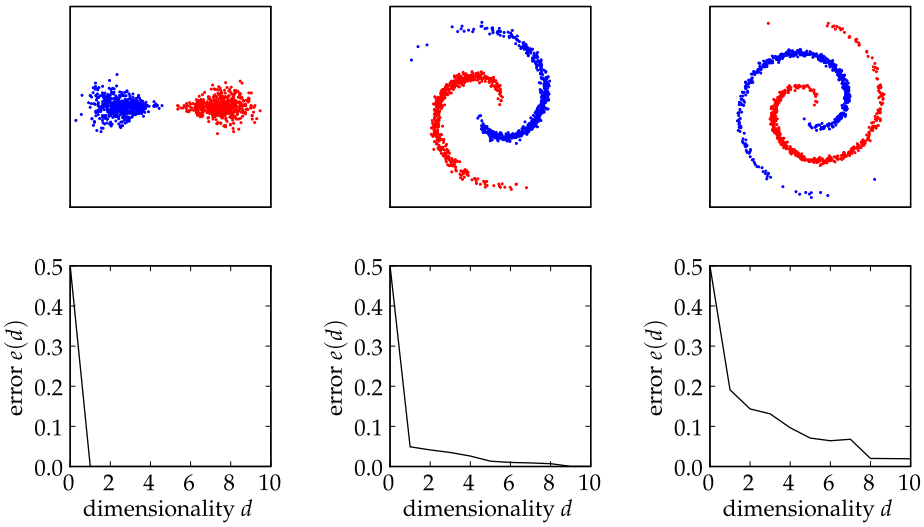


Figure 4: On the top, three increasingly complex representations of a binary classification problem. On the bottom, the curves $e(d)$ quantifying how good the representation of the learning problem is from the perspective of the Gaussian kernel. The original non-distorted learning problem can be solved perfectly with only one kernel principal component. As the input distribution gets distorted more and more, the number of leading components required to solve the learning problem increases, hinting that Gaussian kernels become progressively less suited.

optimal scale parameter typically shrinks as more leading components of the input distribution are observed. This parameter selection method also makes our analysis scale invariant. Scale invariance is desirable since the representation at a given layer of the deep network can take different scales due to the number of nodes contained in each layer or to the multiple types of nonlinearities that can be implemented in deep networks.

3. Methodology

In Section 2, we presented the theory and algorithms required to test our two hypotheses on the evolution of representations in deep networks. To summarize, the main idea of the analysis is to build a set of kernels that subsume the mapping performed at each layer of the deep network and, for each kernel, compute how good the representation is by measuring how many leading components of the kernel feature space are necessary in order to model the learning problem well. It remains to select a set of deep networks and data sets in order to test the hypotheses formulated in Section 1.

We consider the MNIST-10K and CIFAR-bw-10K data sets. These two data sets of 10000 samples each are a trimmed version of the larger MNIST handwritten digits and CIFAR image classification data sets (LeCun et al., 1998; Krizhevsky, 2009). The MNIST-10K data set is a 10-class classification data set that consists of 10000 grayscale images of 28×28 pixels representing handwritten digits and their associated label (a number between 0 and 9). The CIFAR-bw-10K data set is a 10-class classification data set that consists of 10000 grayscale images of 32×32 pixels representing different objects and their associated label (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck).

State-of-the-art performance on these data sets is achieved with architectures made of several layers, suggesting that these data sets are well suited to test the first hypothesis stated earlier in the paper, that is, the progressive simplification of the learning problem performed by deep networks. The second hypothesis on the effect of the structure of deep networks can be tested by taking a set of structured and unstructured deep networks and observing how the layer-wise evolution of the representation differs between these deep networks. We consider a multilayer perceptron (MLP), a pretrained multilayer perceptron (PMLP) and a convolutional neural network (CNN).

The multilayer perceptron (MLP, Rumelhart et al., 1986) is built by alternating linear transformations and nonlinearities applied element-wise to the output of the linear transformations. On the MNIST data set, we apply successively the functions

$$\begin{aligned} f_1(x) &= \text{sigm}(w_1 \cdot x + b_1), \\ f_2(x) &= \text{sigm}(w_2 \cdot x + b_2), \\ f_3(x) &= \text{softmax}(v \cdot x) \end{aligned}$$

to the input where weight matrices w_1, w_2, v and biases b_1, b_2 are learned from data and where the size of hidden layers is set to 1600. The sigmoid and softmax functions are defined as $\text{sigm}(x) = e^x/(1+e^x)$ and $\text{softmax}(x) = e^x/\sum_j e^{x_j}$. On the CIFAR data set, the sigmoid nonlinearity is replaced by the rectifying function defined as $\text{rectify}(x) = \max(0,x)$ and the size of hidden layers is set to 3600. Since it has been observed that overparameterizing deep networks generally improves the generalization error, the size of layers is chosen large with the only constraint of computational cost. The MLP is mostly unstructured as any type of solution can emerge from the random weights initialization.

The pretrained multilayer perceptron (PMLP, Hinton et al., 2006; Bengio et al., 2006) referred in this paper as PMLP is a multilayer perceptron that has been pretrained using a deep belief network (DBN, Hinton et al., 2006) and then fine-tuned on the discriminative task. The pretraining procedure aims to build a deep generative model of the input that can be used as a starting point to learn the supervised task. In order to use the same architecture as for the MLP during the fine-tuning procedure, we set the visible and hidden units of the DBN to be binary on the MNIST data set and respectively Gaussian and rectified linear (Nair and Hinton, 2010) on the CIFAR data set. Here, the structure of the deep network is implicitly given by the weights initialization subsequent to the unsupervised pretraining.

The convolutional neural network (CNN, LeCun et al., 1998) is a deep network inspired by the structure of the primary visual cortex (Hubel and Wiesel, 1962). Its particular convolutional structure exploits the spatial invariance of images in order to learn well-generalizing solutions from few labeled samples. It is built by alternating (1) convolutional layers $y = w \circledast x + b$ transforming a set of input features maps $\{x_1, x_2, \dots\}$ into a set of output features maps $\{y_1, y_2, \dots\}$ such that $y_i = \sum_j w_{ij} * x_j + b_i$ and where w_{ij} are convolution kernels, (2) detection layers where a nonlinearity is applied element-wise to the output of the convolutions in order to extract important features and (3) pooling layers subsampling each feature map by a given factor. On the MNIST data set, we apply successively the functions

$$\begin{aligned} f_1(x) &= \text{pooling}(\text{sigm}(w_1 \circledast x + b_1)), \\ f_2(x) &= \text{pooling}(\text{sigm}(w_2 \circledast x + b_2)), \\ f_3(x) &= \text{softmax}(v \cdot x) \end{aligned}$$

to the input where weight tensors w_1, w_2, weight matrix v and biases b_1, b_2 are learned from data, convolution kernels have size 5×5, pooling layers downsample the input by a factor two and the number of feature maps in each layer is set to 100. On the CIFAR data set, the sigmoid nonlinearity is replaced by the rectifying function described above.

The deep networks described above are trained on a supervised task with backpropagation (Rumelhart et al., 1986) and stochastic gradient descent (Bottou, 1991) with minibatches of size 20. The last layer has a L2 weight penalty. The softmax module (Bishop, 1996) optimizes the deep network for maximum likelihood. Weights of each layer l are initialized so that the output is of constant magnitude, thus falling into the correct regime of the subsequent nonlinearity. These deep networks are analyzed in two different settings:

- *Supervised learning:* the deep network is trained in a supervised fashion on the target task (digit classification for the MNIST data set and image classification for the CIFAR data set).
- *Transfer learning:* the deep network is trained in a supervised fashion on a binary classification task that consists of determining whether the sample has been flipped vertically or not.

These settings allow us to measure how the structure contained in deep networks affects different aspects of learning such as the layer-wise organization of the learned solution or the transferability of features from one task to another.

3.1 Experimental Setup

We train the deep networks on the 10000 samples of the data set until a training error of 2.5% is reached. Such stopping criterion ensures that the subsequent solutions have a constant complexity and that the limited capacity of the deep network has no side effect on the structure of the solution. As a sanity check, each architecture has been trained with the regular early stopping criterion on the full MNIST and CIFAR-bw data sets, leading to test errors that are on par with results published in the literature for similar architectures (MNIST-MLP: 1.6%, MNIST-PMLP: 1.3%, MNIST-CNN: 0.9%, CIFAR-bw-MLP: 48.1%, CIFAR-bw-PMLP: 46.8%, CIFAR-bw-CNN: 32.4%).

In our analysis, we estimate the kernel principal components with the 10000 samples used for training the deep network. Therefore, the empirical estimate of the d leading kernel principal components takes the form of d 10000-dimensional vectors, or similarly, of a data set of 10000 d-dimensional mapped samples. A lower bound of $e(d)$ is obtained by fitting and evaluating the linear model with the 10000 mapped samples. An upper bound of $e(d)$ is obtained by two-fold cross-validation (5000 samples to fit the model and the 5000 remaining samples to evaluate it). The set of candidate kernel widths is composed of the 0.1, 0.5 and 0.9 quantiles of the distribution of distances between pairs of points. It turns out that the effect of the kernel scale is rather small and that no further scale parameters are required. The layers of interest are the input data ($l = 0$) and the output of each layer ($l = 1, 2, \ldots$).

4. Results

In this section, we present the results of our analysis on the evolution of the representation in deep networks. Section 4.1 discusses the empirical observation that deep networks trained on the supervised task produce gradually simpler and more accurate representations of the learning problem.

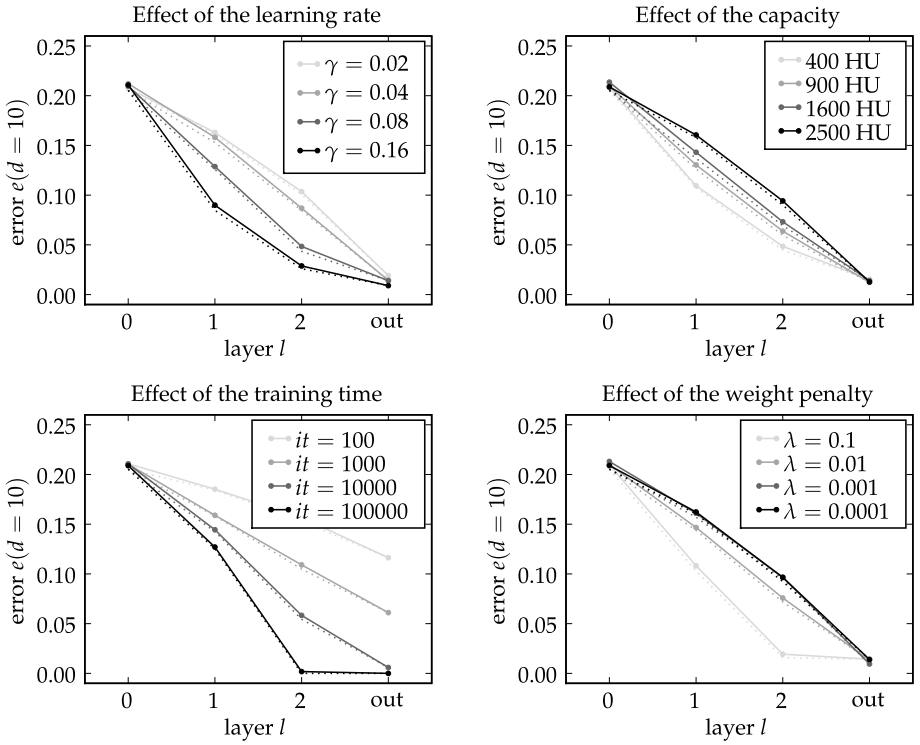


Figure 5: Effect of the learning rate, of the capacity, of the training time and of the weight penalty on the layer-wise evolution of the representation built by an MLP on the MNIST-10K data set. Solid and dotted lines respectively represent the upper and lower approximation bounds of the analysis. As the learning rate increases, the solution tends to make use primarily of the first layers of the deep network. The same effect is observed when we reduce the capacity, increase the weight penalty or increase the training time.

Then, Section 4.2 compares side-by-side the evolution of the representation in different deep networks and discusses the empirical observation that the structure of the deep network controls the layer-wise evolution of the representation in the deep network.

4.1 Better Representations are Built Layer After Layer

It is still an open question how the complex and multimodal form of intelligence observed in living organisms emerges from randomly disposed and locally scoped neurons. Machine learning researchers similarly pointed out that emergent properties also occur in artificial neural networks when trained with simple local algorithms such as Hebbian learning or backpropagation, without having to explicitly define the role of each individual neuron. Also, their ability to simultaneously specialize on specific tasks in output nodes and grow new functionalities from hidden nodes hints that information contained in the underlying distribution of sensed data should be ubiquitous, yet parsimonious where discrimination takes place.

It can be hypothesized that the organization of mapped data distributions within the neural network forms a continuum between general purpose distributions in the middle of the network and

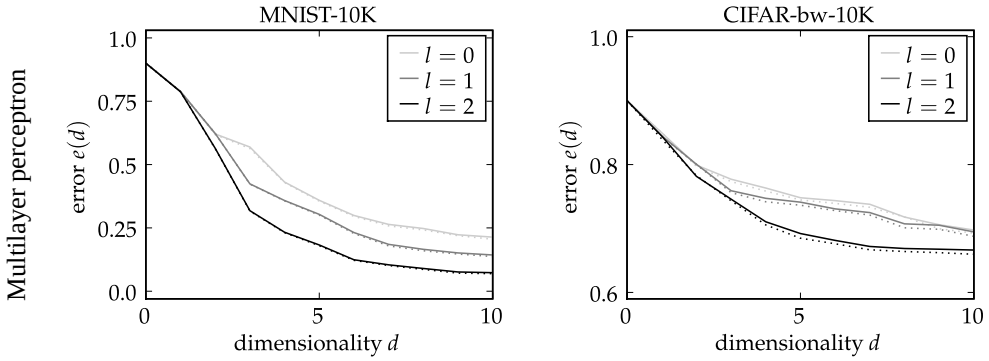


Figure 6: Layer-wise evolution of the error as a function of the number of dimensions when trained on the target task. Solid and dotted lines respectively represent the upper and lower approximation bounds of the analysis. As we move from the first to the last layers, the class information concentrates in the leading components of the mapped data distribution. This observation confirms the first hypothesis depicted in Figure 1.

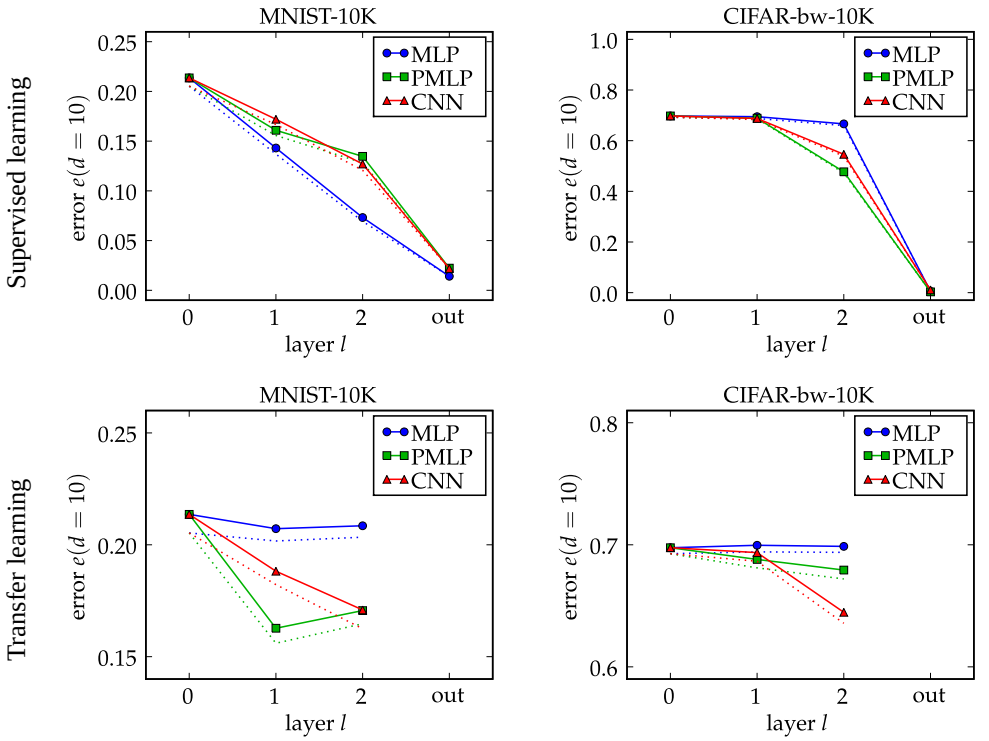


Figure 7: Layer-wise evolution of the error obtained for each training procedure for $d = 10$. Solid and dotted lines respectively represent the upper and lower approximation bounds of the analysis. We observe that the particular structure of the CNN and of the PMLP controls the layer-wise evolution of the representation. This confirms the second hypothesis depicted in Figure 1.

task specific distributions at its discriminative edges. Reformulating this hypothesis in the case of a simple multilayer feedforward network trained on image classification, the sensed distribution would evolve progressively from a distribution representing pixels well to a distribution representing classes well as the distribution is mapped to more and more layers.

We can observe in Figure 6 that this hypothesis holds within the span of our experimental setup and that simultaneously lower-dimensional and more accurate models of the task can be obtained layer after layer. This means that the task-relevant information, initially spread over a large number of principal components, converges progressively towards the leading components of the mapped data distribution.

This layer-wise preservation of the statistical tractability of the learning problem and its progressive simplification is a theoretical motivation for using these deep networks in a modular way (Caruana, 1997; Weston et al., 2008; Collobert and Weston, 2008): additional modules can be plugged on top of intermediate representations and still make sense of it.

4.2 Role of the Structure of Deep Networks

Training deep networks is a complex nonconvex learning problem with many reasonable solutions. As it can be seen in Figure 5, even simple hyperparameters such as the learning rate or the L2 weight penalty can greatly influence the layer-wise structure of the solution. Adding to the fact that those are only a fraction of the hyperparameters that needs to be tuned in order to achieve high generalization (e.g., importance of reconstruction error, orthogonality of hidden representations), it can therefore be tricky—if not, impossible—to find an appropriate combination of hyperparameters that leads to a well-structured solution for the learning problem.

On the other hand, the unsupervised pretraining proposed by Hinton et al. (2006) finds a network of latent variables that better represents the underlying distribution. As a consequence, the structure of the pretrained deep network already contains a certain part of the solution (Larochelle et al., 2009) and possibly makes better use of each layer. Similarly, in the context of sequential data, we can postulate that dedicating the early layers of the architecture to a convolutional preprocessing is also a more effective (LeCun, 1989; Serre et al., 2005) and biologically plausible (Ringach, 2002) way of solving the learning problem. Both approaches have shown empirically to produce better generalization (LeCun, 1989; Salakhutdinov and Hinton, 2007).

We corroborate this argument by comparing in Figure 7 the layer-wise evolution of the representation for different deep networks: a multilayer perceptron (MLP), a pretrained MLP (PMLP) and a convolutional neural network (CNN). On one side, the MLP does not embed any preconditioning on the learning problem. On the other side, the PMLP and the CNN embed respectively a generative model of the input and a spatial invariance prior on the problem. We can think of the mechanisms implemented by the PMLP and the CNN as complex regularizers on the solution of the learning problem.

Figure 7 (top) shows the evolution of the representation with respect to the learning problem when the deep network has been trained on the target task. We observe that the evolution of the representation of the MLP follows a different trend than the representation built by the PMLP and the CNN. The MLP tends to solve the MNIST problem greedily, discriminating from the first layers while the PMLP and the CNN postpone the discrimination to the last layers. On the other hand, on the CIFAR data set, the MLP doesn't discriminate until the last layer while the PMLP and the CNN spread the discrimination to more layers. Figure 7 (bottom) shows the evolution of the

representation with respect to the learning problem when the deep network has been trained on the transfer task. On both data sets, the representation built by the MLP does not improve as the deep network specializes on the transfer task while the PMLP and the CNN still build in the first layers a better representation of the learning problem, corroborating the effect of the PMLP and the CNN on structure of the solution.

These observations suggest that the complex regularizers implemented in the PMLP and the CNN have the effect of facilitating the construction of a structured solution, controlling the rate of discrimination at every layer. Erhan et al. (2010) already described the PMLP as a regularized version of the MLP and showed how it improves the generalization ability of deep networks. Our analysis completes the study, providing a layer-wise perspective on the effect and the role of regularization in deep networks and a unified view on the very different regularizers implemented by the PMLP and the CNN.

5. Conclusion and Discussion

We introduce a method for analyzing deep networks that combines kernel methods and descriptive statistics in order to quantify the layer-wise evolution of the representation in deep networks. Our method abstracts deep networks as a sequence of deeper and deeper kernels subsuming the mapping performed by more and more layers. The kernel framework expresses the relation between the representation built in the deep network and the learning problem.

Our analysis is able to detect and quantify the progressive and layer-wise transformation of the input performed by the deep network. In particular, we find that properly trained deep networks progressively simplify the statistics of complex data distributions, building in their last layers representations that are both simple and accurate.

The analysis also corroborates the hypothesis that a suitable structure for the deep network allows to make efficient use of its representational power by controlling the rate of discrimination at each layer of the deep network. This observation provides a new unified view on the role and effect of regularizers in deep networks.

Conceptually, our analysis is not only restricted to artificial neural networks. We believe that performing a similar analysis on different levels of processing in a biological neural architecture may reveal interesting parallels between artificial and biological neural systems.

Appendix A. More Background Information on kPCA Convergence

In this section, we briefly give some additional results on the convergence properties of kernel PCA. For the full account, please refer to Braun (2006) and Braun et al. (2008).

The rationale behind using the number of kPCA components as an estimate of the dimensionality rather than simpler metrics such as counting the number of support vectors of a trained SVM is that the first method provides an estimate of the dimensionality that is provably robust to the number of samples used in the analysis. This interesting fact was derived from a fundamental result on the approximation error of scalar products with eigenvectors of the kernel matrix with respect to their asymptotic counterparts.

More concretely, if $x_1, \ldots, x_n \in \mathcal{X}$ are points drawn i.i.d. from some probability distribution $P_{\mathcal{X}}$, we define the kernel matrix K of a Mercer kernel k by

$$K_{ij} = k(x_i, x_j) \qquad \text{for } 1 \leq i, j \leq n.$$

As $n \to \infty$, the eigenvalues and eigenvectors of K converge to those of the integral operator

$$T_k(f) = \int_{x \in \mathcal{X}} k(.,x) f(x) dP_{\mathcal{X}}$$

in an appropriate measure. In particular, it has been shown by Braun (2006) that the approximation error between the ith eigenvalue λ_i of K (in descending order) and corresponding eigenvalue l_i of T_k scales essentially multiplicatively, that is,

$$|\lambda_i - l_i| \leq C(n) l_i + \varepsilon(n),$$

where $C(n) \to 0$, $\varepsilon(n) \to 0$ as $n \to \infty$, and even for small n, $\varepsilon(n)$ is small.

So, even for a small number of points, the structure of the principal components in feature space are very similar compared to the asymptotic case. Moreover, as the next result shows, the same also holds for the location of the sample vector of a function with respect to the eigenspaces.

As shown by Braun et al. (2008), for a bounded function g which lies in the range of T_k (that is, there exists a h such that $T_k(h) = g$), one can bound the scalar products between the sample vector $G = (g(x_1), \ldots, g(x_n))$ and the eigenvectors u_i (normalized to unit length) of K by

$$\frac{1}{\sqrt{n}} |u_i^\top G| \leq \lambda_i C(n) + \varepsilon(n),$$

where $\varepsilon(n) \to 0$ as $n \to \infty$. Note that the scalar products with the eigenfunctions ψ_i of T_k also decay as $O(l_i)$, which are again linked to λ_i by the results discussed above.

In essence, this result shows that the scalar products between a subsampled smooth function decays as quickly as the eigenvalues of the kernel matrix, such that the information about g is contained in the leading kPCA components only. Here, smoothness means that g lies in the range of T_k such that it is a smoothed version of some function h after convolution with the kernel function k. On the other hand, any noise which is independent of the x_i is uniformly distributed over all kPCA components. In summary, if one plots the products $u_i^\top Y$ with the label vector $Y = (y_1, \ldots, y_n)$, one obtains a decomposition of the label information Y with respect to the kPCA components. From the above considerations, it follows that the spectrum will typically consist of a flat "noise bed" from which the relevant information in the leading components can clearly be distinguished. This result is illustrated in Figure 8 for a small toy example.

References

Gaston Baudat and Fatiha Anouar. Generalized discriminant analysis using a kernel approach. *Neural Computation*, 12(10):2385–2404, 2000.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*, pages 153–160, 2006.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.

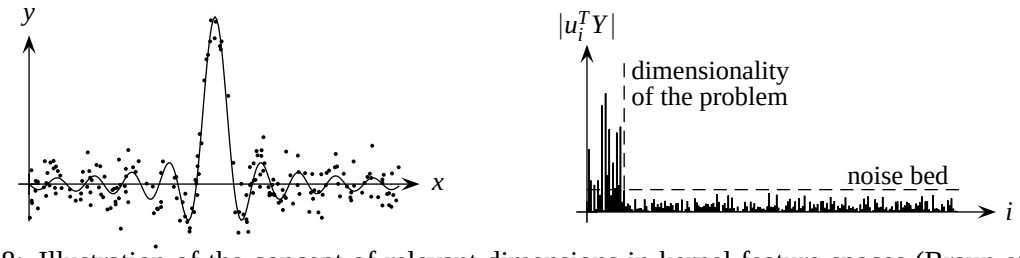


Figure 8: Illustration of the concept of relevant dimensions in kernel feature spaces (Braun et al., 2008). On the left, samples drawn from a toy distribution $p(x,y)$. On the right, label contributions of each kPCA component $u_1,\ldots,u_n$. It can be observed that a small number of leading principal components containing relevant label information emerge from a flat noise bed.

Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes*, 1991.

Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176, pages 169–207. Springer, 2004.

Mikio L. Braun. Accurate bounds for the eigenvalues of the kernel matrix. *Journal of Machine Learning Research*, 7:2303–2328, 2006.

Mikio L. Braun, Joachim Buhmann, and Klaus-Robert Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9:1875–1908, 2008.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems 22*, pages 342–350, 2009.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167, 2008.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

Ian Goodfellow, Quoc Le, Andrew Saxe, and Andrew Y. Ng. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems 22*, pages 646–654, 2009.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160:106–154, January 1962.

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

Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–43, 1990.

Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

Yann LeCun. Generalization and network design strategies. In *Connectionism in Perspective*. Elsevier, 1989. An extended version was published as a technical report of the University of Toronto.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(1):2278–2324, 1998.

Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–48, 1999.

Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Learning discriminative and invariant nonlinear features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):623–628, 2003.

Grégoire Montavon, Mikio Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. In *Advances in Neural Information Processing Systems 23*, pages 1678–1686, 2010.

Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2): 181–202, 2001.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

Genevieve B. Orr and Klaus-Robert Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, 1998. Springer. This book is an outgrowth of a 1996 NIPS workshop.

Dario L. Ringach. Spatial structure and symmetry of simple-cell receptive fields in macaque primary visual cortex. *The Journal of Neurophysiology*, 88(1):455–463, 2002.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

Bernhard Schölkopf, Sebastian Mika, Chris J. C. Burges, Philipp Knirsch, Klaus-Robert Müller, Gunnar Rätsch, and Alex J. Smola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 994–1000, 2005.

Steve Smale, Lorenzo Rosasco, Jack Bouvrie, Andrea Caponnetto, and Tomaso Poggio. Mathematics of the neural response. *Foundations of Computational Mathematics*, 10(1):67–91, 2010.

Alex J. Smola, Bernhard Schölkopf, and Klaus-Robert Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11(4):637–649, 1998.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1168–1175, 2008.

Andre Wibisono, Jake Bouvrie, Lorenzo Rosasco, and Tomaso Poggio. Learning and invariance in a family of hierarchical kernels. Technical report, Massachusetts Institute of Technology, 2010.