The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Deep Semi-Random Features for Nonlinear Function Approximation

Kenji Kawaguchi[*]
Massachusetts Institute of Technology

Bo Xie,[*]**Le Song**
Georgia Institute of Technology

Abstract

We propose semi-random features for nonlinear function approximation. The flexibility of semi-random feature lies between the fully adjustable units in deep learning and the random features used in kernel methods. For one hidden layer models with semi-random features, we prove with no unrealistic assumptions that the model classes contain an arbitrarily good function as the width increases (universality), and despite non-convexity, we can find such a good function (optimization theory) that generalizes to unseen new data (generalization bound). For deep models, with no unrealistic assumptions, we prove universal approximation ability, a lower bound on approximation error, a partial optimization guarantee, and a generalization bound. Depending on the problems, the generalization bound of deep semi-random features can be exponentially better than the known bounds of deep ReLU nets; our generalization error bound can be independent of the depth, the number of trainable weights as well as the input dimensionality. In experiments, we show that semi-random features can match the performance of neural networks by using slightly more units, and it outperforms random features by using significantly fewer units. Moreover, we introduce a new implicit ensemble method by using semi-random features.

Introduction

Many recent advances, such as human-level image classification (Deng et al. 2009) and game playing (Bellemare et al. 2013; Silver et al. 2016) in machine learning are attributed to large-scale nonlinear function models. There are two dominating paradigms for nonlinear modeling in machine learning: kernel methods and neural networks:

- Kernel methods employ *pre-defined* basis functions, $k(x, x')$, called kernels to represent nonlinear functions (Scholkopf and Smola 2001; Shawe-Taylor and Cristianini 2004). Learning algorithms that use kernel methods often come with nice theoretical properties–globally optimal parameters can be found via convex optimization, and statistical guarantees can be provided rigorously. However, kernel methods typically work with matrices that are quadratic in the number of samples, leading to unfavorable computation and storage complexities. A popular approach to tackle such issues is to approximate kernel functions using random features (Rahimi and Recht 2008; Sindhwani, Avron, and Mahoney 2014; Pennington, Yu, and Kumar 2015). One drawback of random features is that its approximation powers suffer from the curse of dimensionality (Barron 1993) because its bases are not adaptive to the data.
- Neural networks use adjustable basis functions and learn their parameters to approximate the target nonlinear function (LeCun, Bengio, and Hinton 2015). Such adaptive nature allows neural networks to be compact yet expressive. As a result, they can be efficiently trained on some of the largest datasets today. By incorporating domain specific network architectures, neural networks have also achieved state-of-the-art results in many applications. However, learning the basis functions involves difficult non-convex optimization. Few theoretical insights are available in the literature and more research is needed to understand the working mechanisms and theoretical guarantees for neural networks (Choromanska, LeCun, and Arous 2015; Swirszcz, Czarnecki, and Pascanu 2016; Shamir 2016).

Can we have the best of both worlds? Can we develop a framework for big nonlinear problems which has the ability to adapt basis functions, has low computational and storage complexity, while at the same time retaining some of the theoretical properties of random features? Towards this goal, we propose semi-random features to explore the space of trade-off between flexibility, provability and efficiency in nonlinear function approximation. We show that semi-random features have a set of nice theoretical properties, like random features, while possessing a (deep) representation learning ability, like deep learning. More specifically:

- Despite the nonconvex learning problem, semi-random feature model with one hidden layer has no bad local minimum;
- Depending on the problems, the generalization bound of deep semi-random features can be exponentially better than known bounds of deep ReLU nets;
- Semi-random features can be composed into multi-layer architectures, and going deep in the architecture leads to more expressive model than going wide;
- Semi-random features also lead to statistical stable func-

[*]Equal contribution

tion classes, where generalization bounds can be readily provided.

Background

We briefly review different ways of representing nonlinear functions in this section.

Hand-designed basis. In a classical machine learning approach for nonlinear function approximation, users or domain experts typically handcraft a set of features $\phi_{\text{expert}} : \mathcal{X} \rightarrow \mathcal{H}$, a map from an input data space $\mathcal{X}$ to a (complete) inner product space $\mathcal{H}$. Many empirical risk minimization algorithms then require us to compute the inner product of the features as $\langle \phi_{\text{expert}}(x), \phi_{\text{expert}}(x') \rangle_{\mathcal{H}}$ for each pair $(x, x') \in \mathcal{X} \times \mathcal{X}$. Computing this inner product can be expensive when the dimensionality of $\mathcal{H}$ is large, and indeed it can be infinite. For example, if $\mathcal{H}$ is the space of square integrable functions, we need to evaluate the integral as $\langle \phi_{\text{expert}}(x), \phi_{\text{expert}}(x') \rangle_{\mathcal{H}} = \int_{\omega} \phi_{\text{expert}}(x; \omega) \overline{\phi_{\text{expert}}(x'; \omega)}$.

Kernel methods. When our algorithms solely depend on the inner product, the kernel trick avoids this computational burden by introducing an easily computable kernel function as $k_{\text{expert}}(x', x) = \langle \phi_{\text{expert}}(x), \phi_{\text{expert}}(x') \rangle_{\mathcal{H}}$, resulting in an implicit definition of the features ϕ_{expert} (Scholkopf and Smola 2001; Shawe-Taylor and Cristianini 2004). However, the kernel approach typically scales poorly on large datasets. Given a training set of m input points $\{x_i\}_{i=1}^{m}$, evaluating a learned function at a new point x requires computing $\hat{f}(x) = \sum_{i=1}^{m} \alpha_i k_{\text{expert}}(x_i, x)$, the cost of which increases linearly with m. Moreover, it usually requires computing (or approximating) inverses of matrices of size $m \times m$.

Random features. In order to scale to large datasets, one can approximate the kernel by a set of random basis functions sampled according to some distributions. That is, $k_{\text{expert}}(x', x) \approx \frac{1}{C} \sum_{j=1}^{C} \phi_{\text{random}}(x; \mathbf{r}_j) \phi_{\text{random}}(x'; \mathbf{r}_j)$, where both the type of basis functions ϕ_{random}, and the sampling distribution for the random parameter $\mathbf{r}_j$ are determined by the kernel function. Due to its computational advantage and theoretical foundation, the random feature approach has many applications and is an active research topic (Rahimi and Recht 2008; Sindhwani, Avron, and Mahoney 2014; Pennington, Yu, and Kumar 2015).

Neural networks. Neural networks approximate functions using weighted combination of *adaptable* basis functions $f(x) = \sum_{k=1}^{n} w_k^{(2)} \phi(x; \mathbf{w}_k^{(1)})$, where both the combination weights $w_k^{(2)}$ and the parameters $\mathbf{w}_k^{(1)}$ in each basis function ϕ are learned from data. Neural networks can be composed into multilayers to express highly flexible nonlinear functions.

Semi-Random Features

When comparing different nonlinear representaitons, we can see that random features are designed to approximate a known kernel, but not for learning these features from the given dataset (i.e., it is not a *representation learning*). As a result, when compared to neural network, it utilizes less amount of information encoded in the dataset, which could be disadvantageous. Neural networks, on the other hand, pose a difficulty for theoretical developments due to non-convexity in optimization.

This suggests a hybrid approach of random feature and neural network, called *semi-random feature* (or semi-random unit), to learn representation (or feature) from datasets. The goal is to obtain a new type of basis functions which can retain some theoretical guarantees via injected randomness (or diversity) in hidden weights, while at the same time have the ability to adapt to the data at hand. More concretely, semi-random features are defined as

$$\phi_s(x; \mathbf{r}, \mathbf{w}) = \sigma_s(\mathbf{x}^\top \mathbf{r}) \left(\mathbf{x}^\top \mathbf{w}\right), \tag{1}$$

where $\mathbf{x} = (1, x^\top)^\top$ is assumed to be in $\mathbb{R}^{1+d}$, $\mathbf{r} = (r_0, r^\top)^\top$ is sampled randomly, and $\mathbf{w} = (w_0, w^\top)^\top$ is adjustable weights to be learned from data (hence, it is "semi-random"). Furthermore, the family of functions σ_s for $s \in \{0, 1, 2, \dots\}$ is defined as $\sigma_s(z) = (z)^s H(z)$, where H is Heaviside step function ($H(z) = 1$ for $z > 0$ and 0 otherwise). For instance, σ_0 is simply Heaviside step function, σ_1 is ramp function, and so on. We call the corresponding semi-random features with $s = 0$ "linear semi-random features (LSR)" and with $s = 1$ "squared semi-random features (SSR)". An illustration of example semi-random features can be found in Appendix.

Unlike dropout, which uses a data independent random switching mechanism (during training), the random switching in semi-random feature depends on the input data x (during both training and testing), inducing highly nonlinear models with practical advantages. By further enhancing this property, we additionally introduce linear semi-random implicit-ensemble (LSR-IE) features in Section "Image classification benchmarks".

Intuitively, models with semi-random features have more expressive power than those with random features because of the learnable unit parameter $\mathbf{w}$. Yet, these models are less flexible compared to neural networks, since the parameters in $\sigma_s(\mathbf{x}^\top \mathbf{r})$ is sampled randomly. Depending on the problems. this property of semi-random feature *can* result in exponential advantage over fully random feature in *expressiveness* (as discussed in Appendix), and exponential advantage over deep ReLU models in *generalization* error bound (as discussed in Section "Generalization Guarantee").

One Hidden Layer Model

With semi-random features ϕ_s in equation (1), we define one hidden layer model for nonlinear function as

$$\hat{f}_n^s(x; w) = \sum_{k=1}^{n} \phi_s(x; \mathbf{r}_k, \mathbf{w}_k^{(1)}) w_k^{(2)}, \tag{2}$$

where $\mathbf{r}_k$ is sampled randomly for $k \in \{2, 3, \cdots, n\}$ as described in Section "Semi-Random Features", and $\mathbf{r}_1$ is fixed to be the first element of the standard basis as $\mathbf{r}_1 = \mathbf{e}_1 = (1, 0, 0, \cdots 0)^\top$ (to compactly represent a constant term in x). We can think of this model as one hidden layer model by considering $\phi_s(x;\mathbf{r},\mathbf{w}_k^{(1)})$ as the output of k-th unit of the hidden layer, and $\mathbf{w}_k^{(1)}$ as adjustable parameters associated with this hidden layer unit. This way of understanding the model will become helpful when we generalize it to a multilayer model in Section "Multilayer Model". Note that $\hat{f}_n^s(x;w)$ is a nonlinear function of x. When it is clear, by the notation w, we denote all adjustable parameters in the entire model.

In matrix notation, the model in (2) can be rewritten as

$$\hat{f}_n^s(x;w) = \left(\sigma_s(\mathbf{x}^\top \mathbf{R}) \odot (\mathbf{x}^\top \mathbf{W}^{(1)})\right) W^{(2)}, \qquad (3)$$

where

$$\mathbf{W}^{(1)} = (\mathbf{w}_1^{(1)}, \mathbf{w}_2^{(1)}, \dots, \mathbf{w}_n^{(1)}) \in \mathbb{R}^{(d+1)\times n},$$

$$\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n) \in \mathbb{R}^{(d+1)\times n}, \text{and}$$

$$\mathbf{W}^{(2)} = (w_1^{(2)}, \dots, w_n^{(2)})^\top \in \mathbb{R}^{n+1}.$$

Here, $(M_1 \odot M_2)$ represents a Hadamard product of two matrices M_1 and M_2. Furthermore $\sigma_s(M)_{ij} = \sigma_s(M_{ij})$, given a matrix M of any size (with overloads of the symbol σ_s).

In the following subsections, we present our theoretical results for one hidden layer model. All proofs in this paper are deferred to the appendix.

Universal Approximation Ability

We show that our model class has universal approximation ability. Given a finite s, our model class is defined as

$$\mathcal{F}_n^s = \{x \mapsto \hat{f}_n^s(x;w) \mid w \in \mathbb{R}^{d_w}\},$$

where $d_w = (d+1)n + n$ is the number of adjustable parameters. Let $L^2(\Omega)$ be the space of square integrable functions on a compact set $\Omega \subseteq \mathbb{R}^d$. Then Theorem 1 states that we can approximate any $f \in L^2(\Omega)$ arbitrarily well as we increase the number of units n. We discuss the importance of the bias term r_0 to obtain the universal approximation power in Appendix.

Theorem 1 (Universal approximation) *Let s be any fixed finite integer and let $\Omega \neq \{0\}$ be any fixed nonempty compact subset of $\mathbb{R}^d$. Then, for any $f \in L^2(\Omega)$, with probability one,*

$$\lim_{n\to\infty} \inf_{\hat{f}\in\mathcal{F}_n^s} \|f - \hat{f}\|_{L^2(\Omega)} = 0.$$

Optimization Theory

As we have confirmed universal approximation ability of our model class $\mathcal{F}_n^s$ in the previous section, we now want to find a good $\hat{f} \in \mathcal{F}_n^s$ via empirical loss minimization. More specifically, given a dataset $\{(x_i, y_i)\}_{i=1}^m$, we will consider the following optimization problem:

$$\underset{w\in\mathbb{R}^{d_w}}{\text{minimize}}\ \mathcal{L}(w) = \frac{1}{2m}\sum_{i=1}^m \left(y_i - \hat{f}_n^s(x_i;w)\right)^2.$$

Let $Y = (y_1, y_2, \dots, y_m)^\top \in \mathbb{R}^m$ and $\hat{Y} = (f_n^s(x_1;w), f_n^s(x_2;w), \dots, f_n^s(x_m;w))^\top \in \mathbb{R}^m$. Given a matrix M, let $\mathbf{P}_{\text{col}(M)}$ and $\mathbf{P}_{\text{null}(M)}$ be the projection matrices onto the column space and null space of M.

Our optimization problem turns out to be characterized by the following m by nd matrix:

$$D = \begin{bmatrix} \sigma_s(\mathbf{x}_1^\top \mathbf{r}_1)\mathbf{x}_1^\top & \cdots & \sigma_s(\mathbf{x}_1^\top \mathbf{r}_n)\mathbf{x}_1^\top \\ \vdots & \ddots & \vdots \\ \sigma_s(\mathbf{x}_m^\top \mathbf{r}_1)\mathbf{x}_m^\top & \cdots & \sigma_s(\mathbf{x}_m^\top \mathbf{r}_n)\mathbf{x}_m^\top \end{bmatrix}, \qquad (4)$$

where $\sigma_s(\mathbf{x}_i^\top \mathbf{r}_j)\mathbf{x}_i^\top$ is a $1 \times d$ block at (i,j)-th block entry. That is, at any global minimum, we have $\mathcal{L}(w) = \frac{1}{2m}\|\mathbf{P}_{\ker(D^T)}Y\|^2$ and $\hat{Y} = \mathbf{P}_{\text{col}(D)}Y$. Moreover, we can achieve a global minimum in polynomial time in d_w based on the following theorem.

Theorem 2 (No bad local minima and few bad critical points) *For any s and any n, the optimization problem of $\mathcal{L}(w)$ has the following properties:*

(i) it is non-convex (if $D \neq 0$)[1],

(ii) every local minimum is a global minimum,

(iii) if $w_k^{(2)} \neq 0$ for all $k \in \{1, 2, \dots, n\}$, every critical point is a global minimum, and

(iv) at any global minimum, $\mathcal{L}(w) = \frac{1}{2m}\|\mathbf{P}_{\text{null}(D^T)}Y\|^2$ and $\hat{Y} = \mathbf{P}_{\text{col}(D)}Y$.

Theorem 2 (optimization) together with Theorem 1 (universality) suggests that not only does our model class contain an arbitrarily good function (as n increases), but also we can find the best function in the model class given a dataset. In the context of understanding the loss surface of neural networks (Kawaguchi 2016; Freeman and Bruna 2016), Theorem 2 implies that the potential problems in the loss surface are due to the inclusion of $\mathbf{r}$ as an optimization variable.

Generalization Guarantee

In the previous sections, we have shown that our model class has universal approximation ability and that we can learn the best model given a finite dataset. A major remaining question is about the generalization property; how well can a learned model generalize to unseen new observations? Theorem 3 bounds the generalization error; the difference between expected risk, $\frac{1}{2}\mathbb{E}_x(f(x) - \hat{f}(x;w^*))^2$, and empirical risk, $\mathcal{L}(w)$. In Theorem 3, we use the following notations: $[\![\sigma, x]\!] = [\sigma_s(\mathbf{x}^\top \mathbf{r}_1)\mathbf{x} \ \cdots \ \sigma_s(\mathbf{x}^\top \mathbf{r}_n)\mathbf{x}] \in \mathbb{R}^{nd}$ and $[\![w]\!] = (w_1^{(2)} w_1^{(1)\top}, w_2^{(2)} w_2^{(1)\top}, \dots, w_n^{(2)} w_n^{(1)\top})^\top \in \mathbb{R}^{nd}$.

Theorem 3 (Generalization bound for shallow model) *Let $s \geq 0$ and $n > 0$ be fixed. Consider any model class $\mathcal{F}_n^s$ with $\|[\![w]\!]\|_2 \leq C_W$ and $\|[\![\sigma, x]\!]\|_2 \leq C_{\sigma x}$ almost surely. Then, with probability at least $1-\delta$, for any $\hat{f} \in \mathcal{F}_n^s$,*

$$\frac{1}{2}\mathbb{E}_x(f(x) - \hat{f}(x;w))^2 - \mathcal{L}(w)$$

$$\leq (C_Y^2 + C_{\hat{Y}}^2)\sqrt{\frac{\log\frac{1}{\delta}}{2m}} + 2(C_Y + C_{\hat{Y}})\frac{C_{\hat{Y}}}{\sqrt{m}},$$

where $C_{\hat{Y}} = C_W C_{\sigma x}$.

[1] In the case of $D = 0$, $\mathcal{L}(w)$ is convex in a trivial way; our model class only contains a single constant function $x \mapsto 0$.

By combining Theorem 2 (optimization) and Theorem 3 (generalization), we obtain the following remark.

Remark 1. *(Expected risk bound)* Let C_W be values such that the global minimal value $\mathcal{L}(w) = \frac{1}{2m}\|\mathbf{P}_{\ker(D^T)}Y\|^2$ is attainable in the model class (e.g., setting $C_W = \|(D^\top D)^\dagger D^\top Y\|$ suffices). Then, at any critical points with $w_k^{(2)} \neq 0$ for all $k \in \{1, 2, \ldots, n\}$ and any local minimum such that $\|[\![w]\!]\|_2 < C_W$, we have

$$\mathbb{E}_x(f(x) - \hat{f}(x; w^*))^2 \leq \frac{\|\mathbf{P}_{\mathrm{null}(D^T)}Y\|^2}{m} + O\left(\sqrt{\frac{\log \frac{1}{\delta}}{m}}\right), \quad (5)$$

with probability at least $1 - \delta$. Here, $O(\cdot)$ notation simply hides the constants in Theorem 3.

In the right-hand side of equation (5), the second term goes to zero as m increases, and the first term goes to zero as n or d increases (because $\mathrm{null}(D^T)$ becomes a smaller and smaller space as n or d increases, eventually containing only 0). Hence, we can minimize the expected risk to zero.

Multilayer Model

We generalize one hidden layer model to H hidden layer model by composing semi-random features in a nested fashion. More specifically, let n_l be the number of units, or width, in the l-th hidden layer for all $l = 1, 2, \ldots, H$. Then we will denote a model of fully-connected feedforward semi-random networks with H hidden layers by

$$\hat{f}^s_{n_1,\ldots,n_H}(x; w) = h_w^{(H)}\, W^{(H+1)}, \quad (6)$$

where for all $l \in \{2, 3, \cdots, H\}$,

$$h_w^{(l)}(x) = h_r^{(l)}(x) \odot (h_w^{(l-1)}(x)\, W^{(l)}) \text{ and}$$
$$h_r^{(l)}(x) = \sigma_s(h_r^{(l-1)}(x)\, R^{(l)})$$

is the output of the l-th *semi-random* hidden layer, and the output of the l-th *random* switching layer respectively. Here, $W^{(l)} = (w_1^{(l)}, w_2^{(l)}, \ldots, w_{n_l}^{(l)}) \in \mathbb{R}^{n_{l-1}\times n_l}$ and $R^{(l)} = (r_1^{(l)}, r_2^{(l)}, \ldots, r_{n_l}^{(l)}) \in \mathbb{R}^{n_{l-1}\times n_l}$. Similarly to one hidden layer model, $r_k^{(l)}$ is sampled randomly for $k \in \{2, 3, \ldots, n_l\}$ but $r_1^{(l)}$ is fixed to be $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^\top$ (to compactly write the effect of constant terms in x). The output of the first hidden layer is the same as that of one hidden layer model:

$$h_w^{(1)}(x) = h_r^{(1)}(x) \odot (\mathbf{x}\mathbf{W}^{(1)}) \text{ and } h_r^{(1)}(x) = \sigma_s(\mathbf{x}\mathbf{R}^{(1)}),$$

where the boldface notation emphasizes that we require the bias terms at least in the first layer. In other words, we keep randomly updating the random switching layer $h_r^{(l)}(x)$, and couple it with a linearly adjustable hidden layer $h_w^{(l)}(x)W^{(l)}$ to obtain the next semi-random hidden layer $h_w^{(l+1)}(x)$.

Convolutional semi-random feedforward neural networks can be defined in the same way as in equation (6) with vector-matrix multiplication being replaced by c dimensional convolution (for some number c). In our experiments, we will test both convolutional semi-random networks as well as fully-connected versions. We will discuss further generalizations of our architecture in Appendix.

Benefit of Depth

We first confirm that our multilayer model class

$$\mathcal{F}^s_{n_1,\ldots,n_H} = \{x \mapsto \hat{f}^s_{n_1,\ldots,n_H}(x; w) | w \in \mathbb{R}^{d_w}\}$$

preserves universal approximation ability.

Corollary 4 (Universal approximation with deep model) *Let s be any fixed finite integer and let $\Omega \neq \{0\}$ be any fixed nonempty compact subset of $\mathbb{R}^d$. Then, for any $f \in L^2(\Omega)$, with probability one,*

$$\lim_{n_1,\ldots,n_H\to\infty} \inf_{\hat{f}\in\mathcal{F}^s_{n_1,\ldots,n_H}} \|f - \hat{f}\|_{L^2(\Omega)} = 0.$$

We now know that both of one hidden layer models and deeper models have universal approximation ability. Then, a natural question arises: how can depth benefit us? To answer the question, note that H hidden layer model only needs $O(nH)$ number of parameters (by setting $n = n_1, n_2, \cdots, n_H$) to create around $n^H d$ paths, whereas one hidden layer model requires $O(n^H)$ number of parameters to do so. Intuitively, because of this, the expressive power would grow exponentially in depth H, if those exponential paths are not redundant to each other. The redundancy among the paths would be minimized via randomness in the switching and exponentially many combinations of nonlinearities σ_s.

We formalize this intuition by considering concrete degrees of approximation powers for our models. To do so, we adopt a type of a degree of "smoothness" on the target functions from a previous work (Barron 1993). Consider Fourier representation of a target function as $f(x) = \int_{\omega\in\mathbb{R}^d} \tilde{f}(\omega)e^{i\omega^\top x}$. Define a class of smooth functions Γ_C:

$$\Gamma_C = \left\{x \mapsto f(x) : \int_{\omega\in\mathbb{R}^d} \|\omega\|_2|\tilde{f}(\omega)| \leq C\right\}.$$

Any $f \in \Gamma_C$ with finite C is continuously differentiable in $\mathbb{R}^d$, and the gradient of f can be written as $\nabla_x f(x) = \int_{\omega\in\mathbb{R}^d} i\omega\tilde{f}(\omega)e^{i\omega^\top x}$. Thus, via Plancherel theorem, we can view C as the bound on $\|\nabla_x f(x)\|_{L(\mathbb{R}^d)}$. See the previous work (Barron 1993) for a more detailed discussion on the properties of this function class Γ_C. Theorem 5 states that a lower bound on the approximation power gets better exponentially in depth H.

Theorem 5 (Lower bound on universal approximation power) *Let $\Omega = [0, 1]^d$. For every fixed finite integer s, for any depth $H \geq 0$, and for any set of nonzero widths $\{n_1, n_2, \ldots, n_H\}$,*

$$\sup_{f\in\Gamma_C} \inf_{\hat{f}\in\mathcal{F}^s_{n_1\cdots n_H}} \|f - \hat{f}\|_{L^2(\Omega)} \geq \frac{\kappa C}{d^2}\left(\prod_{l=1}^{H} n_l\right)^{-1/d},$$

where $\kappa \geq (8\pi e^{(\pi-1)})^{-1}$ is a constant.

By setting $n = n_1 = \cdots = n_H$, the lower bound in Theorem 5 becomes: $\sup_{f\in\Gamma_C} \inf_{\hat{f}\in\mathcal{F}^s_{n_1\cdots n_H}} \|f - \hat{f}\|_{L^2(\Omega)} \geq$

$\frac{\kappa C}{d^2} n^{-H/d}$, where we can easily see the benefit of the depth H.

However, this analysis is still preliminary for the purpose of fully understanding the benefit of the depth. Our hope here is to provide a formal statement to aid intuitions. To help our intuitions further, we discuss about an upper bound on universal approximation power for multilayer model in Appendix.

Optimization Theory

Similarly to one hidden layer case, we consider the following optimization problem:

$$\underset{w \in \mathbb{R}^{d_w}}{\text{minimize}}\, \mathcal{L}^{(H)}(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(y_i - f^s_{n_1,\dots,n_H}(x;w) \right)^2 .$$

Compared to one hidden layer case, our theoretical understanding of multilayer model is rather preliminary. Here, given a function $g(w_1, w_2, \dots, w_n)$, we say that $\bar{w} = (\bar{w}_1, \bar{w}_2, \dots, \bar{w}_n)$ is a global minimum of g with respect to w_1 if $\bar{w}_1$ is a global minimum of $\tilde{g}(w_1) = g(w_1, \bar{w}_2, \dots, \bar{w}_n)$.

Corollary 6 (No bad local minima and few bad critical points w.r.t. last two layers) *For any s, any depth $H \geq 1$, and any set of nonzero widths $\{n_1, n_2, \dots, n_H\}$, the optimization problem of $\mathcal{L}^{(H)}(w)$ has the following property:*

(i) every local minimum is a global minimum with respect to $(W^{(H)}, W^{(H+1)})$, and

(ii) if $w_k^{(H+1)} \neq 0$ for all $k \in \{1, 2, \dots, n_H\}$, every critical point is a global minimum with respect to $(W^{(H)}, W^{(H+1)})$.

Future work is still required to investigate the theoretical nature of the optimization problem with respect to all parameters. Some hardness results of a standard neural network optimization come from the difficulty of learning activation pattern via optimization of the variable $\mathbf{r}$ (Livni, Shalev-Shwartz, and Shamir 2014). In this sense, our optimization problem is somewhat easier, and it would be interesting to see if we can establish meaningful optimization theory for semi-random model as a first step to establish the theory for neural networks in general.

Generalization Guarantee

The following corollary bounds the generalization error. In the statement of the corollary, we can easily see that the generalization error goes to zero as m increases (as long as the relevant norms are bounded). Hence, we can achieve generalization. In Corollary 7, we use the following notations: $[\![\sigma, x]\!]_{k_0,k_1,\dots,k_H} = [\![\sigma]\!]_{k_1,\dots,k_H}(x)\mathbf{x}_{k_0}$ and $[\![w]\!]_{k_0,k_1,\dots,k_H} = \left(\prod_{l=1}^{H} w^{(l)}_{k_{l-1}k_l}\right) w^{(H+1)}_{k_H}$. Let $\mathrm{vec}(M)$ be a vectorization of a tensor M.

Corollary 7 (Generalization bound for deep model) *Let $s \geq 0$ and $H \geq 1$ be fixed. Let $n_l > 0$ be fixed for $l = 1, 2, \dots, H$. Consider the model class $\mathcal{F}^s_{n_1,\dots,n_H}$ with $\|\mathrm{vec}([\![w]\!])\|_2 \leq C_W$ and $\|\mathrm{vec}([\![\sigma, x]\!])\|_2 \leq C_{\sigma x}$ almost surely. Then, with probability at least $1-\delta$, for any $\hat{f} \in \mathcal{F}^s_{n_1,\dots,n_H}$,*

$$\frac{1}{2}\mathbb{E}_x(f(x) - \hat{f}(x; w^*))^2 - \mathcal{L}^{(H)}(w)$$
$$\leq (C_Y^2 + C_{\hat{Y}}^2)\sqrt{\frac{\log\frac{1}{\delta}}{2m}} + 2(C_Y + C_{\hat{Y}})\frac{C_{\hat{Y}}}{\sqrt{m}},$$

where $C_{\hat{Y}} = C_W C_{\sigma x}$.

The generalization bound in Corollary 7 can be exponentially better than the known bounds of ReLU; e.g., see (Sun et al. 2016; Neyshabur, Tomioka, and Srebro 2015; Xie, Deng, and Xing 2015). The known generalization upper bounds of ReLU explicitly contain 2^H factor, which comes from ReLU nonlinearity and grows exponentially in depth H. In contrast, the generalization bound in Corollary 7 does not explicitly depend on any of the depth, the number of trainable weights, and the dimensionality of the domain. Note that $f_{n_1,\dots,n_H}(x) = vec([\![w]\!])^\top vec([\![\sigma, x]\!]) = \|vec([\![w]\!])\|_2 \|vec([\![\sigma, x]\!])\|_2 \cos\theta$. Hence, even though the dimensionality of $\mathrm{vec}([\![w]\!])$ and $\mathrm{vec}([\![\sigma, x]\!])$ grows exponentially in the depth, the norm bound $C_W C_{\sigma x}$ would stay near the norm of the output. Indeed, $C_W C_{\sigma x} \approx f_{n_1,\dots,n_H}(x)/\cos\theta$. As a new related work, the generalization bounds of *standard nets trained by a novel two-phase training procedure* in a recent paper (Kawaguchi, Kaelbling, and Bengio 2017) have this desired qualitative property similarly to our generalization bounds of *semi-random nets*.

Experiments

We compare semi-random features with random features (RF) and neural networks with ReLU on both UCI datasets and image classification benchmarks. We will study two variants of semi-random features for $s = 0$ (LSR: linear semi-random features) and $s = 1$ (SSR: squared semi-random features) in $\sigma_s(\cdot)$ from equation (1). Additional experimental details are presented in Appendix. The source code of the proposed method is publicly available at: http://github.com/zixu1986/semi-random.

Simple Test Function

We first tested the methods with a simple sine function, $f(x) = \sin(x)$, where we can easily understand what is happening. Figure 1 shows the test errors with one standard deviations. As we can see, semi-random network (LSR) performed the best. The problem of ReLU became clear once we visualized the function learned at each iteration: ReLU network had a difficulty to diversify activation units to mimic the frequent oscillations of the sine function (i.e., it took long time to diversely allocate the breaking points of its piecewise linear function). The visualizations of learned functions at each iteration for each method are presented in Appendix. On average, they took 54.39 (ReLU), 43.04 (random), 45.44 (semi-random) seconds. Their training errors are presented in Appendix.

UCI datasets

We have comparisons on six large UCI datasets.The network architecture used on this dataset is multi-layer networks with

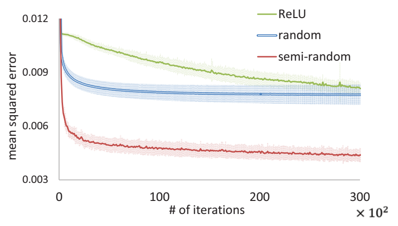


Figure 1: Test error for a simple test function.

Table 1: Performance comparison on UCI datasets. RF for random features, LSR and SSR for linear ($s = 0$) and squared ($s = 1$) semi-random features respectively. m_{tr}: number of training data points; m_{te}: number of test data points; d dimension of the data.

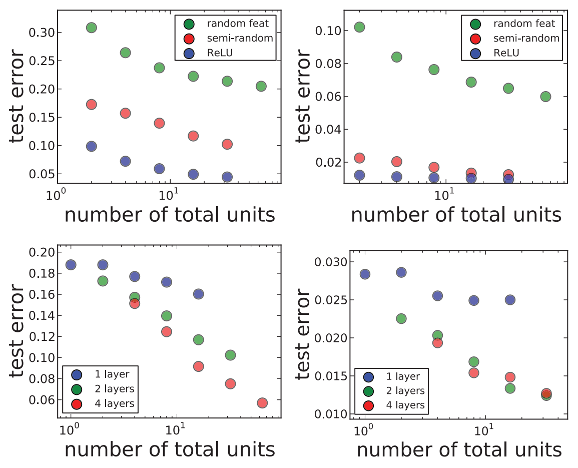
Dataset	method	err (%)	Dataset	method	err (%)
covtype	ReLU	5.6	adult	ReLU	15.0
$m_{\text{tr}} = 522,910$	RF	20.2	$m_{\text{tr}} = 32,561$	RF	14.9
$m_{\text{te}} = 58,102$	LSR	5.7	$m_{\text{te}} = 16,281$	LSR	14.8
$d = 54$	SSR	14.4	$d = 123$	SSR	14.9
webspam	ReLU	1.1	senseit	ReLU	13.6
$m_{\text{tr}} = 280,000$	RF	6.0	$m_{\text{tr}} = 78,823$	RF	16.0
$m_{\text{te}} = 70,000$	LSR	1.1	$m_{\text{te}} = 19,705$	LSR	13.9
$d = 123$	SSR	2.2	$d = 100$	SSR	13.3
letter	ReLU	13.1	sensor	ReLU	2.0
$m_{\text{tr}} = 15,000$	RF	14.9	$m_{\text{tr}} = 48,509$	RF	13.4
$m_{\text{tr}} = 5,000$	LSR	6.5	$m_{\text{tr}} = 10,000$	LSR	1.4
$d = 16$	SSR	5.6	$d = 48$	SSR	5.7

$l = [1, 2, 4]$ hidden layers and $k = [1, 2, 4, 8, 16] \times d$ hidden units per layer where d is the input data dimension.

Comparison of best performance. In Table 1, we listed the best performance among different architectures for semi-random and fully random methods. For ReLU, we fix the number of hidden units to be d. On most datasets, semi-random features achieve smaller errors compared with ReLU by using more units. In addition, semi-random units have significant lower errors than random features.

Matching the performance of ReLU. The top row of Figure 2 demonstrates how many more units are required for random and semi-random features to reach the test errors of networks with ReLU. First, all three methods enjoy lower test errors by increasing the number of hidden units. Second, semi-random units can achieve comparable performance to ReLU with slightly more units, around 2 to 4 times in Webspam dataset. In comparison, random features require many more units, more than 16 times. These experiments clearly show the benefit of adaptivity in semi-random features.

Depth vs width. The bottom row of Figure 2 explores the benefit of depth. Here, "l-layer" indicates l *hidden* layer model. To grow the number of total units, we can either use more layers or more units per layer. Experiment results suggest that we can gain more in performance by going deeper. The ability to benefit from deeper architecture is an important feature that is not possessed by random features. The details on how the test error changes w.r.t. the number of layers and number of units per layer are shown in Figure 3. As we can see, on most datasets, more layers and more units lead to smaller test errors. However, the adult dataset is more noisy and it is easier to overfit. All types of neurons perform relatively the same on this dataset, and more parameters actually lead to worse results. Furthermore, the squared semi-random features have very similar error pattern to neural network with ReLU.



(a) Covtype dataset (b) Webspam dataset

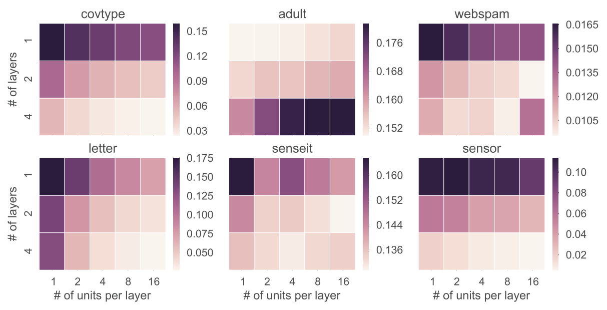
Figure 2: Top row: Linear semi-random features match the performance of ReLU for two hidden layer networks in two datasets. Bottom row: Depth vs width of linear semi-random features. Both plots show performance of semi-random units. Even the total number of units is the same, deeper models achieve lower test error.

Table 2: Test error (in %) of different methods on three image classification benchmark datasets. $2\times$, $4\times$ and $16\times$ mean the number of units used is 2 times, 4 times and 16 times of that used in neural network with ReLU respectively.

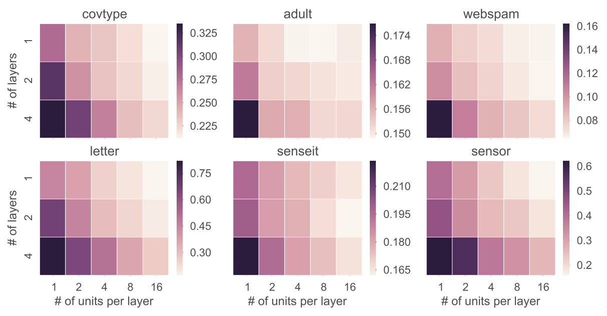
neuron type	MNIST	CIFAR10	SVHN
ReLU	0.70	16.3	3.9
RF	8.80	59.2	73.9
RF $2\times$	5.71	55.8	70.5
RF $4\times$	4.10	49.8	58.4
RF $16\times$	2.69	40.7	37.1
LSR	0.97	21.4	7.6
LSR $2\times$	0.78	17.4	6.9
LSR $4\times$	0.71	18.7	6.4
LSR-IE	0.59	20.0	6.9
LSR-IE $2\times$	$\underline{0.47}$	16.8	5.9
LSR-IE $4\times$	0.54	$\underline{14.9}$	$\underline{4.8}$

Image classification benchmarks

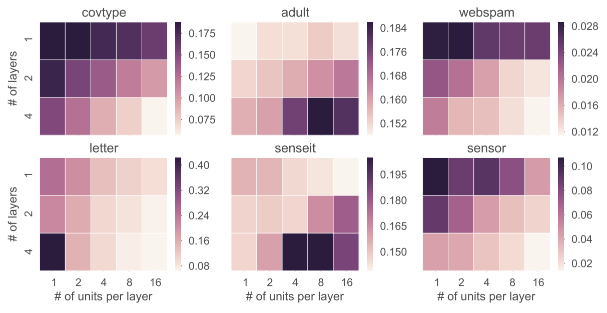
We have also compared different methods on three image classification benchmark datasets. Here we use publicly



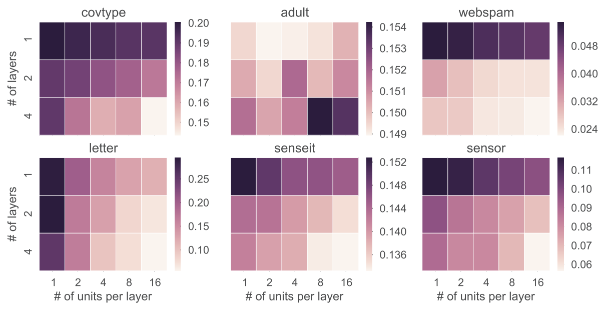
(a) ReLU



(b) random features



(c) linear semi-random features



(d) squared semi-random features

Figure 3: Detailed experiment results for all types of neurons and on all datasets. The heat map for each dataset shows how the test error changes w.r.t. the number of layers and number of units per layer.

available and well-tuned neural network architectures from tensorflow for the experiments. We simply replace ReLU by random and semi-random units respectively. The results are summarized in Table 2.

LSR-IE Unit. To improve the practical performance of LSR unit while preserving its theoretical properties, we additionally introduced a new unit, called linear semi-random *implicit*-ensemble (LSR-IE) unit. Unlike an *explicit* ensemble, LSR-IE trains only a single network and comparable to a single standard network with dropout. Please refer to our latest version for more details on LSR-IE: https://arxiv.org/abs/1702.08882.

MNIST dataset. MNIST is a popular dataset for recognizing handwritten digits. It contains 28×28 grey images, 60,000 for training and 10,000 for test. We use a convolution neural network consisting of two convolution layers, with 5×5 filters and the number of channels is 32 and 64, respectively. Each convolution is followed by a max-pooling layer, then finally a fully-connected layer of 512 units with 0.5 dropout. Increasing the number of units for semi-random leads to better performance. At four times the size of the original network, semi-random feature can achieve very close errors of 0.71%. In contrast, even when increasing the number of units to 16 times more, random features still cannot reach below 1%.

CIFAR10 dataset. CIFAR 10 contains internet images and consists of 50,000 32×32 color images for training and 10,000 images for test. We use a convolutional neural network architecture with two convolution layers, each with 64 5×5 filters and followed by max-pooling. The fully-connected layers contain 384 and 192 units. By using two times more units, semi-random features are able to achieve similar performance with ReLU. However, the performance of random features lags behind by a huge margin.

SVHN dataset. The Street View House Numbers (SVHN) dataset contains house digits collected by Google Street View. We use the 32×32 color images version and only predict the digits in the middle of the image. For training, we combined the training set and the extra set to get a dataset with 604,388 images. We use the same architecture as in the CIFAR10 experiments.

Conclusion

In this paper, we proposed the method of semi-random features. For one hidden layer model, we proved that our model class contains an arbitrarily good function as the width increases (universality), and we can find such a good function (optimization theory) that generalizes to unseen new data (generalization bound). For deep model, we proved universal approximation ability, a lower bound on approximation error, a partial optimization guarantee, and a generalization bound. Furthermore, we demonstrated the advantage of semi-random features over fully-random features via empirical results and theoretical insights.

The idea of semi-random feature itself is more general than what is explored in this paper, and it opens up several intersecting directions for future work. Indeed, we can generalize any deep architecture by having an option to include *semi-random units* per unit level. We also can define a more general semi-random feature as: given some nonconstant functions σ and g, $\phi(x;\mathbf{r},\mathbf{w}) = \sigma(\mathbf{x}^\top \mathbf{r})g\left(\mathbf{x}^\top \mathbf{w}\right)$, where $\mathbf{x} = (1, x)$ is assumed to be in $\mathbb{R}^{1+d}$, $\mathbf{r} = (r_0, r)$ is sampled randomly, and $\mathbf{w} = (w_0, w)$ is adjustable weights to be learned from data. This general formulation would lead to a flexibility to balance expressivity, generalization and theoretical tractability.

References

Barron, A. R. 1993. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory* 39(3):930–945.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Choromanska, A.; LeCun, Y.; and Arous, G. B. 2015. Open problem: The landscape of the loss surfaces of multilayer networks. In *Proceedings of The 28th Conference on Learning Theory*, 1756–1760.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009*, 248–255. IEEE.

Freeman, C. D., and Bruna, J. 2016. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*.

Huang, G.-B.; Chen, L.; Siew, C. K.; et al. 2006. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks* 17(4):879–892.

Kawaguchi, K.; Kaelbling, L. P.; and Bengio, Y. 2017. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.

Kawaguchi, K. 2016. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Leshno, M.; Lin, V. Y.; Pinkus, A.; and Schocken, S. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks* 6(6):861–867.

Livni, R.; Shalev-Shwartz, S.; and Shamir, O. 2014. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, 855–863.

Mohri, M.; Rostamizadeh, A.; and Talwalkar, A. 2012. *Foundations of machine learning*. MIT press.

Neyshabur, B.; Tomioka, R.; and Srebro, N. 2015. Norm-based capacity control in neural networks. In *Proceedings of The 28th Conference on Learning Theory*, 1376–1401.

Pennington, J.; Yu, F.; and Kumar, S. 2015. Spherical random features for polynomial kernels. In *Advances in Neural Information Processing Systems*, 1846–1854.

Rahimi, A., and Recht, B. 2008. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 1177–1184.

Rahimi, A., and Recht, B. 2009. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, 1313–1320.

Scholkopf, B., and Smola, A. J. 2001. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Shamir, O. 2016. Distribution-specific hardness of learning neural networks. *arXiv preprint arXiv:1609.01037*.

Shawe-Taylor, J., and Cristianini, N. 2004. *Kernel methods for pattern analysis*. Cambridge university press.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sindhwani, V.; Avron, H.; and Mahoney, M. W. 2014. Quasi-monte carlo feature maps for shift-invariant kernels. In *International Conference on Machine Learning*.

Sun, S.; Chen, W.; Wang, L.; Liu, X.; and Liu, T.-Y. 2016. On the depth of deep neural networks: a theoretical view. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2066–2072. AAAI Press.

Swirszcz, G.; Czarnecki, W. M.; and Pascanu, R. 2016. Local minima in training of deep networks. *arXiv preprint arXiv:1611.06310*.

Telgarsky, M. 2016. Benefits of depth in neural networks. In *29th Annual Conference on Learning Theory*, 1517–1539.

Xie, P.; Deng, Y.; and Xing, E. 2015. On the generalization error bounds of neural networks under diversity-inducing mutual angular regularization. *arXiv preprint arXiv:1511.07110*.

Xie, B.; Liang, Y.; and Song, L. 2016. Diversity leads to generalization in neural networks. *arXiv preprint arXiv:1611.03131*.