Kernel Learning via Random Fourier Representations

Andi Wang
Department of Statistics
University of Oxford
andi.wang@spc.ox.ac.uk

Leon Law
Department of Statistics
University of Oxford
ho.law@spc.ox.ac.uk

Xenia Miscouridou
Department of Statistics
University of Oxford
xenia.miscouridou@spc.ox.ac.uk

Marcin Mider
Department of Statistics
University of Warwick
marcin.mider@spc.ox.ac.uk

Sherman Ip
Department of Statistics
University of Warwick
sherman.ip@spc.ox.ac.uk

Abstract

Random Fourier Features (RFF) is a kernel approximation method that works well for large datasets, however it makes certain assumptions about the choice of the kernel and so here in this report, we present and validate an alternative method based on fitting neural networks. Our alternative method consists of training the neural network with one hidden layer- approximating the frequencies from Fourier expansion of some translation invariant kernel in a data dependent fashion. The predictive accuracy is comparable to the one from the RFF algorithm, achieving however a lower variance. We then applied a similar idea of fitting neural networks using this time two hidden layers and implementing mean pooling to the two-stage sampled learning. We achieved again similar predictive accuracy, but with reduced variance. However, both methods involve significant computational cost. Furthermore, we have explored various optimisation methods and tested them in the neural networks.

1 Introduction

Kernel methods allows one to work in a high-dimensional feature space, possibly infinite-dimensional, without explicitly working in them. It however comes with an expensive computational cost, especially when the data set is large. Random Fourier Features (RFF) [3] deals with this problem by taking some pre-specified translation invariant kernel and approximates it by a finite dimensional explicit feature map, which is usually computational more efficient to work on. The problem with this is that the choice of the kernel has to be made and choosing a good kernel is still a challenging and open question. To deal with this, we present a method using a one hidden layer neural network, that optimises the parameters of a non-linear map (as given in RFF) directly in a data-dependent fashion, which can be implicitly seen as learning some 'optimal' kernel. Further, we also proposed a method using a two hidden layer neural network and mean pooling on a two-staged sampled learning problem.

As a quick outline of the paper, in Section 2 of this report the background theory of kernels and relevant techniques used are discussed; we aim to give some general intuition into the basic theory of kernels themselves, culminating with the Representer Theorem, and then briefly describe random Fourier representations and learning theory on distributions. In Section 3, we discuss various optimisation methods for neural networks, such as gradient descent, stochastic gradient descent and Quasi-Newton and compare their performance. In Section 4, we using the Adult and Aerosol Data Set to compare and contrast our alternative method for one and two hidden layer with mean pooling versus the RFF algorithm with a RBF kernel, before ending with Conclusion.

2 Background Theory

In the classic learning problem, we have data (x_i, y_i) for $i = 1, \dots, n$, where for each i, $x_i \in \mathcal{X}$, $\mathcal{X}$ being a non-empty set, and for simplicity we'll assume $y_i \in \mathbb{R}$. Note that aside from being non-empty, no assumptions are placed on $\mathcal{X}$; for instance it could be a space of documents. It is natural then to look at the space of functions $f : \mathcal{X} \to \mathbb{R}$ to study the relationship between the inputs x_i and outputs y_i; we want to find an f where $f(x_i) \approx y_i$. Formally, for a loss function $L : (\mathcal{X} \times \mathbb{R} \times \mathbb{R})^n \to \mathbb{R} \cup \{\infty\}$ we seek

$$\operatorname*{argmin}_{f \in \mathcal{H}_k} L\big((x_1, y_1, f(x_1)), \dots, (x_n, y_n, f(x_n))\big) + \Omega\big(\|f\|^2_{\mathcal{H}_k}\big) \tag{1}$$

where $\mathcal{H}_k$ is a Hilbert space of functions mapping $\mathcal{X} \to \mathbb{R}$, which we will construct, and $\Omega : [0, \infty) \to \mathbb{R}$ is an increasing function.

2.1 Kernels

In order to tackle this problem, we first need to impose some structure on $\mathcal{X}$. One way to do this is with a *feature map*, $\phi : \mathcal{X} \to \mathcal{H}$, where $\mathcal{H}$ is a space with lots of structure. Mathematically, we know that a convenient class of spaces to work with are Hilbert spaces. In this case, a natural measure of similarity between points $x, y \in \mathcal{X}$ is $k(x, y) := \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$. We call such a $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a *kernel*. Such a kernel is automatically symmetric and *positive semidefinite*[1]. Remarkably, the Moore-Aronszajn Theorem states that *every* positive semidefinite function k is a kernel for some Hilbert space $\mathcal{H}$ [1]. So, we can start with a positive semidefinite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, appropriate for the problem at hand, and then try and construct the appropriate Hilbert space $\mathcal{H}_k$.

Given the motivation of (1), we want to build our Hilbert space $\mathcal{H}_k$ within the space of functions $f : \mathcal{X} \to \mathbb{R}$. From our kernel k, a collection of functions which are natural to include are the functions $k(\cdot, x) : \mathcal{X} \to \mathbb{R}$, for each fixed $x \in \mathcal{X}$. Since a Hilbert space is a vector space, we will need to include all linear combinations; functions f of the form

$$f(\cdot) = \sum_{i=1}^{n} a_i k(\cdot, x_i)$$

for $n \in \mathbb{N}, (a_1, \dots, a_n) \in \mathbb{R}^n, (x_1, \dots, x_n) \in \mathcal{X}^n$. We also require an inner product, which we will define as follows, for $g(\cdot) = \sum_{j=1}^{m} b_j k(\cdot, y_i)$,

$$\langle f, g \rangle_{\mathcal{H}_k} := \sum_{i=1}^{n} \sum_{j=1}^{m} a_i b_j k(x_i, y_j).$$

We can check this is well-defined (i.e. it does not depend on the particular representations of f and g) and that it satisfies the axioms of an inner product. These follow from our requirement that k is positive semidefinite.

We also note that k has the *reproducing property*: from the definition of our inner product, for f as above and $x \in \mathcal{X}$, $\langle k(\cdot, x), f \rangle_{\mathcal{H}_k} = f(x)$, and in particular, $k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle_{\mathcal{H}_k}$. This last equation demonstrates the Moore-Aronszajn theorem; the explicit feature map ϕ *is* $x \mapsto k(\cdot, x)$. Skipping the technical details, we can *complete* this vector space with respect to the norm associated with our inner product to form a bona fide Hilbert space $\mathcal{H}_k$, which we call the *reproducing kernel Hilbert space* (RKHS) with *reproducing kernel* k[2].

So we have now constructed a very pleasing space $\mathcal{H}_k$; how does this help us with our original learning problem (1)? Fortunately, we have the following theorem.

Theorem 1 *(Representer Theorem) In the setting of (1), there exists a minimiser $f \in \mathcal{H}_k$ of the form*

$$f(\cdot) = \sum_{i=1}^{n} a_i k(\cdot, x_i)$$

for some $(a_1, \dots, a_n) \in \mathbb{R}^n$. If Ω is strictly increasing, then each minimiser of the regularised risk admits such a representation.

[1]In the sense that $\forall n \in \mathbb{N}, \forall (a_1, \dots, a_n) \in \mathbb{R}^n, \forall (x_1, \dots, x_n) \in \mathcal{X}^n$, $\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0$.

[2]We can *define* a RKHS to be a Hilbert space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathbb{R}$ such that $\forall x \in \mathcal{X}$ the evaluation maps $\delta_x : \mathcal{H} \to \mathbb{R}, f \mapsto f(x)$ are bounded (that is, continuous) [2]. From our construction this is immediate by the reproducing property of k and the Cauchy-Schwarz inequality. Conversely, from this definition the existence of a kernel k with its reproducing property follows from the *Riesz representation theorem*. In particular, this definition requires of course that the evaluation maps are well-defined, so for instance the $L^2(\mu)$ spaces are *not* RKHSs.

Note that we have reduced the optimisation from an infinite-dimensional problem (over the infinite-dimensional space $\mathcal{H}_k$) to the finite-dimensional problem of optimising $(a_1, \dots, a_n)$ over $\mathbb{R}^n$. Also note that the $k(\cdot, x_i)$ are not arbitrary, but correspond to our input data points x_i.

2.2 Random Fourier Representations

We now move from a general discussion of kernels to the main topic of our project: random Fourier representations. We say that a kernel $k : V \times V \to \mathbb{R}$ defined on a vector space V is *translation-invariant* if there exists a $h : V \to \mathbb{R}$ such that $k(x, y) = h(x - y), \forall x, y \in V$. For brevity we will write $k(x - y)$.

Theorem 2 *(Bochner's Theorem) [3] A continuous, translation-invariant kernel k on $\mathbb{R}^d$ is the Fourier transform of a non-negative measure.*

If k is suitably scaled, then we can take the measure $p(\omega)$ to be a probability measure:

$$k(x - y) = \int_{\mathbb{R}^d} e^{i\,\omega^T(x-y)}\, p(\omega) d\omega = \mathbb{E}_{\omega \sim p}[\zeta_\omega(x)\zeta_\omega(y)^*]$$

where $\zeta_\omega(x) := e^{i\,\omega^T x}$. Assuming we can simulate from $p(\omega)$, one way to estimate k is to draw independent samples $\omega_1, \dots, \omega_m$ from p and compute

$$\hat{k}(x, y) = \frac{1}{m} \sum_{j=1}^{m} \left[\cos(\omega_j^T x) \cos(\omega_j^T y) + \sin(\omega_j^T x) \sin(\omega_j^T y) \right]. \tag{2}$$

This form suggests we have an explicit finite dimensional feature map, allowing us to potentially save computational power.

$$\phi(x) := \sqrt{\frac{1}{m}} \left[\cos\left(\omega_1^\top x\right), \sin\left(\omega_1^\top x\right) \dots, \cos\left(\omega_m^\top x\right), \sin\left(\omega_m^\top x\right) \right] \in \mathbb{R}^{2m}. \tag{3}$$

In [3] the authors proved uniform convergence in probability on compacts, exponentially as m grows.

We will later approach this from the other direction: by considering (2) to be a collection of kernels parameterised by the m frequencies, and fit a two-layer neural net to optimally choose a kernel, i.e. a set of frequencies.

2.3 Learning on Distributions

In [4] the authors discuss the problem of distribution regression, from a sample: now for each output y_i we have a sample $x_{i,1}, \dots, x_{i,N} \overset{iid}{\sim} x_i$ where x_i is a distribution. The strategy in [4] is to use two layers of kernels: first, the empirical distributions $\hat{x}_i$ corresponding to the $\{x_{i,j}\}_{j=1}^N$ are embedded via a kernel k to $\mu_{\hat{x}_i} = \mathbb{E}_{u \sim \hat{x}_i}[k(\cdot, u)]$. The $\{\mu_{\hat{x}_i}\}_{i=1}^n$ are now viewed as new inputs, and the authors define another kernel K on the space of mean-embedded distributions $X = \{\mu_x := \mathbb{E}_{u \sim x}[k(\cdot, u)] : x \text{ a probability distribution}\} \subseteq \mathcal{H}_k$. By assuming both kernels are translation-invariant we would hope that we can approximate them using random Fourier features as in the preceding section.

For the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ this is fine as we are assuming our input space $\mathcal{X} = \mathbb{R}^d$, for some d. However, for $K : X \times X \to \mathbb{R}$ naively applying the previous section is not immediately straightforward, since X is generally infinite-dimensional, it is not immediately clear if Bochner's theorem even holds, i.e. whether there exists a distribution p on X such that for any probability distributions a, b on $\mathcal{X}$

$$K(\mu_a - \mu_b) = \int_X e^{i\langle \omega, \mu_a - \mu_b \rangle_{\mathcal{H}_k}}\, p(\omega) d\omega.$$

And even if Bochner's theorem were to hold in this case, how would we identify p and sample from it? An interesting question perhaps would be whether or not Gaussian processes, say, could be applied here.

Instead, once we have approximated the first layer, k by random Fourier features $\hat{k}$ as in (2), we are now dealing with objects of the form $\frac{1}{N} \sum_{j=1}^N \hat{k}(\cdot, x_{ij})$, which live in a *finite-dimensional space* parameterised by the frequencies $\omega_1, \dots, \omega_m$. Now we can simply consider the kernel K 'restricted' to this finite-dimensional space, and approximate it by random Fourier features as before.

3 Optimisation Methods

To fit the random Fourier representation model onto data, the objective, T, given below is to be minimized

$$T = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 + \lambda\|\beta\|^2 + \mu\|\Omega\|^2 \tag{4}$$

where

$$z(x_i) = \sqrt{\frac{1}{m}}\left[\sum_{j=1}^{m}\beta_j^{\cos}\cos(w_j^{\mathrm{T}}x_i) + \sum_{j=1}^{m}\beta_j^{\sin}\sin(w_j^{\mathrm{T}}x_i)\right] \tag{5}$$

$$\hat{y}_i = S(z(x_i)) \tag{6}$$

S is the activation function, β is a vector with elements $\{\beta_j^{\cos}, \beta_j^{\sin}\}_{j=1}^{m}$, Ω is a matrix with columns $\{w_j\}_{j=1}^{m}$ and λ, μ are tuning parameters. The objective cannot be assumed to be convex and may have a number of local optima. It can be shown that the gradient of the objective with respect to β and Ω can be calculated in closed form, which will be useful for methods in optimization. For exact detail of the whole algorithm, see Appendix.

There are a number of methods to optimize the objective. Methods such as the Nelder-Mead simplex method, trust region algorithm and Quasi-Newton algorithm come packaged with high-level programming languages, such as *R* and *MATLAB*. In particular, the trust region algorithm requires the gradient of the objective.

Gradient descent is a simpler method for optimization, where the parameters are updated according to the gradient of the objective evaluated at the current value of parameters. That is the update step is

$$\beta \leftarrow \beta - \epsilon\nabla_\beta T \tag{7}$$

and

$$\Omega \leftarrow \Omega - \epsilon D_\Omega T \tag{8}$$

where ϵ is the step size and is user defined. The initial value of the parameters are also user defined.

Stochastic gradient descent is similar to gradient descent, however the evaluation of the gradient only uses one data point at a time. That is let

$$T_i = \left(y_i - \hat{y}_i\right)^2 + \lambda\|\beta\|^2 + \mu\|\Omega\|^2 \tag{9}$$

so that in the ith iteration the update step is

$$\beta \leftarrow \beta - \epsilon\nabla_\beta T_{\text{mod}(i-1,n)+1} \tag{10}$$

and

$$\Omega \leftarrow \Omega - \epsilon D_\Omega T_{\text{mod}(i-1,n)+1}\ . \tag{11}$$

By doing so, this enables the parameter space to be explored more. It should also be faster as less calculations are performed in each iteration,

To give some idea of the performance of the optimization algorithms, the random Fourier representation model was fitted onto a small binary classification dataset *SAheart* with $n = 462$ samples and $p = 9$ features with different number of nodes in the hidden layer, m and the tuning parameters set to zero. The sigmoid function was used as the activation function. The optimization was repeated 48 times with different initial values.

It was found that gradient descent performed much faster than all of the optimization packages such as the Quasi-Newton algorithm. This can be seen from Figure 1, for $m = 64$ the Quasi-Newton algorithm took (176 ± 4) s to complete whereas gradient descent took (53.3 ± 0.7) s. The fastest was stochastic gradient descent which took (27 ± 8) s.

Another observation was that the spread of the training error was bigger for the stochastic version of gradient decent, most likely because of the exploration nature of stochastic gradient descent. However this did not affect the test error thus the performance of the classifier.

In conclusion from the empirical experiment, the fast speed and good performance of stochastic gradient descent makes it a good candidate for optimizing the objective in order to fit the random Fourier representation model onto datasets. The evidence from the experiment will be better if the experiment was conducted on more and larger datasets.

4 Experiments

4.1 Description

In this project we have first derived the backpropagation algorithm for the following architecture given in (6) where $s(\cdot)$ is the activation function to be chosen later.

We used stochastic gradient descent to fit the weights $\{\beta_j^{\cos}, \beta_j^{\sin}, w_j\}_{j=1}^m$ and produce the outputs. We then compared the obtained results to the case of randomly selected frequencies $\{w_j\}_{j=1}^m$ from a Gaussian spectral measure corresponding to approximating a Gaussian RBF kernel. In the latter we only need to estimate the weights β and we do that by formulating a least squares problem with a Ridge penalty. Furthermore we considered a three layer extension. We applied these methods to the data used by experiments in Rahimi and Recht. [3]

4.2 Adult Data Set

The first application we considered uses the Adult data set from the Machine learning repository. This dataset includes 48842 observations which we split into training and test data in a ratio 2:1. There are 108 features per observation including both binary and continuous variables. Note here that we have transformed a every k-categorical variable to k binary ones. The goal is to use these variables to predict the salary of individuals and in particular to classify it to be less or greater than 50000 gbp annually. To this end we ran both a neural network with two layers and a RFF algorithm aiming to compare their output and performance.

4.2.1 Neural Network

For the neural net we computed the fitted values by implementing stochastic gradient descent to minimise the following objective function with respect to β and w:

$$R(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \lambda\|\beta\|_2^2 + \lambda\|\Omega\|_F^2$$

where $L(\cdot,\cdot)$ is the loss function and λ is the penalisation parameter for the $L-2$ and the Frobenius norm respectively. We have created a Class for a Network in python that can train a neural network for any number of hidden layers. Note here that we augment the data X so that to include a bias term in the first layer. We experimented with $\lambda \in \{0.01, 0.01, 0.05, 0.1, 0.5, 1.5\}$, number of layers $\in \{20, 30, 50, 100, 200\}$, step size $\in \{0.001, 0.01, 0.1, 1\}$ and the best accuracy was found to be 15.5% misclassifications and it was obtained with 100 layers, step size of 0.1 and penalty parameter $\lambda = 1.5$. We used the sigmoid function as an activation function $s(\cdot)$ in the output layer and a quadratic loss.

4.2.2 RFF

For the random fourier features case we simulate frequencies $\{w_j\}_{j=1}^m$ from a positive definite shift invariant kernel which we choose to be the RBF kernel and compute the transformed data in (5) and concentrate them all in a matrix Z. We then use Ridge regression to find the fitted weights $\{\beta_j^{\cos}, \beta_j^{\sin}\}_{j=1}^m$ by minimising the following objective function with respect to β:

$$R(\theta) = \|Z^T\beta - y\|_2^2 + \lambda\|\beta\|_2^2$$

We experimented with the following values for the variables: number of layers $\in \{50, 100, 200, 300, 500\}$, $\lambda \in \{0.01, 0.05, 0.5, 1, 1.5\}$. We also had σ^2 which is the variance related to the Gaussian RBF kernel used to sample the frequencies. We chose σ^2 to be 0.5 taking into account that the empirical covariances of the fitted frequencies in the neural network implementation had an average of 0.5. In this case the misclassification percentage was 14.7%.

4.3 Discussion

In our experiments we used RBF kernels for the Random Fourier Features algorithm. In order to approximate these one needs to draw the frequencies at random from the Gaussian distribution with covariance matrix proportional to the identity matrix. By fitting the frequencies with the aid of neural nets we allow ourselves to explore a larger family kernels - those which are still translation invariant, but whose covariances need not be as regular as RBF. One would expect that by extending the search area to a larger family of kernels one should achieve better predictive accuracy - that's why neural nets should in principle achieve better results, as it should somehow learn the 'optimal' kernel through the data.

The covariance matrix should be somehow informative about the reason for the improvement in predictive accuracy. It is therefore of interest to compare the covariance matrix of the frequencies generated at random for the Random Fourier Features to the covariance matrix of the frequencies fitted by the neural net algorithm.

For the structure of the neural nets with 108 dimensional inputs (Adult dataset) and D dimensional first layer (or D samples of frequencies in RFF) we dealt with 108×108 covariance matrix. Examining covariance directly revealed that the diagonal elements are on average $10 - 15$ times larger than the other entries in the matrix. The entries on the diagonal were mostly fluctuating around similar values of 0.3-0.4 (for the particular parameters used in the exercise).

In order to visualise the covariance, we plotted the PCA of the frequencies. This is a reduction of dimension from 108 to 2, so it is expected that the projection ignores most of the variance. Nonetheless, it still has a potential to reveal some sort of skewness that would not be approximated by the RFF covariance, but that might be giving an edge in prediction.

As it turned out the PCA was nearly spherical, as can be seen in Figure 2, which is exactly what one would expect from the n-dimensional data with spherical covariance (this is why RFF looks that way as well, as can be seen in Figure 3). One possibility for this behaviour is that because the weights are initialized using standard Gaussian distribution they might be getting stuck at some local optimum near the initial values. However, the loss function and predictive accuracy can be tracked during the learning process of the neural nets and they directly show that it is not the case, as the algorithm is learning over time and the loss function goes down considerably.

On one hand this is quite surprising as it tells us, that RBF might be the near-optimal kernel to use in this case, but on the other it at least explains why we cannot get the (significantly) better predictive accuracy when using the neural nets to train the model.

4.4 Aerosol Data Set

The aerosol dataset involves 800 bags of 100 observations per bag. In this case we treat the problem in two ways again. In the first we consider a multi-instance problem, where for each output y_i, there is a whole bag of B inputs assumed to arise from a probability distribution P_i. We represent bag i by the corresponding empirical kernel embedding $\mu_i = \frac{1}{B}\sum_{a=1}^{B} k(\cdot, x_{ia})$ with respect to a fixed kernel k and then consider the $\mu_i's$ to be inputs of a new problem where we define another kernel K. We will compare this method with the RFF approach in which we make a kernel choice on a multi-instance regression for aerosol prediction.

4.4.1 Neural Network

The parameters for the three layer neural network are the number of layers D_1 and D_2 respectively, the step size and the penalising parameter λ. The loss function is quadratic loss and the activation is identity function. We tried values of step size $\in \{0.01, 0.1, 1\}$, $\lambda \in \{0.05, 0.5, 1.5\}$. For the layer sizes using values of 100 or 200 overfits the neural network . Having in mind that the training data has 700 observations, a more reasonable layer size is around 50 which results in 50×50 combinations. We measure the accuracy by RMSE which is the root of mean squared error an with $\lambda = 0.05$, step size $= 0.1$ values we obtain RMSE $= 0.091$.

4.4.2 RFF

For the random Fourier features we sample the frequencies twice using in both cases a Gaussian kernel . The parameters used in this case are the penalising parameter λ, the layer sizes D_1 and D_2 and σ_1, σ_2 which define the covariance matrix for the two Gaussian kernels from which we sample the frequencies. Using different combinations of values we finalise our model to $\lambda = 0.05$, $D_1 = D_2 = 50$ and $\sigma_1 = \sigma_2 = 0.5$. The RMSE we get is 0.11.

As it turned out, comparing covariance matrices in the case of the two-stage model used on MISR dataset is very similar to the single-stage one used on Adult dataset - i.e. the covariance matrix of the frequencies again resembles the identity matrix multiplied by a constant and PCA seems spherical. It seems that RBF kernels are again near-optimal.

4.5 Conclusions

For both cases we conclude that both methods give similar results. The random Fourier feature, although very simple in structure as it only uses linear ridge regression competes very well the neural network in terms of accuracy. What is more is that RFF is much faster, involves less parameters to be tuned and has a simpler structure making it easier to interpret. However, this happens because the choice of the RBF kernel is very good in this case which is definitely not a universal approach and the advantages from neural network algorithm in ‘learning’ the kernel cannot be ignored.

References

[1] Hofmann, T., Scholkopf, B. and Smola, A.J. (2008) Kernel Methods in Machine Learning, *Annals of Statistics*, Vol. 36, No. 3, 1171–1220.

[2] Gretton, A. (2015) *Advanced Topics in Machine Learning*, course notes, available at `http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/rkhscourse.html`.

[3] Rahimi, A., Recht, B. (2007) Random Features for Large-Scale Kernel Machines, *Advances in Neural Information Processing Systems (NIPS).*

[4] Szabo, Z., Gretton, A., Poczos, B., Sriperumbudur, B.K. (2015) Two-stage Sampled Learning Theory on Distributions, *International Conference on Artificial Intelligence and Statistics (AISTATS).*

5 Appendix

5.1 Notation

- The data are denoted by $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{Z}_{0+}$ or $y \in \mathbb{R}$, depending on a problem at hand.
- **Layers** of the Neural Network run through: $l = 0, 1, \ldots, L$. Layer (0) is an *artificial* layer whose output are just the input data $x_i = x_i^{(0)}$. Layer (L) (in our case L=2) is an output layer,whose output are the fitted values/predicted class probabilities. Layers $l = 1, \ldots, L-1$ are called the hidden layers.
- $w_{i,j}^{(l)}$ - **weight** for the connection between the i^{th} unit in layer $(l+1)$ with the j^{th} unit in layer (l).
- $b_i^{(l)}$ - **bias** for the i^{th} unit in layer (l).
- $m^{(l)}$ - number of units in layer (l). In general, for our setup we will have $2m$ units in hidden layer, but the weights will be shared among unit pairs, numbered $2i$ and $2i+1$, for $i = 0, \ldots m-1$.
- $\sigma(\cdot)$, $s(\cdot)$, $c(\cdot)$ - generic, sine and cosine **activation functions** respectively.
- $x_i^{(l)} = \sigma(x_i^{(l-1)T} w_i^{(l-1)} + b_i^{(l)})$ - **activation** i.e. output of the i^{th} unit in the $(l)^{th}$ layer.
- $z_i^{(l)} = x_i^{(l-1)T} w_i^{(l-1)} + b_i^{(l)}$ - **score** of the i^{th} unit in the $(l)^{th}$ layer.
- J - **loss** function.

5.2 Neural Network Algorithm

The following steps will be performed for each data point (x_i, y_i):
First, feed forward x_i's through our network to produce $z_{(l)}$ and $x_{(l)}$, for $l = 1, \ldots, L$. Then backpropagate the information from the loss function. To do this we need to know the quantities: $\frac{\partial J}{\partial w_{ij}^{(l)}}$ and $\frac{\partial J}{\partial b_i^{(l)}}$, for all i, j, l. We calculate them separately for the last layer and other layers. Consider the final layer (L) first.

$$\frac{\partial J}{\partial w_{ij}^{(L-1)}} = \frac{\partial J}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial w_{ij}^{(L-1)}} = \frac{\partial J}{\partial z_i^{(L)}} (x^{(L-1)})_j := \delta_i^{(L)} (x^{(L-1)})_j, \tag{12}$$

so if we could calculate a vector $\delta^{(L)}$, whose i^{th} entry is $\frac{\partial J}{\partial z_i^{(L)}}$ (and we can), then we can concisely write:

$$\frac{\partial J}{\partial w^{(L-1)}} = \delta^{(L)} x^{(L-1)T}. \tag{13}$$

There is no term $\frac{\partial J}{\partial b_i^{(L)}}$, as there is no bias in the final layer. Updating the hidden layers is only slightly more complicated (and it is complicated still a bit further due to the dual nature of our units) and update of the first hidden layer is again slightly different due to presence of a bias term. Thus, for $l = L-1, \ldots, 1$:

$$\begin{aligned}\delta_i^{(l)} :=& \frac{\partial J}{\partial z_i^{(l)}} = \sum_k \frac{\partial J}{\partial z_k^{(l+1)}} \left(\frac{\partial z_k^{(l+1)}}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial z_i^{(l)}} + \frac{\partial z_k^{(l+1)}}{\partial x_{(i+m)}^{(l)}} \frac{\partial x_{(i+m)}^{(l)}}{\partial z_i^{(l)}} \right) = \\ =& \left(\sum_k \delta_k^{(l+1)} (w_{ki}^{(l)} + w_{k(i+m)}^{(l)}) \right) \sigma'(z_i^{(l)}),\end{aligned} \tag{14}$$

(notice that we assume each unit i, $i = 1, \dots, m$ produces scores with indices i and $i + m$) which gives:

$$\frac{\partial J}{\partial w_{ij}^{(l-1)}} = \frac{\partial J}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l-1)}} = \delta_i^{(l)} (x^{(l-1)})_j = \left(\sum_k \delta_k^{(l+1)} (w_{ki}^{(l)} + w_{k(i+m)}^{(l)}) \right) \sigma'(z_i^{(l)}) (x^{(l-1)})_j, \tag{15}$$

and this can be written concisely as:

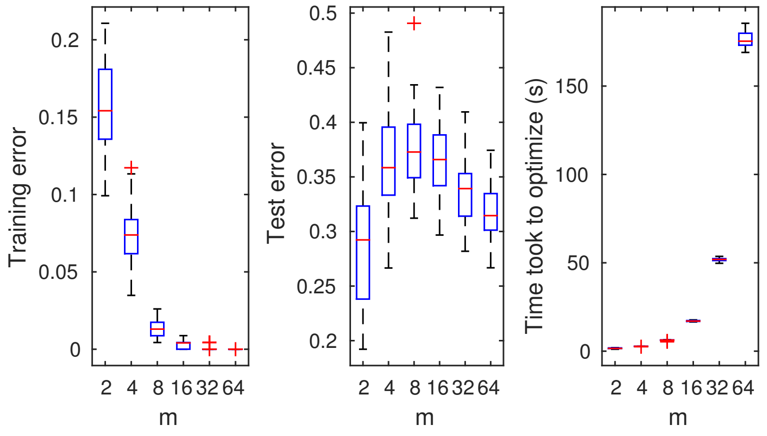
$$\frac{\partial J}{\partial w^{(l-1)}} = \delta^{(l)} x^{(l-1)T} = \left(\left((w_1^{(l)T} + w_2^{(l)T}) \delta^{(l+1)} \right) \cdot \sigma'(z^{(l)}) \right) x^{(l-1)T}. \tag{16}$$

where '$\cdot$' is an element-wise multiplication and $w_1^{(l)T}$ and $w_2^{(l)T}$ are top and bottom halves respectively of $w^{(l)T}$. The bias terms are still not present until the first hidden layer. It is easily seen that:

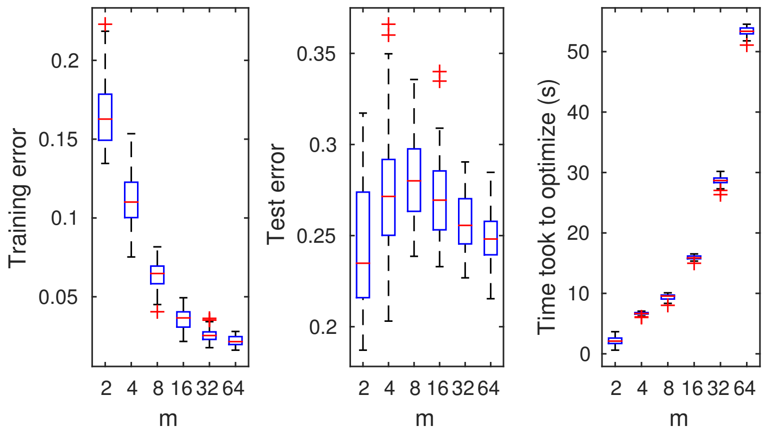
$$\frac{\partial J}{\partial b^{(1)}} = \delta^{(2)}. \tag{17}$$

One can derive the algorithm for the extended neural net with mean pooling in a similar fashion.

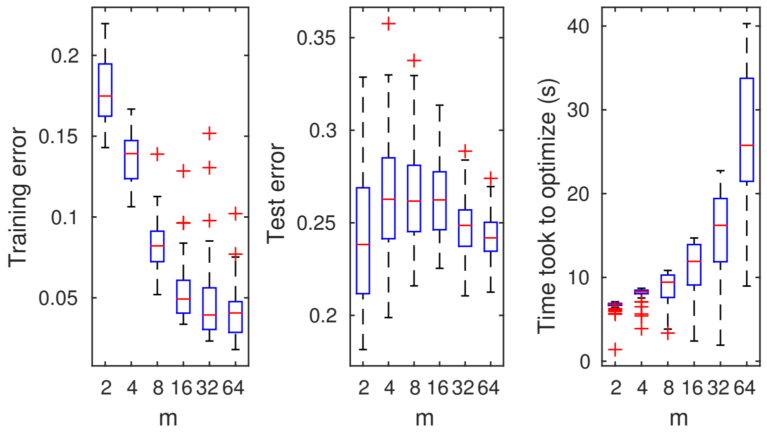
5.3 Graphs



(a) Quasi-Newton



(b) Gradient Descent, step size = 0.1



(c) Stochastic Gradient Descent, step size = 0.1

Figure 1: Training error, test error and time took to optimize the objective for fitting the random Fourier representations onto the dataset *SAheart*. The experiments were repeated 48 times with different initial values. The training and test set was split 50:50.

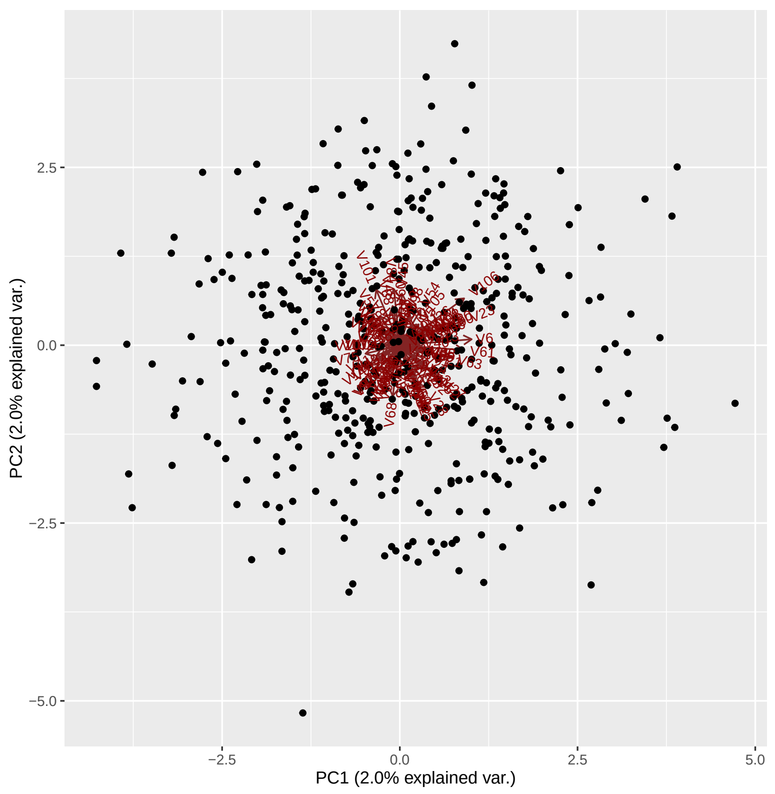


Figure 2: PCA of frequencies fitted using neural nets

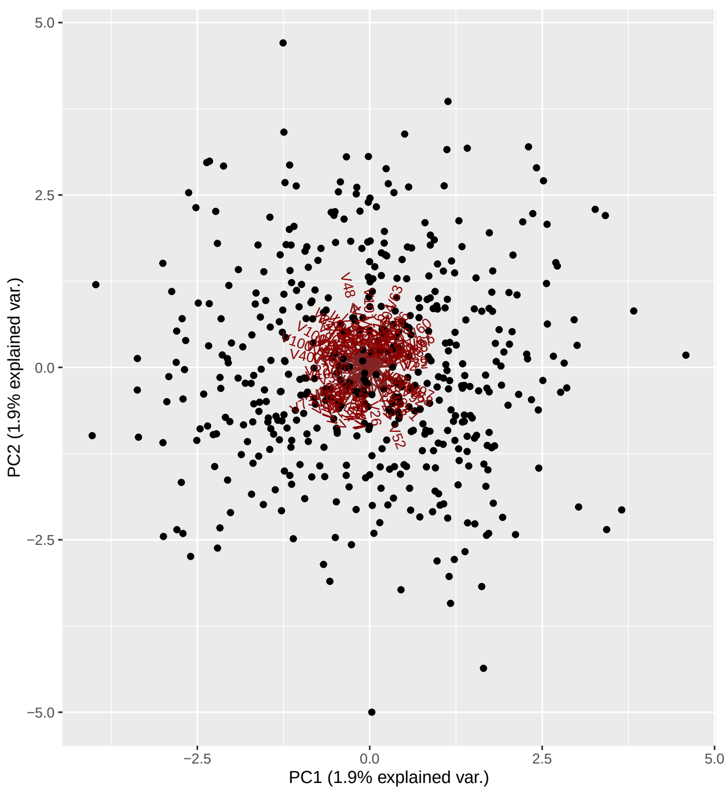


Figure 3: PCA of frequencies simulated at random for RFF