Deep Learning: A Bayesian Perspective

Nicholas G. Polson
Booth School of Business
*University of Chicago**
Vadim O. Sokolov
Systems Engineering and Operations Research
George Mason University

First Draft: May 2017

Abstract

Deep learning is a form of machine learning for nonlinear high dimensional data reduction and prediction. A Bayesian probabilistic perspective provides a number of advantages. Specifically statistical interpretation and properties, more efficient algorithms for optimisation and hyper-parameter tuning, and an explanation of predictive performance. Traditional high-dimensional statistical techniques; principal component analysis (PCA), partial least squares (PLS), reduced rank regression (RRR), projection pursuit regression (PPR) are shown to be shallow learners. Their deep learning counterparts exploit multiple layers of of data reduction which leads to performance gains. Stochastic gradient descent (SGD) training and optimisation and Dropout (DO) provides model and variable selection. Bayesian regularization is central to finding networks and provides a framework for optimal bias-variance trade-off to achieve good out-of sample performance. Constructing good Bayesian predictors in high dimensions is discussed. To illustrate our methodology, we provide an analysis of first time international bookings on Airbnb. Finally, we conclude with directions for future research.

1 Introduction

Deep learning (DL) is a form of machine learning that uses hierarchical layers of latent variables. Deep learners can be viewed as a probabilistic model where the conditional mean is specified as a stacked generalized linear model (sGLM). The current interest in DL stems from its remarkable success in a wide field of applications, including Artificial Intelligence (AI) DeepMind [2016], Kubota [2017], Esteva et al. [2017], image processing Simonyan and Zisserman [2014], learning in games DeepMind [2017], neuroscience Poggio [2016], energy conservation DeepMind [2016], skin cancer diagnostics Kubota [2017], Esteva et al. [2017]. Schmidhuber [2015] provides a comprehensive historical survey of deep learning and its application.

*Polson is Professor of Econometrics and Statistics at the Chicago Booth School of Business. email: ngp@chicagobooth.edu. Sokolov is an assistant professor at George Mason University, email: vsokolov@gmu.edu

Deep learning is a nonlinear high dimensional data reduction scheme. The theoretical roots of DL lie in Kolmogorov's representation of a multivariate response surfaces as a superposition of univariate semi-affine functions Kolmogorov [1963]. DL is algorithmic rather than probabilistic in its nature, see Breiman [2001] for the merits of both approaches. By providing a novel Bayesian perspective of the DL paradigm, a number of new avenues of research are available, such as faster stochastic algorithms, improved hyper-parameter tuning and interpretation of when good predictive performance is to be expected.

On the theoretical side, DL exploits a "universal basis" due to Kolmogorov and Arnold (1957) who show that any multivariate function can be expressed as a superposition of univariate semi-affine functions. Deep learning therefore provides a nonlinear dimension reduction technique for very high dimensional predictor spaces. By construction, the deep learning models are very flexible and gradient information can be efficiently calculated for variety of architectures.

On the empirical side, the advances in deep learning come from three sources

(i) new activation (a.k.a. link) functions, such as ReLU, instead of the historical use of sigmoid

(ii) depth of the architecture and dropout as a variable selection technique

(iii) computationally efficient routines to train and evaluate the models as well as accelerated computing via graphics processing unit (GPU) and tensor processing unit (TPU)

The rest of the paper is outlined as follows. Section 1.1 provides a review of deep learning. Section 2 provides a Bayesian probabilistic interpretation and Many traditional statistical techniques (PCA, PCR, SIR, LDA) are shown to be "shallow learners". That is, they are deep learning architectures with two layers. Much of the recent success in DL applications has been showing the advantages of including deeper layers and these gains pass over to traditional structural models. Section 3 provides heuristics on why Bayes procedure provide good predictors in high dimensional data reduction problems. Section 4 describes how to train, validate and test a deep learning models. Computational details associated with stochastic gradient descent (SGD) are given. Section 5 provides an application to bookings data from the Airbnb website. Finally, Section 5 concludes with directions for future research.

1.1 Deep Learning

Machine learning finds a predictor of an output Y given a high dimensional input X. A learning machine is an input-output mapping, $Y = F(X)$, where the input space is high-dimensional and we write

$$Y = F(X) \text{ where } X = (X_1, \ldots, X_p).$$

The output Y can be continuous, discrete as in classification, or mixed. For example, in a classification problem, we need to learn a mapping $F : X \to Y$, where $Y \in \{1, \ldots, K\}$ indexes categories. A predictor is denoted by $\hat{Y}(X) := F(X)$.

To construct a multivariate function, $F(X)$, we start with building blocks. Let $f_1, \ldots, f_l$ be univariate activation functions. A semi-affine activation rule is given by

$$f_l^{W,b} = f_l\left(\sum_{j=1}^{N_j} W_{ij} x_j + b_l\right) = f_l(W_l x_l + b_l)$$

Given L layers, the superposition (composite) prediction is given by

$$\hat{Y}(x) = F_{W,b}(x) = (f_1^{W_1,b_1} \circ \ldots \circ f_L^{W_L,b_L})(x)$$

Our deep predictor, given the number of layers L, then becomes the composite map

$$\hat{Y}(X) := \left(f_1^{W_1,b_1} \circ \ldots \circ f_L^{W_L,b_L}\right)(X).$$

Put simply, a high dimensional mapping, F, is modeled via the superposition of univariate semi-affine functions. Similar to a classic basis decomposition, the deep approach uses univariate activation functions to decompose a high dimensional X. To select the number of hidden units (a.k.a neurons), N_l, at each layer we will use dropout. The offset vector is essential. For example, using $f(x) = sin(x)$ without b would have difficulty recovering an even function like $\cos(x)$ cannot be approximated using this family. An offset element (e.g. $\sin(x + \pi/2) = \cos(x)$) immediately corrects this problem.

Let $Z^{(l)}$ denote the l-th layer, and so $X = Z^{(0)}$. The final output is the response Y, which can be numeric or categorical. A deep prediction rule is then

$$\begin{aligned}
Z^{(1)} &= f^{(1)}\left(W^{(0)}X + b^{(0)}\right), \\
Z^{(2)} &= f^{(2)}\left(W^{(1)}Z^{(1)} + b^{(1)}\right), \\
&\ldots \\
Z^{(L)} &= f^{(L)}\left(W^{(L-1)}Z^{(L-1)} + b^{(L-1)}\right), \\
\hat{Y}(X) &= W^{(L)}Z^{(L)} + b^{(L)}.
\end{aligned}$$

Here, $W^{(l)}$ are weight matrices, and $b^{(l)}$ are threshold or activation levels. Designing a good predictor depends crucially on the choice of univariate activation functions $f^{(l)}$. Kolmogorov's representation requires only two layers in principle Vitushkin [1964], A. G. Vitushkin [1967] prove the remarkable fact that a discontinuous link is required at the second layer even though the multivariate function is continuous. Neural networks (NN) simply approximate univariate function as mixtures of sigmoids, typically with an exponential number of neurons, which does not generalize well. They can simply be viewed as projection pursuit regression $F(x) = \sum_{i=1}^{N} g_i(WX + b))$ with the only difference being that in a neural network the nonlinear link functions, f_l, are parameter dependent and learned from training data.

Figure 1 shows a number of commonly used structures; for example, feed-forward architectures, neural Turing machines. Once you have learned the dimensionality of the weight matrices which are non-zero, there's an implied network structure.

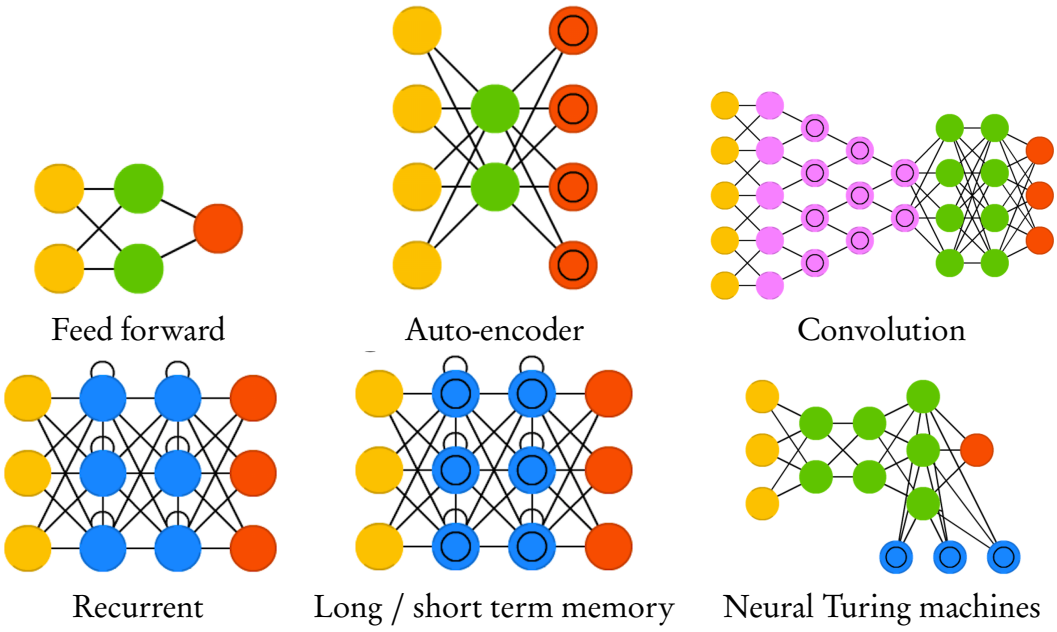


Figure 1: *Most commonly used deep learning architectures for modeling. Source:* http://www.asimovinstitute.org/neural-network-zoo.

Recently, deep architectures (indicating non-zero weights) include convolutional neural networks (CNN), recurrent NN (RNN), long-short term memory (LSTM), neural Turing machines (NTM). See Figure 1. Pascanu et al. [2013] and Montúfar and Morton [2015] provide results on the advantage of representing some functions compactly with deep layers, where they require exponential size of neurons with two layers. Poggio [2016] extends theoretical results on when deep learning can be exponentially better than shallow learning. Bryant [2008] implements Sprecher [1972], a constructive algorithm to estimate the inner link function which is not smooth. In practice, deep layers allow for given smooth activation functions and the key are the "learned" hyper-planes which find the underlying complex interactions and regions without having to see an exponentially large number of training samples.

Bayesian neural networks have a long history. Early results on stochastic recurrent neural networks (a.k.a Boltzmann machines) were published in Ackley et al. [1985]. Accounting for uncertainty by integrating over parameters is discussed in Denker et al. [1987]. MacKay [1992] proposed a general Bayesian framework for tuning network architecture and training parameters for feed forward architectures. Neal [1993] proposed using Hamiltonian Monte Carlo (HMC) to sample from posterior distribution over the set of model parameters and then averaging outputs of multiple models. Markov Chain Monte Carlo algorithms was proposed by Müller and Insua [1998] to jointly identify parameters of a feed forward neural network as well as the architecture. A connection of neural networks with Bayesian nonparametric techniques was demonstrated in Lee [2004]. There's an ongoing development of connections between Bayesian inference and modern deep learning. Recent results shows how dropout regularization technique can be used to represent uncertainty in deep learning models Gal and Ghahramani [2015].

2 Deep Probabilistic Learning

Probabilistically, output Y can be viewed as a random variable being generated by a probability model $p(Y|Y^{W,b}(X))$, conditional on the parameter in the predictor $Y^{W,b}(X)$, the negative log-likelihood defines $\mathscr{L}$ as

$$\mathscr{L}(Y,\hat{Y}) = -\log p(Y|Y^{\hat{W},\hat{b}}(X)).$$

The L_2-norm, $\mathscr{L}(Y_i,\hat{Y}(X_i)) = ||Y_i - \hat{Y}(X_i)||_2^2$ is traditional least squares, and the cross-entropy $\mathscr{L}(Y_i,\hat{Y}(X_i)) = \sum_{i=1}^n Y_i \log \hat{Y}(X_i)$ provides the equivalent of multi-class logistic classification.

The regularization term, $\lambda\phi(W,b)$, has a probabilistic interpretation as a negative log-prior distribution over parameters, namely

$$\begin{aligned} -\log p(\phi(W,b)) &= \lambda\phi(W,b), \\ p(\phi(W,b)) &\propto \exp(-\lambda\phi(W,b)). \end{aligned}$$

Deep predictors are regularized maximum a posteriori (MAP) estimators, where

$$\begin{aligned} p(W,b|D) &\propto p(Y|Y^{W,b}(X))p(W,b) \\ &\propto \exp\big(-\log p(Y|Y^{W,b}(X)) - \log p(W,b)\big), \end{aligned}$$

Training requires the solution of a highly nonlinear optimization

$$\hat{Y} := Y^{\hat{W},\hat{b}}(X) \text{ where } (\hat{W},\hat{b}) := \arg\min_{W,b} \log p(W,b|D),$$

where the log-posterior is optimised over the training data, $D = \{Y^{(i)},X^{(i)}\}_{i=1}^T$ is

$$-\log p(W,b|D) = \sum_{i=1}^T \mathscr{L}(Y^{(i)},Y^{W,b}(X^{(i)})) + \lambda\phi(W,b).$$

Deep learning has the key property that $\nabla_{W,b}\log p(Y|Y^{W,b}(X))$ is computationally very cheap to evaluate using tensor methods for very complicated architectures and fast implementation on large datasets. `TensorFlow` and `TPUs` provide a state-of-the-art framework for a plethora of architectures. From a statistical perspective, a caveat is that the posterior is highly multi-modal and providing good hyper-parameter tuning can be expensive. There is clearly a fruitful area of research applying state-of-the-art stochastic Bayesian MCMC algorithms to provide more efficient algorithms. For shallow architectures, we propose the alternating direction method of multipliers (ADMM) as a very efficient solution to the optimization problem.

2.1 Dropout for Model and Variable Selection

Dropout is a model selection technique designed to avoid over-fitting in the training process, and does so by removing input dimensions in X randomly with a given probability p. It is instructive to see how this affects the underlying loss function and optimization problem. For example, suppose that we wish to minimise MSE, $\mathscr{L}(Y,\hat{Y}) = ||Y - \hat{Y}||_2^2$, then, when marginalizing over the randomness, we have a new objective

$$\arg\min_W \mathbb{E}_{D\sim\text{Ber}(p)}||Y - W(D\star X)||_2^2,$$

which is equivalent to

$$\arg\min_W ||Y - pWX||_2^2 + p(1-p)||\Gamma W||_2^2,$$

where $\Gamma = (\text{diag}(X^\top X))^{\frac{1}{2}}$.

Dropout then simply uses a Bayesian ridge regression with a g-prior as an objective function. This reduces the likelihood of over-reliance on small sets of input data in training, see Hinton and Salakhutdinov (2006) and Srivastava et al. (2014). Dropout can be viewed as the optimization version of the traditional spike-and-slab prior that has proven so popular in Bayesian model averaging.

For example, in a simple model with one hidden layer, we replace the network

$$\begin{aligned} Y_i^{(l)} &= f(Z_i^{(l)}), \\ Z_i^{(l)} &= W_i^{(l)} X^{(l)} + b_i^{(l)}, \end{aligned}$$

with the dropout architecture

$$\begin{aligned} D_i^{(l)} &\sim \text{Ber}(p), \\ \tilde{Y}_i^{(l)} &= D^{(l)} \star X^{(l)}, \\ Y_i^{(l)} &= f(Z_i^{(l)}), \\ Z_i^{(l)} &= W_i^{(l)} X^{(l)} + b_i^{(l)}. \end{aligned}$$

In effect, this replaces the input X by $D \star X$, where $\star$ denotes the element-wise product and D is a matrix of independent Bernoulli Ber(p) distributed random variables.

Dropout also regularizes the choice of the number of hidden units in a layer. This can be achieved if we drop units of the hidden rather than the input layer and then establish which probability p gives the best results. It is worth recalling though, as we have stated before, that one of the dimension reduction properties of a network structure is that once a variable from a layer is dropped, all terms above it in the network also disappear.

2.2 Shallow Learners

Almost all shallow data reduction techniques can be viewed as consisting of a low dimensional auxiliary variable Z and a prediction rule specified by a composition of functions

$$\hat{Y} = f_1^{W_1, b_1}(f_2(W_2 X + b_2)) = f_1^{W_1, b_1}(Z), \text{ where } Z := f_2(W_2 X + b_2).$$

The problem of high dimensional data reduction is to find the Z-variable and to estimate the layer functions (f_1, f_2) correctly. In the layers, we want to uncover the low-dimensional Z-structure in a way that does not disregard information about predicting the output Y.

Principal component analysis (PCA), partial least squares (PLS), reduced rank regression (RRR), linear discriminant analysis (LDA), project pursuit regression (PPR), and logistic regression are all shallow learners. Mallows [1973] provides an interesting perspective on how Bayesian shrinkage provides good predictors in regression settings. Frank and Friedman [1993] provide

excellent discussions of PLS and why Bayesian shrinkage methods provide good predictors. Wold (1956), Diaconis and Shahshahani (1984), Ripley (1996), Cook (2007), and Hastie et al. (2009) provide further discussion of dimension reduction techniques. Other connection exists for Fisher's Linear Discriminant classification rule which is simply fitting $H(wX+b)$, where H is a Heaviside function. Polson and Scott (2011) provide a Bayesian version of support vector machines (SVMs) and a comparison with logistic regression for classification.

PCA reduces X to $f_2(X)$ using a singular value decomposition of the form

$$Z = f_2(X) = W^\top X + b\,, \tag{1}$$

where the columns of the weight matrix W form an orthogonal basis for directions of greatest variance (which is in effect an eigenvector problem). Similarly PPR reduces X to $f_2(X)$ by setting

$$Z = f_2(X) = \sum_{i=1}^{N_1} f_i(W_{i1}X_1 + \ldots + W_{ip}X_p).$$

Example: Interaction terms, x_1x_2 and $(x_1x_2)^2$, and max functions, $\max(x_1,x_2)$ can be expressed as nonlinear functions of semi-affine combinations. Specifically,

$$x_1x_2 = \frac{1}{4}(x_1+x_2)^2 - \frac{1}{4}(x_1-x_2)^2$$

$$\max(x_1,x_2) = \frac{1}{2}|x_1+x_2| + \frac{1}{2}|x_1-x_2|$$

$$(x_1x_2)^2 = \frac{1}{4}(x_1+x_2)^4 + \frac{7}{4\cdot 3^3}(x_1-x_2)^4 - \frac{1}{2\cdot 3^3}(x_1+2x_2)^4 - \frac{2^3}{3^3}(x_1+\frac{1}{2}x_2)^4$$

Diaconis and Shahshahani [1981] provide further discussion for Projection Pursuit Regression, where the network uses a layered model $\sum_{i=1}^N f(w_i^\top X)$. Diaconis et al. [1998] provide an ergodic view of composite iterated functions, a precursor to the use of multiple layers of single operators that can model complex multivariate systems. Sjöberg et al. [1995] provide the approximation theory for composite functions.

Example: Deep ReLU architectures can be viewed as Max-Sum networks via the following simple identity. Define $x^+ = \max(x,0)$. Let $f_x(b) = (x+b)^+$ where b is an offset. Then $(x+y^+)^+ = \max(0,x,x+y)$. This is generalized in Feller [1971] (p.272) who shows by induction that

$$(f_{x_1} \circ \ldots \circ f_{x_k})(0) = (x_1 + (x_2 + \ldots + (x_{k-1} + x_k^+)^+)^+)^+ = \max_{1\le j\le k}(x_1+\ldots+x_j)^+$$

A composition or convolution of max-layers is then a one layer max-sum network.

2.3 Stacked Auto-Encoders

Auto-encoding is an important data reduction technique. An auto-encoder is a deep learning architecture designed to replicate X itself, namely $X = Y$, via a *bottleneck* structure. This means we select a model $F^{W,b}(X)$ which aims to concentrate the information required to recreate X.

See Heaton et al (2017) for an application to smart indexing in finance. Suppose that we have N input vectors $X = \{x_1, \ldots, x_N\} \in \mathbb{R}^{M \times N}$ and N output (or target) vectors $\{x_1, \ldots, x_N\} \in \mathbb{R}^{M \times N}$. Setting biases to zero, for the purpose of illustration, and using only one hidden layer ($L = 2$) with $K < N$ factors, gives for $j = 1, \ldots, N$

$$\begin{aligned} Y_j(x) = F_W^m(X)_j &= \sum_{k=1}^{K} W_2^{jk} f\left(\sum_{i=1}^{N} W_1^{ki} x_i\right) \\ &= \sum_{k=1}^{K} W_2^{jk} Z_j \text{ for } Z_j = f\left(\sum_{i=1}^{N} W_1^{ki} x_i\right). \end{aligned}$$

Since, in an auto-encoder, we fit the model $X = F_W(X)$, and *train* the weights $W = (W_1, W_2)$ with regularization penalty in a

$$\begin{aligned} \mathscr{L}(W) &= \arg\min_W ||X - F_W(X)||^2 + \lambda\phi(W) \\ \text{with} \quad \phi(W) &= \sum_{i,j,k} |W_1^{jk}|^2 + |W_2^{ki}|^2. \end{aligned}$$

Writing the DL objective as an augmented Lagrangian (as in ADMM) with a hidden factor Z, leads to a two step algorithm of two steps, an encoding step (a penalty for Z), and a decoding step for reconstructing the output signal via

$$\arg\min_{W,Z} ||X - W_2 Z||^2 + \lambda\phi(Z) + ||Z - f(W_1, X)||^2,$$

where the regularization on W_1 induces a penalty on Z. The last term is the encoder, the first two the decoder.

If W_2 is estimated from the structure of the training data matrix, then we have a traditional factor model, and the W_1 matrix provides the factor loadings. PCA, PLS, SIR fall into this category, see Cook (2007) for further discussion. If W_2 is trained based on the pair $\hat{X} = \{Y, X\}$ then we have a sliced inverse regression model. If W_1 and W_2 are simultaneously estimated based on the training data X, then we have a two layer deep learning model.

Auto-encoding demonstrates that deep learning does not directly model variance-covariance matrix explicitly as the architecture is already in predictive form. Given a hierarchical non-linear combination of deep learners, an implicit variance-covariance matrix exists, but that is not the driver of the algorithm.

Another interesting area for future research are long short term memory models (LSTMs). For example, a dynamic one layer auto-encoder for a financial time series (Y_t) is a coupled system of the form

$$Y_t = W_x X_t + W_y Y_{t-1} \text{ and } \begin{pmatrix} X_t \\ Y_{t-1} \end{pmatrix} = W Y_t.$$

Here, the state equation encodes and the matrix W decodes the Y_t vector into its history Y_{t-1} and the current state X_t.

3 Finding Good Bayes Predictors

Bayesian methods tackle the problem of good predictive performance in a number of ways. The goal is to find a good predictive MSE $E_{Y,\hat{Y}}(||\hat{Y} - Y||^2)$. First, Stein shrinkage (a.k.a regularization

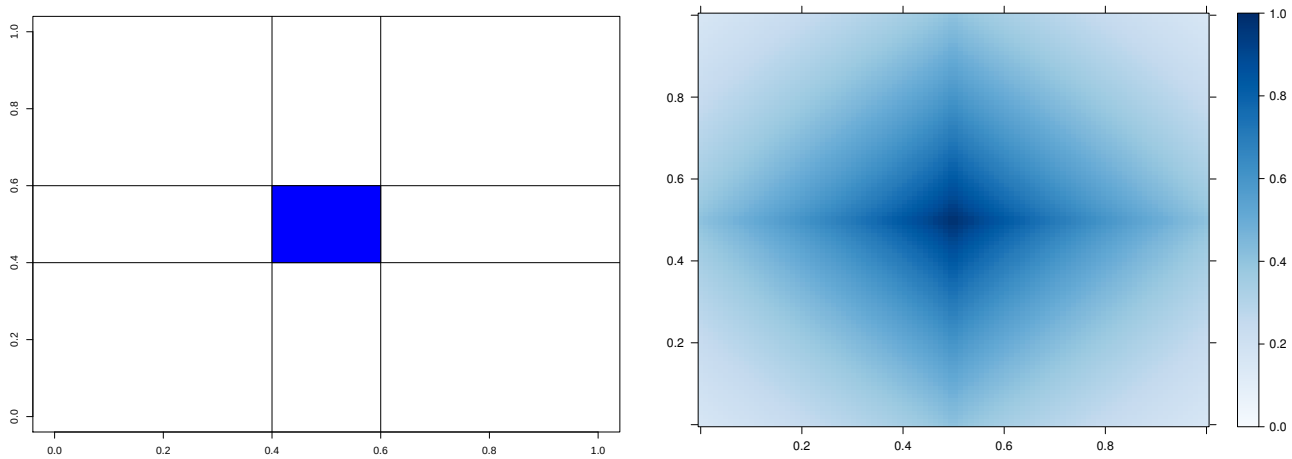
with an ℓ_2 norm) has long been know to provide good mean squared error properties in estimation, namely $E(||\hat{\theta} - \theta)||^2)$ as well. These gains translate into predictive performance (in an iid setting) for $E(||\hat{Y} - Y||^2)$. One of the main issues is how to tune the amount of regularisation (a.k.a prior hyper-parameters). Stein's unbiased estimator of risk provides a simple empirical rule to address this problem, as does cross-validation. From a Bayes perspective, the marginal likelihood (and full marginal posterior) provides a natural method for hyper-parameter tuning. The issue is computational tractability and scalability. The posterior for (W, b) is extremely high dimensional and multimodal and posterior MAP provides good predictors $\hat{Y}(X)$.

Bayes conditional averaging can also perform well in high dimensional regression and classification problems. High dimensionality brings with it the curse of dimensionality and it is instructive to understand why certain kernel can perform badly.

Adaptive Kernel predictors (a.k.a. smart conditional averager) are of the form

$$\hat{Y}(X) = \sum_{r=1}^{R} K_r(X_i, X)\hat{Y}_r(X).$$

Here $\hat{Y}_r(X)$ is a deep predictor with its own trained parameters. For tree models, the kernel $K_r(X_i, X)$ is a *cylindrical* region R_r (open box set). Figure 2 illustrates the implied kernels for trees (cylindrical sets) and random forests. Not too many points will be neighbors in high dimensional input space.



(a) Tree Kernel (b) Random Forest Kernel

Figure 2: Kernels for Cylindrical region and Random Forests

Constructing the regions is fundamental to reduce the curse of dimensionality. It is useful to imagine a very large dataset, e.g. 100k images and think about how a new image's input coordinates, X, are neighbors" to data point in the training set. Our predictor will then be a smart conditional average of the observed outputs, Y, for our neighbors, When p is large, spheres (L^2 balls or Gaussian kernels) are terrible, either no points or all points are neighbors" of the new input variable. Trees are good as not too many "neighbors'.

To illustrate the problem further, Figure 3 below shows the 2D image of 1000 uniform samples from a 50-dimensional ball B_{50}. The image is calculated as $w^T Y$, where $w = (1, 1, 0, \ldots, 0)$ and

$Y \sim U(B_{50})$. Samples are centered around the equators and none of the samples fall nowhere close to the boundary of the set.

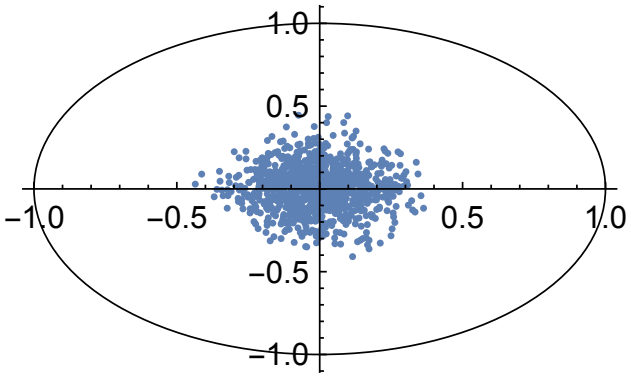
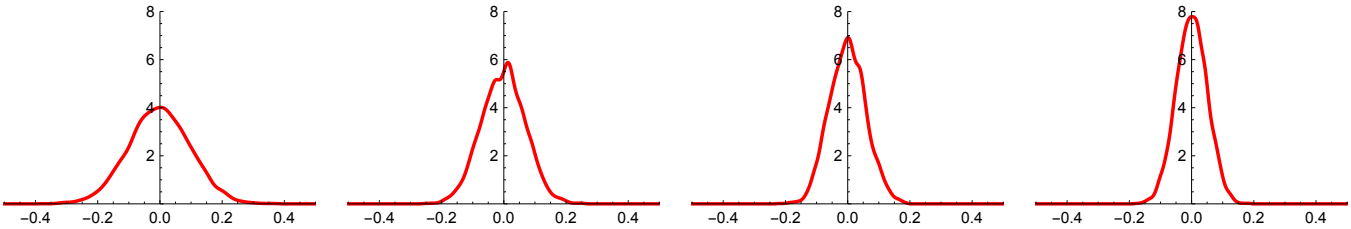


Figure 3: The 2D image of the 50-dimensional ball and the result of the Monte Carlo sampling

As dimensionality of the space grows, the variance of the marginal distribution goes to zero. We can empirically see it from Figure 4, which shows histogram of 1D image of uniform sample from balls of different dimensionality, i.e. $e_1^T Y$, where $e_1 = (1, 0, \ldots, 0)$.



(a) $p = 100$ (b) $p = 200$ (c) $p = 300$ (d) $p = 400$

Figure 4: Histogram of marginal distribution of $Y \sim U(B_p)$ for different dimensions p.

Similar central limit results were known to Maxwell who shown that random variable $w^T Y$ is close to standard normal, when $Y \sim U(B_p)$, p is large, and w is a unit vector (lies on the boundary of the ball). For the history of this fact, see Diaconis and Freedman [1987]. More general results in this direction were obtained in Klartag [2007]. Further, Milman and Schechtman [2009] presents many analytical and geometrical results for finite dimensional normed spaces, as the dimension grows to infinity.

Deep learning improves on this by performing a sequence of GLM-like transformations, effectively DL learns a distributed partition of the input space. Specifically, suppose that we have K partitions. Then the DL predictor takes the form of a weighted average or soft-max of the weighted average in case of classification of observations in this partition. Given a new high dimensional input X_{new}, many deep learners are an average of learners obtained by our hyper-plane decomposition, Generically, we have

$$\hat{Y}(X) = \sum_{k \in K} w_k(X) \hat{Y}_k(X),$$

where w_k are the weights learned in region K, and $w_k(X)$ is an indicator of the region with appropriate weighting given the training data. Where w_k is a weight which also indicates which partition the new $X_n e w$ lies in.

The partitioning of the input space by a deep learner is similar to the one performed by decision trees and partition-based models such as CART, MARS, RandomForests, BART. However, use trees and are more local in the regions that there use to construct their estimators within a region. Each neuron in deep learning model corresponds to a manifold that divides the input space. In case of ReLU activation function $f(x) = \max(0, x)$ the manifold is simply a hyperplane and neuron gets activates, when the new observation is on the "right" side of this hyperplane, the activation amount is equal to how far from the boundary the given point is. For example in two dimensions, three neurons with ReLU activation functions will divide the space into seven regions, as shown on Figure 5.

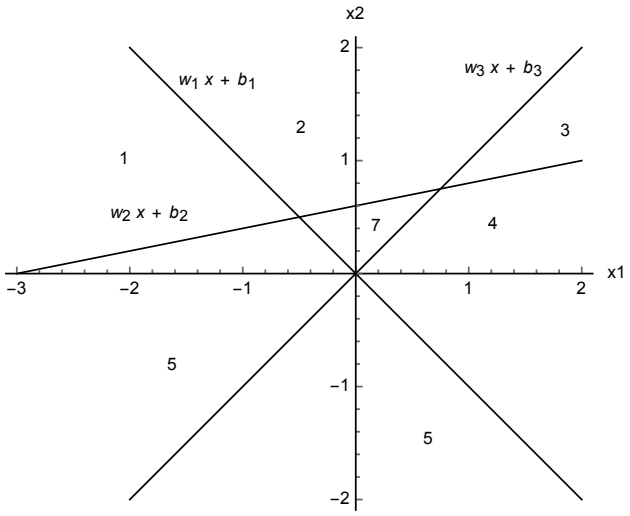


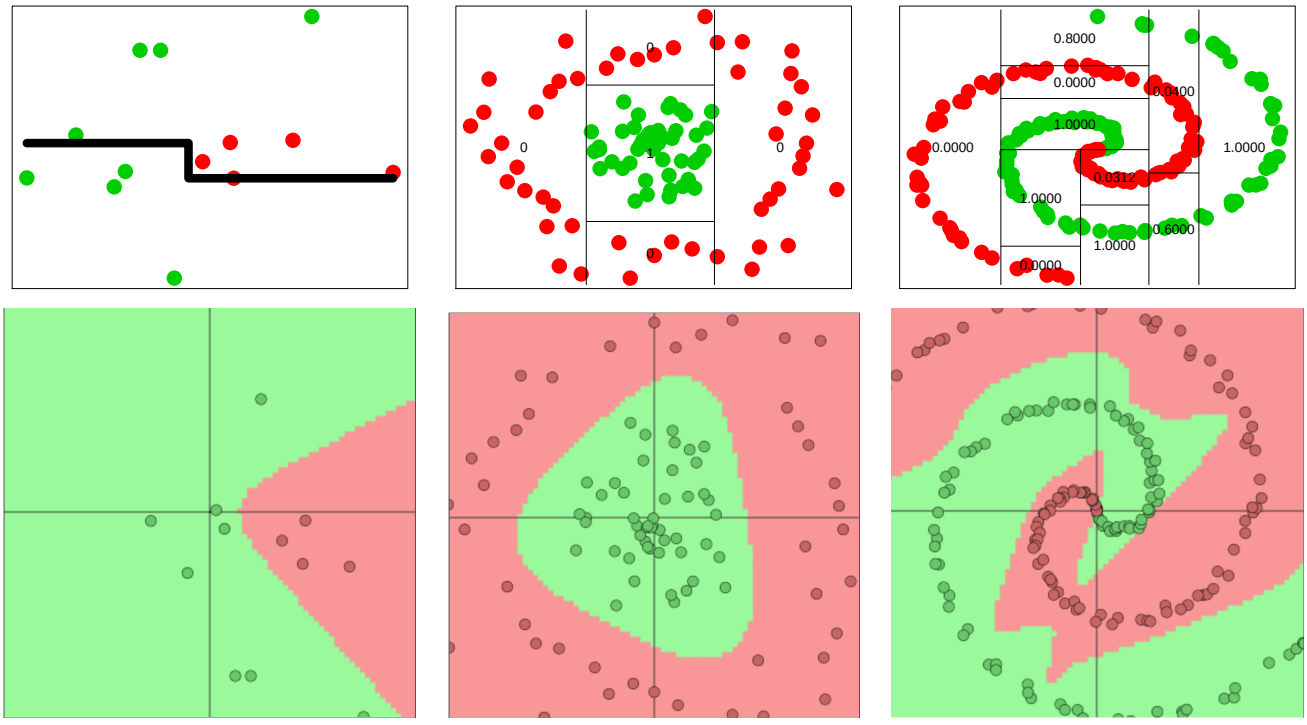
Figure 5: Hyperplanes defined by three neurons with ReLU activation functions.

The key difference then between tree-based architecture and neural network based models is the way hyper-planes are combined. Figure 6 shows the comparison of space decomposition by hyperplanes as performed by a tree-based and neural network architectures. We compare a neural network with two layers (bottom row) with tree mode trained with CART algorithm (top row). The network architecture used is

$$\begin{aligned} Y &= \text{softmax}(w^0 Z^2 + b^0) \\ Z^2 &= \tanh(w^2 Z^1 + b^2) \\ Z^1 &= \tanh(w^1 X + b^1)\,. \end{aligned}$$

The weight matrices for simple data $W^1, W^2 \in \mathbb{R}^{2\times 2}$, for circle data $W^1 \in \mathbb{R}^{2\times 2}$ and $W^2 \in \mathbb{R}^{3\times 2}$, for spiral data we have $W^1 \in \mathbb{R}^{2\times 2}$ and $W^2 \in \mathbb{R}^{4\times 2}$. In our notations, we assume that the activation function is applied point-vise at each layer. An advantage of deep architectures is that the number of hyper-planes grow exponentially with the number of layers. The key property of an activation function (link) is $f(0) = 0$ and it has zero value in certain regions. For example, hinge or rectified learner $\max(x, 0)$ box car (differences in Heaviside) functions are very common. As compared

to a logistic regression, rather then using softmax$(1/(1+e^{-x}))$ in deep learning tanh is typically used for training.



(a) simple data (b) circle data (c) spiral data

Figure 6: Space partition by tree architectures (top row) and deep learning architectures (bottom row) for three different data sets.

Amit and Geman [1997] provide an interesting discussion. Formally, a Bayesian probabilistic approach (if computationally feasible) known how to optimally weight predictors via a model averaging approach

$$\hat{Y}(X) = \sum_{r=1}^{R} w_k \hat{Y}_k(X),$$

where $\hat{Y}_k(x) = E(Y \mid X_k)$. Such rules can achieve great out-of-sample performance. Amit et al. [2000] discuss the striking success of multiple randomized classifiers. Using a simple set of binary local features, one classification tree can achieve 5% error on the NIST data base with 100,000 training data points. On the other hand 100 trees, trained under one hour, when aggregated yield an error rate under 7%. We believe that this stems from the fact that a sample from a very rich and diverse set of classifiers produces on average weakly dependent classifiers conditional on class. A Bayesian model of weak dependence is exchangeability.

Suppose that we have K exchangeable, $\mathbb{E}(\hat{Y}_i) = \mathbb{E}(\hat{Y}_{\pi(i)})$, predictors

$$\hat{Y} = (\hat{Y}_1, \ldots, \hat{Y}_K)$$

Find w to attain $\arg\min_W El(Y, w^T\hat{Y})$ where l convex in the second argument;

$$El(Y, w^T\hat{Y}) = \frac{1}{K!}\sum_{\pi} El(Y, w^T\hat{Y}) \geq El\left(Y, \frac{1}{K!}\sum_{\pi} w_{\pi}^T\hat{Y})\right) = El\left(Y, (1/K)\iota^T\hat{Y}\right)$$

where $\iota = (1, \ldots, 1)$. Hence, the randomised multiple predictor with weights $w = (1/K)\iota$ provides close to optimal Bayes predictive performance. We now turn to algorithmic issues.

4 Algorithmic Issues

4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a default standard for minimizing the loss function $f(W, b)$ (maximizing the likelihood) to find the deep learning weights and offsets. SGD simply minimizes the function by taking a negative step along an estimate g^k of the gradient $\nabla f(W^k, b^k)$ at iteration k. The gradients are available via the chain rule applied to the superposition of semi-affine functions. The approximate gradient is estimated by calculating

$$g^k = \frac{1}{|E_k|}\sum_{i \in E_k} \nabla \mathscr{L}_{w,b}(Y_i, \hat{Y}^k(X_i)),$$

where $E_k \subset \{1, \ldots, T\}$ and $|E_k|$ is the number of elements in E_k.

When $|E_k| > 1$ the algorithm is called batch SGD and simply SGD otherwise. Typically, the subset E is chosen by going cyclically and picking consecutive elements of $\{1, \ldots, T\}$, $E_{k+1} = [E_k \mod T] + 1$. The direction g^k is calculated using a chain rule (a.k.a. back-propagation) providing an unbiased estimator of $\nabla f(W^k, b^k)$. Specifically, we have

$$\mathrm{E}(g^k) = \frac{1}{T}\sum_{i=1}^{T} \nabla \mathscr{L}_{w,b}(Y_i, \hat{Y}^k(X_i)) = \nabla f(W^k, b^k).$$

At each iteration, the SGD updates the solution

$$(W, b)^{k+1} = (W, b)^k - t_k g^k$$

Deep learning algorithms use step size t_k (a.k.a learning rate) that is either kept constant or a simple step size reduction strategy, such as $t_k = a\exp(-kt)$ is used. The hyper parameters of reduction schedule are usually found empirically from numerical experiments and observations of the loss function progression.

One caveat of SGD is that the descent in f is not guaranteed or can be very slow at every iteration. Stochastic Bayesian approaches ought to alleviate these issues. The variance of the gradient estimate g^k can also be near zero as the iterates converge to a solution. To tackle those problems a coordinate descent (CD) and momentum-based modifications can be applied. Alternative directions method of multipliers (ADMM) can also provide a natural alternative and lead to non-linear alternating updates, see Carreira-Perpinán and Wang [2014].

The CD evaluates a single component E_k of the gradient ∇f at the current point and then updates the E_kth component of the variable vector in the negative gradient direction. The momentum-based versions of SGD or so-called accelerated algorithms were originally proposed by Nesterov

[1983]. For more recent discussion, see Nesterov [2013]. Momentum adds memory to the search process by combining new gradient information with the previous search directions. Empirically momentum-based methods have been shown a better convergence for deep learning networks Sutskever et al. [2013]. The gradient only influences changes in the velocity of the update, which then updates the variable

$$\begin{aligned} v^{k+1} &= \mu v^k - t_k g((W,b)^k) \\ (W,b)^{k+1} &= (W,b)^k + v^k \end{aligned}$$

The hyper-parameter μ controls the dumping effect on the rate of update of the variables. The physical analogy is the reduction in kinetic energy that allows to "slow down" the movements at the minima. This parameter can also chosen empirically using cross-validation.

Nesterov's momentum method (a.k.a. Nesterov acceleration) calculates the gradient at the point predicted by the momentum. One can view this as a look-ahead strategy with updating scheme

$$\begin{aligned} v^{k+1} &= \mu v^k - t_k g((W,b)^k + v^k) \\ (W,b)^{k+1} &= (W,b)^k + v^k. \end{aligned}$$

Another popular modification are the AdaGrad methods Zeiler [2012], which adaptively scales each of the learning parameter at each iteration

$$\begin{aligned} c^{k+1} &= c^k + g((W,b)^k)^2 \\ (W,b)^{k+1} &= (W,b)^k - t_k g(W,b)^k)/(\sqrt{c^{k+1}} - a), \end{aligned}$$

where is usually a small number, e.g. $a = 10^{-6}$ that prevents from dividing by zero. `PRMSprop` takes the `AdaGrad` idea further and puts more weight on recent values of gradient squared to scale the update direction, i.e. we have

$$c^{k+1} = d c^k + (1-d) g((W,b)^k)^2$$

The `Adam` method Kingma and Ba [2014] combines both `PRMSprop` and momentum methods, it leads to the following update equations

$$\begin{aligned} v^{k+1} &= \mu v^k - (1-\mu) t_k g((W,b)^k + v^k) \\ c^{k+1} &= d c^k + (1-d) g((W,b)^k)^2 \\ (W,b)^{k+1} &= (W,b)^k - t_k v^{k+1}/(\sqrt{c^{k+1}} - a) \end{aligned}$$

Second order methods solve the optimization problem by solving a system of nonlinear equations $\nabla f(W,b) = 0$ by applying the Newton's method

$$(W,b)^+ = (W,b) - \{\nabla^2 f(W,b)\}^{-1} \nabla f(W,b).$$

We can see that SGD simply approximates $\nabla^2 f(W,b)$ by $1/t$. The advantages of a second order method include much faster convergence rates and insensitivity to the conditioning of the problem. In practice, second order methods are rarely used for deep learning applications Dean et al. [2012b]. The major disadvantage is inability to train model using batches of data as SGD does. Since typical deep learning model relies on large scale data sets, the second order methods become memory and computationally prohibitive at even modest-sized training data sets.

4.2 Learning Shallow Predictors

Traditional factor models use linear combination of K latent factors, $\{F_1, F_2, \ldots, F_K\}$,

$$Y_i = \sum_{k=1}^{K} w_{ik} F_k \, , \; \forall i = 1, \ldots, N.$$

Here factors F_k and weights B_{ik} can be found by solving the following problem

$$\text{argmin}_{w,F} \sum_{n=1}^{N} ||z_n - \sum_{k=1}^{K} w_{nk} F_k||^2 + \lambda \sum_{k,n=1}^{N,K} ||w_{nk}||_l,$$

where $l = 1$ or $l = 2$. Here we minimize the reconstruction error (a.k.a. accuracy) plus the regularization penalty to control the variance-bias trade-off for out-of-sample prediction. Algorithms exist to solve this problem very efficiently. Such a model can be represented as a neural network model with $L = 2$ with identity activation function.

The basic sliced inverse regression Li [1991] (SIR) model takes the form

$$Y = G(WX, \epsilon),$$

where $G(\cdot)$ is a nonlinear function and $W \in R^{k \times p}$, with $k < p$, in other words, Y is a function of k linear combinations of X. To find W, we first slice the feature matrix, then analyze data's covariance matrix and covariance matrix of the slice means of X, weighted by the size of slice. Function G is found empirically by visually exploring relation. Advantage of deep learning approach is that functional relation G is found automatically. As an extension to the original SIR fitting algorithm, Jiang and Liu [2013] proposed a variable selection under the SIR modeling framework. A partial least squares regression (PLS) Wold et al. [2001] finds T, a lower dimensional representation of $X = TP^T$ and then regresses it onto Y, $Y = TBC^T$.

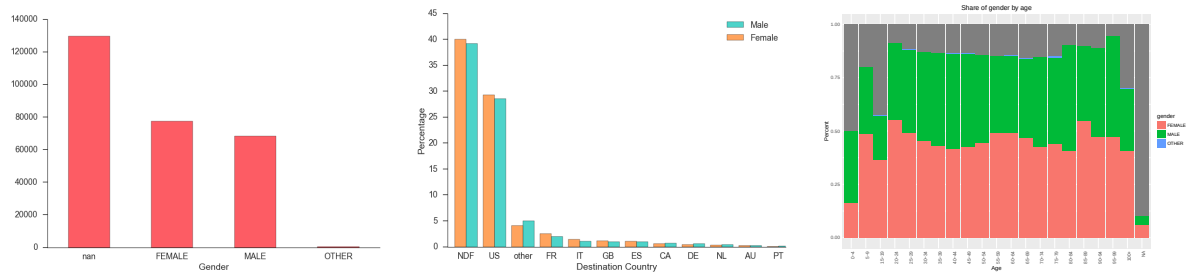
Deep learning least squares networks arrive at a criterion function given by a negative log-posterior which need to be minimized. Penalized log-posterior, with ϕ denoting a generic regularization penalty is given by

$$L(G, F) = \sum_{i=1}^{n} ||y_i - g(F(x_i))||_2 + \lambda_g \phi(G, F)$$
$$\phi_i = \sum_{k} f_k^L(a_{ik}^L), \; a_{ik}^L = z_{ik}^L w^L, \; z_{ik}^L = \sum_{j} f^{L-1}(a_{jk}^{L-1}).$$

Carreira-Perpinán and Wang [2014] proposed a method of auxiliary coordinates which allows to replace the original unconstrained optimization problem associate with model training with an alternative function in a constrained space that can be optimized using alternating directions method and thus is highly parallelizable. An extension of these methods are ADMM and Divide and Concur (DC) algorithms, see Polson et al. [2015]. The gains for applying these to deep layered models, in an iterative fashion, appear to be large but have yet to be quantified empirically.

5 Application: Predicting Airbnb Bookings

To illustrate the deep learning paradigm, we use a dataset provided by Airbnb for Kaggle competition was used for our empirical analysis. The goal is to build a predictive model that can predict in which country a new user will make his or her first booking. Though Airbnb offers bookings in more then 190 countries, there are 10 countries where users make frequent bookings. We treat the problem as classification into one of the 12 classes (10 major countries + other + NDF); where *other* corresponds to any other country which is not in the list of top 10 and *NDF* corresponds to situations where no booking was made. The data consists of two tables, one contains the attributes of each of the users and the other contains data about sessions of each user at the Airbnb website. The user data contains demographic characteristics, type of device and browser used to sign up, and the destination country of the first booking, which is our dependent variable Y. The data involves 213,451 users and 1,056,7737 individual sessions. The sessions data contains information about actions taken during each session, duration and devices used. Both datasets has a large number of missing values. For example age information is missing for 42% of the users. Figure 7(a) shows that nearly half of the gender data is missing and there is slight imbalance between the genders.



(a) Number of observations for each gender

(b) Percent of reservations per destination

(c) Relationship between age and gender

Figure 7: Gender and Destination Summary Plots for Airbnb users.

Figure 7(b) shows the country of origin for the first booking by gender. Most of the entries in the destination columns are NDF, meaning no booking was made by the user. Further, Figure 7(c) shows relationship between gender and age, the gender value is missing for most of the users who did not identify their age.

As there is little difference in booking behavior between the genders, we assume missing at random case and that there is little effect of the gender variable on the outcome. However, as we will see later, the fact that gender was specified is an important predictor. Intuitively, users who filled the gender field are more "serious" and thus are more likely to book.

On the other hand, as Figure 8 shows, age does play a role.

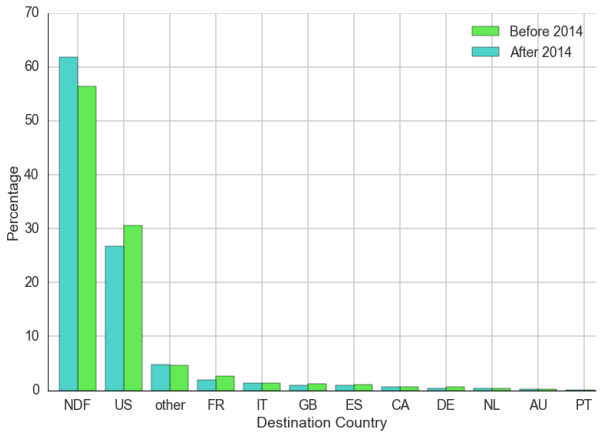
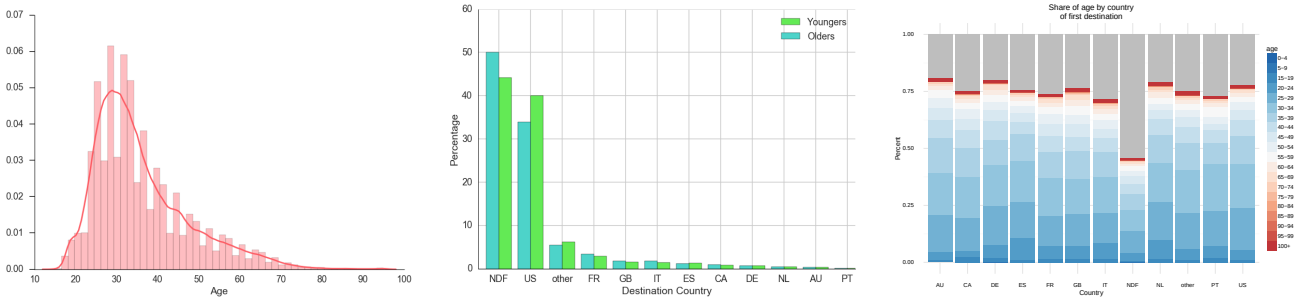


Figure 9: Booking behavior for users who opened their accounts before 2014



(a) Empirical distribution of user's age

(b) Destination by age category

(c) Destination by age group

Figure 8: Age Information for Airbnb users.

Figure 8(a) shows that most of the users are of age between 25 and 40. Furthermore, looking at booking behavior between two different age groups, younger than 45 cohort and older than 45 cohort, (see from Figure 8(b)) have very different booking behavior. Further, as we can see from Figure 8(c) half of the users who did not book did not identify their age either.

Another effect of interest is the non-linearity between the time account was created and booking behavior. Figure 9 shows that "old timers" are more likely to book when compared to recent users. Since the number of records in sessions data is different for each users we developed features from those records so that sessions data can be used for prediction. The general idea is to covert multiple session records to a single set of features per user. The list of the features we calculate is

(i) Number of sessions records

(ii) For each action type, we calculate the count and standard deviation

(iii) For each device type, we calculate the count and standard deviation

(iv) For session duration we calculate mean, standard deviation and median

Furthermore, we use one-hot encoding for categorical variables from the user table, e.g. gender, language, affiliate provider, etc. One-hot encoding replaces categorical variable with K categories by K binary dummy variable.

We build a deep learning model with two hidden dense layers and ReLU activation function $f(x) = \max(0, x)$. We use ADAGRAD optimization to train the model. We predict probabilities of future destination booking for each of the new users. The evaluation metric for this competition is NDCG (Normalized discounted cumulative gain). It uses top five predicted destinations and is calculated as:

$$\text{NDCG}_k = \frac{1}{n}\sum_{i=1}^{n}\text{DCG}_5^i,$$

where $\text{DCG}_5^i = 1/\log_2(p(i)+1)$ and $p(i)$ is the position of the true destination in the list of five predicted destinations. For example, if for a particular user i the destination is FR, and FR was at the top of the list of five predicted countries, than

$$\text{DCG}_5^i = \frac{1}{\log_2(1+1)} = 1.0.$$

when FR is second, e.g. model prediction (US, FR, DE, NDF, IT) gives a

$$\text{DCG}_5^i = \frac{1}{\log_2(2+1)} = 1/1.58496 = 0.6309$$

We trained our deep learning network with 20 epochs and batch size of 256. We used 10% of the sample, i.e. 21346 observations for evaluating the model. We used a two-hidden layer architecture with ReLU activation functions

$$\begin{aligned} Y &= \text{softmax}(w^0 Z^2 + b^0) \\ Z^2 &= \max(w^2 Z^1 + b^2, 0) \\ Z^1 &= \max(w^1 X + b^1, 0)\,. \end{aligned}$$

The weight matrices for simple data $W^1 \in \mathbb{R}^{64\times p}$, $W^2 \in \mathbb{R}^{64\times 64}$. In our notations, we assume that the activation function is applied point-vise at each layer.

The resulting model has out-of-sample $NDCG$ of —0.8351. The classes are imbalanced in this problem. Table 1 shows percent of each class in out-of-sample data set.

Dest	AU	CA	DE	ES	FR	GB	IT	NDF	NL	PT	US	other
% obs	0.3	0.6	0.5	1	2.2	1.2	1.2	59	0.31	0.11	29	4.8

Table 1: Percent of each class in out-of-sample data set

Figure 10 shows out-of-sample NDCG for each of the destinations.

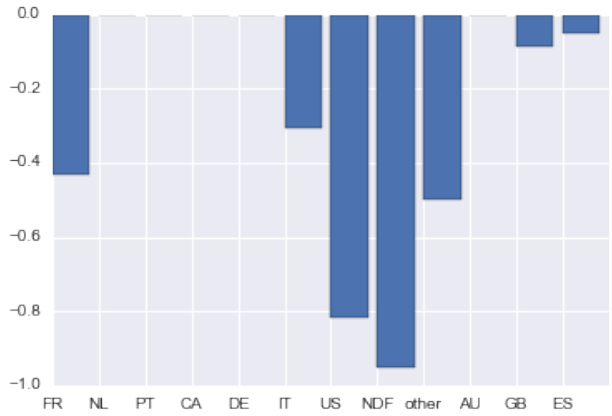
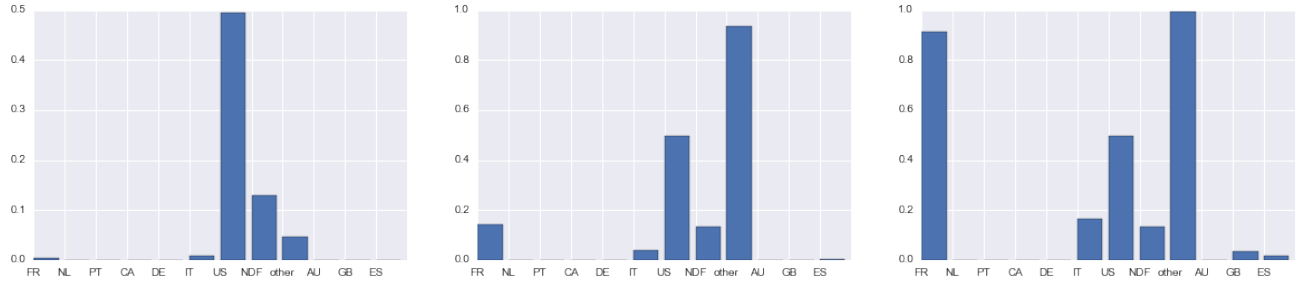


Figure 10: Out-of-sample NDCG for each of the destinations

Figure 11 shows accuracy of prediction for each of the destination countries.



(a) first (b) second (c) third

Figure 11: Prediction accuracy for deep learning model. Panel (a) shows accuracy of prediction when only top predicted destination is used. Panel (b) shows correct percent of correct predictions when correct country is in the top two if the predicted list. Panel (c) shows correct percent of correct predictions when correct country is in the top three if the predicted list

The model accurately predicts bookings in the US and FR and other when top three predictions are considered.

Furthermore, we compared performance of the deep learning model with the XGBoost algorithms Chen and Guestrin [2016] for fitting gradient boosted tree model. The performance of the model is comparable and yields NGD of —0.8476. One of the advantages of the tree-based model is ability to calculate importance of each of the features Hastie et al. [2016]. Figure 12 shows the variable performance calculated from our XGBoost model.

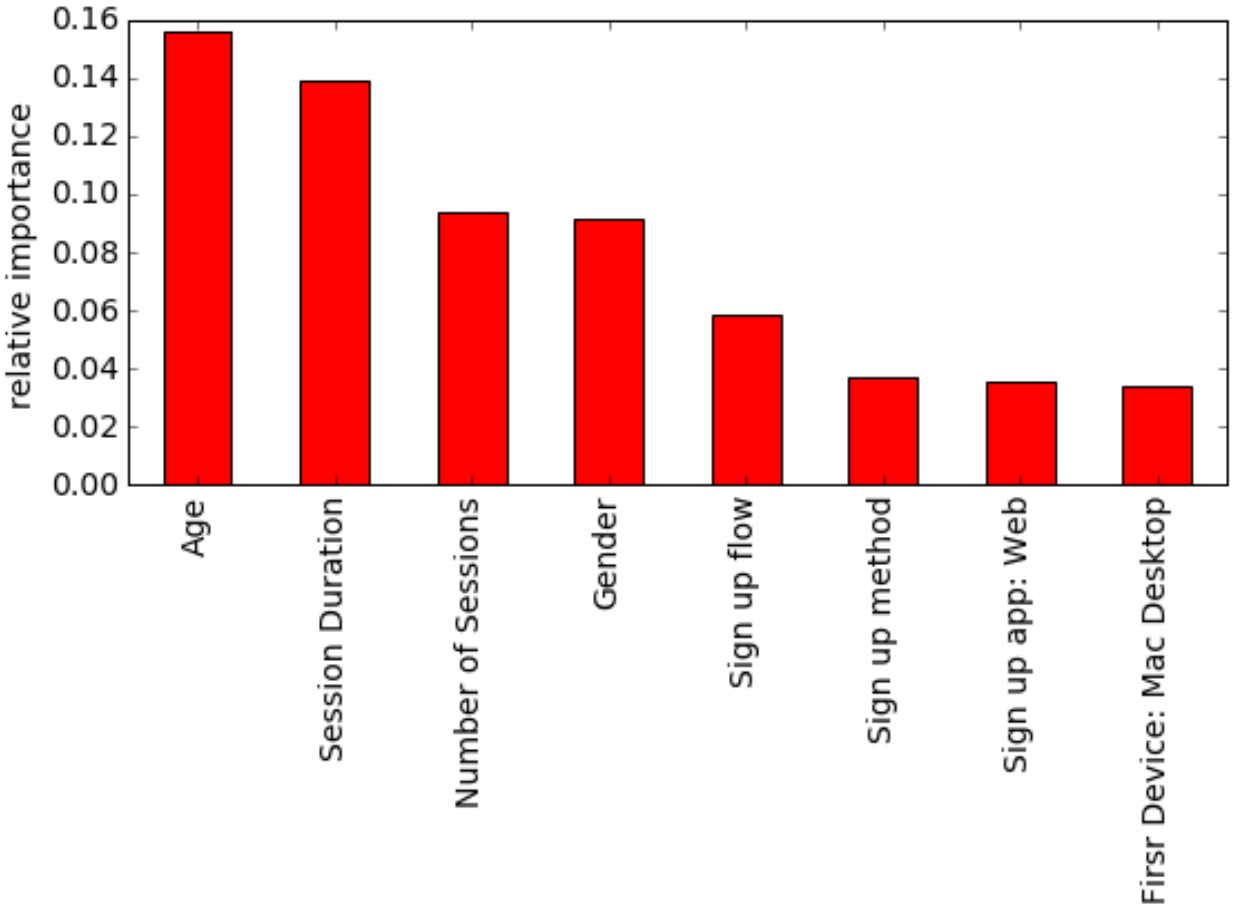


Figure 12: Fifteen most important features as identified by the XGBoost model

Importance scores calculated by the XGBoost model model confirms the findings of our exploratory data analysis. In particular, we see that fact that a user specified gender is a strong predictor. Number of sessions on Airbnb site recorded for a given user before booking is a strong predictor as well. Intuitively, users who visited the site multiple times are more likely to book. Further, web-users who are likely signed up via devices with large screens are likely to book as well.

6 Discussion

Deep learning can be viewed as a high dimensional nonlinear data reduction scheme. A Bayesian probabilistic model underlying DL is a stacked generalized linear model (GLM). Thus, it sheds light on training of a deep architecture using SGD, which is a first order gradient methods for finding a posterior mode is a very high dimensional space. By taking a predictive approach where regularization provides good answers, deep learning has been very successful.

There are many areas of future for Bayesian deep learning.

(i) By viewing deep learning probabilistically as stacked GLMs opens up many statistical models including exponential family models and heteroscedastic errors.

(ii) Bayesian hierarchical models have similar advantages to deep learners. Hierarchical models include extra stochastic layer and provide extra interpretability and flexibility.

(iii) Another avenue is combining proximal algorithms and MCMC.

(iv) With gradient information easily available via the chain rule (a.k.a. back propagation), a new avenue of stochastic methods to fit networks exists, such as MCMC, HMC, proximal methods, and ADMM, could dramatically speed up the time to train deep learners.

(v) Hyper-parameter tuning

(vi) Comparison with traditional Bayesian non-parametric approaches, such as treed Gaussian Models Gramacy [2005], and BART Chipman et al. [2010]. Using hyperplanes in Bayesian non-parametric methods ought to yield good predictors Francom [2017].

(vii) Deep learning has very well developed computational software that can be used for Bayesian calculations where pure MCMC is too slow.

(viii) Better Bayesian algorithms for for hyper-parameter training and optimization Tran et al. [2016]. Langevin diffusion MCMC, proximal MCMC and Hamiltonian Monte Carlo (HMC) can exploit the derivatives as well as Hessian information. Polson et al. [2015], Dean et al. [2012a].

Rather than searching a grid of values with a goal of minimising out-of-sample means squared error, one could place further regularisation penalties (priors) on these parameters and integrate them into the algorithm. The MCMC methods that have been developed in the past thirty years in the Bayesian community have lots to offer here. Given the availability of high performance computing, it is now possible to implement high dimensional posterior inference on large data sets, see Dean et al. [2012a]. The same advantages are now available for Bayesian inference. Further, we believe deep learning models have bright future in many fields of applications, such as finance, where DL is a form of nonlinear factor models Heaton et al. [2016a,b] with each layer capturing different tine scale effects and spatio-temporal data can also be viewed as an image in space-time and DL provides a pattern matching technique for recovering nonlinear complex relations Dixon et al. [2017], Polson and Sokolov [2017].

References

G. M. Henkin A. G. Vitushkin. Linear superpositions of functions. *Uspekhi Mat. Nauk*, 22, 1967.

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Y. Amit and D. Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9(7):1545–1588, July 1997.

Yali Amit, Gilles Blanchard, and Kenneth Wilder. Multiple randomized classifiers: Mrcl. 2000.

Leo Breiman. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, August 2001.

Donald W. Bryant. *Analysis of Kolmogorov's superpostion theorem and its implementation in applications with low and high dimensional data*. University of Central Florida, 2008.

Miguel A Carreira-Perpinán and Weiran Wang. Distributed optimization of deeply nested systems. In *AISTATS*, pages 10–19, 2014.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

Hugh A Chipman, Edward I George, Robert E McCulloch, et al. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large Scale Distributed Deep Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012a.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and others. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012b.

DeepMind. DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/, 2016.

DeepMind. The story of AlphaGo so far. https://deepmind.com/research/alphago/, 2017.

John Denker, Daniel Schwartz, Ben Wittner, Sara Solla, Richard Howard, Lawrence Jackel, and John Hopfield. Large automatic learning, rule extraction, and generalization. *Complex systems*, 1(5):877–922, 1987.

Persi Diaconis and David Freedman. A dozen de finetti-style results in search of a theory. In *Annales de l'IHP Probabilités et statistiques*, volume 23, pages 397–423, 1987.

Persi Diaconis and Mehrdad Shahshahani. Generating a random permutation with random transpositions. *Probability Theory and Related Fields*, 57(2):159–179, 1981.

Persi W Diaconis, David Freedman, et al. Consistency of bayes estimates for nonparametric regression: normal theory. *Bernoulli*, 4(4):411–444, 1998.

Matthew F. Dixon, Nicholas G. Polson, and Vadim O. Sokolov. Deep Learning for Spatio-Temporal Modeling: Dynamic Traffic Flows and High Frequency Trading. *arXiv:1705.09851 [stat]*, May 2017. arXiv: 1705.09851.

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

William Feller. *An introduction to probability theory and its applications*. Wiley, 1971. ISBN 978-0-471-25709-7.

Devin Francom. *BASS: Bayesian Adaptive Spline Surfaces*, 2017. URL https://CRAN.R-project.org/package=BASS. R package version 0.2.2.

Ildiko E Frank and Jerome H Friedman. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):109–135, 1993.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2, 2015.

Robert B Gramacy. *Bayesian treed Gaussian process models*. PhD thesis, University of California Santa Cruz, 2005.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer, New York, NY, 2nd edition edition, 2016. ISBN 978-0-387-84857-0.

JB Heaton, NG Polson, and JH Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016a.

JB Heaton, NG Polson, and JH Witte. Deep portfolio theory. *arXiv preprint arXiv:1605.07230*, 2016b.

Bo Jiang and Jun S Liu. Sliced inverse regression with variable selection and interaction detection. *arXiv preprint arXiv:1304.4056*, 652, 2013.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Bo'az Klartag. A central limit theorem for convex sets. *Inventiones mathematicae*, 168(1):91–131, 2007.

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *American Mathematical Society Translation*, 28(2):55–59, 1963.

Taylor Kubota. Artificial intelligence used to identify skin cancer. http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/, January 2017.

H. Lee. *Bayesian Nonparametrics via Neural Networks*. ASA-SIAM Series on Statistics and Applied Mathematics. Society for Industrial and Applied Mathematics, January 2004. ISBN 978-0-89871-563-7.

Ker-Chau Li. Sliced Inverse Regression for Dimension Reduction. 86(414):316–327, 1991.

David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Colin L Mallows. Some comments on Cp. *Technometrics*, 15(4):661–675, 1973.

Vitali D Milman and Gideon Schechtman. *Asymptotic theory of finite dimensional normed spaces: Isoperimetric inequalities in riemannian manifolds*, volume 1200. Springer, 2009.

Guido F. Montúfar and Jason Morton. When Does a Mixture of Products Contain a Product of Mixtures? *SIAM Journal on Discrete Mathematics*, 29(1):321–347, 2015.

Peter Müller and David Rios Insua. Issues in Bayesian Analysis of Neural Network Models. *Neural Computation*, 10(3):749–770, April 1998.

Radford M Neal. Bayesian learning via stochastic dynamics. *Advances in neural information processing systems*, pages 475–475, 1993.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to Construct Deep Recurrent Neural Networks. *arXiv:1312.6026 [cs, stat]*, December 2013. arXiv: 1312.6026.

T. Poggio. Deep Learning: Mathematics and Neuroscience. *A Sponsored Supplement to Science*, Brain-Inspired intelligent robotics: The intersection of robotics and neuroscience:9–12, 2016.

Nicholas G. Polson and Vadim O. Sokolov. Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, 79:1–17, June 2017.

Nicholas G. Polson, James G. Scott, Brandon T. Willard, and others. Proximal algorithms in statistics and machine learning. *Statistical Science*, 30(4):559–581, 2015.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

David A. Sprecher. A survey of solved and unsolved problems on superpositions of functions. *Journal of Approximation Theory*, 6(2):123–134, 1972.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

Anatoli Georgievich Vitushkin. Some properties of linear superpositions of smooth functions. In *Dokl. Akad. Nauk SSSR*, volume 156, pages 1003–1006, 1964.

Svante Wold, Michael Sjöström, and Lennart Eriksson. Pls-regression: a basic tool of chemometrics. *Chemometrics and intelligent laboratory systems*, 58(2):109–130, 2001.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.