Bayesian Analysis (2017) **12**, Number 4, pp. 1275–1304

# Deep Learning: A Bayesian Perspective

Nicholas G. Polson* and Vadim Sokolov†

**Abstract.** Deep learning is a form of machine learning for nonlinear high dimensional pattern matching and prediction. By taking a Bayesian probabilistic perspective, we provide a number of insights into more efficient algorithms for optimisation and hyper-parameter tuning. Traditional high-dimensional data reduction techniques, such as principal component analysis (PCA), partial least squares (PLS), reduced rank regression (RRR), projection pursuit regression (PPR) are all shown to be shallow learners. Their deep learning counterparts exploit multiple deep layers of data reduction which provide predictive performance gains. Stochastic gradient descent (SGD) training optimisation and Dropout (DO) regularization provide estimation and variable selection. Bayesian regularization is central to finding weights and connections in networks to optimize the predictive bias-variance trade-off. To illustrate our methodology, we provide an analysis of international bookings on Airbnb. Finally, we conclude with directions for future research.

**Keywords:** deep learning, machine learning, Artificial Intelligence, LSTM models, prediction, Bayesian hierarchical models, pattern matching, TensorFlow.

## 1 Introduction

Deep learning (DL) is a form of machine learning that uses hierarchical abstract layers of latent variables to perform pattern matching and prediction. Deep learners are probabilistic predictors where the conditional mean is a stacked generalized linear model (sGLM). The current interest in DL stems from its remarkable success in a wide range 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), and skin cancer diagnostics (Kubota, 2017; Esteva et al., 2017). Schmidhuber (2015) provides a comprehensive historical survey of deep learning and their applications.

Deep learning is designed for massive data sets with many high dimensional input variables. For example, Google's translation algorithm (Sutskever et al., 2014) uses $\sim$ 1–2 billion parameters and very large dictionaries. Computational speed is essential, and automated differentiation and matrix manipulations are available on `TensorFlow` Abadi et al. (2015). Baidu successfully deployed speech recognition systems (Amodei et al., 2016) with an extremely large deep learning model with over 100 million parameters, 11 layers and almost 12 thousand hours of speech for training. DL is an algorithmic approach rather than probabilistic in its nature, see Breiman (2001) for the merits of both approaches.

*University of Chicago, Booth School of Business, ngp@chicagobooth.edu
†George Mason University, Volgenau School of Engineering, vsokolov@gmu.edu

 DOI: 10.1214/17-BA1082

Our approach is Bayesian and probabilistic. We view the theoretical roots of DL in Kolmogorov's representation of a multivariate response surface as a superposition of univariate activation functions applied to an affine transformation of the input variable (Kolmogorov, 1963). An affine transformation of a vector is a weighted sum of its elements (linear transformation) plus an offset constant (bias). Our Bayesian perspective on DL leads to new avenues of research including faster stochastic algorithms, hyper-parameter tuning, construction of good predictors, and model interpretation.

On the theoretical side, we show how DL exploits a Kolmogorov's "universal basis". By construction, deep learning models are very flexible and gradient information can be efficiently calculated for a variety of architectures. On the empirical side, we show that the advances in DL are due to:

(i) New activation (a.k.a. link) functions, such as rectified linear unit ($\text{ReLU}(x) = \max(0, x)$), instead of sigmoid function

(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)

(iv) Deep learning has very well developed computational software where pure Markov Chain Monte Carlo (MCMC) is too slow.

To illustrate DL, we provide an analysis of a dataset from Airbnb on first time international bookings. Different statistical methodologies can then be compared, see Kaggle (2015) and Ripley (1994) who provides a comparison of traditional statistical methods with neural network based approaches for classification.

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 of many traditional statistical techniques, such as principal component analysis (PCA), principal component regression (PCR), sliced inverse regression (SIR), and linear discriminant analysis (LDA) which are shown to be "shallow learners" with two layers. Much of the recent success in DL applications has been achieved by including deeper layers and these gains pass over to traditional statistical models. Section 3 provides heuristics on why Bayes procedures provide good predictors in high dimensional data reduction problems. Section 4 describes how to train, validate and test deep learning models. We provide computational details associated with stochastic gradient descent (SGD). Section 5 provides an application to bookings data from the Airbnb website. Finally, Section 6 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,

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

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

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

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

Here $W$ and $z$ are the weight matrix and inputs of the $l$th layer.

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 a stochastic search technique known as dropout.

The offset vector is essential. For example, using $f(x) = \sin(x)$ without bias term $b$ would not allow to recover an even function like $\cos(x)$. 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 and Khenkin (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 a 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, are parameter dependent and learned from training data.

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

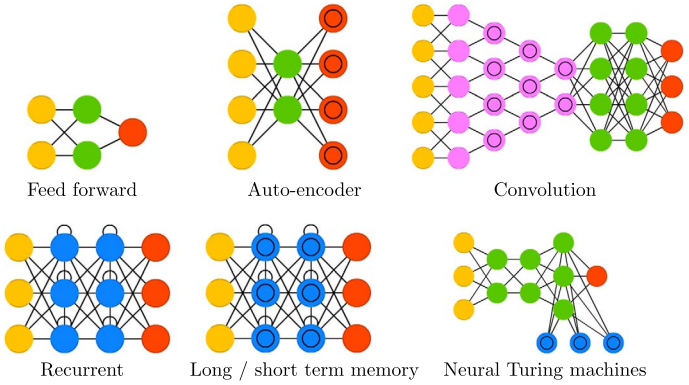



Figure 1: Most commonly used deep learning architectures. Each circle is a neuron which calculates a weighted sum of an input vector plus bias and applies a non-linear function to produce an output. Yellow and red colored neurons are input-output cells correspondingly. Pink colored neurons apply wights inputs using a kernel matrix. Green neurons are hidden ones. Blue neurons are recurrent ones and they append its values from previous pass to the input vector. Blue neuron with circle inside a neuron corresponds to a memory cell. 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), and neural Turing machines (NTM). Pascanu et al. (2013) and Montúfar and Morton (2015) provide results on the advantage of representing some functions compactly with deep layers. Poggio (2016) extends theoretical results on when deep learning can be exponentially better than shallow learning. Bryant (2008) implements Sprecher (1972) algorithm to estimate the non-smooth inner link function. In practice, deep layers allow for smooth activation functions to provide "learned" hyper-planes which find the underlying complex interactions and regions without having to see an exponentially large number of training samples.

## 2 Deep Probabilistic Learning

Probabilistically, the output $Y$ can be viewed as a random variable being generated by a probability model $p(Y|Y^{W,b}(X))$. Given $\hat{W}, \hat{b}$, the negative log-likelihood defines $\mathcal{L}$ as

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

The $L_2$-norm, $\mathcal{L}(Y_i, \hat{Y}(X_i)) = \|Y_i - \hat{Y}(X_i)\|_2^2$ is traditional least squares, and negative cross-entropy loss is $\mathcal{L}(Y_i, \hat{Y}(X_i)) = -\sum_{i=1}^{n} Y_i \log \hat{Y}(X_i)$ for multi-class logistic classification. The procedure to obtain estimates $\hat{W}, \hat{b}$ of the deep learning model parameters is described in Section 4.

To control the predictive bias-variance trade-off we add a regularization term and optimize

$$\mathcal{L}_\lambda(Y, \hat{Y}) = -\log p(Y|Y^{\hat{W},\hat{b}}(X)) - \log p(\phi(W, b) \mid \lambda).$$

Probabilistically this is a negative log-prior distribution over parameters, namely

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

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

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

Training requires the solution of a highly nonlinear optimization

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

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

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

Deep learning has the key property that $\nabla_{W,b} \log p(Y|Y^{W,b}(X))$ is computationally inexpensive 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, one caveat is that the posterior is highly multi-modal and providing good hyper-parameter tuning can be expensive. This is clearly a fruitful area of research for state-of-the-art stochastic Bayesian MCMC algorithms to provide more efficient algorithms. For shallow architectures, the alternating direction method of multipliers (ADMM) is an 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. This is achieved 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, $\mathcal{L}(Y, \hat{Y}) = \|Y - \hat{Y}\|_2^2$, then, when marginalizing over the randomness, we have a new objective

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

where $\star$ denotes the element-wise product. It is equivalent to, with $\Gamma = (\mathrm{diag}(X^\top X))^{\frac{1}{2}}$

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

Dropout then is simply Bayes 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 also be viewed as the optimization version of the traditional spike-and-slab prior, which 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

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

In effect, this replaces the input $X$ by $D \star X$, where $D$ is a matrix of independent $\mathrm{Ber}(p)$ 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, 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

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

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 (1994), Cook (2007); Hastie et al. (2016) provide further discussion of dimension reduction techniques. Other connections exists for Fisher's Linear Discriminant classification rule, which is simply fitting $H(wX + b)$, where $H$ is a Heaviside function. Polson et al. (2015) 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,

$$\begin{aligned} x_1x_2 &= \frac{1}{4}(x_1 + x_2)^2 - \frac{1}{4}(x_1 - x_2)^2, \\ \max(x_1, x_2) &= \frac{1}{2}|x_1 + x_2| + \frac{1}{2}|x_1 - x_2|, \\ (x_1x_2)^2 &= \frac{1}{4}(x_1 + x_2)^4 + \frac{7}{4 \cdot 3^3}(x_1 - x_2)^4 - \frac{1}{2 \cdot 3^3}(x_1 + 2x_2)^4 - \frac{2^3}{3^3}(x_1 + \frac{1}{2}x_2)^4. \end{aligned}$$

Diaconis and Shahshahani (1981) provide further discussion for Projection Pursuit Regression, where the network uses a layered model of the form $\sum_{i=1}^{N} f(w_i^\top X)$. 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 $j = 1, \ldots, N$

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

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

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

Writing our DL objective as an augmented Lagrangian (as in ADMM) with a hidden factor $Z$, leads to a two step algorithm, 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 $\hat{X} = \{Y, X\}$ than 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 focus 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

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

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

## 2.4 Bayesian Inference for Deep Learning

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).

A Bayesian extension of a feed forward network architectures has been considered by several authors (Neal, 1990; Saul et al., 1996; Frey and Hinton, 1999; Lawrence, 2005; Adams et al., 2010; Mnih and Gregor, 2014; Kingma and Welling, 2013; Rezende et al., 2014). Recent results show how dropout regularization can be used to represent uncertainty in deep learning models. In particular, Gal (2015) shows that dropout technique provides uncertainty estimates for the predicted values. The predictions generated by the deep learning models with dropout are nothing but samples from predictive posterior distribution.

Graphical models with deep learning encode a joint distribution via a product of conditional distributions and allow for computing (inference) many different probability distributions associated with the same set of variables. Inference requires the calculation of a posterior distribution over the variables of interest, given the relations between the variables encoded in a graph and the prior distributions. This approach is powerful when learning from samples with missing values or predicting with some missing inputs.

A classical example of using neural networks to model a vector of binary variables is the Boltzmann machine (BM), with two layers. The first layer encodes latent variables and the second layer encodes the observed variables. Both conditional distributions $p(\text{data} \mid \text{latent variables})$ and $p(\text{latent variables} \mid \text{data})$ are specified using logistic function parametrized by weights and offset vectors. The size of the joint distribution table grows exponentially with the number of variables and Hinton and Sejnowski (1983) proposed using Gibbs sampler to calculate update to model weights on each iteration. The multimodal nature of the posterior distribution leads to prohibitive computational times required to learn models of a practical size. Tieleman (2008) proposed a variational approach that replaces the posterior $p(\text{latent variables} \mid \text{data})$ and approximates

it with another easy to calculate distribution was considered in Salakhutdinov (2008). Several extensions to the BMs have been proposed. An exponential family extensions have been considered by Smolensky (1986); Salakhutdinov (2008); Salakhutdinov and Hinton (2009); Welling et al. (2005).

There have also been multiple approaches to building inference algorithms for deep learning models MacKay (1992); Hinton and Van Camp (1993); Neal (1992); Barber and Bishop (1998). Performing Bayesian inference on a neural network calculates the posterior distribution over the weights given the observations. In general, such a posterior cannot be calculated analytically, or even efficiently sampled from. However, several recently proposed approaches address the computational problem for some specific deep learning models (Graves, 2011; Kingma and Welling, 2013; Rezende et al., 2014; Blundell et al., 2015; Hernández-Lobato and Adams, 2015; Gal and Ghahramani, 2016).

The recent successful approaches to develop efficient Bayesian inference algorithms for deep learning networks are based on the reparameterization techniques for calculating Monte Carlo gradients while performing variational inference. Given the data $D = (X, Y)$, the variation inference relies on approximating the posterior $p(\theta \mid D)$ with a variation distribution $q(\theta \mid D, \phi)$, where $\theta = (W, b)$. Then $q$ is found by minimizing the based on the Kullback–Leibler divergence between the approximate distribution and the posterior, namely

$$\mathrm{KL}(q \mid\mid p) = \int q(\theta \mid D, \phi) \log \frac{q(\theta \mid D, \phi)}{p(\theta \mid D)} d\theta.$$

Since $p(\theta \mid D)$ is not necessarily tractable, we replace minimization of $\mathrm{KL}(q \mid\mid p)$ with maximization of evidence lower bound (ELBO)

$$\mathrm{ELBO}(\phi) = \int q(\theta \mid D, \phi) \log \frac{p(Y \mid X, \theta) p(\theta)}{q(\theta \mid D, \phi)} d\theta.$$

The *log* of the total probability (evidence) is then

$$\log p(D) = \mathrm{ELBO}(\phi) + \mathrm{KL}(q \mid\mid p).$$

The sum does not depend on $\phi$, thus minimizing $\mathrm{KL}(q \mid\mid p)$ is the same that maximizing $\mathrm{ELBO}(q)$. Also, since $\mathrm{KL}(q \mid\mid p) \geq 0$, which follows from Jensen's inequality, we have $\log p(D) \geq \mathrm{ELBO}(\phi)$. Thus, the evidence lower bound name. The resulting maximization problem $\mathrm{ELBO}(\phi) \to \max_\phi$ is solved using stochastic gradient descent.

To calculate the gradient, it is convenient to write the ELBO as

$$\mathrm{ELBO}(\phi) = \int q(\theta \mid D, \phi) \log p(Y \mid X, \theta) d\theta - \int q(\theta \mid D, \phi) \log \frac{q(\theta \mid D, \phi)}{p(\theta)} d\theta.$$

The gradient of the first term $\nabla_\phi \int q(\theta \mid D, \phi) \log p(Y \mid X, \theta) d\theta = \nabla_\phi E_q \log p(Y \mid X, \theta)$ is not an expectation and thus cannot be calculated using Monte Carlo methods. The idea is to represent the gradient $\nabla_\phi E_q \log p(Y \mid X, \theta)$ as an expectation of some random variable, so that Monte Carlo techniques can be used to calculate it. There are two

standard methods to do it. First, the log-derivative trick, uses the following identity $\nabla_x f(x) = f(x)\nabla_x \log f(x)$ to obtain $\nabla_\phi E_q \log p(Y \mid \theta)$. Thus, if we select $q(\theta \mid \phi)$ so that it is easy to compute its derivative and generate samples from it, the gradient can be efficiently calculated using Monte Carlo technique. Second, we can use reparametrization trick by representing $\theta$ as a value of a deterministic function, $\theta = g(\epsilon, x, \phi)$, where $\epsilon \sim r(\epsilon)$ does not depend on $\phi$. The derivative is given by

$$\begin{aligned}\nabla_\phi E_q \log p(Y \mid X, \theta) &= \int r(\epsilon)\nabla_\phi \log p(Y \mid X, g(\epsilon, x, \phi))d\epsilon \\ &= E_\epsilon[\nabla_g \log p(Y \mid X, g(\epsilon, x, \phi))\nabla_\phi g(\epsilon, x, \phi)].\end{aligned}$$

The reparametrization is trivial when $q(\theta \mid D, \phi) = N(\theta \mid \mu(D, \phi), \Sigma(D, \phi))$, and $\theta = \mu(D, \phi) + \epsilon\Sigma(D, \phi)$, $\epsilon \sim N(0, I)$. Kingma and Welling (2013) propose using $\Sigma(D, \phi) = I$ and representing $\mu(D, \phi)$ and $\epsilon$ as outputs of a neural network (multilayer perceptron), the resulting approach was called variational auto-encoder. A generalized reparametrization has been proposed by Ruiz et al. (2016) and combines both log-derivative and reparametrization techniques by assuming that $\epsilon$ can depend on $\phi$.

## 3 Finding Good Bayes Predictors

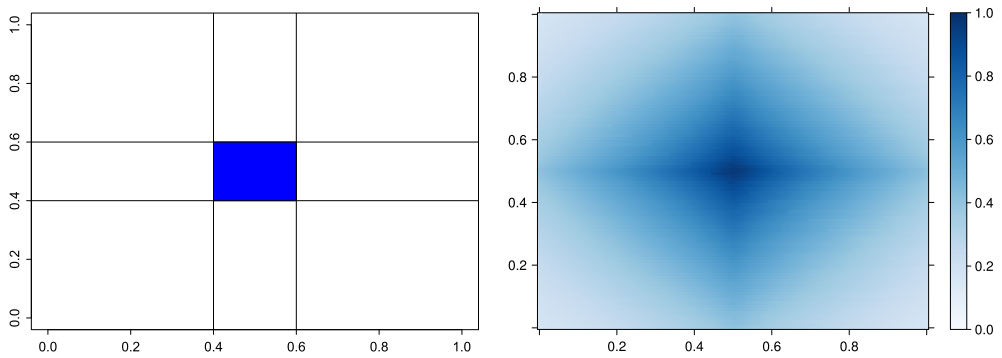
The Bayesian paradigm provides novel insights into how to construct estimators with good predictive performance. The goal is simply to find a good predictive MSE, namely $E_{Y,\hat{Y}}(\|\hat{Y} - Y\|^2)$, where $\hat{Y}$ denotes a prediction value. Stein shrinkage (a.k.a regularization with an $L^2$ norm) in known to provide good mean squared error properties in estimation, namely $E(||\hat{\theta} - \theta)||^2)$. These gains translate into predictive performance (in an iid setting) for $E(||\hat{Y} - Y||^2)$.

The main issue is how to tune the amount of regularisation (a.k.a prior hyperparameters). 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. In the context of DL, the posterior for $(W, b)$ is extremely high dimensional and multimodal and posterior MAP provides good predictors $\hat{Y}(X)$.

Bayes conditional averaging performs well in high dimensional regression and classification problems. High dimensionality, however, 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




(a) Tree Kernel (b) Random Forest Kernel

Figure 2: Kernel Weight. The intensity of the color is proportional to the size of the weight. Left panel (a) shows weights for tree-based model, with non-zero values only inside a cylindrical region (a box), and (b) shows weights for a random forest model, with non-zero wights everywhere in the domain and sizes decaying away from the location of the new observation.

implied kernels for trees (cylindrical sets) and random forests. Not too many points will be neighbors in a high dimensional input space.

Constructing the regions to preform conditional averaging is fundamental to reduce the curse of dimensionality. Imagine a large dataset, e.g. 100k images and think about how a new image's input coordinates, $X$, are "neighbors" to data points in the training set. Our predictor is a smart conditional average of the observed outputs, $Y$, from our neighbors. When $p$ is large, spheres ($L^2$ balls or Gaussian kernels) are terrible, degenerate cases occur when either no points or all of the points are "neighbors" of the new input variable will appear. Tree-based models address this issue by limiting the number of "neighbors.

Figure 3 further illustrates the challenge with 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 anywhere close to the boundary of the set.

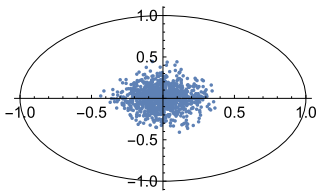
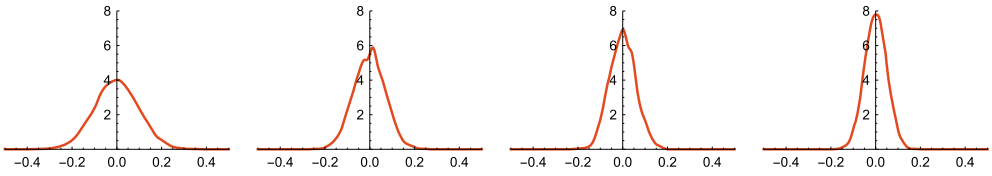



Figure 3: The 2D image of the Monte Carlo samples from a the 50-dimensional ball.

As dimensionality of the space grows, the variance of the marginal distribution goes to zero. Figure 4 shows the histogram of 1D image of uniform sample from balls of different dimensionality, that is $e_1^T Y$, where $e_1 = (1, 0, \ldots, 0)$.




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

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

Similar central limit results were known to Maxwell who has shown that the 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), see Diaconis and Freedman (1987). More general results in this direction were obtained in Klartag (2007) and Milman and Schechtman (2009) who presents many analytical and geometrical results for finite dimensional normed spaces, as the dimension grows to infinity.

Deep learning can improve on traditional methods by performing a sequence of GLM-like transformations. Effectively DL learns a distributed partition of the input space. For example, suppose that we have $K$ partitions and a DL predictor that takes the form of a weighted average or soft-max of the weighted average for classification. Given a new high dimensional input $X_{\text{new}}$, many deep learners are then an average of learners obtained by our hyper-plane decomposition. Our predictor takes the form

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

where $w_k$ are the weights learned in region $K$, and $k$ is an indicator of the region with appropriate weighting given the training data.

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, and Gaussian Processes. Each neuron in a deep learning model corresponds to a manifold that divides the input space. In the case of ReLU activation function $f(x) = \max(x, 0)$ the manifold is simply a hyperplane and the neuron gets activated 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.

The key difference 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.

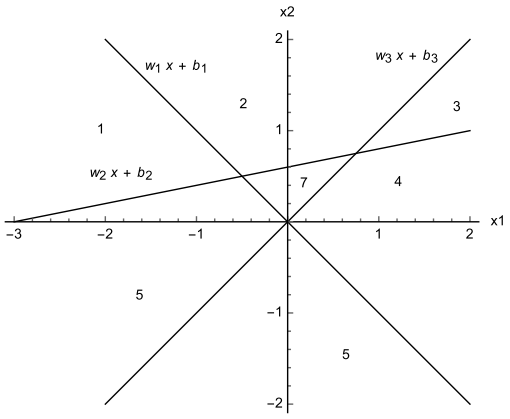



Figure 5: Hyperplanes defined by three neurons with ReLU activation functions.

We compare a neural network with two layers (bottom row) with tree mode trained with CART algorithm (top row). The network architecture is given by

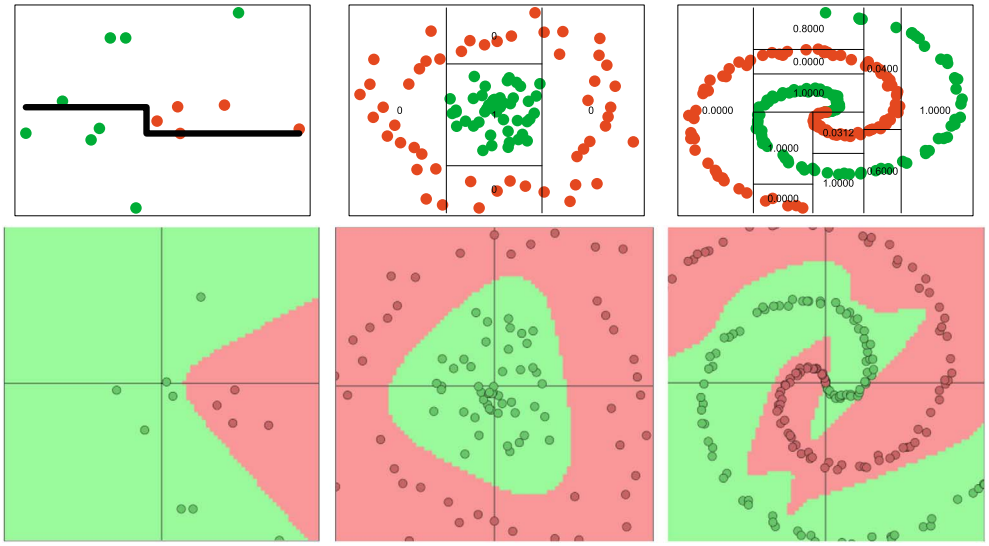
$$\begin{aligned} Y &= \text{softmax}(w^0 Z^2 + b^0), \\ Z^2 &= \tanh(w^2 Z^1 + b^2), \\ Z^1 &= \tanh(w^1 X + b^1)\ . \end{aligned}$$

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 than using $\text{softmax}(1/(1+e^{-x}))$ in deep learning $\tanh(x)$ is typically used for training, as $\tanh(0) = 0$.

Amit and Geman (1997) provide an interesting discussion of efficiency. Formally, a Bayesian probabilistic approach (if computationally feasible) optimally weights predictors via model averaging with $\hat{Y}_k(x) = E(Y \mid X_k)$

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

Such rules can achieve optimal out-of-sample performance. Amit et al. (2000) discusses 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 hand written digit 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%. This stems from the




(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.

fact that a sample from a very rich and diverse set of classifiers produces, on average, weakly dependent classifiers conditional on class.

To further exploit this, consider the Bayesian model of weak dependence, namely exchangeability. Suppose that we have $K$ exchangeable, $\mathbb{E}(\hat{Y}_i) = \mathbb{E}(\hat{Y}_{\pi(i)})$, and stacked predictors

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

Suppose that we wish to find weights, $w$, 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 equal weights $w = (1/K)\iota$ provides the optimal Bayes predictive performance.

We now turn to algorithmic issues.

## 4 Algorithmic Issues

In this section we discuss two types of algorithms for training learning models. First, stochastic gradient descent, which is a very general algorithm that efficiently works for large scale datasets and has been used for many deep learning applications. Second, we

discuss specialized statistical learning algorithms, which are tailored for certain types of traditional statistical models.

## 4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a default gold standard for minimizing the a 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

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

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, this leads to

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

At each iteration, SGD updates the solution

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

Deep learning algorithms use a 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 it 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 leads to non-linear alternating updates, see Carreira-Perpinán and Wang (2014).

The CD evaluates a single component $E_k$ of the gradient $\nabla f$ at the current point and then updates the $E_k$th 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). The momentum term 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 $\mu$ 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 be 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 dividing by zero. `PRMSprop` takes the `AdaGrad` idea further and places more weight on recent values of gradient squared to scale the update direction, i.e. we have

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

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

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

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

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

Here 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 its inability to train models using batches of data as SGD does. Since a typical deep learning model relies on large scale data sets, 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, $\{z_1, z_2, \ldots, z_K\}$,

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

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

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

Then, 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 (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 we analyze the data's covariance matrices and slice means of $X$, weighted by the size of slice. The function $G$ is found empirically by visually exploring relations. The key advantage of deep learning approach is that functional relation $G$ is found automatically. To extend 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 $Y$ via $Y = TBC^T$.

A deep learning least squares network arrives at a criterion function given by a negative log-posterior, which needs to be minimized. The penalized log-posterior, with $\phi$ denoting a generic regularization penalty is given by

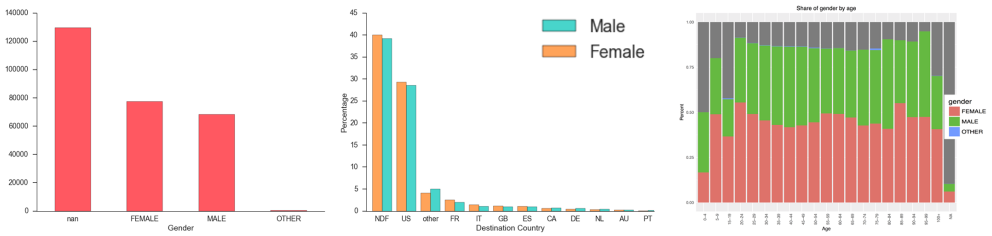
$$\begin{aligned} L(G, F) &= \sum_{i=1}^{n} \|y_i - g(F(x_i))\|_2 + \lambda_g \phi(G, F), \\ \phi_i &= \sum_k f_k^L(a_{ik}^L), \ a_{ik}^L = z_{ik}^L w^L, \ z_{ik}^L = \sum_j f^{L-1}(a_{jk}^{L-1}). \end{aligned}$$

Carreira-Perpinán and Wang (2014) propose a method of auxiliary coordinates which replaces the original unconstrained optimization problem, associated 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, for further discussion 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 our methodology, we use the dataset provided by the Airbnb Kaggle competition. This dataset whilst not designed to optimize the performance of DL provides a useful benchmark to compare and contrast traditional statistical models. The goal is to build a model that can predict which country a new user will make his or her first booking. Though Airbnb offers bookings in more than 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

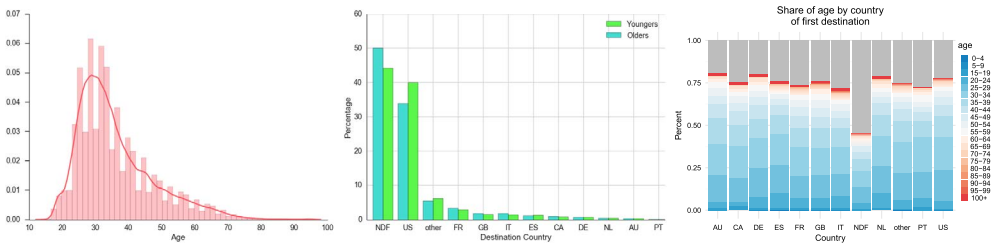
Figure 7: Gender and Destination Summary Plots for Airbnb users.

Figure 7(b) shows the country of origin for the first booking by gender. Most of the entries in the destination columns are NDF (not defined), 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.

We find that there is little difference in booking behavior between the genders. 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, the age variable does play a role.

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




(a) Empirical distribution of user's age

(b) Destination by age category

(c) Destination by age group

Figure 8: Age Information for Airbnb users.

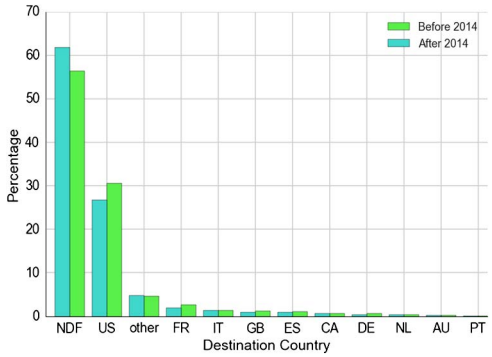



Figure 9: Booking behavior for users who opened their accounts before 2014.

and older than 45 cohort, (see 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 the 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 convert 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 a 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 $f(x) = \max(x, 0)$. 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). We use 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, then

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

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 mini-batch size of 256. For a hold-out sample we used 10% of the data, namely 21346 observations. The fitting algorithm evaluates the $DCG$ function at every epoch to monitor the improvements of quality of predictions from epoch to epoch. It takes approximately 10 minutes to train, whereas the variational inference approach is computationally prohibitive at this scale.

Our model uses 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.

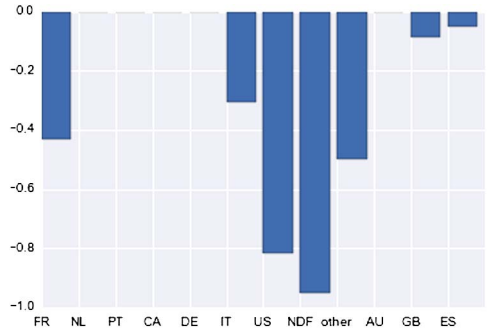
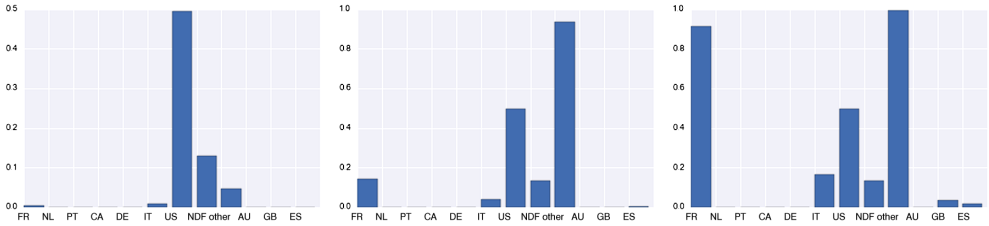



Figure 10: Out-of-sample NDCG for each of the destinations.




(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.

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

Figure 11 shows accuracy of prediction for each of the destination countries. The model accurately predicts bookings in the US and FR and other when top three predictions are considered.

Furthermore, we compared the performance of our 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 its ability to calculate the importance of each of the features (Hastie et al., 2016). Figure 12 shows the variable performance calculated from our XGBoost model.

The importance scores calculated by the XGBoost model confirm our exploratory data analysis findings. In particular, we see the 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 signed up via devices with large screens are also likely to book as well.

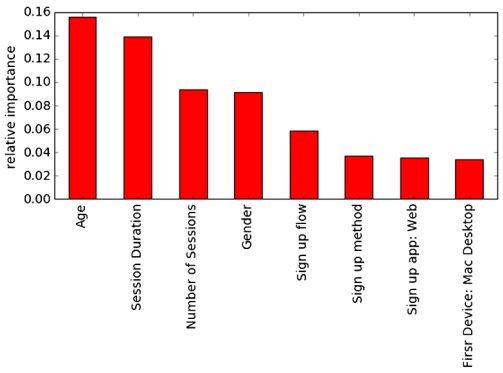



Figure 12: The fifteen most important features as identified by the XGBoost model.

## 6 Discussion

Our view of deep learning is a high dimensional nonlinear data reduction scheme, generated probabilistically as a stacked generalized linear model (GLM). This sheds light on how to train a deep architecture using SGD. This is a first order gradient method for finding a posterior mode in a very high dimensional space. By taking a predictive approach, where regularization learns the architecture, deep learning has been very successful in many fields.

There are many areas of future research for Bayesian deep learning which include

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

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

(iii) By viewing deep learning as a Gaussian Process allows for exact Bayesian inference Neal (1996); Williams (1997); Lee et al. (2017). The Gaussian Process connection opens opportunities to develop more flexible and interpretable models for engineering Gramacy and Polson (2011) and natural science applications Banerjee et al. (2008).

(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, which could dramatically speed up the time to train deep learners.

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

(vi) Improved Bayesian algorithms for hyper-parameter training and optimization (Snoek et al., 2012). Langevin diffusion MCMC, proximal MCMC and Hamiltonian Monte Carlo (HMC) can exploit the derivatives as well as Hessian information (Polson et al., 2015,b; Dean et al., 2012a).

(vii) 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 out.

MCMC methods also have lots to offer to DL and can be included seamlessly in `TensorFlow` (Abadi et al., 2015). Given the availability of high performance computing, it is now possible to implement high dimensional posterior inference on large data sets is now a possibility, see Dean et al. (2012a). The same advantages are now available for Bayesian inference. Further, we believe deep learning models have a bright future in many fields of applications, such as finance, where DL is a form of nonlinear factor models (Heaton et al., 2017), with each layer capturing different time scale effects and spatio-temporal data is viewed as an image in space-time (Dixon et al., 2017; Polson and Sokolov, 2017). In summary, the Bayes perspective adds helpful interpretability, however, the full power of a Bayes approach has still not been explored. From a practical perspective, current regularization approaches have provided great gains in predictive model power for recovering nonlinear complex data relationships.

## References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." Software available from tensorflow.org. URL `http://tensorflow.org/`. 1275, 1298

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). "A learning algorithm for Boltzmann machines." *Cognitive Science*, 9(1): 147–169. 1283

Adams, R., Wallach, H., and Ghahramani, Z. (2010). "Learning the structure of deep sparse graphical models." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 1–8. 1283

Amit, Y., Blanchard, G., and Wilder, K. (2000). "Multiple randomized classifiers: MRCL." 1288

Amit, Y. and Geman, D. (1997). "Shape Quantization and Recognition with Randomized Trees." *Neural Computation*, 9(7): 1545–1588. 1288

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). "Deep speech 2: End-

to-end speech recognition in english and mandarin." In *International Conference on Machine Learning*, 173–182. 1275

Banerjee, S., Gelfand, A. E., Finley, A. O., and Sang, H. (2008). "Gaussian predictive process models for large spatial data sets." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(4): 825–848. MR2523906. 1297

Barber, D. and Bishop, C. M. (1998). "Ensemble learning in Bayesian neural networks." *Neural Networks and Machine Learning*, 168: 215–238. 1284

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). "Weight uncertainty in neural networks." *arXiv preprint arXiv:1505.05424*. 1284

Breiman, L. (2001). "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)." *Statistical Science*, 16(3): 199–231. MR1874152. 1275

Bryant, D. W. (2008). *Analysis of Kolmogorov's superpostion theorem and its implementation in applications with low and high dimensional data*. University of Central Florida. 1278

Carreira-Perpinán, M. A. and Wang, W. (2014). "Distributed optimization of deeply nested systems." In *AISTATS*, 10–19. 1290, 1292

Chen, T. and Guestrin, C. (2016). "XGBoost: A Scalable Tree Boosting System." *CoRR*, abs/1603.02754. 1296

Chipman, H. A., George, E. I., McCulloch, R. E., et al. (2010). "BART: Bayesian additive regression trees." *The Annals of Applied Statistics*, 4(1): 266–298. 1297

Cook, R. D. (2007). "Fisher Lecture: Dimension Reduction in Regression." *Statistical Science*, 22(1): 1–26. 1281, 1282

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., and Ng, A. Y. (2012a). "Large Scale Distributed Deep Networks." In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, 1223–1231. Curran Associates, Inc. 1298

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., and others (2012b). "Large Scale Distributed Deep Networks." In *Advances in Neural Information Processing Systems*, 1223–1231. 1291

DeepMind (2016). "DeepMind AI Reduces Google Data Centre Cooling Bill by 40%." https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/. 1275

DeepMind (2017). "The story of AlphaGo so far." https://deepmind.com/research/alphago/. 1275

Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. (1987). "Large automatic learning, rule extraction, and generalization." *Complex Systems*, 1(5): 877–922. MR0931844. 1283

Diaconis, P. and Freedman, D. (1987). "A dozen de Finetti-style results in search of a theory." In *Annales de l'IHP Probabilités et statistiques*, volume 23, 397–423. MR0898502. 1287

Diaconis, P. and Shahshahani, M. (1981). "Generating a random permutation with random transpositions." *Probability Theory and Related Fields*, 57(2): 159–179. 1281

— (1984). "On Nonlinear Functions of Linear Combinations." *SIAM Journal on Scientific and Statistical Computing*, 5(1): 175–191. MR0731890. 1281

Diaconis, P. W., Freedman, D., et al. (1998). "Consistency of Bayes estimates for nonparametric regression: normal theory." *Bernoulli*, 4(4): 411–444. 1281

Dixon, M. F., Polson, N. G., and Sokolov, V. O. (2017). "Deep Learning for Spatio-Temporal Modeling: Dynamic Traffic Flows and High Frequency Trading." *arXiv:1705.09851 [stat]*. ArXiv: 1705.09851. 1298

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). "Dermatologist-level classification of skin cancer with deep neural networks." *Nature*, 542(7639): 115–118. 1275

Feller, W. (1971). *An introduction to probability theory and its applications*. Wiley. 1281

Francom, D. (2017). *BASS: Bayesian Adaptive Spline Surfaces*. R package version 0.2.2. URL `https://CRAN.R-project.org/package=BASS`. 1297

Frank, I. E. and Friedman, J. H. (1993). "A statistical view of some chemometrics regression tools." *Technometrics*, 35(2): 109–135. 1281

Frey, B. J. and Hinton, G. E. (1999). "Variational learning in nonlinear Gaussian belief networks." *Neural Computation*, 11(1): 193–213. 1283

Gal, Y. (2015). "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks." *arXiv:1512.05287*. 1283

Gal, Y. and Ghahramani, Z. (2016). "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning." In *International conference on machine learning*, 1050–1059. 1284

Gramacy, R. B. (2005). "Bayesian treed Gaussian process models." Ph.D. thesis, University of California Santa Cruz. 1297

Gramacy, R. B. and Polson, N. G. (2011). "Particle learning of Gaussian process models for sequential design and optimization." *Journal of Computational and Graphical Statistics*, 20(1): 102–118. MR2816540. 1297

Graves, A. (2011). "Practical variational inference for neural networks." In *Advances in Neural Information Processing Systems*, 2348–2356. 1284

Hastie, T., Tibshirani, R., and Friedman, J. (2016). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. New York, NY: Springer, 2nd edition. 1281, 1296

Heaton, J., Polson, N., and Witte, J. H. (2017). "Deep learning for finance: deep portfolios." *Applied Stochastic Models in Business and Industry*, 33(1): 3–12. 1282, 1298

Hernández-Lobato, J. M. and Adams, R. (2015). "Probabilistic backpropagation for scalable learning of Bayesian neural networks." In *International Conference on Machine Learning*, 1861–1869. 1284

Hinton, G. E. and Salakhutdinov, R. R. (2006). "Reducing the dimensionality of data with neural networks." *Science (New York, N.Y.)*, 313(5786): 504–507. MR2242509. 1280

Hinton, G. E. and Sejnowski, T. J. (1983). "Optimal perceptual inference." In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 448–453. IEEE New York. 1283

Hinton, G. E. and Van Camp, D. (1993). "Keeping the neural networks simple by minimizing the description length of the weights." In *Proceedings of the sixth annual conference on Computational learning theory*, 5–13. ACM. 1284

Jiang, B. and Liu, J. S. (2013). "Sliced inverse regression with variable selection and interaction detection." *arXiv preprint arXiv:1304.4056*, 652. 1292

Kaggle (2015). "Airbnb New User Bookings." https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings. Accessed: 2017-09-11. 1276

Kingma, D. and Ba, J. (2014). "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*. 1291

Kingma, D. P. and Welling, M. (2013). "Auto-encoding variational Bayes." *arXiv preprint arXiv:1312.6114*. 1283, 1284, 1285

Klartag, B. (2007). "A central limit theorem for convex sets." *Inventiones Mathematicae*, 168(1): 91–131. MR2285748. 1287

Kolmogorov, A. N. (1963). "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. 1276

Kubota, T. (2017). "Artificial intelligence used to identify skin cancer." http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/. 1275

Lawrence, N. (2005). "Probabilistic non-linear principal component analysis with Gaussian process latent variable models." *Journal of Machine Learning Research*, 6(Nov): 1783–1816. MR2249872. 1283

Lee, H. (2004). *Bayesian Nonparametrics via Neural Networks*. ASA-SIAM Series on Statistics and Applied Mathematics. Society for Industrial and Applied Mathematics. 1283

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein,

J. (2017). "Deep Neural Networks as Gaussian Processes." *arXiv preprint arXiv:1711.00165*. 1297

Li, K.-C. (1991). "Sliced Inverse Regression for Dimension Reduction." 86(414): 316–327.

MacKay, D. J. (1992). "A practical Bayesian framework for backpropagation networks." *Neural Computation*, 4(3): 448–472. 1283, 1284

Mallows, C. L. (1973). "Some comments on Cp." *Technometrics*, 15(4): 661–675. 1281

Milman, V. D. and Schechtman, G. (2009). *Asymptotic theory of finite dimensional normed spaces: Isoperimetric inequalities in riemannian manifolds*, volume 1200. Springer. MR0856576. 1287

Mnih, A. and Gregor, K. (2014). "Neural variational inference and learning in belief networks." *arXiv preprint arXiv:1402.0030*. 1283

Montúfar, G. F. and Morton, J. (2015). "When Does a Mixture of Products Contain a Product of Mixtures?" *SIAM Journal on Discrete Mathematics*, 29(1): 321–347. 1278

Müller, P. and Insua, D. R. (1998). "Issues in Bayesian Analysis of Neural Network Models." *Neural Computation*, 10(3): 749–770. 1283

Neal, R. M. (1990). "Learning stochastic feedforward networks." *Department of Computer Science, University of Toronto*, 64. 1283

Neal, R. M. (1992). "Bayesian training of backpropagation networks by the hybrid Monte Carlo method." Technical report, Technical Report CRG-TR-92-1, Dept. of Computer Science, University of Toronto. 1284

Neal, R. M. (1993). "Bayesian learning via stochastic dynamics." *Advances in Neural Information Processing Systems*, 475–475. 1283

— (1996). "Priors for infinite networks." In *Bayesian Learning for Neural Networks*, 29–53. Springer. 1297

Nesterov, Y. (1983). "A method of solving a convex programming problem with convergence rate O (1/k2)." In *Soviet Mathematics Doklady*, volume 27, 372–376. 1290

Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media. 1290

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). "How to Construct Deep Recurrent Neural Networks." *arXiv:1312.6026 [cs, stat]*. ArXiv: 1312.6026. 1278

Poggio, T. (2016). "Deep Learning: Mathematics and Neuroscience." *A Sponsored Supplement to Science*, Brain-Inspired intelligent robotics: The intersection of robotics and neuroscience: 9–12. 1275, 1278

Polson, N. G., Scott, J. G., Willard, B. T., and others (2015). "Proximal algorithms in statistics and machine learning." *Statistical Science*, 30(4): 559–581. 1281, 1292, 1298

Polson, N. G. and Sokolov, V. O. (2017). "Deep learning for short-term traffic flow prediction." *Transportation Research Part C: Emerging Technologies*, 79: 1–17. 1298

Polson, N. G., Willard, B. T., and Heidari, M. (2015b). "A statistical theory of deep learning via proximal splitting." *arXiv preprint arXiv:1509.06061*. 1298

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). "Stochastic backpropagation and approximate inference in deep generative models." *arXiv preprint arXiv:1401.4082*. 1283, 1284

Ripley, B. D. (1994). "Neural networks and related methods for classification." *Journal of the Royal Statistical Society. Series B (Methodological)*, 409–456. 1276, 1281

Ruiz, F. R., AUEB, M. T. R., and Blei, D. (2016). "The generalized reparameterization gradient." In *Advances in Neural Information Processing Systems*, 460–468. 1285

Salakhutdinov, R. (2008). "Learning and evaluating Boltzmann machines." *Tech. Rep., Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto*. 1284

Salakhutdinov, R. and Hinton, G. (2009). "Deep boltzmann machines." In *Artificial Intelligence and Statistics*, 448–455. 1284

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). "Mean field theory for sigmoid belief networks." *Journal of Artificial Intelligence Research*, 4: 61–76. 1283

Schmidhuber, J. (2015). "Deep learning in neural networks: An overview." *Neural Networks*, 61: 85–117. 1275

Simonyan, K. and Zisserman, A. (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition." 1275

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A. (1995). "Nonlinear black-box modeling in system identification: a unified overview." *Automatica*, 31(12): 1691–1724. 1281

Smolensky, P. (1986). "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." 194–281. Cambridge, MA, USA: MIT Press. 1284

Snoek, J., Larochelle, H., and Adams, R. P. (2012). "Practical bayesian optimization of machine learning algorithms." In *Advances in neural information processing systems*, 2951–2959. 1298

Sprecher, D. A. (1972). "A survey of solved and unsolved problems on superpositions of functions." *Journal of Approximation Theory*, 6(2): 123–134. MR0348347. 1278

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, 15(1): 1929–1958. 1280

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). "On the importance of initialization and momentum in deep learning." In *International conference on machine learning*, 1139–1147. 1291

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). "Sequence to sequence learning with neural networks." In *Advances in neural information processing systems*, 3104–3112. 1275

Tieleman, T. (2008). "Training restricted Boltzmann machines using approximations to the likelihood gradient." In *Proceedings of the 25th international conference on Machine learning*, 1064–1071. ACM. 1283

Vitushkin, A. G. and Khenkin, G. M. (1967). "Linear superpositions of functions." *Russian Mathematical Surveys*, 22(1): 77. 1277

Welling, M., Rosen-Zvi, M., and Hinton, G. E. (2005). "Exponential family harmoniums with an application to information retrieval." In *Advances in neural information processing systems*, 1481–1488. 1284

Williams, C. K. (1997). "Computing with infinite networks." In *Advances in neural information processing systems*, 295–301. 1297

Wold, H. (1956). "Causal inference from observational data: A review of end and means." *Journal of the Royal Statistical Society. Series A (General)*, 119(1): 28–61. MR0079863. 1281

Wold, S., Sjöström, M., and Eriksson, L. (2001). "PLS-regression: a basic tool of chemometrics." *Chemometrics and Intelligent Laboratory Systems*, 58(2): 109–130. 1292

Zeiler, M. D. (2012). "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701*. 1291