A Statistical View of Deep Learning

Shakir Mohamed

4 July 2015

I've taken to writing this series of posts on a statistical view of deep learning with two principal motivations in mind. The first was as a personal exercise to make concrete and to test the limits of the way that I think about and use deep learning in my every day work. The second, was to highlight important statistical connections and implications of deep learning that I have not seen made in the popular courses, reviews and books on deep learning, but which are extremely important to keep in mind. This document forms a collection of these essays originally posted at blog.shakirm.com.

CONTENTS

1

RECURSIVE GENERALISED LINEAR MODELS

Deep learning and the use of deep neural networks [1] are now established as a key tool for practical machine learning. Neural networks have an equivalence with many existing statistical and machine learning approaches and I would like to explore one of these views in this post. In particular, I'll look at the view of deep neural networks as recursive generalised linear models (RGLMs). Generalised linear models form one of the cornerstones of probabilistic modelling and are used in almost every field of experimental science, so this connection is an extremely useful one to have in mind. I'll focus here on what are called feed-forward neural networks and leave a discussion of the statistical connections to recurrent networks to another post.

1.1 GENERALISED LINEAR MODELS

The basic linear regression model is a linear mapping from P-dimensional input features (or covariates) x, to a set of targets (or responses) y, using a set of weights (or regression coefficients) β and a bias (offset) β_0 . The outputs can also by multivariate, but I'll assume they are scalar here. The full probabilistic model assumes that the outputs are corrupted by Gaussian noise of unknown variance σ^2.

$$\eta = \beta^\top x + \beta_0$$
$$y = \eta + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

In this formulation, η is the systematic component of the model and ϵ is the random component. Generalised linear models (GLMs)[2] allow us to extend this formulation to problems where the distribution on the targets is not Gaussian but some other distribution (typically a distribution in the exponential family). In this case, we can write the generalised regression problem, combining the coefficients and bias for more compact notation, as:

$$\eta = \beta^\top x, \qquad \beta = [\hat{\beta}, \beta_0], x = [\hat{x}, 1]$$

$$\mathbb{E}[y] = \mu = g^{-1}(\eta)$$

where $g(\cdot)$ is the link function that allows us to move from natural parameters η to mean parameters μ. If the inverse link function used in the definition of μ above were the logistic sigmoid, then the mean parameters correspond to the probabilities of y being a 1 or 0 under

Table 1: Correspondence between link and activations functions in generalised regression.

Target	Regression	Link	Inv link	Activation
Real	Linear	Identity	Identity	
Binary	Logistic	Logit $\log \frac{\mu}{1-\mu}$	Sigmoid $\frac{1}{1+\exp(-\eta)}$	Sigmoid
Binary	Probit	Inv Gauss CDF $\Phi^{-1}(\mu)$	Gauss CDF $\Phi(\eta)$	Probit
Binary	Gumbel	Compl. log-log $\log(-\log(\mu))$	Gumbel CDF $e^{-e^{-x}}$	
Binary	Logistic		Hyperbolic Tangent $\tanh(\eta)$	Tanh
Categorical	Multinomial		Multin. Logit $\frac{\eta_i}{\sum_j \eta_j}$	Softmax
Counts	Poisson	$\log(\mu)$	$\exp(\nu)$	
Counts	Poisson	$\sqrt{(}\mu)$	ν^2	
Non-neg.	Gamma	Reciprocal $\frac{1}{\mu}$	$\frac{1}{\nu}$	
Sparse	Tobit		max $\max(0;\nu)$	ReLU
Ordered	Ordinal		Cum. Logit $\sigma(\phi_k - \eta)$	

the Bernoulli distribution.

There are many link functions that allow us to make other distributional assumptions for the target (response) y. In deep learning, the link function is referred to as the activation function and I list in the table below the names for these functions used in the two fields. From this table we can see that many of the popular approaches for specifying neural networks that have counterparts in statistics and related literatures under (sometimes) very different names, such multinomial regression in statistics and softmax classification in deep learning, or rectifier in deep learning and tobit models is statistics.

1.2 RECURSIVE GENERALISED LINEAR MODELS

Constructing a recursive GLM or deep deep feed-forward neural network using the linear predictor as the basic building block. GLMS have a simple form: they use a linear combination of the input using weights β, and pass this result through a simple non-linear function. In deep learning, this basic building block is called a layer. It is easy to see that such a building block can be easily repeated to form more complex, hierarchical and non-linear regression functions. This recursive application of the basic regression building block is why models in deep learning are described as having multiple layers and are described as deep.

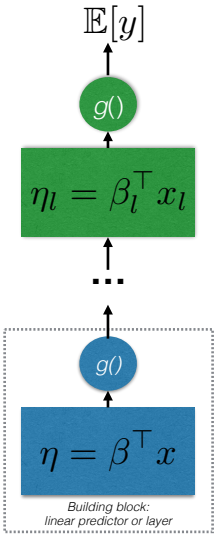

Figure 1: Constructing a recursive GLM or deep deep feedforward neural network using the linear predictor as the basic building block.

If an arbitrary regression function h, for layer l, with linear predictor , and inverse link or activation function f, is specified as:

$$h_l(x) = f_l(\eta_l)$$

then we can easily specify a recursive GLM by iteratively applying or composing this basic building block:

$$\mathbb{E}[y] = \mu_L = h_L \circ \ldots \circ h_1 \circ h_o(x)$$

This composition is exactly the specification of an L-layer deep neural network model. There is no mystery in such a construction (and hence in feedforward neural networks) and the utility of such a model is easy to see, since it allows us to extend the power of our regressors far beyond what is possible using only linear predictors.

This form also shows that recursive GLMs and neural networks are one way of performing basis function regression. What such a formulation adds is a specific mechanism by which to specify the basis functions: by application of recursive linear predictors.

1.3 LEARNING AND ESTIMATION

Given the specification of these models, what remains is an approach for training them, i.e. estimation of the regression parameters β for every layer. This is where deep learning has provided a great deal of insight and has shown how such models can be scaled to very high-dimensional inputs and on very large data sets.

A natural approach is to use the negative log-probability as the loss function and maximum likelihood estimation [3]:

$$\mathcal{L} = -\log p(y|\mu_L)$$

where if using the Gaussian distribution as the likelihood function we obtain the squared loss, or if using the Bernoulli distribution we obtain the cross entropy loss. Estimation or learning in deep neural networks corresponds directly to maximum likelihood estimation in recursive GLMs. We can now solve for the regression parameters by computing gradients w.r.t. the parameters and updating using gradient descent. Deep learning methods now always train such models using stochastic approximation (using stochastic gradient descent), using automated tools for computing the chain rule for derivatives throughout the model (i.e. back-propagation), and perform the computation on powerful distributed systems and GPUs. This allows such a model to be scaled to millions of data points and to very large models with potentially millions of parameters [4].

From the maximum likelihood theory, we know that such estimators can be prone to overfitting and this can be reduced by incorporating model regularisation, either using approaches such as penalised regression and shrinkage, or through Bayesian regression. The importance of regularisation has also been recognised in deep learning and further exchange here could be beneficial.

1.4 SUMMARY

Deep feed-forward neural networks have a direct correspondence to recursive generalised linear models and basis function regression in statistics – which is an insight that is useful in demystifying deep networks and an interpretation that does not rely on analogies to sequential processing in the brain. The training procedure is an application of (regularised) maximum likelihood estimation, for which we now have a large set of tools that allow us to apply these models to very large-scale, real-world systems. A statistical perspective on deep learning points to a broad set of knowledge that can be exchanged between the two fields, with the potential for further advances in efficiency and understanding of these regression problems. It is thus one I believe we all benefit from by keeping in mind. There are other viewpoints such as the connection to graphical models, or for recurrent networks, to dynamical systems, which I hope to think through in the future.

2

AUTO-ENCODERS AND FREE ENERGY

With the success of discriminative modelling using deep feedforward neural networks (or using an alternative statistical lens, recursive generalised linear models) in numerous industrial applications, there is an increased drive to produce similar outcomes with unsupervised learning. In this post, I'd like to explore the connections between denoising auto-encoders as a leading approach for unsupervised learning in deep learning, and density estimation in statistics. The statistical view I'll explore casts learning in denoising auto-encoders as that of inference in latent factor (density) models. Such a connection has a number of useful benefits and implications for our machine learning practice.

2.1 GENERALISED DENOISING AUTO-ENCODERS

Denoising auto-encoders are an important advancement in unsupervised deep learning, especially in moving towards scalable and robust representations of data. For every data point y, denoising auto-encoders begin by creating a perturbed version of it y', using a known corruption process $\mathcal{C}(\mathbf{y}'|\mathbf{y})$. We then create a network that given the perturbed data y', reconstructs the original data y. The network is grouped into two parts, an encoder and a decoder, such that the output of the encoder z can be used as a representation/features of the data. The objective function is [5]:

$$\text{Perturbation:} \quad \mathbf{y}' \sim \mathcal{C}(\mathbf{y}'|\mathbf{y})$$

$$\text{Encoder:} \quad \mathbf{z}(\mathbf{y}') = f_\phi(\mathbf{y}') \qquad \text{Decoder:} \quad \mathbf{y} \approx g_\theta(\mathbf{z})$$

$$\text{Objective:} \quad \mathcal{L}_{DAE} = \log p(\mathbf{y}|\mathbf{z})$$

where $\log p(\cdot)$ is an appropriate likelihood function for the data, and the objective function is averaged over all observations. Generalised denoising auto-encoders (GDAEs) realise that this formulation may be limited due to finite training data, and introduce an additional penalty term $\mathcal{R}(\cdot)$ for added regularisation [6]:

$$\mathcal{L}_{GDAE} = \log p(\mathbf{y}|\mathbf{z}) - \lambda\mathcal{R}(\mathbf{y}, \mathbf{y}')$$

GDAEs exploit the insight that perturbations in the observation space give rise to robustness and insensitivity in the representation

z. Two key questions that arise when we use GDAEs are: how to choose a realistic corruption process, and what are appropriate regularisation functions.

2.2 SEPARATING MODEL AND INFERENCE

The difficulty in reasoning statistically about auto-encoders is that they do not maintain or encourage a distinction between a model of the data (statistical assumptions about the properties and structure we expect) and the approach for inference/estimation in that model (the ways in which we link the observed data to our modelling assumptions). The auto-encoder framework provides a computational pipeline, but not a statistical explanation, since to explain the data (which must be an outcome of our model), you must know it beforehand and use it as an input. Not maintaining the distinction between model and inference impedes our ability to correctly evaluate and compare competing approaches for a problem, leaves us unaware of relevant approaches in related literatures that could provide useful insight, and makes it difficult for us to provide the guidance that allows our insights to be incorporated into our community's broader knowledge-base.

To ameliorate these concerns we typically re-interpret the auto-encoder by seeing the decoder as the statistical model of interest (and is indeed how many interpret and use auto-encoders in practice). A probabilistic decoder provides a generative description of the data, and our task is inference/learning in this model. For a given model, there are many competing approaches for inference, such as maximum likelihood (ML) and maximum a posteriori (MAP) estimation, noise-contrastive estimation, Markov chain Monte Carlo (MCMC), variational inference, cavity methods, integrated nested Laplace approximations (INLA), etc. The role of the encoder is now clear: the encoder is one mechanism for inference in the model described by the decoder. Its structure is not tied to the model (decoder), and it is just one from the smorgasbord of available approaches with its own advantages and tradeoffs.

2.3 APPROXIMATE INFERENCE IN LATENT VARIABLE MODELS

Encoder-decoder view of inference in latent variable models. Another difficulty with DAEs is that robustness is obtained by considering perturbations in the data space such a corruption process will, in general, not be easy to design. Furthermore, by carefully reasoning about the induced probabilities, we can show [5] that the DAE objective function $\mathcal{L}_{\mathrm{DAE}}$ corresponds to a lower bound obtained by applying the variational principle to the log-density of the corrupted data $\log p(\mathbf{y}')$ this though, is not a quantity we are interested in reasoning about.

A way forward would be to instead apply the variational principle

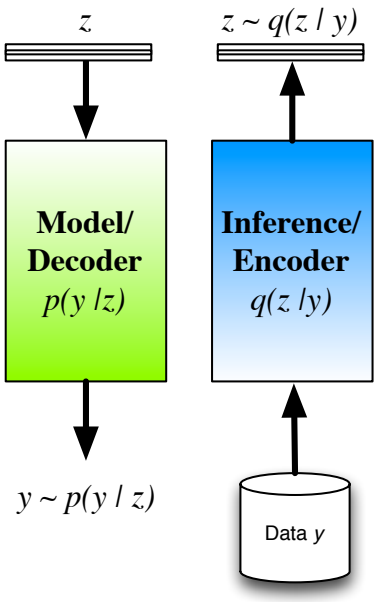

Figure 2: Encoder-decoder view of inference in latent variable models.

to the quantity we are interested in, the log-marginal probability of the observed data $\log p(\mathbf{y})$ [7] [8]. The objective function obtained by applying the variational principle to the generative model (probabilistic decoder) is known as the variational free energy:

$$\mathcal{L}_{\mathrm{VFE}} = \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{y}|\mathbf{z})] - \mathrm{KL}[q(\mathbf{z})\|p(\mathbf{z})]$$

By inspection, we can see that this matches the form of the GDAE objective. There are notable differences though:

- Instead of considering perturbations in the observation space, we consider perturbations in the hidden space, obtained by using a prior $p(z)$. The hidden variables are now random, latent variables. Auto-encoders are now generative models that are straightforward to sample from.
- The encoder $q(z|y)$ is a mechanism for approximating the true posterior distribution of the latent/hidden variables $p(z|y)$.
- We are now able to explain the introduction of the penalty function in the GDAE objective in a principled manner. Rather than designing the penalty by hand, we are able to derive the form this penalty should take, appearing as the KL divergence between the the prior and the encoder distribution.

Auto-encoders reformulated in this way, thus provide an efficient way of implementing approximate Bayesian inference. Using an encoder-decoder structure, we gain the ability to jointly optimise all parameters using the single computational graph; and we obtain an efficient way of doing inference at test time, since we only need a single forward pass through the encoder. The cost of taking this ap-

proach is that we have now obtained a potentially harder optimisation, since we have coupled the inferences for the latent variables together through the parameters of the encoder. Approaches that do not implement the q-distribution as an encoder have the ability to deal with arbitrary missingness patterns in the observed data and we lose this ability, since the encoder must be trained knowing the missingness pattern it will encounter. One way we explored these connections is in a model we called Deep Latent Gaussian Models (DLGM) with inference based on stochastic variational inference (and implemented using an encoder) [7], and is now the basis of a number of extensions [9] [10].

2.4 SUMMARY

Auto-encoders address the problem of statistical inference and provide a powerful mechanism for inference that plays a central role in our search for more powerful unsupervised learning. A statistical view, and variational reformulation, of auto-encoders allows us to maintain a clear distinction between the assumed statistical model and our approach for inference, gives us one efficient way of implementing inference, gives us an easy-to-sample generative model, allows us to reason about the statistical quantity we are actually interested in, and gives us a principled loss function that includes the important regularisation terms. This is just one perspective that is becoming increasingly popular, and is worthwhile to reflect upon as we continue to explore the frontiers of unsupervised learning.

3

MEMORY AND KERNELS

Memory, the ways in which we remember and recall past experiences and data to reason about future events, is a term used frequently in current literature. All models in machine learning consist of a memory that is central to their usage. We have two principal types of memory mechanisms, most often addressed under the types of models they stem from: parametric and non-parametric (but also all the shades of grey in-between). Deep networks represent the archetypical parametric model, in which memory is implemented by distilling the statistical properties of observed data into a set of model parameters or weights. The poster-child for non-parametric models would be kernel machines (and nearest neighbours) that implement their memory mechanism by actually storing all the data explicitly. It is easy to think that these represent fundamentally different ways of reasoning about data, but the reality of how we derive these methods points to far deeper connections and a more fundamental similarity.

Deep networks, kernel methods and Gaussian processes form a continuum of approaches for solving the same problem - in their final form, these approaches might seem very different, but they are fundamentally related, and keeping this in mind can only be useful for future research. This connection is what I explore in this post.

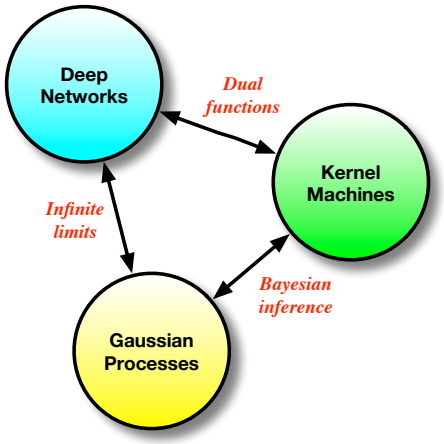

Figure 3: Connecting machine learning methods for regression.

3.1 BASIS FUNCTIONS AND NEURAL NETWORKS

All the methods in this post look at regression: learning discriminative or input-output mappings. All such methods extend the humble linear model, where we assume that linear combinations of the input data x, or transformations of it $\phi(x)$, explain the target values y. The $\phi(x)$ are basis functions that transform the data into a set of more interesting features. Features such as SIFT for images or MFCCs for audio have been popular in the past – in these cases, we still have a linear regression, since the basis functions are fixed. Neural networks give us the ability to use adaptive basis functions, allowing us to learn what the best features are from data instead of designing these by-hand, and allowing for a non-linear regression.

A useful probabilistic formulation separates the regression into systematic and random components: the systematic component is a function f we wish to learn, and the targets are noisy realisations of this function. To connect neural networks to the linear model, I'll explicitly separate the last linear layer of the neural network from the layers that appear before it. Thus for an L-layer deep neural network, I'll denote the first $L-1$ layers by the mapping $\phi(x;\theta)$ with parameters θ, and the final layer weights w; the set of all model parameters is $q = \{\theta, w\}$.

$$\text{Sytematic:} \quad f = \mathbf{w}^\top \phi(\mathbf{x};\theta) \qquad \mathbf{q} \sim \mathcal{N}(0, \sigma_q^2 \mathbf{I}),$$

$$\text{Random:} \quad y = f(\mathbf{x}) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_y^2)$$

Once we have specified our probabilistic model, this implies an objective function for optimising the model parameters given by the negative log joint-probability. We can now apply back-propagation and learn all the parameters, performing MAP estimation in the neural network model. Memory in this model is maintained in the parametric modelling framework; we do not save the data but compactly represent it by the parameters of our model. This formulation has many nice properties: we can encode properties of the data into the function f, such as being a 2D image for which convolutions are sensible, and we can choose to do a stochastic approximation for scalability and perform gradient descent using mini-batches instead of the entire data set. The loss function for the output weights is of particular interest, since it will offers us a way to move from neural networks to other types of regression.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^\top \phi(\mathbf{x}_n;\theta))^2 + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}.$$

3.2 KERNEL METHODS

If you stare a bit longer at this last objective function, especially as formulated by explicitly representing the last linear layer, you'll very

quickly be tempted to compute its dual function [11, pp. 293]. We'll do this by first setting the derivative w.r.t. w to zero and solving for it:

$$\nabla J(\mathbf{w}) = 0 \implies \mathbf{w} = \frac{1}{\lambda}\sum_n (y_n - \mathbf{w}^\top \phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

$$\mathbf{w} = \sum_n \alpha_n \phi(\mathbf{x}_n) = \mathbf{\Phi}^\top \boldsymbol{\alpha} \qquad \alpha_n = -\frac{1}{\lambda}(\mathbf{w}^\top \phi(\mathbf{x}_n) - y_n)$$

We've combined all basis functions/features for the observations into the matrix Φ. By taking this optimal solution for the last layer weights and substituting it into the loss function, two things emerge: we obtain the dual loss function that is completely rewritten in terms of a new parameter α, and the computation involves the matrix product or Gram matrix $K = \Phi\Phi^\top$. We can repeat the process and solve the dual loss for the optimal parameter , and obtain:

$$\nabla J(\boldsymbol{\alpha}) = 0 \implies \boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$$

And this is where the kernel machines deviate from neural networks. Since we only need to consider inner products of the features $\phi)x)$ (implied by maintaining K), instead of parameterising them using a non-linear mapping given by a deep network, we can use kernel substitution (aka, the kernel trick) and get the same behaviour by choosing an appropriate and rich kernel function $k(x, x')$. This highlights the deep relationship between deep networks and kernel machines: they are more than simply related, they are duals of each other.

The memory mechanism has now been completely transformed into a non-parametric one - we explicitly represent all the data points (through the matrix K). The advantage of the kernel approach is that is is often easier to encode properties of the functions we wish to represent e.g., functions that are up to p-th order differentiable or periodic functions, but stochastic approximation is now not possible. Predictions for a test point x* can now be written in a few different ways:

$$f = \mathbf{w}_{MAP}^\top \phi(\mathbf{x}^*) = \boldsymbol{\alpha}^\top \mathbf{\Phi}(\mathbf{x})\phi(\mathbf{x}^*) = \sum_n \alpha_n k(\mathbf{x}^*, \mathbf{x}_n) = k(\mathbf{X}, \mathbf{x}^*)^\top (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}$$

The last equality is a form of solution implied by the Representer theorem and shows that we can instead think of a different formulation of our problem: one that directly penalises the function we are trying to estimate, subject to the constraint that the function lies within a Hilbert space (and providing a direct non-parametric view):

$$J(f) = \frac{1}{2}\sum_{n=1}^{N} (y_n - f(\mathbf{x}_n))^2 + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2.$$

3.3 GAUSSIAN PROCESSES

We can go even one step further and obtain not only a MAP estimate of the function f, but also its variance. We must now specify a probability model that yields the same loss function as this last objective function. This is possible since we now know what a suitable prior over functions is, and this probabilistic model corresponds to Gaussian process (GP) regression [12]:

$$p(f) = \mathcal{N}(\mathbf{0}, \mathbf{K}) \qquad p(y|f) = \mathcal{N}(y|f, \lambda)$$

We can now apply the standard rules for Gaussian conditioning to obtain a mean and variance for any predictions x^*. What we obtain is:

$$p(f^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \mathcal{N}(\mathbb{E}[f^*], \mathbb{V}[f^*])$$

$$\mathbb{E}[f^*] = k(\mathbf{X}, \mathbf{x}^*)^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

$$\mathbb{V}[f^*] = k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{X}, \mathbf{x}^*)^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} k(\mathbf{X}, \mathbf{x}^*)$$

Conveniently, we obtain the same solution for the mean whether we use the kernel approach or the Gaussian conditioning approach. We now also have a way to compute the variance of the functions of interest, which is useful for many problems (such as active learning and optimistic exploration). Memory in the GP is also of the non-parametric flavour, since our problem is formulated in the same way as the kernel machines. GPs form another nice bridge between kernel methods and neural networks: we can see GPs as derived by Bayesian reasoning in kernel machines (which are themselves dual functions of neural nets), or we can obtain a GP by taking the number of hidden units in a one layer neural network to infinity [13].

3.4 SUMMARY

Deep neural networks, kernel methods and Gaussian processes are all different ways of solving the same problem - how to learn the best regression functions possible. They are deeply connected: starting from one we can derive any of the other methods, and they expose the many interesting ways in which we can address and combine approaches that are ostensibly in competition. I think such connections are very interesting, and should prove important as we continue to build more powerful and faithful models for regression and classification.

4

RECURRENT NETWORKS AND DYNAMICAL SYSTEMS

Recurrent neural networks (RNNs) are now established as one of the key tools in the machine learning toolbox for handling large-scale sequence data. The ability to specify highly powerful models, advances in stochastic gradient descent, the availability of large volumes of data, and large-scale computing infrastructure, now allows us to apply RNNs in the most creative ways. From handwriting generation, image captioning, language translation and voice recognition, RNNs now routinely find themselves as part of large-scale consumer products.

On a first encounter, there is a mystery surrounding these models. We refer to them under many different names: as recurrent networks in deep learning, as state space models in probabilistic modelling, as dynamical systems in signal processing, and as autonomous and non-autonomous systems in mathematics. Since they attempt to solve the same problem, these descriptions are inherently bound together and many lessons can be exchanged between them: in particular, lessons on large-scale training and deployment for big data problems from deep learning, and even more powerful sequential models such as changepoint, factorial or switching state-space models. This post is an initial exploration of these connections.

4.1 RECURRENT NEURAL NETWORKS

Recurrent networks [14] take a functional viewpoint to sequence modelling. They describe sequence data using a function built using recursive components that use feedback from hidden units at time points in the past to inform computations of the sequence at the present. What we obtain is a neural network where activations of one of the

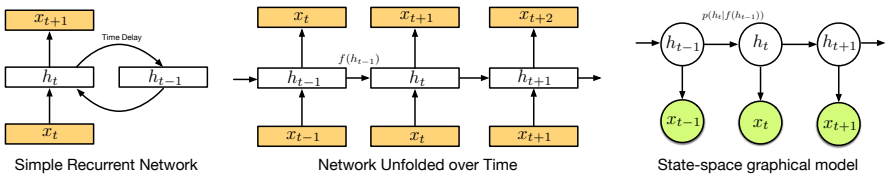

Figure 4: Equivalent models: recurrent networks and state-space models.

hidden layers feeds back into the network along with the input (see figures). Such a recursive description is unbounded and to practically use such a model, we unfold the network in time and explicitly represent a fixed number of recurrent connections. This transforms the model into a feed-forward network for which our familiar techniques can be applied.

If we consider an observed sequence x, we can describe a loss function for RNNs unfolded for T steps as:

$$\text{Feedback} : \mathbf{h_t} = \mathbf{f}(\mathbf{h_{<t}}, \mathbf{x_{t-1}})$$

$$\text{Loss} : J(\theta) = \sum_{t=1}^{T} d\left(\mathbf{x_t}, \mathbf{h_t}\right)$$

The model and corresponding loss function is that of a feed-forward network, with $d(\cdot)$ an appropriate distance function for the data being predicted, such as the squared loss. The difference from standard feed-forward networks is that the parameters of the recursive function f are the same for all time points, i.e. they are shared across the model. We can perform parameter estimation by averaging over a mini-batch of sequences and using stochastic gradient descent with application of the backpropagation algorithm. For recurrent networks, this combination of unfolding in time and backpropagation is referred to as backpropagation through time (BPTT) [15].

Since we have simplified our task by always considering the learning algorithm as the application of SGD and backprop, we are free to focus our energy on creative specifications of the recursive function. The simplest and common recurrent networks use feedback from one past hidden layer earlier examples include the Elman or Jordan networks. But the true workhorse of current recurrent deep learning is the Long Short-Term Memory (LSTM) network [16]. The transition function in an LSTM produces two hidden vectors: a hidden layer h, and a memory cell c, and applies the function f composed of soft-gating using sigmoid functions $\sigma(\cdot)$ and a number of weights and biases (e.g., A, B, a, b):

$$\text{Input} : \mathbf{i}_t = \sigma(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{h}_{t-1} + \mathbf{D}\mathbf{c}_{t-1} + \mathbf{a})$$

$$\text{Forget} : \mathbf{f}_t = \sigma(\mathbf{E}\mathbf{x}_t + \mathbf{F}\mathbf{h}_{t-1} + \mathbf{G}\mathbf{c}_{t-1} + \mathbf{b})$$

$$\text{Cell} : \mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh(\mathbf{H}\mathbf{x}_t + \mathbf{G}\mathbf{h}_{t-1} + \mathbf{d})$$

$$\text{Output} : \mathbf{o}_t = \sigma(\mathbf{K}\mathbf{x}_t + \mathbf{L}\mathbf{h}_{t-1} + \mathbf{M}\mathbf{c}_t + \mathbf{e})$$

$$\text{Hidden} : \mathbf{h}_t = \mathbf{o}_t \tanh(\mathbf{c}_t)$$

We can also view the recurrent network construction above using a probabilistic framework (relying on reasoning used in part I of this series). Instead of viewing the recurrent network as a recursive function followed by unfolding for T time steps, we can directly model a sequence of length T with latent (or hidden) dynamics and specify a probabilistic graphical model. Both the latent states h and the observed data x are assumed to be probabilistic. The transition probability is the same for all time, so this is equivalent to assuming the parameters of the transition function are shared. We could refer to these models as stochastic recurrent networks; the established convention is to refer to them as dynamical systems or state-space models.

In probabilistic modelling, the core quantity of interest is the probability of the observed sequence x, computed as follows:

$$p(\mathbf{x}_1, \dots \mathbf{x}_T) = \prod_t \int p(\mathbf{x}_t, \mathbf{h}_t) d\mathbf{h}_t$$

$$p(\mathbf{x}_t, \mathbf{h}_t) = p(\mathbf{x}_t|\mathbf{h}_t) p(\mathbf{h}_t|\mathbf{h}_{t-1})$$

Using maximum likelihood estimation, we can obtain a loss function based on the log of this marginal likelihood. Since for recurrent networks the transition dynamics is assumed to be deterministic, we can easily recover the RNN loss function:

$$\text{Det. Dynamics} : p_\theta(\mathbf{h}_t|\mathbf{h}_{t-1}) = \delta\left(\mathbf{h}_t = f_\theta(\mathbf{h}_{t-1}, \mathbf{x}_{t-1})\right)$$

$$\text{Loss} : J(\theta) = \sum_t \log \int p(\mathbf{h}_t|\mathbf{h}_{t-1}) p(\mathbf{x}_t|\mathbf{h}_t) d\mathbf{h}_t$$

$$\implies J(\theta) = \sum_t \log p(\mathbf{x}_t|f_\theta(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}))$$

which recovers the original loss function with the distance function given by the log of the chosen likelihood function. It is no surprise that the RNN loss corresponds to maximum likelihood estimation with deterministic dynamics.

As machine learners we never really trust our data, so in some cases we will wish to consider noisy observations and stochastic transitions. We may also wish to explore estimation beyond maximum likelihood. A great deal of power is obtained by considering stochastic transitions that transform recurrent networks into probabilistic generative temporal models [17, 18] models that account for missing data, allow for denoising and built-in regularisation, and that model the sequence density. We gain new avenues for creativity in our transitions: we can now consider states that jump and random times between different operational modes, that might reset to a base state, or that interact with multiple sequences simultaneously.

But when the hidden states h are random, we are faced with the problem of inference. For certain assumptions such as discrete or Gaussian transitions, algorithms for hidden Markov models and Kalman filters, respectively, demonstrate ways in which this can be done. More recent approaches use variational inference or particle MCMC [17]. In general, efficient inference for large-scale state-space models remains an active research area.

4.3 PREDICTION, FILTERING AND SMOOTHING

Dynamical systems are often described to make three different types of inference problems explicit: prediction, filtering and smoothing [18].

- **Prediction (inferring the future)** is the first use of most machine learning models. Having seen training data we are asked to forecast the behaviour of the sequence at some point k timesteps in the future. Here, we compute the predictive distribution of the hidden state, since knowing this allows us to predict or generate what would be observed:

$$p(\mathbf{h}_{t+k}|\mathbf{y}_{1,\dots t})$$

- **Filtering (inferring the present)** is the task of computing the marginal distribution of the hidden state given only the past states and observations.

$$p(\mathbf{h}_t|\mathbf{y}_{1,\dots,t})$$

- **Smoothing (inferring the past)** is the task of computing the marginal distribution of the hidden state given knowledge of the past and future observations.

$$p(\mathbf{h}_t|\mathbf{y}_{1,\dots,T}), t < T$$

.

These operations neatly separate the different types of computations that must be performed to correctly reason about the sequence with random hidden states. For RNNs, due to their deterministic nature, computing predictive distributions and filtering are realised by the feed-forward operations in the unfolded network. Smoothing is an operation that does not have a counterpart, but architectures such as bi-directional recurrent nets attempt to fill this role.

4.4 SUMMARY

Recurrent networks and state space models attempt to solve the same problem: how to best reason from sequential data. As we continue

research in this area, it is the intersection of deterministic and probabilistic approaches that will allow us to further exploit the power of these temporal models. Recurrent networks have been shown to be powerful, scalable, and applicable to an incredibly diverse set of problems. They also have much to teach in terms of initialisation, stability issues, gradient management and the implementation of large-scale temporal models. Probabilistic approaches have much to offer in terms of better regularisation, different types of sequences we can model, and the wide range of probabilistic queries we can make with models of sequence data. There is much more that can be said, but these initial connections make clear the way forward.

5

GENERALISATION AND REGULARISATION

We now routinely build complex, highly-parameterised models in an effort to address the complexities of modern data sets. We design our models so that they have enough 'capacity', and this is now second nature to us using the layer-wise design principles of deep learning. But some problems continue to affect us, those that we encountered even in the low-data regime, the problem of overfitting and seeking better generalisation.

The classical description of deep feedforward networks in part 1 or of recurrent networks in part 4 established maximum likelihood as the the underlying estimation principle for these models. Maximum Likelihood (ML) [19] is an elegant, conceptually simple and easy to implement estimation framework. And it has several statistical advantages, including consistency and asymptotic efficiency. Deep learning has shown just how effective ML can be. But it is not without its disadvantages, the most prominent being a tendency for overfitting. Overfitting is the problem of all statistical sciences, and ways of dealing with this are abound. The general solution reduces to considering an estimation framework other than maximum likelihood this penultimate post explores some of the available alternatives.

5.1 REGULARISERS AND PRIORS

The principle technique for addressing overfitting in deep learning is by regularisation adding additional penalties to our training objective that prevents the model parameters from becoming large and from fitting to the idiosyncrasies of the training data. This transforms our estimation framework from maximum likelihood into a maximum penalised likelihood, or more commonly maximum a posteriori (MAP) estimation (or a shrinkage estimator). For a deep model with loss function $L(\theta)$ and parameters θ, we instead use the modified loss that includes a regularisation function $\mathcal{R}$:

$$L(\theta) = -\sum_n \log p(y_n|x_n, \theta) + \tfrac{1}{\lambda}\mathcal{R}(\theta)$$

λ is a regularisation coefficient that is a hyperparameter of the model. It is also commonly known that this formulation can be derived by considering a probabilistic model that instead of a penalty,

Table 2: Common priors and regularisers

Name	$\mathcal{R}(\subseteq)$	$p(\theta)$
L2/Gaussian/Weight Decay	$\frac{1}{\lambda}\|\theta\|_2^2$	$\mathcal{N}(\theta\|0;\lambda)$
L1/Laplace/Lasso	$\frac{1}{\lambda}\|\theta\|_1$	$\mathcal{L}\dashv\sqrt{(\theta\|0;\lambda)}$
p-norms	$\|\theta\|_p; p > 0$	$exp(-\lambda\|\theta\|_p)$
Total variation	$\lambda\|\Delta\theta\|; \Delta\theta = (\theta_j - \theta_{j-1})$	
Fused Lasso	$\alpha\|\theta\| + \beta\|\Delta\theta\|$	
Cauchy	$-\sum_i \log(\theta_i^2 + \gamma^2)$	$\frac{1}{\pi\gamma}\frac{\gamma^2}{(\theta-\mu)^2+\gamma^2}$

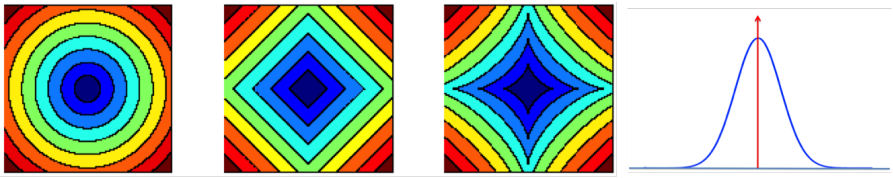

Figure 5: Contours showing the shrinkage effects of different priors.

introduces a prior probability distribution over the parameters. The loss function is the negative of the log joint probability distribution:

$$\log p(y, \theta) = \sum_n \log p(y_n|x_n, \theta) + \log p(\theta|\lambda)$$

The table shows some common regularisers, of which the L1 and L2 penalties are used in deep learning. Most other regularisers in the probabilistic literature cannot be added as a simple penalty function, but are instead given by a hierarchical specification (and whose optimisation is also more involved, requiring some form of alternating optimisation). Amongst the most effective are the sparsity inducing penalties such as Automatic Relevance Determination, the Normal-Inverse Gaussian, the Horseshoe, and the general class of Gaussian scale-mixtures.

5.2 INVARIANT MAP ESTIMATORS

While these regularisers may prevent overfitting to some extent, the underlying estimator still has a number of disadvantages. One of these is that MAP estimators are not invariant to smooth reparameterisations of the model. MAP estimators reason only using the density of the posterior distribution on parameters and their solution thus depends arbitrarily on the units of measurement we use. The effect of this is that we get very different gradients depending on our units, with different scaling and behaviour that impacts our optimisation. The most general way of addressing this is to reason about the entire distribution on parameters instead. Another approach is to design an invariant MAP estimator [20], where we instead maximise the modified probabilistic model:

$$p(x|\theta)p(\theta)|I(\theta)|^{-\frac{1}{2}}$$

where $I(\theta)$ is the Fisher information matrix. It is the introduction of the Fisher information that gives us the transformation invariance, although using this objective is not practically feasible (requiring up to 3rd order derivatives). But this is an important realisation that highlights an important property we seek in our estimators. Inspired by this, we can use the Fisher information in other ways to obtain invariant estimators (and better-behaved gradients). This builds the link to, and highlights the importance of the natural gradient in deep learning, and the intuition and use of the minimum message length from information theory [21].

5.3 DROPOUT: WITH AND WITHOUT INFERENCE

Since the L2 regularisation corresponds to a Gaussian prior assumption on the parameters, this induces a Gaussian distribution on the hidden variables of a deep network. It is thus equally valid to introduce regularisation on the hidden variables directly. This is what dropout [22], one of the major innovations in deep learning, uses to great effect. Dropout also moves a bit further away from MAP estimation and closer to a Bayesian statistical approach by using randomness and averaging to provide robustness.

Consider an arbitrary linear transformation layer of a deep network with link/activation function $\sigma(\cdot)$, input $\mathbf{h}$, parameters $\mathbf{W}$ and the dimensionality of the hidden variable D. Rather than describing the computation as a warped linear transformation, dropout uses a modified probabilistic description. For $i = 1, ...D$, we have two types of dropout:

$$\text{Bernoulli: } z_i \sim \mathcal{B}ern(z_i|\pi_i) \quad \pi_i = \tfrac{1}{2}(default)$$

$$\text{Gaussian: } z_i \sim \mathcal{N}(z_i|1, \sigma^2) \quad \sigma^2 = 1(default)$$

$$\text{Dropout Layer: } \mathbf{y} = \sigma(\mathbf{W}(\mathbf{h} \circ \mathbf{z}) + \mathbf{b})$$

In the Bernoulli case, we draw a $1/0$ indicator for every variable in the hidden layer and include the variable in the computation if it is 1 and drop it out otherwise. The hidden units are now random and we typically call such variables latent variables. Dropout introduces sparsity into the latent variables, which in recent times has been the subject of intense focus in machine learning and an important way to regularise models. A feature of dropout is that it assumes that the the dropout (or sparsity probability) is always known and fixed for the training period. This makes it simple to use and has shown to provide an invaluable form of regularisation.

You can view the indicator variables z as a way of selecting which

of the hidden features are important for computation of the current data point. It is natural to assume that the best subset of hidden features is different for every data point and that we should find and use the best subset during computation. This is the default viewpoint in probabilistic modelling, and when we make this assumption the dropout description above corresponds to an equally important tool in probabilistic modelling that of models with spike-and-slab priors [23]. A corresponding spike-and-slab-based model, where the indicator variables are called the spikes and the hidden units, the slabs, would be:

$$\text{Spike and Slab: } z_i \sim \mathcal{B}\text{ern}(z_i|\pi_i) \qquad p(\mathbf{y}|\mathbf{z},\mathbf{h},\boldsymbol{\Sigma}) = \prod_i \mathcal{N}(y_i|z_i h_i, z_i \sigma_i^2)$$

We can apply spike-and-slab priors flexibly: it can be applied to individual hidden variables, to groups of variables, or to entire layers. In this formulation, we must now infer the sparsity probability $p(z|y,h)$ this is the hard problem dropout sought to avoid by assuming that the probability is always known. Nevertheless, there has been much work in the use of models with spike-and-slab priors and their inference, showing that these can be better than competing approaches [24]. But an efficient mechanism for large-scale computation remains elusive.

5.4 SUMMARY

The search for more efficient parameter estimation and ways to overcome overfitting leads us to ask fundamental statistical questions about our models and of our chosen approaches for learning. The popular maximum likelihood estimation has the desirable consistency properties, but is prone to overfitting. To overcome this we moved to MAP estimation that help to some extent, but its limitations such as lack of transformation invariance leads to scale and gradient sensitivities that we can seek to ameliorate by incorporating the Fisher information into our models. We could also try other probabilistic regularisers whose unknown distribution we must average over. Dropout is one way of achieving this without dealing with the problem of inference, but were we to consider inference, we would happily use spike-and-slab priors. Ideally, we would combine all types of regularisation mechanisms, those that penalise both the weights and activations, assume they are random and that average over their unknown configuration. There are many diverse views on this issue; all point to the important research still to do.

6

WHAT IS DEEP?

Throughout this series, we have discussed deep networks by examining prototypical instances of these models, e.g., deep feed-forward networks, deep auto-encoders, deep generative models, but have not yet interrogated the key word we have been using. We have not posed the question what does 'deep' mean, and what makes a model deep. There is little in way of a detailed discussion as to what constitutes a 'deep' model and can be hard to do — it seems appropriate as a closing attempt to provide one such view.

Arguably, deep learning today means much more than a description of a class of useful models. It espouses the use of powerful non-linear models, models that provide unified loss functions that allow for end-to-end training, machine learning approaches designed from the beginning to be scalable and amenable to large data sets, and to computational methods that fully exploit modern computing resources. While these other factors have been most remarkably demonstrated with deep learning, these are goals shared with all other areas of machine learning. What is of central importance is 'deep' as a characterisation of models and their desirable features.

6.1 DEEP AND HIERARCHICAL MODELS

If we look into the existing literature, deep learning is generally described as the machine learning of deep models. And a deep model is any model that involves multiple levels of computation, in particular, computation achieved by the repeated application of non-linear transformations [25] . This is a general framework and the number of transformations used forms the depth of the model. This is well-suited as a description of neural networks (or recursive GLMs), since we can easily construct a model by recursively applying a linear transformation followed by an element-wise non-linearity, allowing us to move from linear regression models that use only one (non-)linear transformation (so called 'shallow' models) to more complex non-linear regressions that use three or more non-linear transformations (i.e. 'deep' models).

To provide a statistical view, we need a slightly more precise framework — and will use that of hierarchical models. As a start, we will characterise deep feed-forward models and then generalise from this

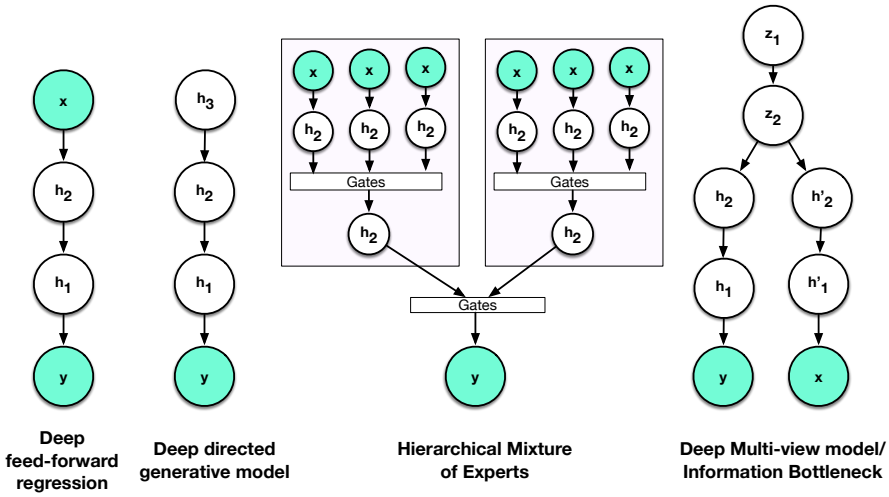

Figure 6: Deep and Hierarchical models are abound in machine learning.

type of model. Feed-forward networks are constructed from a succession of non-linear transformations that form a mapping from inputs x to targets y with parameters θ. Representing each transformation using a 'layer' $\mu_l(\mathbf{z}) = f_l(\mathbf{W}\mathbf{z})$, the final output is obtained by the composition of layers:

$$!y = \mu_L \circ \mu_{L-1} \circ \ldots \circ \mu_0(\mathbf{x}); \quad \theta = \{\mathbf{W}\}_{l=0}^{L}$$

For sigmoidal networks, the activation function f is a sigmoid function. We were previously able to recast this model as a general probabilistic regression model $p(y|g(x;\theta))p(\theta)$, corresponding to a familiar regularised deep neural network whose loss, with regulariser $\mathcal{R} = \log p(\theta)$, is:

$$\mathcal{L} = \log p(y|g(x;\theta)) + \mathcal{R}(\theta)$$

As a probabilistic model, we could instead think of the output of every layer μ_l as a random (latent) variable, with the property that the expectation of a layer is given by the non-linear transformation, and the sequence of expectations producing the output:

$$\mathbb{E}[h_l] = \boldsymbol{\mu}_l = f_l(\mathbf{W}^{(l)}\mathbf{z}) \quad \mathbb{E}[y] = \boldsymbol{\mu}_L = \mathbb{E}[\mathbb{E}[\ldots\mathbb{E}[h_0(\mathbf{x})]]]$$

It is this characterisation that brings us to hierarchical models: models where its (prior) probability distributions can be decomposed into a sequence of conditional distributions [26, Ch. 10]:

$$p(z) = p(z_1|z_2)p(z_2|z_3)\ldots p(z_{L-1}|z_L)p(z_L)$$

This specification implies that the prior is composed of a sequence of stochastic computations, and satisfies the aspirations we established for deep models. A hierarchical construction is not restricted to regression and is a general framework for all models, including

density estimation, time-series models, spatial statistics, etc. Our sigmoidal networks can instead be written as the following hierarchical regression model:

$$p(y|g(x;\theta)) = p(y|h_L)\text{Bern}(h_L|\mathbf{W}_{L-1}\mathbf{h}_{L-1}))\dots\text{Bern}(h_1|\mathbf{W}_0\mathbf{x}_0))p(\theta)$$

At layer l, the inputs from the previous layer

$$h_l$$

are transformed using a linear mapping into natural parameters of the Bernoulli distribution (into the pre-synaptic activations). Since we perform maximum likelihood estimation in the canonical or mean parameters of this model, there is an implicit transformation of the natural parameters using a sigmoid function — a link function for the Bernoulli distribution.

The conclusion from this is that one way to view deep feed-forward networks are as hierarchical probabilistic models. What is important though, is that this hierarchy is a hierarchy formed through the means of the layer-distributions. Only the mean parameters at every layer of the hierarchy depend on computations from previous parts of the hierarchy, i.e. hierarchies whose dependency is through the first-order structure at every layer of the model.

6.2 CHARACTERISING DEEP MODELS

Almost all models we use in deep learning are models formed through hierarchies of the mean. Deep generative models are another popular model class with this characteristic. One widely-known example is a Sigmoid belief network (SBN), a deep directed graphical models with Bernoulli latent variables. The hierarchical formulation through the means is:

$$p(\mathbf{h}_2) = \text{Bern}(\mathbf{h}_2|\boldsymbol{\pi}) \quad p(\mathbf{h}_1|\mathbf{h}_2) = \text{Bern}(\mathbf{h}_1|W_2\mathbf{h}_2) \quad p(\mathbf{x}|\mathbf{h}_1) = \text{Bern}(\mathbf{x}|W_1\mathbf{h}_1)$$

Other examples are not hard to find:

- **Non-linear Gaussian belief networks** (NLGBNs) follow the same hierarchy as SBNs, but use Gaussian latent variables, and form the hierarchy through the Gaussian means. And is closely related to hierarchical ICA.
- **Deep Latent Gaussian Models** (DLGMs) [7] and **Deep Autoregressive Networks** (DARN)[27] form their hierarchy through the means of Gaussian and auto-regressive Bernoulli distributions, respectively.
- **Deep Gaussian Processes**, a non-parametric analog of the NLGBNs, are formed through a hierarchical dependency through its mean functions.

- **Deep Exponential Families** (DEF), similar to the description above for deep feed-forward networks, construct a hierarchical model using one-parameter exponential families. This single (canonical) parameter controls all moments of the distribution and often directly encodes the mean, so any hierarchy formed in this way is a hierarchical model of the means.

- **Deep Boltzmann Machines** (DBM) are graphical undirected models (i.e. all conditional probabilities and restrictions are fully specified by its graphical depiction) also form hierarchical log-linear models using one-parameter exponential families.

The intuition we obtain from deep learning is that every stage of computation, every non-linear transformation, allows us to form increasingly abstract representations of the data. Statistically, every hidden layer allows us to capture more long-range and higher order correlations in the data (after being integrated out). In either view, these hierarchies are important since they provide a highly efficient way to construct complex models, e.g., in a mixture model we can use a 2-layer hierarchy using K and L clusters at each layer, effectively modelling K^L clusters — something infeasible with a standard (flat) mixture model. Parts of our hierarchy far away from the data are indeed more abstract (diffuse and close to the prior), since they have a small effect on the data distribution: this implies that to effectively learn with and use deep models, we require large amounts of data.

6.3 BEYOND HIERARCHIES OF THE MEAN

Deep models forming their hierarchies through the mean parameters are amongst the most powerful and flexible models in the machine learning toolbox. If you are going to build any hierarchical model, a hierarchy through the mean is a very good idea indeed. There are two aspects that follow from this: firstly, there are many models that are formed through mean-hierarchies that are not labelled as deep; secondly, a hierarchy through the mean represents just one way to build such a hierarchical model.

There are many other interesting models that are formed through hierarchical constructions, and some include:

- **Hierarchies on variances:** this is a natural step and is used in many Bayesian models where learning variances is involved. This does raise interesting research questions as to what assumptions and distributions to use beyond the simple one-parameter exponential families that are widely used.

- **Hierarchical mixture models, mixed-membership models and admixture models:** These models form a mixture of mixture-models. These are not typically called deep, though they could be called that. As mentioned above, we can easily can represent K^L mixture components using such constructions. They show

how different representations can be combined in useful ways, e.g., if the mixture is over deep feed-forward networks. And some other interesting instances:

- Bayesian networks and multi-level regression models.
- Hierarchical Dirichlet processes.
  As Wray Buntine points out (see comment on original post) these are also hierarchical models of the mean, and points to another aspect that has not been discussed: that of distributed and partitioned representations. Deep learning emphasises distributed representations (using multivariate continuous latent variables), and models such as the HDP show how both distributed and partitioned representations can be used together to provide powerful and adaptive models.
- Canonical Correlation, Information Bottleneck and multi-view models.
- Multi-level spike-and-slab models.

6.4 SUMMARY

One way to characterise models described as deep are as hierarchical models of means: hierarchical models where the mean at every layer of the hierarchy depends on computation in previous parts of the hierarchy. This is a correspondence we find in almost all models characterised as deep in our current practice, whether these be deep neural networks, deep latent Gaussian models, deep exponential families or deep Gaussian processes. This is always our first approach in building modern machine learning models, since we can capture a great deal of the structure underlying our data in this way. But we also know how to extend our hierarchies in ways that allow us to specify other structural aspects we may be interested in. While it is conceptually easy to extend our hierarchies in many ways, techniques for dealing with hierarchies other than the mean in computationally efficient ways are still missing, and remains one of the important research questions that we face in machine learning.

A SHORT REVIEW

Each post is necessarily short since my aim was to test how concrete I could frame my thinking within around 1200 words (posts are on average 1500 words though). Thus there are many more discussions, references and connections that could be added , and is one limitation of these essays. I do not explicitly discuss convolutional networks anywhere. Since convolution is a special linear operation we will not need any special reasoning to form a statistical view. What does require more reasoning is the statistical connections to pooling operations and something I'll hopefully cement in the future. The invariant MAP estimators discussed in part 5 show that you could get an update rule that will involve the inverse Fisher, but which is different from that obtained using the natural gradient, and is a connection that I was unable to establish directly. I did not provide many examples of the ways that popular deep and statistical methods can be combined. Kernel methods (in part 3) and deep learning can easily be combined by parameterising the kernel with a neural network, giving the best of both worlds. I have chosen to view dropout (in part 5) as a prior assumption that does not require inference, and connected this to spike-and-slab priors. But there are many other views that are complementary and valid for this, making a longer discussion of just this topic something for the future.

I have enjoyed writing and learning through this series; it has been a wonderful exploration and fun to write. Thanks to the many people who have read, shared and sent me feedback.

BIBLIOGRAPHY

[1] C. M. Bishop, "Neural networks for pattern recognition," 1995.

[2] P. McCullagh and J. A. Nelder, "Generalized linear models.," 1989.

[3] P. J. Bickel and K. A. Doksum, "Mathematical statistics, volume i," 2001.

[4] L. Bottou, *Stochastic Gradient Descent Tricks*. 2012.

[5] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[6] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Advances in Neural Information Processing Systems*, pp. 899–907, 2013.

[7] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of The 31st International Conference on Machine Learning*, pp. 1278–1286, 2014.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2014.

[9] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

[10] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[11] C. M. Bishop, *Pattern recognition and machine learning*, vol. 4, ch. Kernel Methods, p. 293. springer New York, 2006. pp. 293.

[12] C. E. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[13] R. M. Neal, *Bayesian Learning for Neural Networks*, ch. Priors for Infinite Networks, pp. 29–53. 1994.

[14] Y. Bengio, I. Goodfellow, and A. Courville, *Deep Learning*. MIT Press (To appear), 2015.

[15] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[16] F. Gers, *Long short-term memory in recurrent neural networks*. PhD thesis, cole Polytechnique Fdrale de Lausanne, Lausanne, Switzerland, 2011.

[17] D. Barber, A. T. Cemgil, and S. Chiappa, *Bayesian time series models*. Cambridge University Press, 2011.

[18] S. Sarkka, *Bayesian filtering and smoothing*, vol. 3. Cambridge University Press, 2013.

[19] L. Le Cam, "Maximum likelihood: an introduction," *International Statistical Review/Revue Internationale de Statistique*, pp. 153–171, 1990.

[20] P. Druilhet, J.-M. Marin, *et al.*, "Invariant {HPD} credible sets and {MAP} estimators," *Bayesian Analysis*, vol. 2, no. 4, pp. 681–691, 2007.

[21] I. H. Jermyn, "Invariant bayesian estimation on manifolds," *Annals of statistics*, pp. 583–605, 2005.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[23] H. Ishwaran and J. S. Rao, "Spike and slab variable selection: frequentist and bayesian strategies," *Annals of Statistics*, pp. 730–773, 2005.

[24] S. Mohamed, Z. Ghahramani, and K. A. Heller, "Bayesian and l1 approaches for sparse unsupervised learning," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 751–758, 2012.

[25] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[26] C. Robert, *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

[27] K. Gregor, I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra, "Deep autoregressive networks," in *Proceedings of The 31st International Conference on Machine Learning*, 2014.