Proc. 2013 International Conference on Neural Information Processing

Training Neural Networks with Implicit Variance

Justin Bayer, Christian Osendorfer, Sebastian Urban, and Patrick van der Smagt

Technische Universität München, Fakultät für Informatik, Lehrstuhl für Robotik und Echtzeitsysteme, Boltzmannstraße 3, 85748 München

Abstract. We present a novel method to train predictive Gaussian distributions $p(z|x)$ for regression problems with neural networks. While most approaches either ignore or explicitly model the variance as another response variable, it is trained implicitly in our case. Establishing stochasticty by the injection of noise into the input and hidden units, the outputs are approximated with a Gaussian distribution by the forward propagation method introduced for fast dropout [1]. We have designed our method to respect that probabilistic interpretation of the output units in the loss function. The method is evaluated on a synthetic and a inverse robot dynamics task, yielding superior performance to plain neural networks, Gaussian processes and LWPR in terms of mean squared error and likelihood.

1 Introduction

Deep learning stands at the center of several key advancements in visual and audio recognition in the past few years [2, 3]. The stacking of several layers of computation results in a hierarchy of feature detectors of which each conveys new intermediate representations of the data. Prediction is assumed to be substantially easier in these representational spaces than in input space, since previously entangled "factors of variation" [4] are well separated. Initially, research on deep learning was started by [5] of which a crucial ingredient was the greedy layer wise pretraining in an unsupervised fashion. Yet, the increase in availabilty of computational power (especially in the form of GPUs) showed that deep neural networks can as well be trained if lots of data is available [6], special non-saturating units are used [7] or with the help of a powerful regularizer named dropout [8]. The latter randomly discards units in the network during training, making co-adaption of units and thus poor generalization less likely.

Perceiving the units of a network as stochastic entities goes back at least to [9]. While neural networks can be used to represent any output distribution which can be summarized by a finite number of sufficient statistics, these have to be defined a priori and are not part of the learning. Contrary, the promise of stochastic networks is that $p(z|x)$ can have a number of maxima exponential in the number of units, allowing complex one-to-many relations.

Research on these models has seen notable papers most recently. For one, [10] introduces a deep density estimator $p(x)$ which can be efficiently trained via back-propagation [11] and is a consistent estimator of the underlying data distribution. Learning a stochastic feed-forward net leading to a multi-modal output distribution $p(z|x)$ is made practical in [12]. Novel techniques to train stochastic neural networks with back-propagation have been presented in [13].

Using deep architectures for the estimation of predictive distributions has been tackled before in [14, 15]. Our contribution is to make use of findings from [1] to approximate each unit in a stochastic neural network up to second order with a Gaussian and reflect this in the construction of the loss. This leads to unimodal Gaussian predictive distributions which play nicely with dropout regularization for deep neural networks.

2 Approach

The well known method of mixture density networks [16] and a recent development in approximating dropout [1] lie at the heart of our work. A brief overview of both will be given in order to establish a base upon which our contribution can be described. We begin with a short review of neural networks.

2.1 Neural Networks

Neural networks can be described as a stack of *layers* of which each consists of an adaptable affine transformation and a subsequent nonlinear function. Let $x \in \mathbb{R}^I$ be an input to the network from which we wish to produce an output $y \in \mathbb{R}^O$. Given a network of K layers, we compute the output u of a layer given the output of the previous layer u' via the following equation

$$u = f(u'W + b), . \tag{1}$$

The weight matrix W, the bias term b and the transfer function f are layer specific. The whole set of adaptable parameters is referred to as $\theta = \{(W^k, b^k)\}_{k=1}^K$ where we added the top index to distinguish between parameters from different layers. Typical choices for the transfer functions $\{f^k\}_{k=1}^K$ are the *sigmoid* $f(\kappa) = \frac{1}{1+\exp(-\kappa)}$, the *rectifier* $f(\kappa) = \max(\kappa, 0)$ or the identity $f(\kappa) = \kappa$, where $\kappa \in \mathbb{R}$. Transfer functions are applied component wise. A single component of a layer is referred to as a unit or neuron.

The most popular and arguably most efficient way to adapt the behaviour as desired is backpropagation [11, 17, 18]. Doing so involves the definition of a loss function $\mathcal{L}(\theta)$ which can be differentiated with respect to the parameters θ and fed into an optimizer such as stochastic gradient descent or nonlinear conjugate gradient. One of the most common choices for a loss function is the mean squared error:

$$\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{N} \sum_i^N ||y_i - z_i||_2^2. \tag{2}$$

It is defined as a sum over a data set $\mathcal{D} = \{(x_i, z_i)\}_{i=1}^N$ which consists of independent samples from a function which the network shall mimic. A probabilistic interpretation is that the network models the data with a Gaussian conditioned on the input: $p(z|x) = \mathcal{N}(\mu(x), \sigma^2)$, where the mean is defined by the output of the network, i.e. $\mu(x) = y$. The variance σ^2 is assumed to be constant, which is commonly referred to as *homoscedastic* variance. Taking the log of the likelihood of the data $\prod_i p(z_i|x_i)$ and neglecting constant terms irrelevant to optimization leaves us with Equation (2).

2.2 Density Networks

Mixture density networks [16] are ordinary neural networks with special transfer function at the last layer and a matching loss. The output represents the sufficient statistics of a mixture of Gaussians: priors, means and covariances of each component. The resulting model can be trained via maximum likelihood by numerical minimization of the negative loglikelihood. We consider mixture density networks with only a single component and a diagonal covariance, and thus call them density networks for brevity. Given a regression problem of target dimensionality D, that is $\mathcal{D} = \{(x_i, z_i)\}$ with $z_i \in \mathbb{R}^D$, the output layer is designed to be of size $O = 2D$. The first D components represent the mean while the second D give the variance of a Gaussian:

$$\mu(x_{i,d}) = y_{i,d}, \tag{3}$$

$$\sigma^2(x_{i,d}) = y_{i,D+d}^2. \tag{4}$$

The square assures positive variance. We use these values to specify a conditional Gaussian distribution of $p(z|x) = \mathcal{N}(\mu(x), \sigma^2(x))$ which is *heteroscedastic* since the variance depends on the input. Computing and differentiating the log likelihood of the targets given the inputs is now straightforward and leads to training via the average negative log likelihood:

$$\mathcal{L}_{NLL}(\theta) = \frac{1}{N} \sum_{i,d} \frac{(z_{i,d} - \mu(x_{i,d}))^2}{2\sigma^2(x_{i,d})} + \log \sqrt{2\pi\sigma^2(x_{i,d})}. \tag{5}$$

In practice, optimization can be difficult: the variances might collapse to very small numbers, leading to very high likelihoods and numerical instabilities [19]. As a counter measure we added a constant term of 0.0001 to the variances in the objective function and its gradients.

2.3 Fast Dropout

Dropout [8] is a powerful regularizer for neural networks. By randomly neglecting input and hidden units from the network during forward- and back-propagation, the method prevents different units of the network from coadapting and subsequently depending on each other too much. This results in a vast improvement

of performance and lead to several significant improvements in visual and audio recognition.

A problem of dropout is that training times tend to be rather long due to the necessesity of sampling and the resulting noisy gradients. To circumvent this, the author's of [1] proposed to approximate the input to each layer of the net $a = u'W + b$. Using a diagonal Gaussian $\hat{a} \sim \mathcal{N}(\mu, s^2)$ is reasonable due to the central limit theorem. Given some mild conditions on the distribution of u' as well as its mean ν' and variance τ'^2 the first two moments of $\hat{a}$ can be computed exactly:

$$\mu = d(\nu' W + b), \tag{6}$$

$$s^2 = \text{diag}(d(1-d)\nu'^2 W^2 + d\tau'^2 W^2). \tag{7}$$

Here, d refers to the dropout rate. Obtaining the moments ν and τ^2 of $o = f(\hat{a})$ can then be done by propagating μ and s^2 through f. This is possible in closed form for the rectifing linear and approximately for the sigmoid. In other cases, the unscented transform [20] or sampling can be used. The authors of [1] provide more details; no significant loss in performance and yet significant improvements in training time is reported, due to less sampling operations.

2.4 Implicit Variance Networks

A consequence of treating each unit in a network as stochastic and approximating it up to second order is that the output of the network is also stochastic. Since fast dropout already handles the forward propagation of variance, in contrast to plain neural networks, the variance of the output units is readily available. We propose to not to neglect the variance at the output layer. Instead we incorporate it into the negative log likelihood (Equation (5)), where we model the statistics with the output of the last layer, i.e. $\mu(x_i) = m_i^K$ and $\sigma^2(x_i) = (s_i^K)^2$.

We extended the model further in two ways: incorporating input variance and a special bias for the variance of units. Variance of inputs arises naturally in many settings (e.g. in noisy sensor data) and can be respected easily during forward propagation from the inputs to the first hidden layer. A principled treatment for inferring the variance of the inputs can employed by performing a factor analysis for example. For simplicity we assume the variance to be constant over the data set and dimensions, essentially treating it as another hyper parameter which is set during cross validation.

During preliminary experiments, we noticed that the network duplicates units at the last hidden layer to reduce variance at the output layer where required for further minimizing the loss. This effect, which we call "pseudo pruning" is due to the (invalid) assumption of independency between the activations of a layer. The network can just copy a unit and half their outgoing weights to reduce the variance contribution while maintaining the mean contribution to the following layer. [1]

[1] We omit a formal argument due to space restrictions.

While pseudo priuning might seem desirable as it reduces overfitting, it can lead to extreme underfitting and is also computationally not efficient. The forward propagation of variance (given in Equation (7)) was thus adapted to the following:

$$s^2 = \text{diag}(d(1-d)\nu'^2 W^2 + d\tau'^2 W^2) \odot \beta, \tag{8}$$

where $\odot$ is the element wise product and $\beta > B$ is constrained to be greater than some positive number B; this can be done by elegantly by reparametrizing $\beta = \exp(\hat{\beta}) + B$ and optimizing with respect to $\hat{\beta}$. Again, B is treated as another hyper parameter and optimized via cross validation.

3 Experiments

We present experiments on a synthetic data set involving heteroscedastic behaviour as a sanity check for our model. We then move to a real world benchmark where inverse robot dynamics are to be learned; this task has been tackled previously in [21, 22]. [2]

For all experiments, the inputs as well as the targets were normalized to zero mean and unit variance; the former helps with optimization [23] and the latter leads to more comparable results for evaluation as it is equivalent to using the normalized MSE. Determination of good hyper parameters was performed via a random search as recommended by [24] for which the search distribution is given in Table 2; batch size and number of hidden units are data set specific and given in the respective section. We picked 32 random configurations for each experiment. Training took place for a fixed number of epochs after which the network with the best score on a held out validation set was picked for final evaluation on the test data. We minimized the $\mathcal{L}_{\text{MSE}}$ in case of NN and FD and $\mathcal{L}_{\text{NLL}}$ for DN and IVN. Optimization was performed with rmsprop [25] using Nesterov momentum [26, 27]. For the plain neural networks, where no variance is modelled, we assume homoscedastic variance which we estimate after training as the variance of the residuals. We report the mean squared error and the negative log likelihood, both averaged over the test sets, in Table 1.

3.1 Toy Data

This data set was proposed by [28] in order to evaluate the ability of a model to express heteroscedasticity in an easily inspectable way. It is governed by the following two equations,

$$\mu_i = 2(\exp(-30(x_i - \frac{1}{4})^2) + \sin(\pi x_i^2)),$$
$$\sigma_i^2 = \exp(\sin(2\pi x_i)),$$

[2] We also considered the "Abalone" data set, but could not make the performance of IVNs competitive with Gaussian processes (which reached an MSE of about 0.29, compared to 0.39 for IVNS) and thus discarded that data set as a good way to compare IVNs to other neural models.

which specify Gaussian distributions $p(y_i|x_i) = \mathcal{N}(\mu_i, \sigma_i^2)$. To generate a data set we sample $\{x_i\}$ points uniformly from the input range $[-0.1, 1]$, and then sample $\{y_i\}$ accordingly. To compare plain neural networks (NN), density networks (DN), networks trained with fast dropout (FD) and implicit variance networks (IVN), we constructed a setting which is far from tailored towards neural networks: very little data.

The data set contained only 50 points for training, 10 for validation and 50 additional points to asses the performance as a test set. We trained the networks for 5000 epochs with batch size either 10, 25, 50 and 10, 25, 50 or 100 hidden units.

Discussion Notably, all methods except ours perform as bad as expected for a neural network model in this setting. While the density networks are en par with our method in terms of mean squared error, they overfit extremly with respect to their predictive distribution. Neural networks neither trained classically nor with fast dropout achieve good results; the variance of fast dropout seems to be meaningless, which is not surprising as it is not trained.

Table 1. Results on the toy benchmark and the sarcos data set.

	Toy		**Sarcos**	
Method	MSE	NLL	MSE	NLL
NN	4.2395	2.2694	0.0047	-1.1893
DN	3.8706	9.7303	0.0096	-1.2532
FD	4.3491	43486.7	0.0065	1.2667
IVN	3.8985	1.6187	0.0079	-1.3606

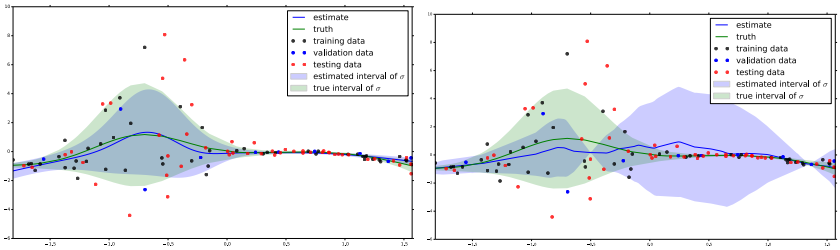
Table 2. Hyper parameter ranges common over different data sets.

Hyper parameter	Choices
#hidden layers	$1, 2, 3$
Transfer function	rectifier, sigmoid
Step rate	$10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$
Momentum	$0, 0.5, 0.9, 0.99, 0.999$
Decay	$0.7, 0.8, 0.9$
Input variance	$0, 0.1, 0.2$
Variance offset B	$0, 0.5, 1$

3.2 Sarcos: Inverse Robot Dynamics

We evaluated the models under consideration on a standard benchmark for learning robot inverse dynamics, the "Sarcos" data set. We trained the networks for

Fig. 1. Predictive distributions of IVNs and DNs on the toy benchmark.



500 epochs and picked the batch size from $\{64, 128, 256, 512\}$ and the number of hidden units from $\{50, 100, 200, 300\}$. We want to stress several observations. For one, IVNs seem to be the best choice if one is interested in good performance of both MSE and NLL. Secondly, plain neural networks perform surprisingly well in our experiments. While both Gaussian processes and LWPR models have different advantages compared to neural networks (model uncertainty and efficient incremental online learning, respectively) our experiments show that both are outperformed in terms of predictive quality.

4 Conclusion and Future Work

We presented a novel method to estimate predictive distributions via deep neural networks that plays nicely with fast dropout. The results are competitive or superior to other neural approaches in our experiments and en par with Gaussian processes and LWPR in a robotics task. Future work will focus on multi-modal output distributions by employing a mixture density like output architecture; application to recurrent neural networks is straightforward. We will also evaluate how principled detemrination of the input variances plays with our approach.

References

1. Wang, S., Manning, C.: Fast dropout training. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13). (2013) 118–126
2. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25. (2012) 1106–1114
3. Dahl, G.E., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on **20**(1) (2012) 30–42
4. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on Machine learning, ACM (2007) 473–480
5. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507

6. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE (2012) 3642–3649
7. Zeiler, M., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J., et al.: On rectified linear units for speech processing, ICASSP (2013)
8. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
9. Neal, R.M.: Connectionist learning of belief networks. Artificial intelligence **56**(1) (1992) 71–113
10. Bengio, Y., Thibodeau-Laufer, .: Deep generative stochastic networks trainable by backprop. (2013)
11. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088) (1986) 533–536
12. Tang, Y., Salakhutdinov, R.: A new learning algorithm for stochastic feedforward neural nets. (2013)
13. Bengio, Y.: Estimating or propagating gradients through stochastic neurons. arXiv preprint arXiv:1305.2982 (2013)
14. Salakhutdinov, R., Hinton, G.: Using deep belief nets to learn covariance kernels for gaussian processes. Advances in neural information processing systems **20** (2008) 1249–1256
15. Uria, B., Murray, I., Renals, S., Richmond, K.: Deep architectures for articulatory inversion. In: Proceedings of Interspeech. (2012)
16. Bishop, C.M.: Mixture density networks. (1994)
17. Werbos, P.: Beyond regression: New tools for prediction and analysis in the behavioral sciences. (1974)
18. Le Cun, Y.: Learning process in an asymmetric threshold network. In: Disordered systems and biological organization. Springer (1986) 233–240
19. Bishop, C.M., et al.: Pattern recognition and machine learning. Volume 1. springer New York (2006)
20. Julier, S.J., Uhlmann, J.K.: New extension of the kalman filter to nonlinear systems. In: AeroSense'97, International Society for Optics and Photonics (1997) 182–193
21. Vijayakumar, S., D'souza, A., Schaal, S.: Incremental online learning in high dimensions. Neural computation **17**(12) (2005) 2602–2634
22. Rasmussen, C.E.: Gaussian processes for machine learning. Citeseer (2006)
23. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural networks: Tricks of the trade. Springer (1998) 9–50
24. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. The Journal of Machine Learning Research **13** (2012) 281–305
25. Tieleman, T., Hinton, G.: Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning (2012)
26. Sutskever, I.: Training Recurrent Neural Networks. PhD thesis, University of Toronto (2013)
27. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. (2013)
28. Le, Q.V., Smola, A.J., Canu, S.: Heteroscedastic gaussian process regression. In: Proceedings of the 22nd international conference on Machine learning, ACM (2005) 489–496