Practical Variational Inference for Neural Networks

Alex Graves
University of Toronto
graves@cs.toronto.edu

Introduction

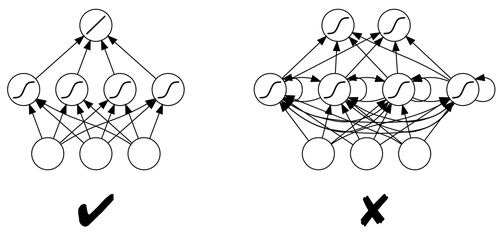
Variational methods offer a powerful approximation to Bayesian inference for parametric models. The basic idea is to replace the true posterior $\Pr(\mathbf{w}|\mathcal{D})$ over the parameters $\mathbf{w}$ given the data $\mathcal{D}$ with an approximate distribution $Q(\mathbf{w})$ that is easier to work with.

$$Q(\mathbf{w}) \approx \Pr(\mathbf{w}|\mathcal{D})$$

In particular $Q(\mathbf{w})$ is usually chosen so that the variational free energy $\mathcal{F}$, which is needed to perform Bayesian inference, can be calculated exactly. However the corresponding integral is difficult to solve for neural network models, where the data distribution $\Pr(\mathcal{D}|\mathbf{w})$ is a complex and highly nonlinear function of the parameters.

$$\mathcal{F} = -\left\langle \ln\left(\frac{\Pr(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{Q(\mathbf{w})}\right)\right\rangle_{\mathbf{w}\sim Q(\mathbf{w})} = \ ?$$

As a result, previous variational methods have been restricted to simple neural networks such as Multilayer Perceptrons with single hidden layers and linear output units. The goal of this work is a method that also works for complex architectures such as recurrent neural networks.



Our approach is to forget about solving the integrals exactly and look instead for ways to approximately calculate and differentiate $Q(\mathbf{w})$ using sampling methods. In other words we approximate the variational approximation.

$$\mathcal{F} \approx -\frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{\Pr(\mathcal{D}|\mathbf{w}^i)P(\mathbf{w}^i)}{Q(\mathbf{w}^i)}\right)$$

The result is a simple, stochastic form of variational inference that can be applied to any differentiable log-loss parametric model (which includes most directed neural networks).

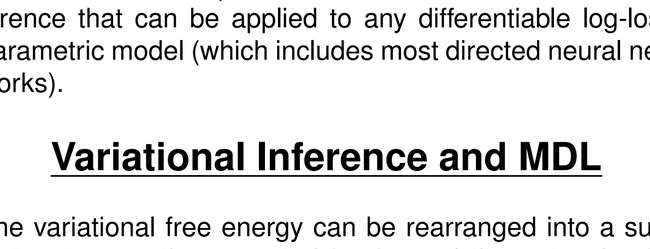
Variational Inference and MDL

The variational free energy can be rearranged into a sum of two terms: the expected log-loss of the network with weights drawn from $Q(\mathbf{w})$, and the Kullback-Leibler divergence between $Q(\mathbf{w})$ and the weight prior $P(\mathbf{w})$.

$$\mathcal{F} = \underbrace{-\left\langle \ln \Pr(\mathcal{D}|\mathbf{w})\right\rangle_{\mathbf{w}\sim Q}}_{\text{Error cost}} + \underbrace{D_{KL}(Q||P)}_{\text{Complexity cost}}$$

The first term is the expected number of nats required to describe the data to someone who has a network with weights drawn from $Q(\mathbf{w})$ (using, for example, arithmetic coding to compress the network errors). We therefore identify it as the **error cost**.

The second term is the expected number of nats required to describe the weights (using 'bits-back' coding) to someone who has the weight prior. Intuitively, the more different $Q(\mathbf{w})$ is from $P(\mathbf{w})$ the more it costs to describe; in particular, distributions that precisely describe individual weights are expensive. Because this term measures the amount of information the network can store in its weights we identify it as the **complexity cost**.

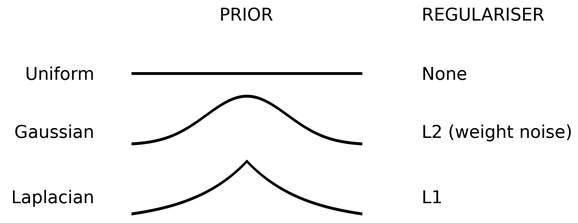


$\mathcal{F}$ is therefore equivalent to a **minimum description length** (MDL) loss function that measures the total cost of describing the model and the data. One advantage of the MDL interpretation is that it separates model complexity from predictive accuracy. Another is that it makes it clear whether the data has really been compressed or not.

Choice of Distributions

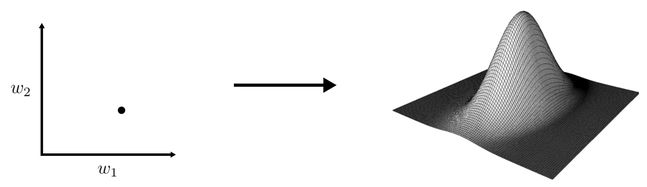
To train the network we must first choose a parametric form for the weight prior $P(\mathbf{w})$ and the variational posterior $Q(\mathbf{w})$, then minimise $\mathcal{F}$ with respect to these parameters.

If $Q(\mathbf{w})$ is a **delta distribution** (all probability is concentrated on a single point in weight space) we recover ordinary maximum likelihood training with a regularisation term that depends on the prior.



Normal $\mathbf{L_1}$ and $\mathbf{L_2}$ regularisation are equivalent to priors with location zero and fixed scale (the smaller the scale the stronger the regularisation). However it is also possible to adapt the prior parameters during training.

If $Q(\mathbf{w})$ is a **diagonal Gaussian** then instead of optimising a point in weight space, we optimise a multivariate Gaussian centred on a point.



The key mathematical identities that make this possible are the following, valid for any multivariate Gaussian $\mathcal{N}(\mu, \Sigma)$:

$$\nabla_\mu \left\langle \ln \Pr(\mathcal{D}|\mathbf{w})\right\rangle_{\mathbf{w}\sim\mathcal{N}} = \left\langle \nabla_{\mathbf{w}} \ln \Pr(\mathcal{D}|\mathbf{w})\right\rangle_{\mathbf{w}\sim\mathcal{N}} \quad (1)$$

$$\nabla_\Sigma \left\langle \ln \Pr(\mathcal{D}|\mathbf{w})\right\rangle_{\mathbf{w}\sim\mathcal{N}} = \frac{1}{2}\left\langle \nabla^2_{\mathbf{w}} \ln \Pr(\mathcal{D}|\mathbf{w})\right\rangle_{\mathbf{w}\sim\mathcal{N}} \quad (2)$$

These allow us to approximate the gradient of the error cost with respect to the parameters of $Q(\mathbf{w})$ by sampling weights from $Q(\mathbf{w})$ and averaging the usual gradient of the log-loss with respect to the weights. For networks whose second derivatives are difficult to calculate, the Hessian in Eq. 2 can be approximated with the negative Fisher matrix.

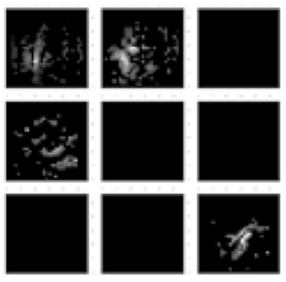
If the variances in $Q(\mathbf{w})$ are fixed and only the means are optimised, the optimisation is equivalent to training with Gaussian **weight noise**. For normal weight-noise training $P(\mathbf{w})$ is implicitly uniform. However it is also possible to use a Gaussian prior with or without adaptive parameters.

Adapting the variances using Eq. 2 gives **adaptive weight noise**. This allows the network to specify 'important' weights precisely (at the cost of higher complexity) while allowing other, less important weights to tend to the prior.

Weight Costs

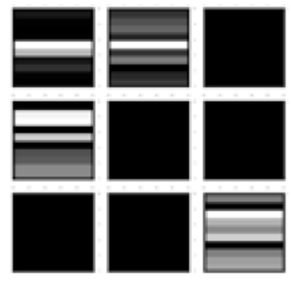
Visualising the complexity costs of the weights can give valuable insight into the workings of the network. Let's look at a single-layer feedforward network trained to classify handwritten digits from the MNIST database.

Here are the costs of the input weights to some of the hidden units.



Each square corresponds to a unit in the hidden layer, and each pixel corresponds to an input weight. The lighter the pixels the more the weight costs. Note that some of the squares have uniformly low cost weights (meaning that the hidden unit is essentially ignored) and that weights with high cost are distributed over the part of the image where the digits are drawn.

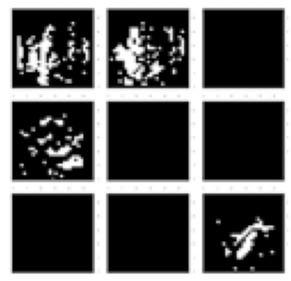
Here are the costs of the weights from the same hidden units to the ten output units (one for each digit class).



Note that the same units are ignored here as above. This makes sense, as the network should not pay for precise weights from a unit whose activation does not carry useful information. Also note that the hidden units have varying degrees of importance to the different output units; presumably they embody input features that are more useful for identifying some digits than others.

Weight Pruning

A simple pruning heuristic can be defined for adaptive weight noise by removing weights that are almost as likely to be zero as they are to be anything else. This tends to prune low cost weights, as shown in the following figure for the same input to hidden weights as above (black weights are pruned, white weights are not).



Experimental Results

We compared the different choices of $P(\mathbf{w})$ and $Q(\mathbf{w})$ for a hierarchical, multidimensional recurrent neural network doing phoneme recognition on the TIMIT database.

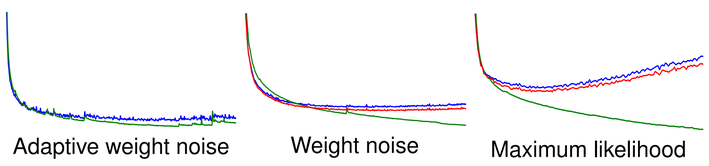
Name	Phoneme Error Rate	Training Epochs
Adaptive mean L2	28.0%	53
L2	27.4%	59
Maximum likelihood	27.1%	44
L1	26.0%	545
Adaptive mean L1	25.4%	765
Weight noise	25.4%	220
Adaptive prior weight noise	24.7%	260
Adaptive weight noise	23.8%	384

We also assessed the effect of weight pruning at different thresholds on the error rate (before and after retraining)

Weights Remaining	Initial error	Retrain error
77.4%	23.8%	24.0%
45.2%	23.9%	23.5%
30.9%	23.9%	23.7%
22.3%	24.0%	23.3%
16.3%	24.5%	24.1%
11.5%	28.0%	24.5%

Overfitting

A major advantage of adaptive weight noise is that it does not overfit on the training data (as long as the training targets are sensibly defined). Therefore early-stopping is not needed and all available data can be used for training. This point is illustrated by the training error curves below (green for training set, blue for test set, red for validation set).



Future Work

An obvious way to extend this work would be to find richer distributions for $P(\mathbf{w})$ and $Q(\mathbf{w})$ for which the free energy can still be easily approximated. We are currently investigating non diagonal Gaussians and spike-and-slab distributions for $Q(\mathbf{w})$ and Gaussian mixtures for $P(\mathbf{w})$.