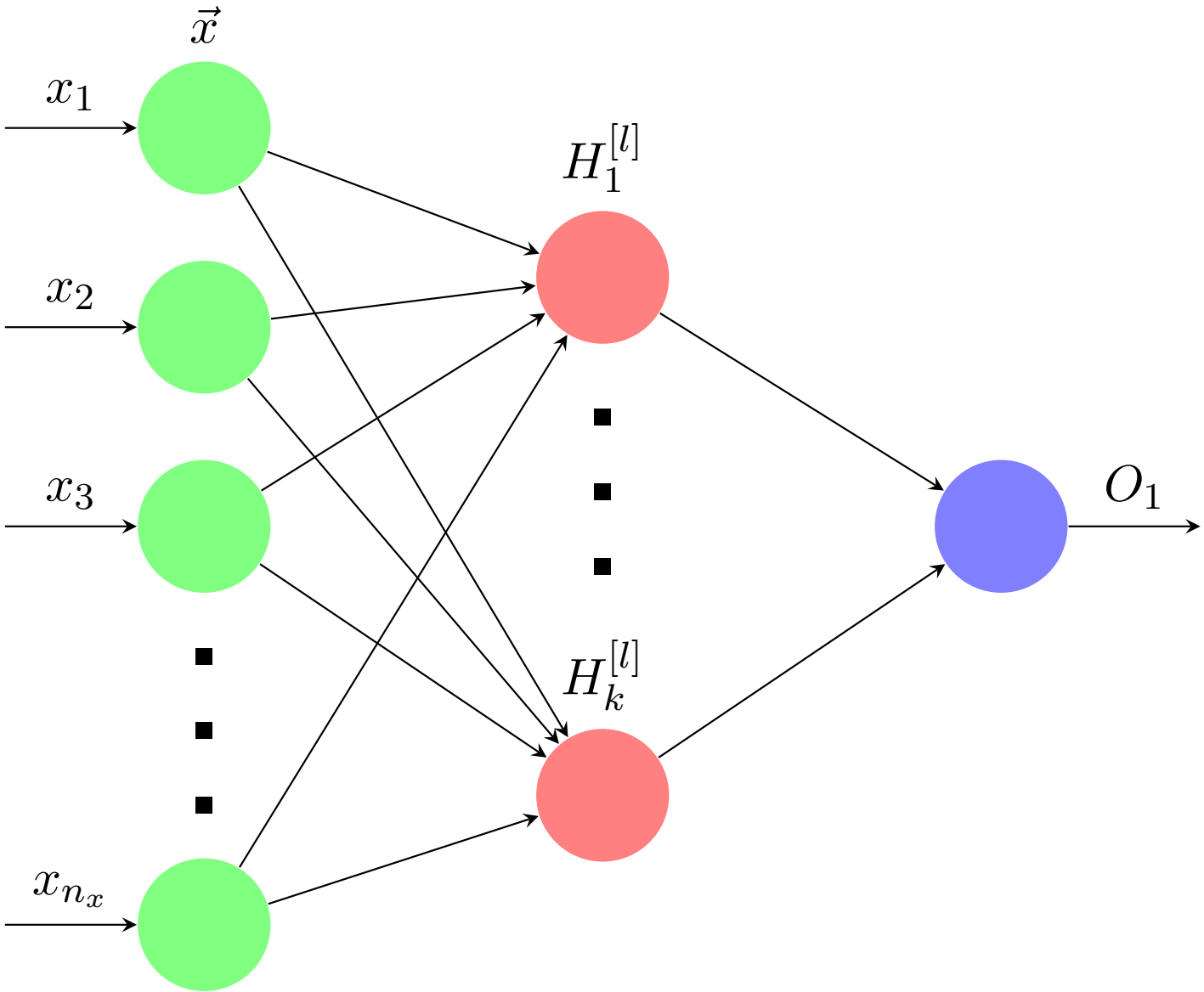
A Bayesian Perspective of Neural Networks

Ethan Goan

Supervised by
Prof. Clinton Fookes, Dr. Dimitri Perrin and Prof. Kerrie Mengersen

Queensland University of Technology

Neural Network Graphical Model



Inside a Neuron

Neuron consists of a linear mapping of the input, followed by a non-linear activation.

$$a_j^{[i]} = g(\mathbf{w}_j^{[i]T}\mathbf{x} + b_j^{[i]})$$

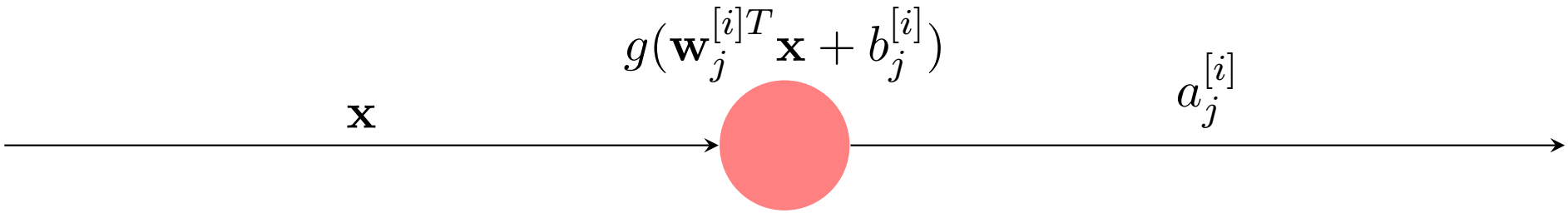
$\mathbf{w}_j^{[i]T}$ = weight vector

$b_j^{[i]T}$ = bias

$g(\cdot)$ = non-linear activation function

$[i]$ = layer number

j = node number within the i^{th} layer



Vectorisation

Can implement the neural network model more efficiently by using matrix operations.

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ & \vdots & \\ \cdots & w_{k_l}^{[1]T} & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{k_l}^{[1]} \end{bmatrix}$$

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]})$$

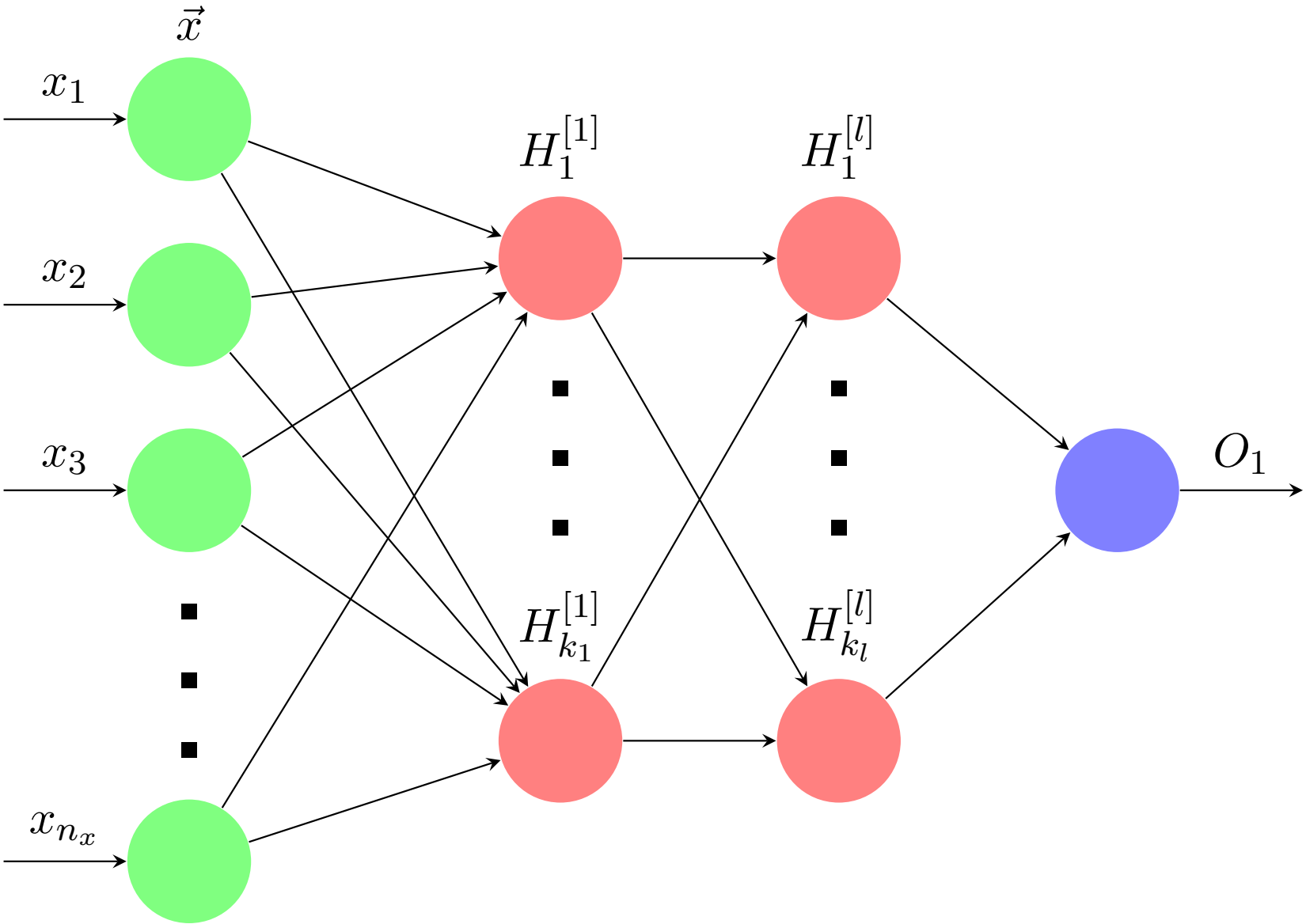
Further Vectorisation - Multiple Input Vectors

Can pass all (or as much as physically possible) into the network in a single iteration by storing the input vectors $\mathbf{x}_i$ as columns of a matrix X.

$$Z^{[1]} = W^{[1]}X + \mathbf{b}^{[1]} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ & \vdots & \\ \cdots & w_{k_l}^{[1]T} & \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{k_l}^{[1]} \end{bmatrix}$$

$$A^{[1]} = g(Z^{[1]})$$

More Hidden Layers



Training - Gradient Descent

Forward Propagation in a 2 layer network (1 hidden layer)

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g(Z^{[2]}) = \hat{y}$$

To perform gradient descent, first need to define an objective to minimise.

$$\mathcal{L}(\hat{y}, y) = -\Big(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\Big)$$

$$J(w, b) = \frac{1}{m}\Sigma_{i=1}^{m}\mathcal{L}(\hat{y}_i, y_i)$$

Training - Gradient Descent

With our cost objective to minimise,

$$J(w, b) = \frac{1}{m} \Sigma_{i=1}^{m} \mathcal{L}(\hat{y}_i, y_i)$$

we can find the partial derivative of this objective and use it to update out network parameters.

$$\theta = \theta - \alpha \frac{\partial J}{\partial \theta}$$

where α is our learning rate, and θ is any of our model weights $W^{[l]}$ or bias' $\mathbf{b}^{[l]}$.

Comparison - Regression

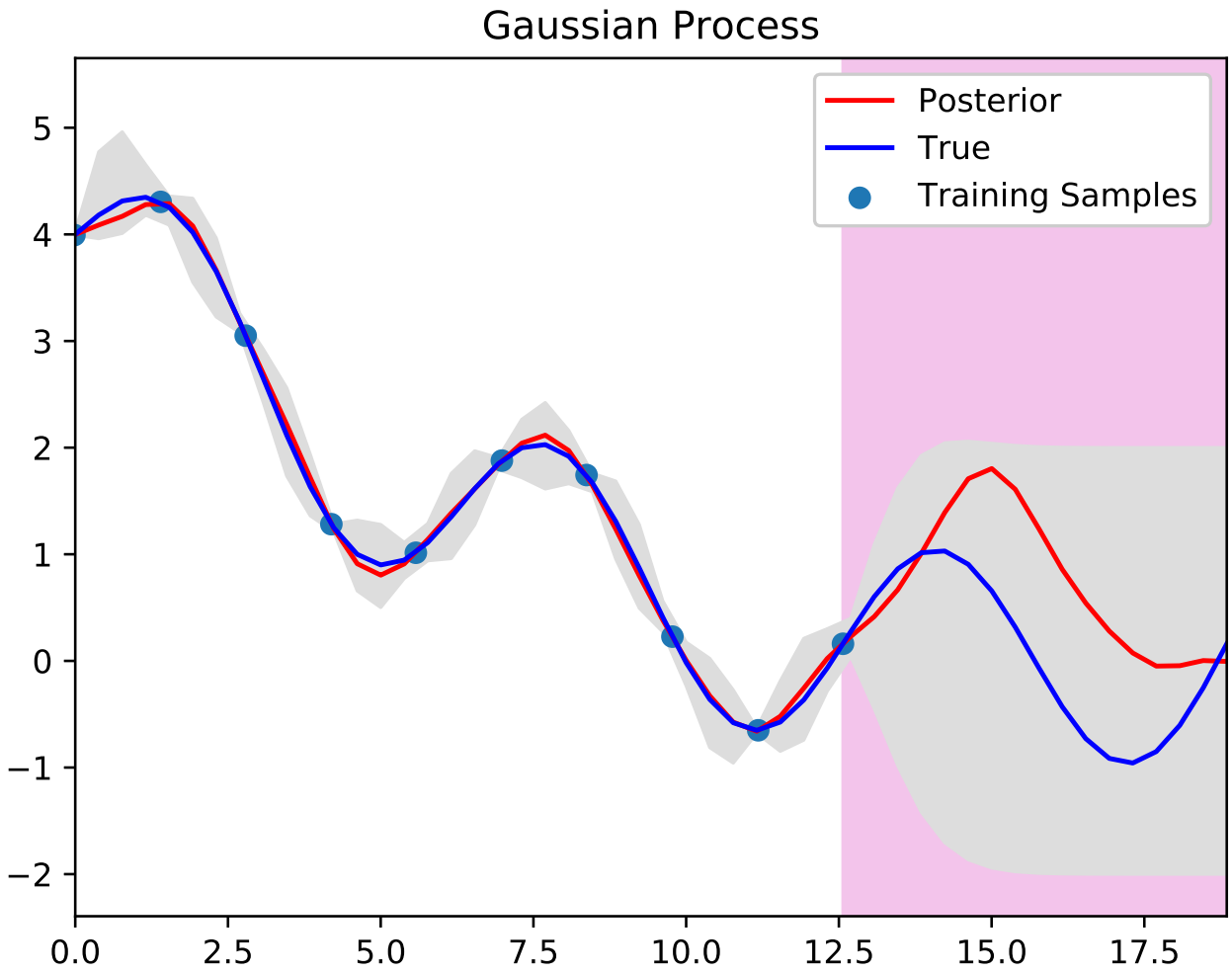


Figure: Regression using Gaussian Process

Comparison - Regression

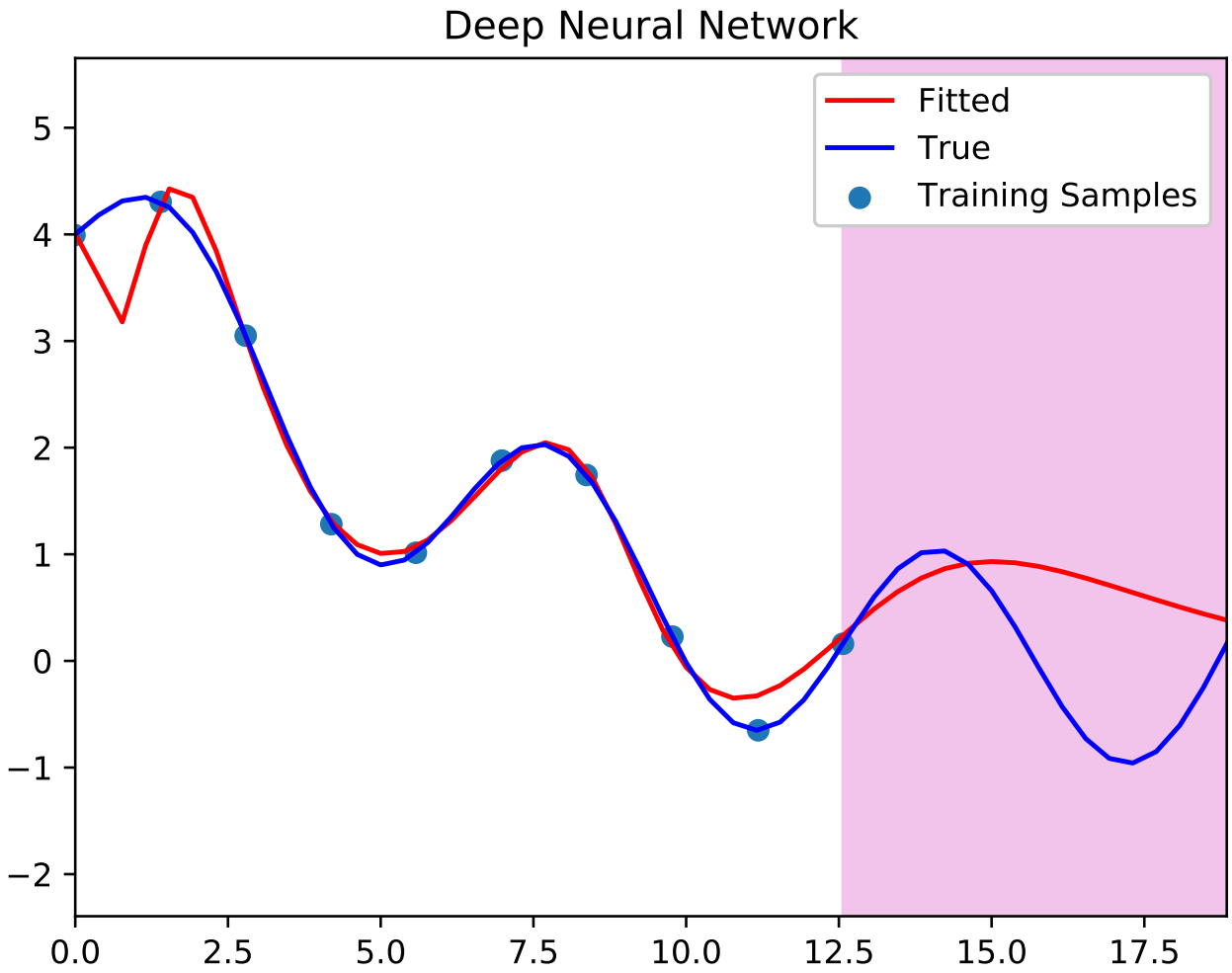


Figure: Regression using neural network with two hidden layers

Stochastic Regularisation Techniques

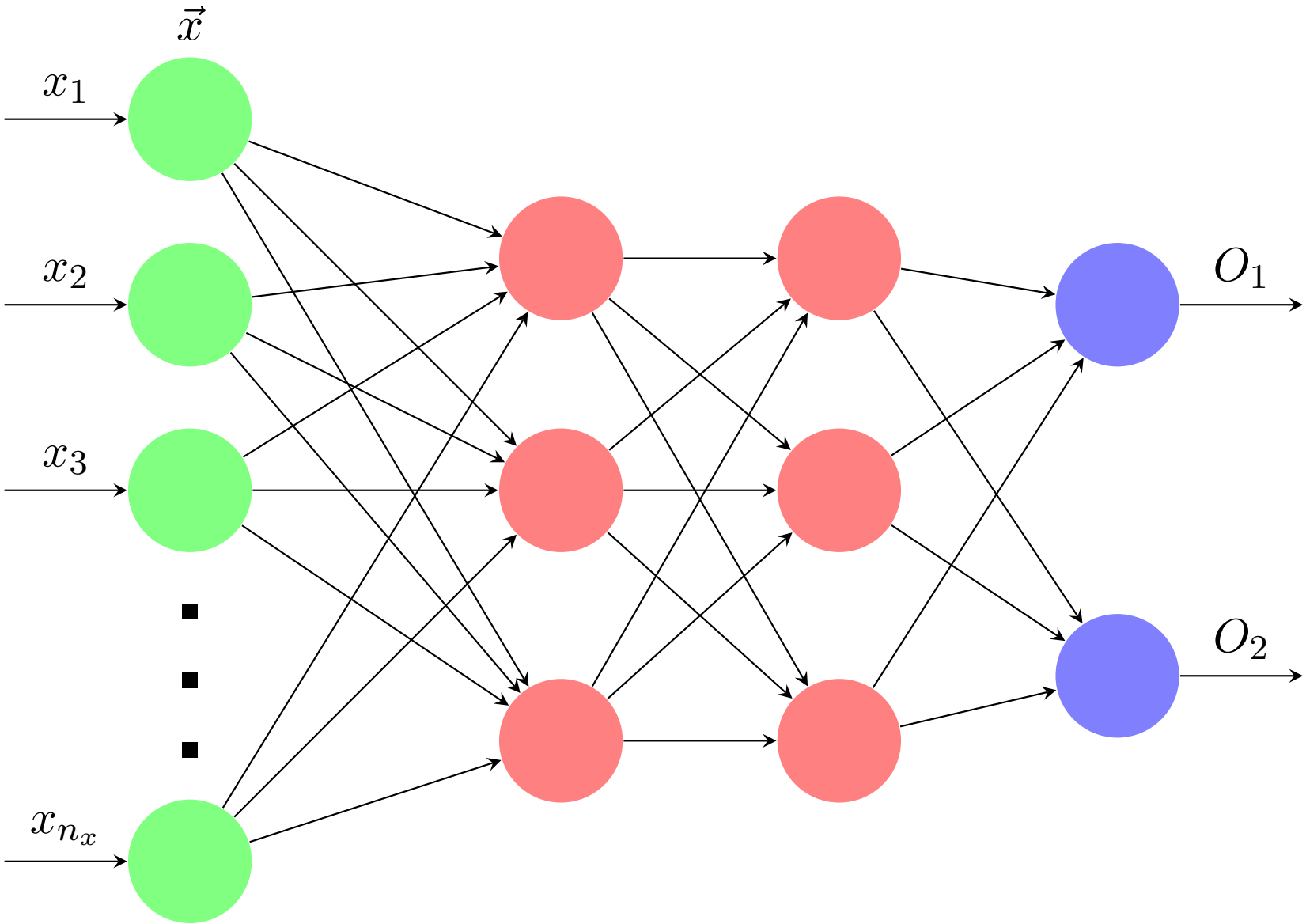
- Neural networks are prone to overfitting training data
- Stochastic Regularisation Techniques (SRTs) are introduced to combat this
- Most prominent technique is Dropout, where output of a unit is attenuated by multiplying element with a Bernoulli distributed RV Srivastava et al. (2014)

Dropout

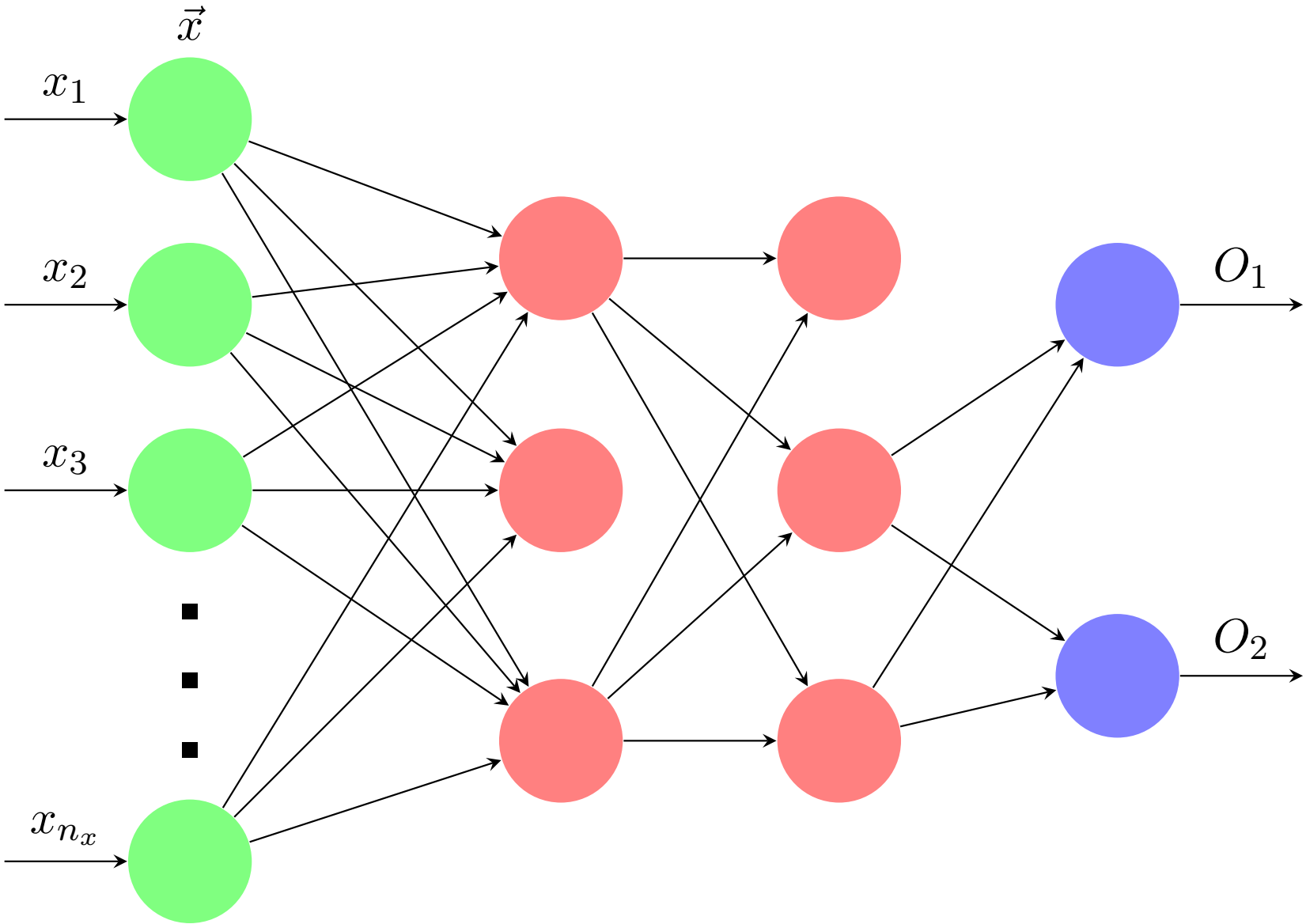
With dropout, the feedforward opertation becomes,

$$
\begin{aligned}
r_j^{[i]} &\sim \text{Bernoulli}(p) \\
\widetilde{\mathbf{a}}^{[i-1]} &= \mathbf{r}^{[i]} \odot \mathbf{a}^{[i-1]} \\
\widetilde{\mathbf{z}}^{[1]} &= W^{[1]} \widetilde{\mathbf{a}}^{[i-1]} + b^{[1]} \\
\widetilde{\mathbf{a}}^{[i+1]} &= g(\widetilde{\mathbf{z}}^{[1]})
\end{aligned}
$$

Before Applying Dropout



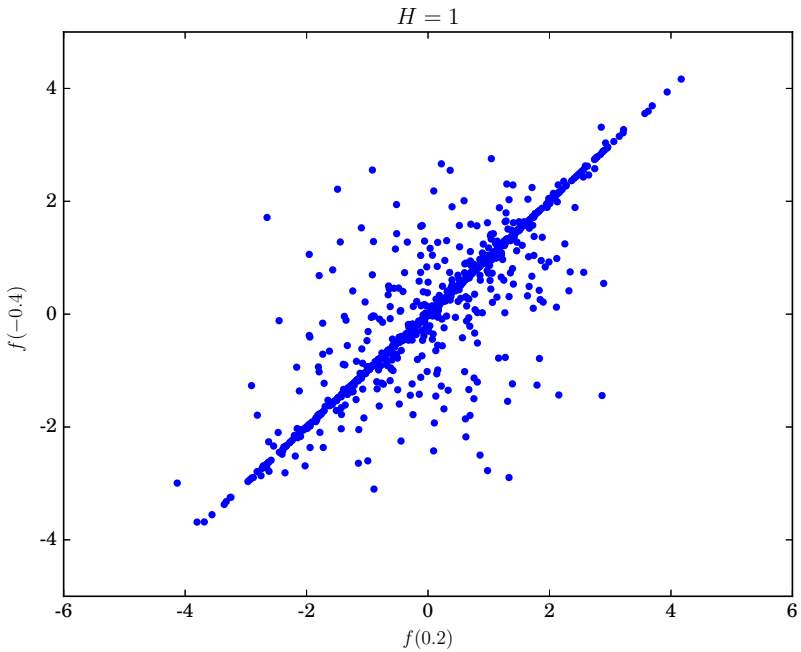
Example After Applying Dropout



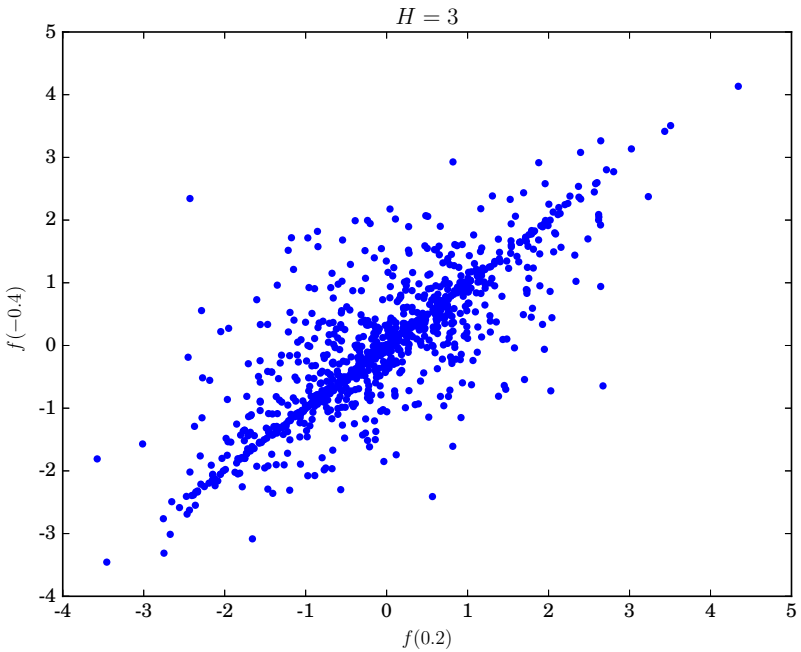
A Bayesian Perspective

- Want to relate neural networks to probabilistic methods
- Development of Bayesian Neural Networks (BNN)
- BNN is a neural network with a prior placed over the network parameters $W^{[i]}, \mathbf{b}^{[\mathbf{i}]}$; Tishby, Levin, and Solla (1989); Neal (1996)
- Work in Neal (1996) showed how when a Gaussian prior is placed over network parameters for a single hidden layer network, the prior on the network output is a Gaussian Process

Gaussian Process Prior

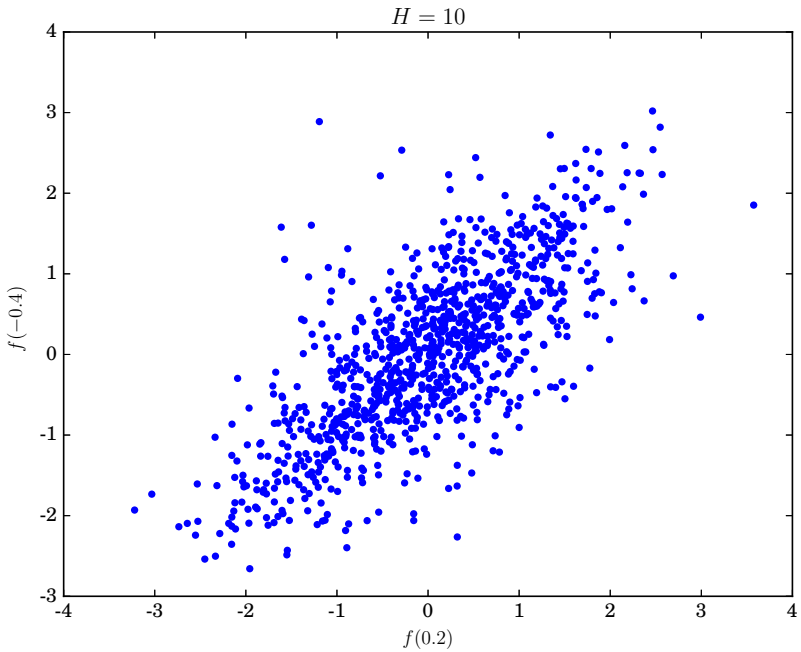


(a) 1 hidden units

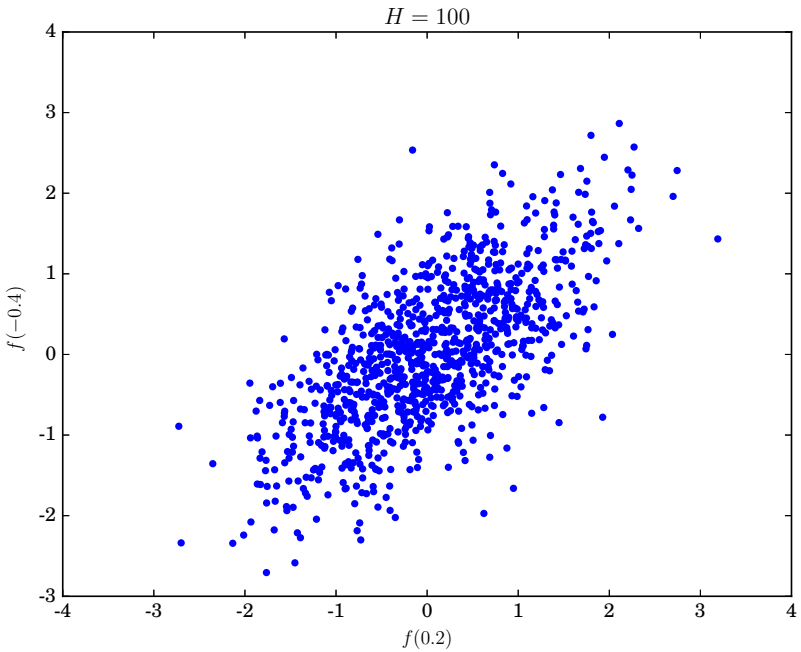


(b) 3 hidden units

Gaussian Process Prior



(a) 10 hidden units



(b) 100 hidden units

The Bayesian Way

Still looking for a for Bayesian treatment of Neural Networks

$$p(\omega|\mathbf{Y},\mathbf{X}) = \frac{p(\omega)p(\mathbf{Y}|\mathbf{X},\omega)}{p(\mathbf{Y}|\mathbf{X})}$$

$$p(\mathbf{y}^*|\mathbf{x}^*,\mathbf{Y},\mathbf{X},\omega) = \int p(\mathbf{y}^*|\mathbf{x}^*,\omega)p(\omega|\mathbf{Y},\mathbf{X})d\omega$$

But as expected, posterior for a deep neural network is intractable

Current Research

- Preliminary work done in Neal (1996), MacKay (1992)
- Only recently resurfaced as a topic of interest
- Neural Networks are hard to perform inference on
- Recent work done in Graves (2011) is promising to address this issue
- How to use Bayesian methods to model uncertainty in our predictions Kingma, Salimans, and Welling (2015), Gal (2016)

Variational Methods

Select an approximate posterior $q_\theta(\omega)$ and minimise KL Divergence between approximate and true posterior

$$\hat{\mathcal{L}}_{VI}(\theta) := -\int q_\theta(\omega) \log\Big(p(\mathbf{Y}|\mathbf{X},\omega)\Big) d\omega + \mathsf{KL}\Big(q_\theta(\omega)||p(\omega)\Big)$$

$$\hat{\mathcal{L}}_{VI}(\theta) := -\sum_{i=1}^{n_x} \int q_\theta(\omega) \log\Big(p(\mathbf{y}_i|\mathbf{f}^\omega(\mathbf{x}_i)\Big) d\omega + \mathsf{KL}\Big(q_\theta(\omega)||p(\omega)\Big)$$

First term corresponds to expected log-liklihood

Reparameterisation Trick

Expected log-liklihood term is of the form,

$$I(\theta) = \frac{\partial}{\partial \theta} \int f(x) p_\theta(x) dx$$

We can use the reparameterisation trick proposed in Kingma and Welling (2013), where the latent variable $\omega \sim q_\theta(\omega)$ is expressed as a deterministic function $g(\epsilon, \theta)$, with $\epsilon \sim p(\epsilon) = \Pi_{l,i} p(\epsilon_{l,i})$.

For example, if $q_\theta(\omega) \sim \mathcal{N}(\mu, \sigma^2)$, can have $g(\theta, \epsilon) = \mu + \epsilon\sigma$, where $p(\epsilon) = \mathcal{N}(0, I)$

Reparameterisation Trick

With this, we can rewrite our KL divergence term between the true and approximate posterior Gal (2016).

$$\hat{\mathcal{L}}_{VI}(\theta) := -\sum_{i=1}^{n_x} \int p(\epsilon) \log \Big(p(\mathbf{y}_i | \mathbf{f}^{g(\epsilon,\theta)}(\mathbf{x}_i)) \Big) d\epsilon + \mathsf{KL}\Big(q_\theta(\omega) || p(\omega) \Big)$$

This expression can then be approximated using Monte Carlo methods to find our expression for the approximate posterior.

Link to Dropout

With this, we can rewrite our KL divergence term between the true and approximate posterior Gal (2016).

$$\hat{\mathcal{L}}_{VI}(\theta) := -\sum_{i=1}^{n_x} \int p(\epsilon) \log \Big(p(\mathbf{y}_i | \mathbf{f}^{g(\epsilon,\theta)}(\mathbf{x}_i)) \Big) d\epsilon + \mathsf{KL}\Big(q_\theta(\omega) || p(\omega) \Big)$$

In this expression, the ter $\mathbf{f}^{g(\epsilon,\theta)}$ corresponds to the output of the network, with dropout parameterised by $p(\epsilon)$ is applied to the networks units. This expression can then be approximated using Monte Carlo methods to find our expression for the approximate posterior.

$$\hat{\mathcal{L}}_{MC}(\theta) := -\sum_{i=1}^{n_x} \log \Big(p(\mathbf{y}_i | \mathbf{f}^{g(\hat{\epsilon},\theta)}(\mathbf{x}_i)) \Big) + \mathsf{KL}\Big(q_\theta(\omega) || p(\omega) \Big)$$

This expression can be optimised using gradient descent to approximate optimal parameters θ.

Link to Dropout

This expression can then be approximated using Monte Carlo methods to find our expression for the approximate posterior.

$$\hat{\mathcal{L}}_{MC}(\theta) := -\sum_{i=1}^{n_x} \log\left(p(\mathbf{y}_i|\mathbf{f}^{g(\hat{\epsilon},\theta)}(\mathbf{x}_i)\right) + \mathsf{KL}\Big(q_\theta(\omega)||p(\omega)\Big)$$

This expression can be optimised using gradient descent to approximate optimal parameters θ. From this, we can form our predictive posterior, and perform Monte Carlo integration to again approximate it and extract uncertainty estimates.

Topics of interest

- How can we better design neural networks with practical inference in mind
- Look at model design, ie. can we let a Bayesian method actually design our model
- Bayesian Domain Adaptation: how to use pretrained models as a prior?
- Can we incorporate output uncertainty in the training process?
- How to make decisions with uncertainty estimations?
- How good is our uncertainty estimations?
- Big one: How to use a Bayesian framework to better understand deep nets?

References I

Gal, Yarin (2016). "Uncertainty in deep learning". In: *University of Cambridge*.

Graves, Alex (2011). "Practical Variational Inference for Neural Networks". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., pp. 2348–2356. URL: http://papers.nips.cc/paper/4329-practical-variational-inference-for-neural-networks.pdf.

Kingma, Diederik P, Tim Salimans, and Max Welling (2015). "Variational dropout and the local reparameterization trick". In: *Advances in Neural Information Processing Systems*, pp. 2575–2583.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

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

References II

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc.: Secaucus, NJ, USA. ISBN: 0387947248.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15(1), pp. 1929–1958.

Tishby, Naftali, Esther Levin, and Sara A Solla (1989). "Consistent inference of probabilities in layered networks: Predictions and generalization". In: *IJCNN International Joint Conference on Neural Networks*. Vol. 2. IEEE New York, pp. 403–409.