Reading Group on Deep Learning
Session 2

Stephane Lathuiliere & Pablo Mesejo

10 June 2016

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

5.5. Regularization in Neural Networks

First approach: Number of hidden units

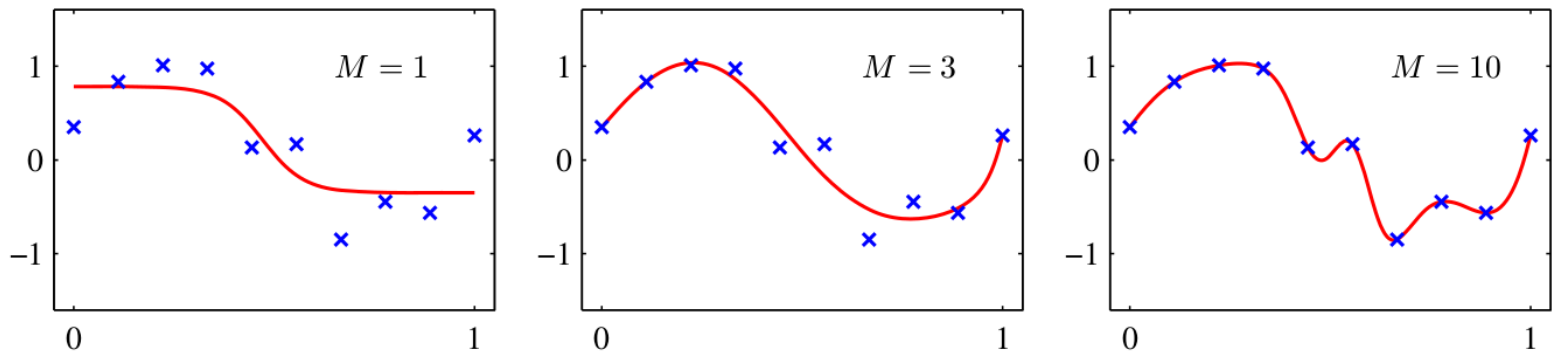


Figure: 5.9. Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having M = 1, 3 and 10 hidden units, respectively.

5.5. Gaussian priors

Second approach:

Before we proposed to train the network by optimizing:

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t \,\middle|\, y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right) \tag{5.12}$$

Now, we optimize the posterior probability:

$$p(\mathbf{w}|t, \mathbf{x}, \lambda) \propto p(t|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\lambda)$$

with the following prior

$$p(\mathbf{w}|\lambda) = \mathcal{N}\left(\mathbf{w}\middle|0, \lambda^{-1} I\right)$$

Taking the negative logarithm, we obtain the error function

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2} \boldsymbol{w}^\top \boldsymbol{w} \tag{5.112}$$

5.5. Consistent Gaussian priors

Let's illustrate the inconsistency of a simple Gaussian prior.

First hidden layer:

$$z_j = h\Big(\sum_i (w_{ji}x_i + w_{j0})\Big) \tag{5.113}$$

Output units:

$$y_k = \sum_j (w_{kj}z_j + w_{k0}) \tag{5.114}$$

We apply a linear transformation on the data: $\tilde{x}_i = ax_i + b$

Equivalent Network:

- $\tilde{w}_{ji} = \frac{1}{a} w_{ji}$
- $\tilde{w}_{j0} = w_{j0} - \frac{b}{a}\sum_i w_{ji}$

5.5. Consistent Gaussian priors

We apply a similar transformation on the output: $\tilde{y}_k = cy_k + d$

Equivalent Network:

- $\tilde{w}_{kj} = cw_{kj}$
- $\tilde{w}_{k0} = cw_{k0} + d$

Problem:

There is no $\tilde{\lambda}$ such that $\tilde{\lambda}\tilde{\boldsymbol{w}}^\top\tilde{\boldsymbol{w}} = \lambda\boldsymbol{w}^\top\boldsymbol{w}$

Thus, if we train a MLP on the transformed data, we do not obtain the equivalent network described above.

5.5. Consistent Gaussian priors

Solution:

We consider the following prior

$$p(\mathbf{w}|\alpha_1, \alpha_2) \propto exp\big(-\frac{\alpha_1}{2}\sum_{w\in\mathcal{W}_1} w^2 - \frac{\alpha_2}{2}\sum_{w\in\mathcal{W}_2} w^2\big) \qquad (5.122)$$

and we obtain the following regularizer

$$\frac{\lambda_1}{2}\sum_{w\in\mathcal{W}_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in\mathcal{W}_2} w^2 \qquad (5.121)$$

Then we can chose: $\tilde{\lambda_1} = a^{\frac{1}{2}}\lambda_1$ and $\tilde{\lambda_2} = c^{\frac{-1}{2}}\lambda_2$

5.5. Consistent Gaussian priors

$$p(\mathbf{w}|\alpha_1,\alpha_2) \propto exp\Big(-\frac{\alpha_1^w}{2}\sum_{w\in\mathcal{W}_1^w} w^2 - \frac{\alpha_2^w}{2}\sum_{w\in\mathcal{W}_2^w} w^2 - \frac{\alpha_1^b}{2}\sum_{w\in\mathcal{W}_1^b} w^2 - \frac{\alpha_2^b}{2}\sum_{w\in\mathcal{W}_2^b} w^2\Big)$$

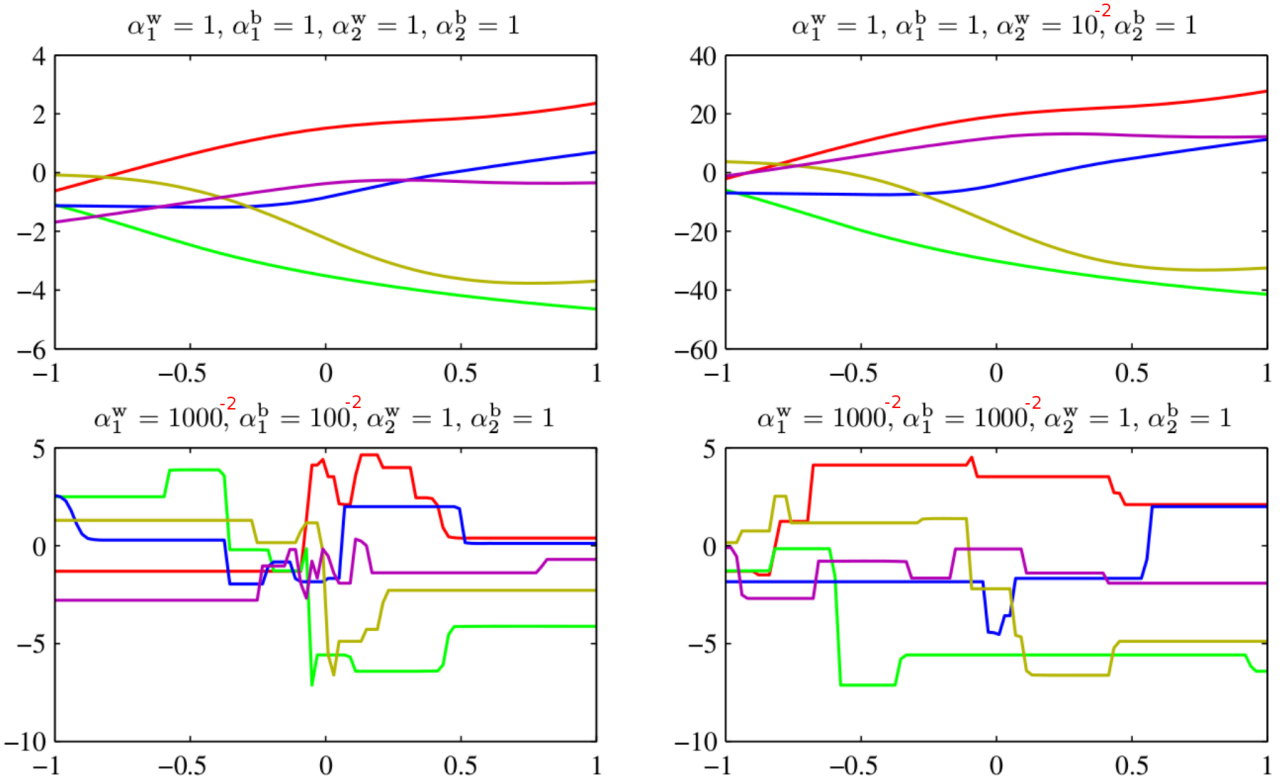


Figure: 5.11. Samples from the prior and plotting the corresponding network functions

5.5. Early stopping

Third approach:

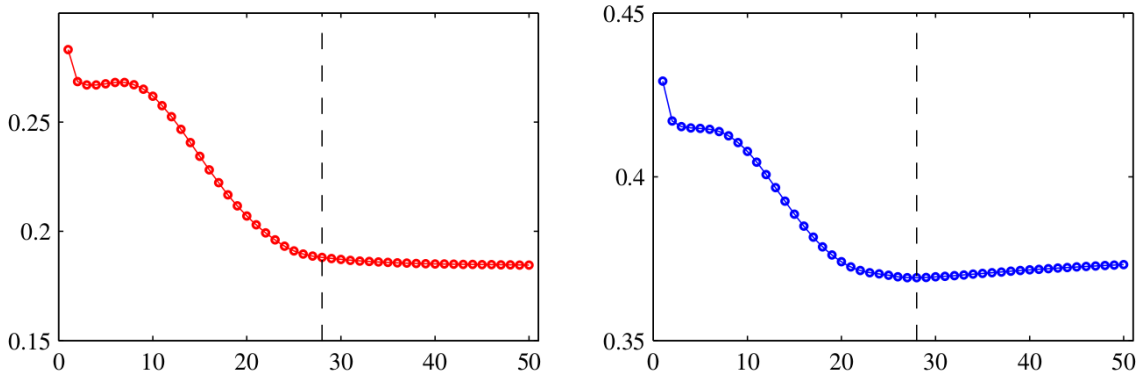


Figure: 5.12. Illustration of the behaviour of training set error and validation set error

5.5. Early stopping

Third approach:

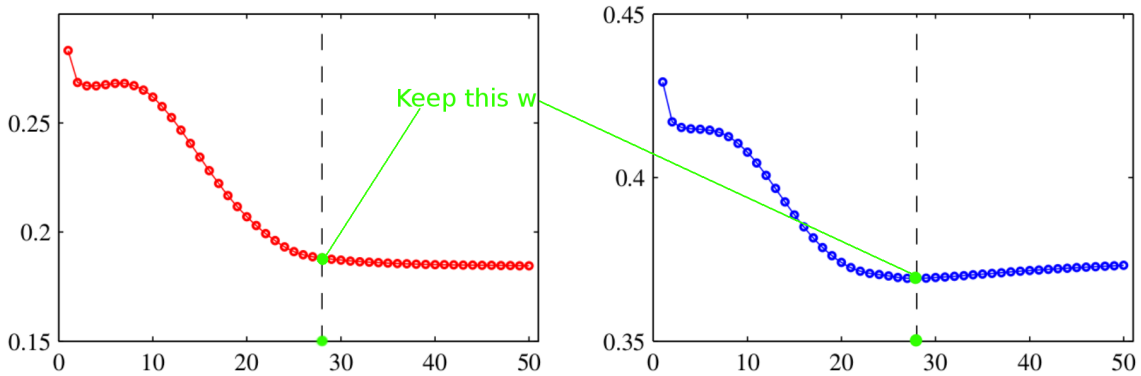


Figure: 5.12. Illustration of the behaviour of training set error and validation set error

5.5. Learning Invariances by Augmenting the Data

Fourth approach:

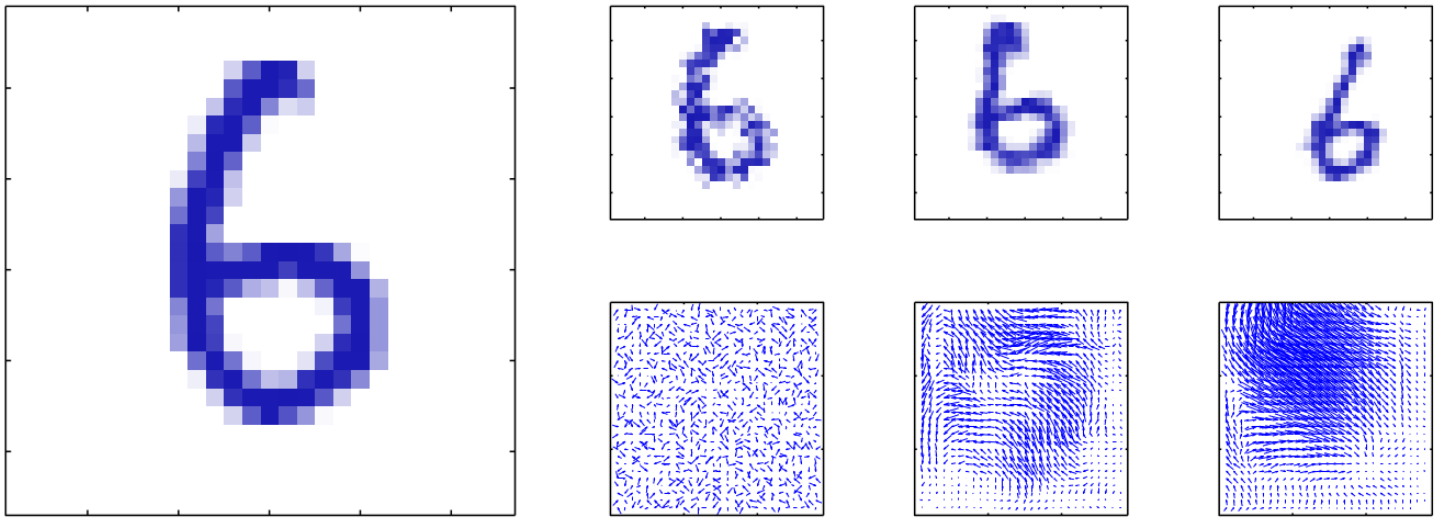


Figure: 5.14. Illustration of the synthetic warping of a handwritten digit. The original image is shown on the left. On the right, the top row shows three examples of warped digits, with the corresponding displacement fields shown on the bottom row.

5.5. Learning Invariances by Augmenting the Data

We consider:

- Transformation governed by a single parameter ξ
- Transformation $s(\boldsymbol{x}, \xi)$ with $s(\boldsymbol{x}, 0) = \boldsymbol{x}$
- Sum-of-squares error function

Error function for the untransformed data (infinite dataset):

$$E = \frac{1}{2} \iint \{y(\boldsymbol{x}) - t\}^2 p(t|\boldsymbol{x}) p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \tag{5.129}$$

Error function for the expanded data:

$$\tilde{E} = \frac{1}{2} \iiint \{y(s(\boldsymbol{x}, \xi)) - t\}^2 p(t|\boldsymbol{x}) p(\boldsymbol{x}) p(\xi) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \, \mathrm{d}\xi \tag{5.130}$$

5.5. Learning Invariances by Augmenting the Data

Expand s as a Taylor series:

$$s(\boldsymbol{x}, \xi) = s(\boldsymbol{x}, 0) + \xi \frac{\partial s}{\partial \xi}(\boldsymbol{x}, \xi)\Big|_{\xi=0} + \frac{\xi^2}{2}\frac{\partial^2 s}{\partial \xi^2}(\boldsymbol{x}, \xi)\Big|_{\xi=0} + O(\xi^3)$$
$$= \boldsymbol{x} + \xi\boldsymbol{\tau} + \frac{1}{2}\xi^2\boldsymbol{\tau}' + O(\xi^3)$$

This allows us to expand the model function:

$$y(s(\boldsymbol{x}, \xi)) = y(x) + \xi\boldsymbol{\tau}^\top \nabla y(\boldsymbol{x}) + \frac{\xi^2}{2}[(\boldsymbol{\tau}')^\top \nabla y(\boldsymbol{x}) + \boldsymbol{\tau}^\top \nabla\nabla y(\boldsymbol{x})\boldsymbol{\tau}] + O(\xi^3)$$

5.5. Learning Invariances by Augmenting the Data

After replacing in $\tilde{E}$:

$$\tilde{E} = E + \lambda\Omega \tag{5.131}$$

with

$$\Omega = \frac{1}{2}\int \left(\tau^\top \nabla y(\boldsymbol{x})\right)^2 p(\boldsymbol{x})\,\mathrm{d}x \tag{5.134}$$

and if we choose $s : \boldsymbol{x} \to \boldsymbol{x} + \xi$

$$\Omega = \frac{1}{2}\int ||\nabla y(\boldsymbol{x})||^2 p(\boldsymbol{x})\,\mathrm{d}x \tag{5.135}$$

This is known as Tikhonov regularization.

5.5. Convolutional networks (Convnets)

Fifth approach:

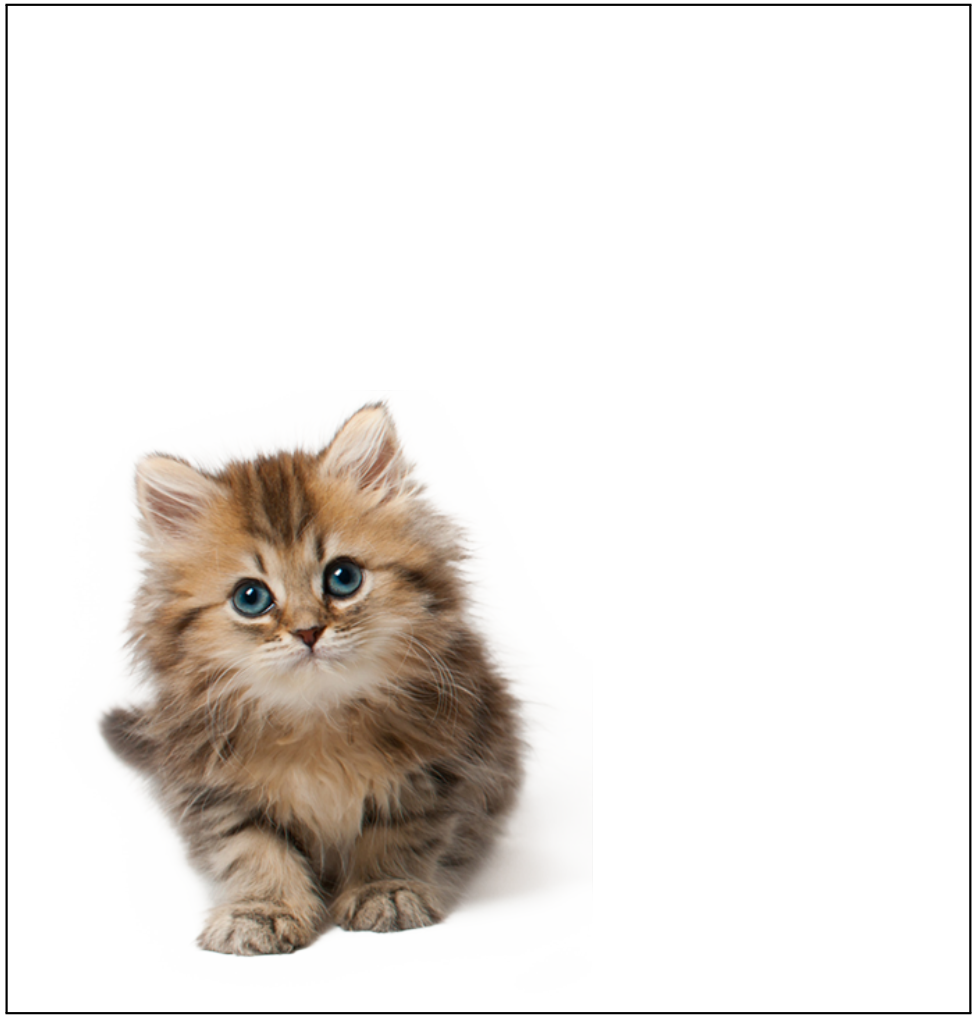


Figure: Cat recognition

5.5. Convolutional networks (Convnets)

Fifth approach:

Figure: Cat recognition

5.5. Convolutional networks (Convnets)

Fifth approach:

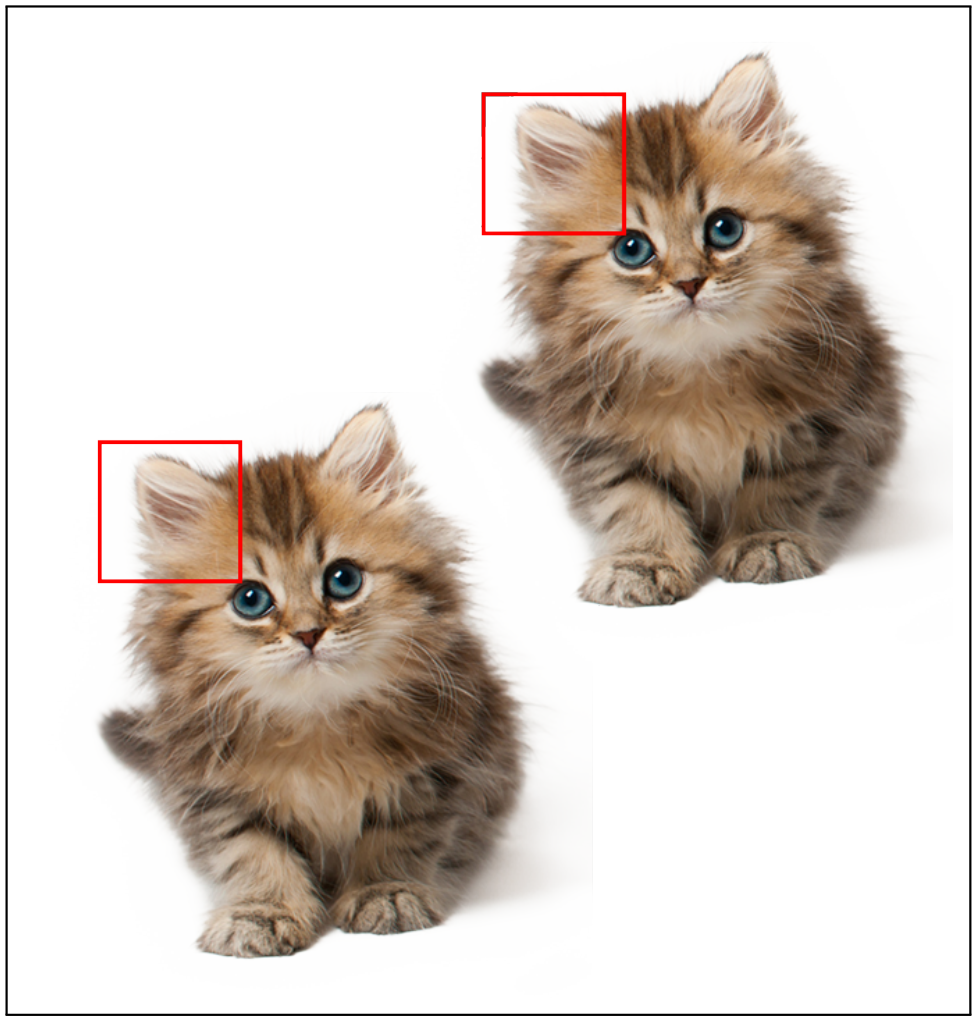


Figure: Cat recognition

Weight Sharing!

5.5. Convolutional networks (Convnets)

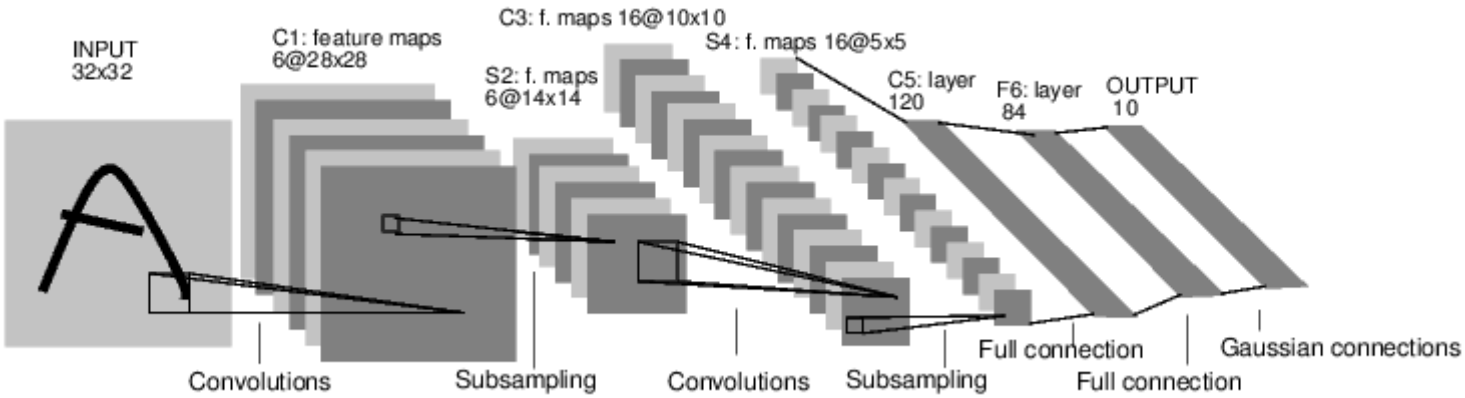


Figure: Architecture of LeNet-5, a Convolution Neural network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.[1]

First layer:

- Without weight sharing: 5x5x6x28x28 = 117600
- with weight sharing: 5x5x6 = 150

[1] Y. LeCun, et al.: Gradient-Based Learning Applied to Document Recognition,1998

5.5. Soft weight sharing

Sixth approach:

$$p(\boldsymbol{w}) = \prod_i p(w_i) \tag{5.136}$$

with

$$p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}\left(w_i | \mu_j, \sigma_j^2\right) \tag{5.137}$$

Total error function:

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \tag{5.139}$$

with

$$\Omega(\boldsymbol{w}) = -\sum_i \ln \left(\sum_{j=1}^{M} \pi_j \mathcal{N}\left(w_i | \mu_j, \sigma_j^2\right) \right) \tag{5.138}$$

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

5.6. Mixture Density Networks

Goal of supervised learning: model $p(\mathbf{t}|\mathbf{x})$.

- Generative approach: model $p(\mathbf{t}, \mathbf{x}) = p(\mathbf{x}|\mathbf{t})p(\mathbf{t})$ and use Bayes' Theorem $p(\mathbf{t}|\mathbf{x}) = {p(\mathbf{x}|\mathbf{t})p(\mathbf{t})}/{p(\mathbf{x})}$
- Discriminative approach: model $p(\mathbf{t}|\mathbf{x})$ directly.

With inverse problems the distribution can be multimodal.

- Forward problem: Model parameters $\rightarrow$ Data/Predictions.
- Inverse problem: Data $\rightarrow$ Model parameters.

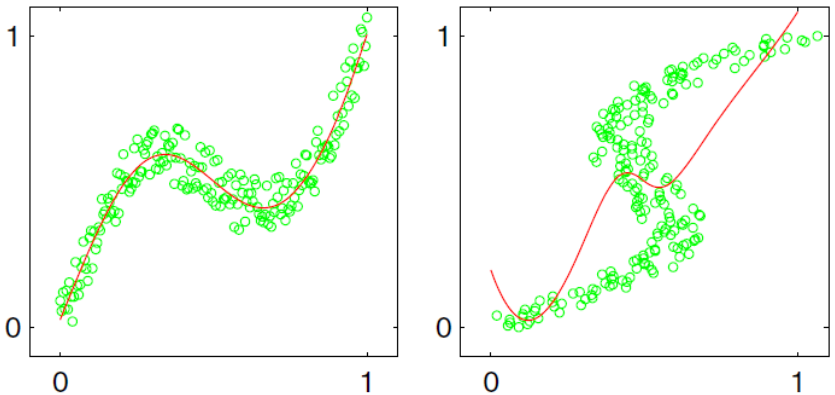


Figure: 5.19: Left: forward problem (x $\rightarrow$ t). Right: inverse problem (t $\rightarrow$ x). A standard ANN trained by least squares approximates the conditional mean.

5.6. Mixture Density Networks

Goal of supervised learning: model $p(\mathbf{t}|\mathbf{x})$.

- Generative approach: model $p(\mathbf{t}, \mathbf{x}) = p(\mathbf{x}|\mathbf{t})p(\mathbf{t})$ and use Bayes' Theorem $p(\mathbf{t}|\mathbf{x}) = {p(\mathbf{x}|\mathbf{t})p(\mathbf{t})}/{p(\mathbf{x})}$
- Discriminative approach: model $p(\mathbf{t}|\mathbf{x})$ directly.

With inverse problems the distribution can be multimodal.

- Forward problem: Model parameters $\rightarrow$ Data/Predictions.
- Inverse problem: Data $\rightarrow$ Model parameters.

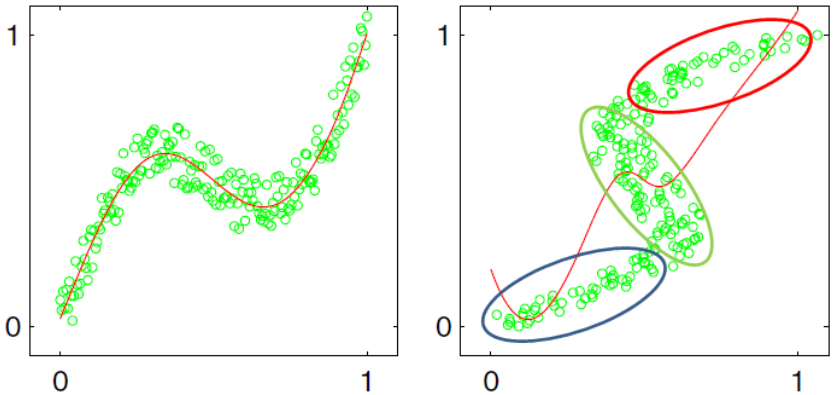


Figure: 5.19: Left: forward problem (x $\rightarrow$ t). Right: inverse problem (t $\rightarrow$ x). A standard ANN trained by least squares approximates the conditional mean.

5.6. Mixture Density Networks

We seek a general framework for modeling $p(\mathbf{t}|\mathbf{x})$.

Mixture Density Network (MDN): mixture model in which both the mixing coefficients $\pi_k(\mathbf{x})$ as well as the component densities $(\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x}))$ are flexible functions of the input vector $\mathbf{x}$.

$$p(\mathbf{t}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}|\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\mathbf{I}\right) \tag{5.148}$$

Recall that the Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{\mu}_k, \Sigma_k\right) \tag{9.7}$$

Note: we can use other distributions for the components, such as Bernoulli distributions if the target variables are binary rather than continuous.

5.6. Mixture Density Networks

- Mixing coefficients must satisfy the constraints

$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, 0 \leq \pi_k(\mathbf{x}) \leq 1 \tag{5.149}$$

which can be achieved using a set of softmax outputs.

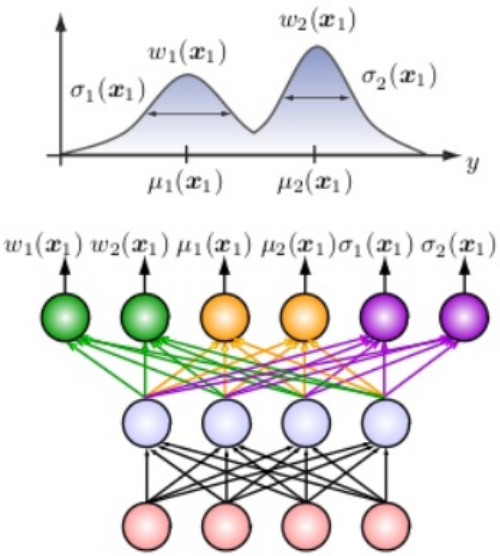
$$\pi_k(\mathbf{x}) = \frac{exp(a_k^\pi)}{\sum_{l=1}^{K} exp(a_l^\pi)} \tag{5.150}$$

- Variances must satisfy $\sigma_k^2 \geq 0$ and so can be represented using

$$\sigma_k(\mathbf{x}) = exp(a_k^\sigma) \tag{5.151}$$

- Means have real components, they can be represented directly by the network output activations $\mu_{kj}(\mathbf{x}) = a_{kj}^\mu$

5.6. Mixture Density Networks



Inputs of activation function

$$z_j = \sum_{i=1}^{4} h_i w_{ij}$$

: Weights → Softmax activation function

$$w_1(\boldsymbol{x}) = \frac{\exp(z_1)}{\sum_{m=1}^{2}\exp(z_m)} \qquad w_2(\boldsymbol{x}) = \frac{\exp(z_2)}{\sum_{m=1}^{2}\exp(z_m)}$$

: Means → Linear activation function

$$\mu_1(\boldsymbol{x}) = z_3 \qquad \mu_1(\boldsymbol{x}) = z_4$$

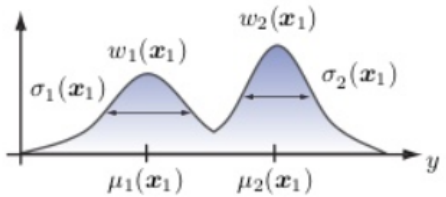
: Variances → Exponential activation function

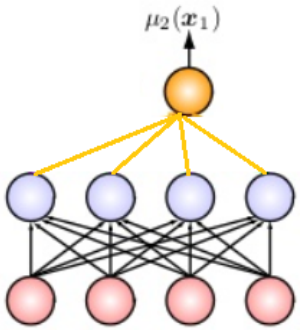
$$\sigma_1(\boldsymbol{x}) = \exp(z_5) \qquad \sigma_2(\boldsymbol{x}) = \exp(z_6)$$

Figure: MDN architecture (from Heiga Zen's slides, researcher at Google working in statistical speech synthesis and recognition)

The total number of network outputs is given by $(L+2)K$, having K (=2) components in the mixture model and L (=1) components in $\mathbf{t}$.

5.6. Mixture Density Networks





Inputs of activation function

$$z_j = \sum_{i=1}^{4} h_i w_{ij}$$

: Weights → Softmax activation function

$$w_1(\boldsymbol{x}) = \frac{\exp(z_1)}{\sum_{m=1}^{2}\exp(z_m)} \qquad w_2(\boldsymbol{x}) = \frac{\exp(z_2)}{\sum_{m=1}^{2}\exp(z_m)}$$

: Means → Linear activation function

$$\mu_1(\boldsymbol{x}) = z_3 \qquad \mu_1(\boldsymbol{x}) = z_4$$

: Variances → Exponential activation function

$$\sigma_1(\boldsymbol{x}) = \exp(z_5) \qquad \sigma_2(\boldsymbol{x}) = \exp(z_6)$$

Figure: MDN architecture (from Heiga Zen's slides, researcher at Google working in statistical speech synthesis and recognition)

The total number of network outputs is given by $(L+2)K$, having K (=2) components in the mixture model and L (=1) components in $\mathbf{t}$.

5.6. Mixture Density Networks

Minimization of the negative logarithm of the likelihood:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w}) \mathbf{I}\right) \right\} \tag{5.153}$$

In order to minimize the error function, we need to calculate the derivatives of the error $E(\mathbf{w})$ with respect to the components of $\mathbf{w}$.

These can be evaluated using backpropagation. We need the derivatives of the error wrt the output-unit activations (δ's).

$$\delta_k^{\pi} = \frac{\partial E_n}{\partial a_k^{\pi}} = \pi_k - \gamma_{nk}, \; \delta_k^{\mu} = \frac{\partial E_n}{\partial a_{kl}^{\mu}} = \gamma_{nk} \left\{ \frac{\mu_{kl} - t_{nl}}{\sigma_k^2} \right\},$$
$$\delta_k^{\sigma} = \frac{\partial E_n}{\partial a_k^{\sigma}} = \gamma_{nk} \left\{ L - \frac{\|\mathbf{t}_n - \boldsymbol{\mu}_k\|^2}{\sigma_k^2} \right\}$$

where $\gamma_{nk} = \gamma_k(\mathbf{t}_n | \mathbf{x}_n) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}}$ and $\mathcal{N}_{nk}$ denotes $\mathcal{N}(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n), \sigma_k^2(\mathbf{x}_n) \mathbf{I})$. γ_{nk} represents $p(\mathbf{x}_n \in \mathcal{G}_k)$

5.6. Mixture Density Networks

Once an MDN has been trained, it can predict the conditional density function of the target data for any given input vector. This conditional density represents a complete description of the generator of the data.

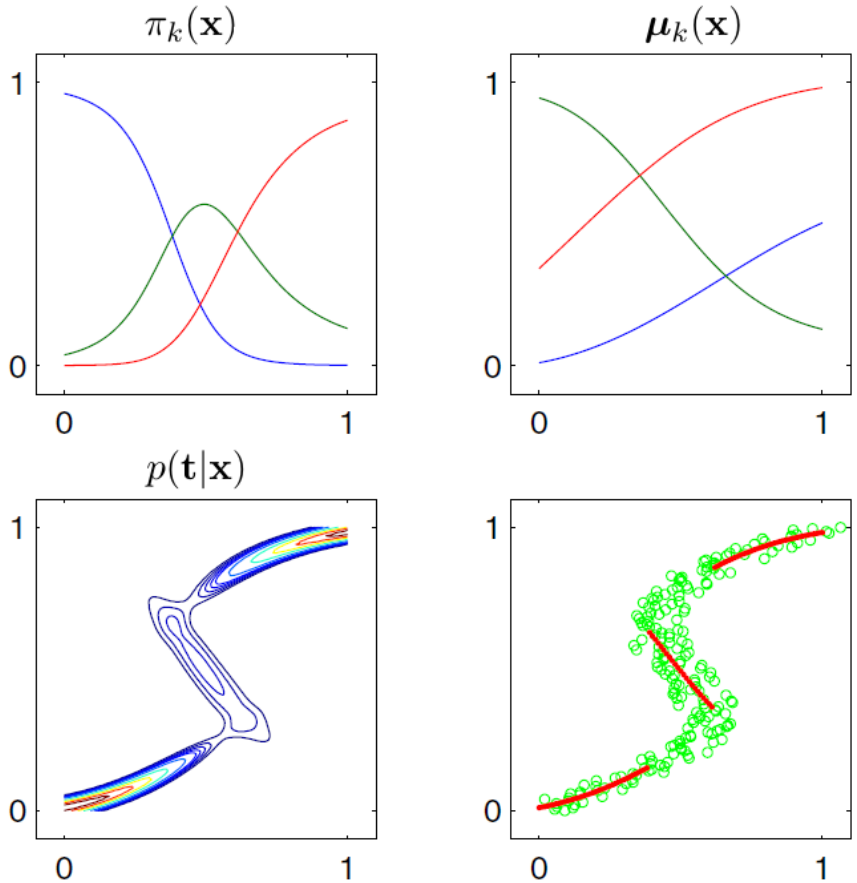


Figure: 5.21. Bottom right: Mean of the most probable component (i.e., the one with the largest mixing coefficient) at each value of $\mathbf{x}$

5.6. Mixture Density Networks

Once an MDN has been trained, it can predict the conditional density function of the target data for any given input vector. This conditional density represents a complete description of the generator of the data.

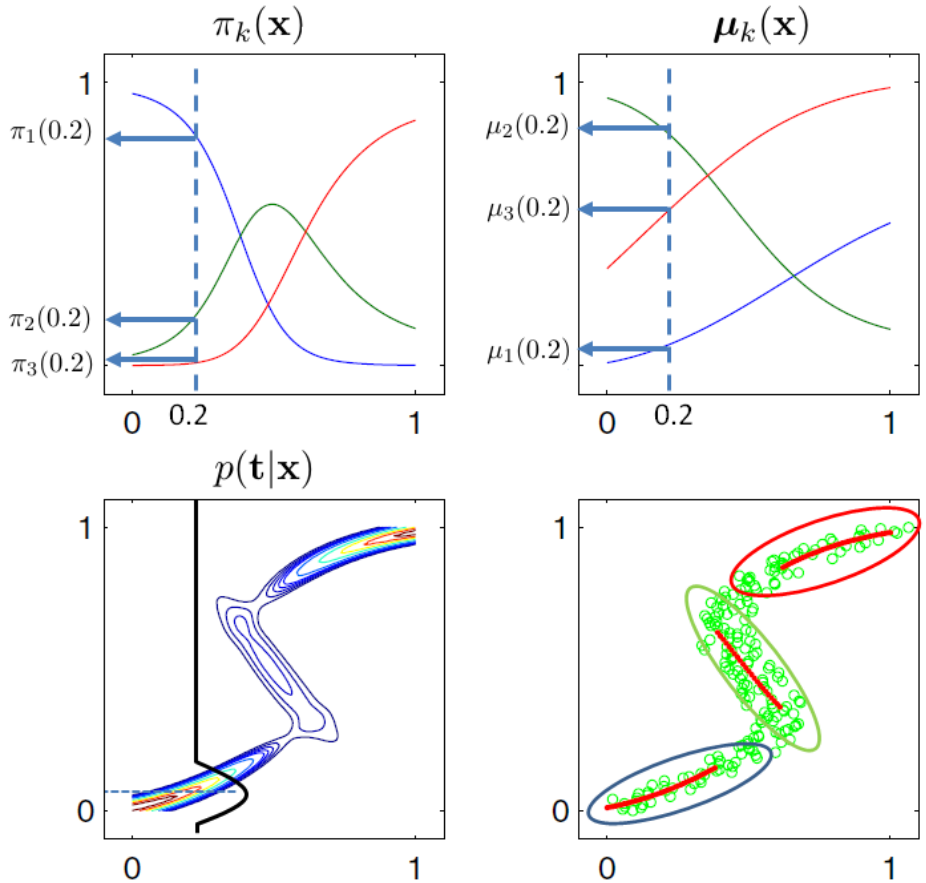


Figure: 5.21. Bottom right: Mean of the most probable component (i.e., the one with the largest mixing coefficient) at each value of $\mathbf{x}$

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

5.7. Bayesian Neural Networks

Main idea:

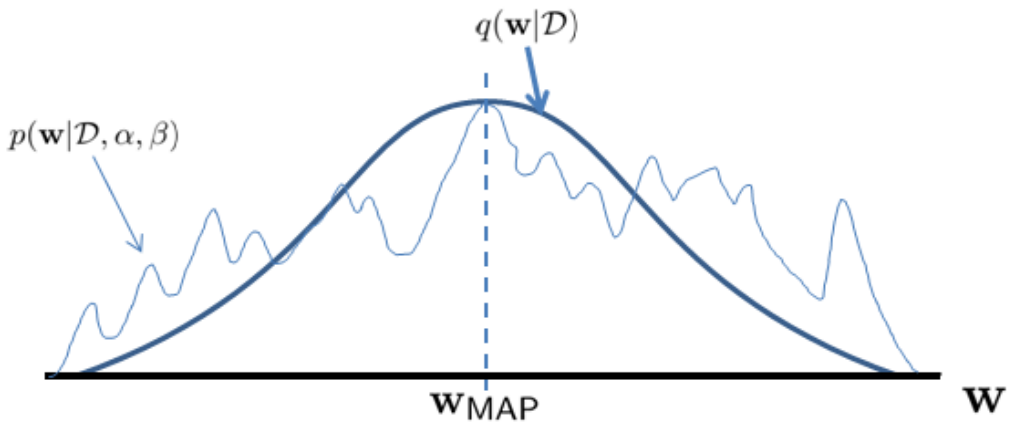
- Apply Bayesian inference to get an entire probability distribution over the network weights, $\mathbf{w}$, given the training data: $p(\mathbf{w}|\mathcal{D})$
  - Classical neural networks training algorithms get a single solution

Approach	**Training**	**Equivalent Problem / Solution**
ML	$\arg\max_{\mathbf{w}} p(t|\mathbf{x},\mathbf{w}) =$ $\arg\max_{\mathbf{w}} \mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1})$	$E(\mathbf{w}) =$ $\frac{1}{2}\sum_{n=1}^{N}\left\{y(\mathbf{x}_n,\mathbf{w})-t_n\right\}^2$
MAP	$\arg\max_{\mathbf{w}} p(\mathbf{w}|t,\mathbf{x},\lambda) =$ $\arg\max_{\mathbf{w}} p(t|\mathbf{x},\mathbf{w})p(\mathbf{w}|\lambda)$ with $p(\mathbf{w}|\lambda)=\mathcal{N}\left(\mathbf{w}\middle|0,\lambda^{-1}I\right)$	$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$
Posterior	$p(\mathbf{w}|\mathcal{D})$	????

5.7. Bayesian Neural Networks

- In multilayered network
  - Highly nonlinear dependence of $y(\mathbf{x}, \mathbf{w})$ on $\mathbf{w}$
    - Exact Bayesian treatment not possible because log of posterior distribution is non-convex
  - Approximate methods are therefore necessary

- First approximation:
  - Laplace approximation: replace posterior by a Gaussian centered at a mode of true posterior



5.7. Bayesian Neural Networks: Simple Regression Case

- Predict single continuous target t from vector $\mathbf{x}$ of inputs assuming hyperparameters α and β are fixed and known.

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \tag{5.161}$$

- Prior over weights assumed Gaussian

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}\mathbf{I}) \tag{5.162}$$

- Likelihood function

$$p(\mathcal{D}|\mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) \tag{5.163}$$

- Posterior is

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta) \propto p(\mathbf{w}|\alpha)p(\mathcal{D}|\mathbf{w}, \beta) \tag{5.164}$$

  - It will be non-Gaussian as a consequence of the nonlinear dependence of $y(\mathbf{x}, \mathbf{w})$ on $\mathbf{w} \rightarrow$ Laplace approximation

5.7. Bayesian Neural Networks: Gaussian approx. to $p(\mathbf{w}|\mathcal{D}, \alpha, \beta)$

- Convenient to maximize the logarithm of the posterior (which corresponds to the regularized sum-of-squares error)

$$\ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2}\sum\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + const \tag{5.165}$$

  The maximum is denoted as $\mathbf{w}_{\text{MAP}}$ which is found using nonlinear optimization methods (e.g. conjugate gradient).

- Having found the mode $\mathbf{w}_{\text{MAP}}$, we can build a local Gaussian approximation

$$q(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}) \tag{5.167}$$

  where $\mathbf{A}$ is in terms of the Hessian of the sum-of-squares error function

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H} \tag{5.166}$$

5.7. Bayesian Neural Networks

Approach	Training	Equivalent Problem / Solution	Testing
ML	$\arg\max_{\mathbf{w}} p(t\|\mathbf{x}, \mathbf{w})$	$\arg\min_{\mathbf{w}} E(\mathbf{w})$ where $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{y(\mathbf{x}_n, \mathbf{w}) - t_n\right\}^2$	$\arg\max_t p(t\|\mathbf{w}_{\text{ML}}, \mathbf{x}) =$ $\arg\max_t \mathcal{N}(t\|y(\mathbf{x}, \mathbf{w}_{\text{ML}}), \beta^{-1})$
MAP	$\arg\max_{\mathbf{w}} p(\mathbf{w}\|t, \mathbf{x}, \lambda)$	$\arg\min_{\mathbf{w}} \tilde{E}(\mathbf{w})$ where $\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$	$\arg\max_t p(t\|\mathbf{w}_{\text{MAP}}, \mathbf{x}) =$ $\arg\max_t \mathcal{N}(t\|y(\mathbf{x}, \mathbf{w}_{\text{MAP}}), \beta^{-1})$
Posterior	$p(\mathbf{w}\|\mathcal{D}, \alpha, \beta)$	$q(\mathbf{w}\|\mathcal{D})$	????

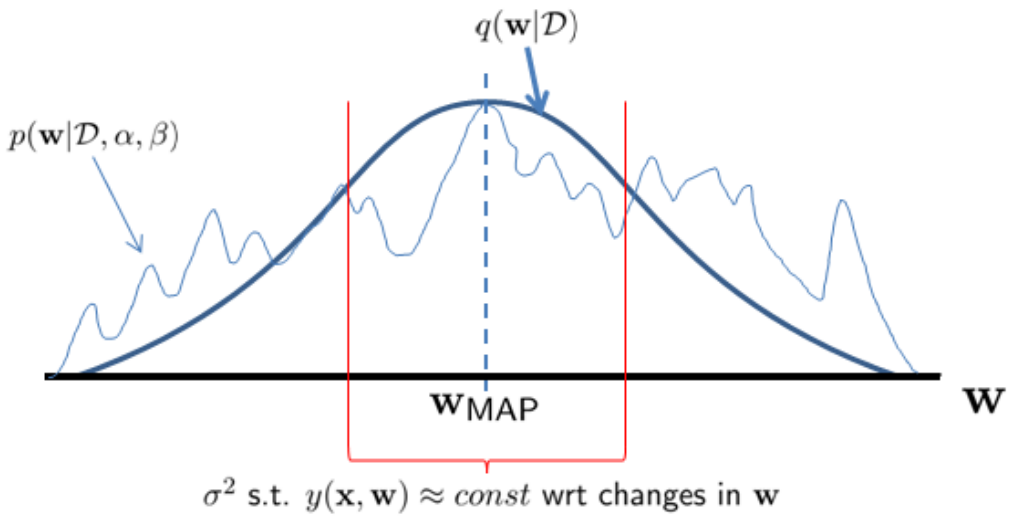
Table: Summary in case of using Gaussian distributions.

5.7. Bayesian Neural Networks: $p(t|\mathbf{x}, \mathcal{D})$

- Similarly, the predictive distribution is obtained by marginalizing wrt this posterior distribution

$$p(t|\mathbf{x}, \mathcal{D}) = \int p(t|\mathbf{x}, \mathbf{w}) q(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{5.168}$$

  - This integration is still analytically intractable due to the nonlinearity of $y(\mathbf{x}, \mathbf{w})$ as a function of $\mathbf{w}$.
- Second approximation: the posterior has small variance compared with the characteristic scales of $\mathbf{w}$ over which $y(\mathbf{x}, \mathbf{w})$ is varying.



5.7. Bayesian Neural Networks: $p(t|\mathbf{x}, \mathcal{D})$

- Similarly, the predictive distribution is obtained by marginalizing wrt this posterior distribution

$$p(t|\mathbf{x}, \mathcal{D}) = \int p(t|\mathbf{x}, \mathbf{w}) q(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{5.168}$$

  - This integration is still analytically intractable due to the nonlinearity of $y(\mathbf{x}, \mathbf{w})$ as a function of $\mathbf{w}$.

- Second approximation: the posterior has small variance compared with the characteristic scales of $\mathbf{w}$ over which $y(\mathbf{x}, \mathbf{w})$ is varying.
  This allows us to make a Taylor series expansion of the network function around $\mathbf{w}_{\text{MAP}}$ and retain only the linear terms:

$$y(\mathbf{x}, \mathbf{w}) \simeq y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) + \mathbf{g}^T(\mathbf{w} - \mathbf{w}_{\text{MAP}}) \tag{5.169}$$

  where $\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\text{MAP}}}$

5.7. Bayesian Neural Networks: $p(t|\mathbf{x}, \mathcal{D})$

- We now have a linear-Gaussian model with a Gaussian distribution for $p(\mathbf{w})$ and a Gaussian for $p(t|\mathbf{w})$ whose mean is a linear function of $\mathbf{w}$ of the form

$$p(t|\mathbf{x}, \mathbf{w}, \beta) \simeq \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) + \mathbf{g}^T(\mathbf{w} - \mathbf{w}_{\text{MAP}}), \beta^{-1}) \tag{5.171}$$

- We can make use of the general result (2.115) for the marginal $p(t)$ to give

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\text{MAP}}), \sigma^2(\mathbf{x})) \tag{5.172}$$

where $\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^T \mathbf{A}^{-1} \mathbf{g}$

2.3.3 Marginal and Conditional Gaussians: Given a marginal Gaussian distribution for $\mathbf{x}$ and a conditional Gaussian distribution for $\mathbf{y}$ given $\mathbf{x}$ in the form $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ and $p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1})$, the marginal distribution of $\mathbf{y}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^T) \tag{2.115}$$

5.7. Bayesian Neural Networks

Approach	Training	Equivalent Problem / Solution	Testing
ML	$\arg\max_{\mathbf{w}} p(t\|\mathbf{x},\mathbf{w})$	$\arg\min_{\mathbf{w}} E(\mathbf{w})$ where $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{y(\mathbf{x}_n,\mathbf{w}) - t_n\right\}^2$	$\arg\max_t p(t\|\mathbf{w}_{\mathsf{ML}},\mathbf{x}) =$ $\arg\max_t \mathcal{N}(t\|y(\mathbf{x},\mathbf{w}_{\mathsf{ML}}),\beta^{-1})$
MAP	$\arg\max_{\mathbf{w}} p(\mathbf{w}\|t,\mathbf{x},\lambda)$	$\arg\min_{\mathbf{w}} \tilde{E}(\mathbf{w})$ where $\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$	$\arg\max_t p(t\|\mathbf{w}_{\mathsf{MAP}},\mathbf{x}) =$ $\arg\max_t \mathcal{N}(t\|y(\mathbf{x},\mathbf{w}_{\mathsf{MAP}}),\beta^{-1})$
Posterior	$p(\mathbf{w}\|\mathcal{D},\alpha,\beta)$	$q(\mathbf{w}\|\mathcal{D})$	$\arg\max_t p(t\|\mathbf{x},\mathcal{D}) =$ $\arg\max_t \mathcal{N}(t\|y(\mathbf{x},\mathbf{w}_{\mathsf{MAP}}),\sigma^2(\mathbf{x}))$

Table: Summary in case of using Gaussian distributions.

5.7. Bayesian Neural Networks: Hyperparameter Optimization

- Estimates are obtained for α and β by maximizing $\ln p(\mathcal{D}|\alpha, \beta)$ given by

$$p(\mathcal{D}|\alpha, \beta) = \int p(\mathcal{D}|\mathbf{w}, \beta) p(\mathbf{w}|\alpha) d\mathbf{w} \tag{5.174}$$

- This is easily evaluated using the Laplace approximation results (4.135). Taking logarithms then gives

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\text{MAP}}) - \frac{1}{2} \ln |\mathbf{A}| + \frac{W}{2} \ln \alpha + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) \tag{5.175}$$

where W is the total number of parameters in $\mathbf{w}$, and the regularized error function is defined by

$$E(\mathbf{w}_{\text{MAP}}) = \frac{\beta}{2} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}_{\text{MAP}}) - t_n\}^2 + \frac{\alpha}{2} \mathbf{w}_{\text{MAP}}^T \mathbf{w}_{\text{MAP}} \tag{5.176}$$

$$Z = \int f(\mathbf{z}) d\mathbf{z} \simeq f(\mathbf{z}_0) \frac{(2\pi)^{W/2}}{|\mathbf{A}|^{1/2}} \tag{4.135}$$

5.7. Bayesian Neural Networks: Hyperparameter Optimization

- Consider first the maximization with respect to α.

$$\alpha = \frac{\gamma}{\mathbf{w}_{\text{MAP}}^{\mathsf{T}} \mathbf{w}_{\text{MAP}}} \tag{5.178}$$

where γ represents the effective number of parameters and is defined by

$$\gamma = \sum_{i=1}^{W} \frac{\lambda_i}{\alpha + \lambda_i} \tag{5.179}$$

and $\beta \mathbf{H} \mathbf{u}_i = \lambda_i \mathbf{u}_i$ is the eigenvalue equation.

- Maximizing the evidence with respect to β gives

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}_{\text{MAP}}) - t_n\}^2 \tag{5.180}$$

5.7. Bayesian Neural Networks: Comparison of Models

- Consider a set of candidate models $\mathcal{H}_i$ to compare.

- We can apply Bayes' Theorem to compute the posterior distribution over models, then pick the model with the largest posterior $p(\mathcal{H}_i|\mathcal{D}) \propto p(\mathcal{D}|\mathcal{H}_i)p(\mathcal{H}_i)$
  - The term $p(\mathcal{D}|\mathcal{H}_i)$ is called the evidence for $\mathcal{H}_i$ and is given by $p(\mathcal{D}|\mathcal{H}_i) = \int p(\mathcal{D}|\mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\mathcal{H}_i)d\mathbf{w}$
  - Assuming that we have no reason to assign strongly different priors $p(\mathcal{H}_i)$, models $\mathcal{H}_i$ are ranked by evaluating the evidence.

- The evidence is approximated by taking (5.175) and substituting the values of α and β obtained from the iterative optimization of these hyperparameters:

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\text{MAP}}) - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{5.175}$$

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.
- Possible Future Topics.

Possible Future Topics: Models

- Self-Organizing Maps (SOM)
  - Ch. 9 "Neural Networks and Learning Machines" (S. Haykin, 2009)
- Radial Basis Function Networks (RBFN)
  - Ch. 5 "Neural Networks for Pattern Recognition" (C.M. Bishop, 1995)
- Convolutional Neural Networks (CNN)
  - AlexNet: "ImageNet Classification with Deep Convolutional Neural Networks" (Krizhevsky et al., NIPS'12)
  - LeNet-5, convolutional neural networks `http://yann.lecun.com/exdb/lenet/`
  - VGGNet: "Very Deep Convolutional Networks for Large-Scale Image Recognition" (Simonyan and Zisserman, arXiv, 2014)
  - GoogleLeNet: "Going deeper with convolutions" (Szegedy et al., arXiv, 2014)
  - DeepFace: "DeepFace: Closing the Gap to Human-Level Performance in Face Verification" (Taigman et al., CVPR'14)
- Recurrent Neural Networks (RNN)
  - LSTM: "Long short-term memory" (Hochreiter and Schmidhuber, Neural Computation, 1997)
  - Boltzmann Machine: Ch. 11.7 from Haykin'09, Ch. 7 from "Introduction to the theory of neural computation" (Hertz et al., 1991)
  - Hopfield Network: Ch. 13.7 from Haykin'09, Ch. 2 from Hertz'91, Ch. 13 from "Neural Networks - A Systematic Introduction" (Raul Rojas, 1996)
- Deep Belief Networks (DBN)
  - Ch. 11.9 from Haykin'09, "A fast learning algorithm for deep belief nets" (Hinton et al., Neural Computation, 2006)

Webpage: `https://project.inria.fr/deeplearning`
Mailing list: `deeplearning@inria.fr`

Possible Future Topics: Problems

- Transfer Learning (or domain adaptation)
  - How to transfer the knowledge gained solving one problem to a different but related problem?
    - Machine learning methods work well under the assumption that the training and test data are drawn from the same feature space and the same distribution. When the distribution changes, it would be nice to reduce the need and effort to recollect a new training data.
- Integrating supervised and unsupervised learning in a single algorithm
  - It seems that Deep Boltzmann Machines do this, but there are issues of scalability

Webpage: `https://project.inria.fr/deeplearning`
Mailing list: `deeplearning@inria.fr`