



THE UNIVERSITY *of* EDINBURGH
informatics

Machine Learning Practical (MLP)

Deep Neural Networks (2)

Changjian Li

Example: Multi-layer network



$$h^{(1)} = \text{sigmoid}(w^{(1)}x + b^{(1)}) \quad h^{(2)} = \text{sigmoid}(w^{(2)}h^{(1)} + b^{(2)}) \quad y = \text{sigmoid}(w^{(3)}h^{(2)} + b^{(3)})$$

Example: Multi-layer network



$$h^{(1)} = \text{sigmoid}(w^{(1)}x + b^{(1)}) \quad h^{(2)} = \text{sigmoid}(w^{(2)}h^{(1)} + b^{(2)}) \quad y = \text{sigmoid}(w^{(3)}h^{(2)} + b^{(3)})$$

A general notation for a multi-layer network:

$$h^{(l)} = a(z^{(l)}) \quad l = 1, 2, 3$$

where $z^{(l)} = w^{(l)}h^{(l-1)} + b^{(l)}$ and $h^{(0)} = x$ and $h^{(3)} = y$

Training multi-layer networks

Training multi-layer networks: Credit assignment

- Hidden units make training the weights more complicated, since the hidden units affect the error function indirectly via all the outputs
- The credit assignment problem
 - what is the “error” of a hidden unit?
 - how important is input-hidden weight $w_{ji}^{(l)}$ at layer l to output unit k ?

Training multi-layer networks: Credit assignment

- Hidden units make training the weights more complicated, since the hidden units affect the error function indirectly via all the outputs
- The credit assignment problem
 - what is the “error” of a hidden unit?
 - how important is input-hidden weight $w_{ji}^{(l)}$ at layer l to output unit k ?
- *What is the gradient of the error with respect to each weight?*
(How to compute grads_wrt_params?)

Numerical differentiation

$$\frac{\partial E}{\partial w_{ji}^{(l)}} \approx \frac{E(w_{ji}^{(l)} + \epsilon) - E(w_{ji}^{(l)})}{\epsilon}$$

where ϵ is a small number.

- Computationally expensive, each evaluation of E requires 2 forward passes ($O(|W|)$ evaluations) where $|W|$ is the number of weights. We have to evaluate the gradient for each weight ($O(|W|^2)$).
- Gradients might be noisy, need to average over multiple evaluations
- In practice, numerical differentiation is not used for training neural networks but provides a powerful check on the correctness of the software

Symbolic differentiation

- Transform an expression composed of arithmetic operations and symbols, into an expression representing its derivative.
 - For example: $g(x) = \cos(x) + 2x - e^x$, $f(g) = 4g^2$
 $\rightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x} = 8 * (\cos(x) + 2x - e^x) * (-\sin(x) + 2 - e^x)$

Symbolic differentiation

- Transform an expression composed of arithmetic operations and symbols, into an expression representing its derivative.

- For example: $g(x) = \cos(x) + 2x - e^x$, $f(g) = 4g^2$

$$\rightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x} = 8 * (\cos(x) + 2x - e^x) * (-\sin(x) + 2 - e^x)$$

- Resulting expressions can be complex and computationally expensive (*expression swell*).

- $f(x) = \frac{e^{wx+b} + e^{-(wx+b)}}{e^{wx+b} - e^{-(wx+b)}}$

$$\rightarrow \frac{\partial f}{\partial w} = \frac{(-xe^{(-b-wx)} - xe^{b+wx})(e^{-b-wx} + e^{b+wx})}{(-e^{-b-wx} + e^{b+wx})^2} + \frac{-xe^{-b-wx} + xe^{b+wx}}{-e^{-b-wx} + e^{b+wx}}$$

Hand-derived differentiation

To derive the backpropagation equations by hand and then to implement them explicitly in software

- is error-prone and tedious
- can also result in redundant computations as the forward propagation equations are coded separately from the backpropagation equations

Automatic differentiation (AD)

- AD is a set of techniques to evaluate the mathematical functions, as well as compute their derivatives beside them.
 - express composite functions into variables and elementary operations (e.g., addition, subtraction, multiplication, and division)

Automatic differentiation (AD)

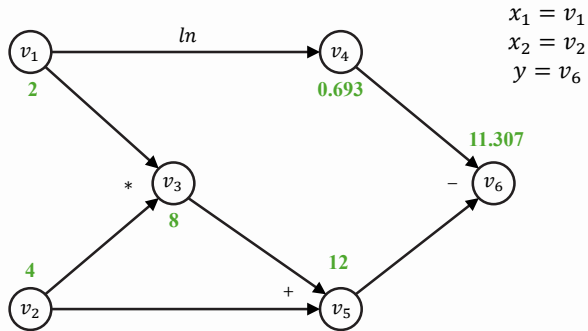
- AD is a set of techniques to evaluate the mathematical functions, as well as compute their derivatives beside them.
 - express composite functions into variables and elementary operations (e.g., addition, subtraction, multiplication, and division)
 - **Forward-mode**: compute the derivative of the function with respect to each input
 - **Reverse-mode**: compute the derivative of the function with respect to each output

Automatic differentiation (AD)

- AD is a set of techniques to evaluate the mathematical functions, as well as compute their derivatives beside them.
 - express composite functions into variables and elementary operations (e.g., addition, subtraction, multiplication, and division)
 - **Forward-mode**: compute the derivative of the function with respect to each input
 - **Reverse-mode**: compute the derivative of the function with respect to each output
- Example: $y = f(x_1, x_2) = x_1x_2 + x_2 - \ln(x_1)$, $x_1 = 2$, $x_2 = 4$
- Chain rule: $z = h(y)$, $y = g(x)$, the derivative $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$

Step One: forward pass

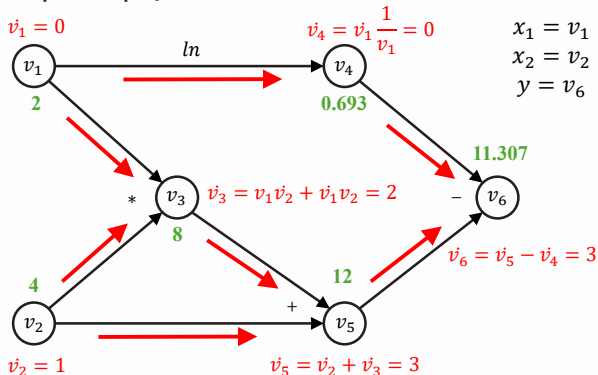
Computational graph with intermediate variables $\{v_i\}$



Forward-mode AD: forward passes

Derivative $\dot{v}_i^j$: the derivative of node v_i to the input x_j .

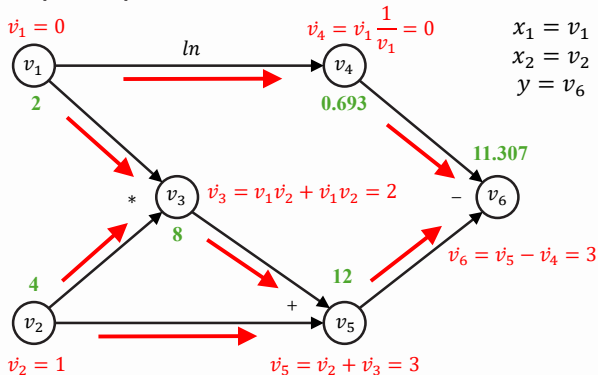
→ Note: we omit the superscript j , and take $\dot{v}_i$ as the derivative to x_2 .



Forward-mode AD: forward passes

Derivative $\dot{v}_i^j$: the derivative of node v_i to the input x_j .

→ Note: we omit the superscript j , and take $\dot{v}_i$ as the derivative to x_2 .

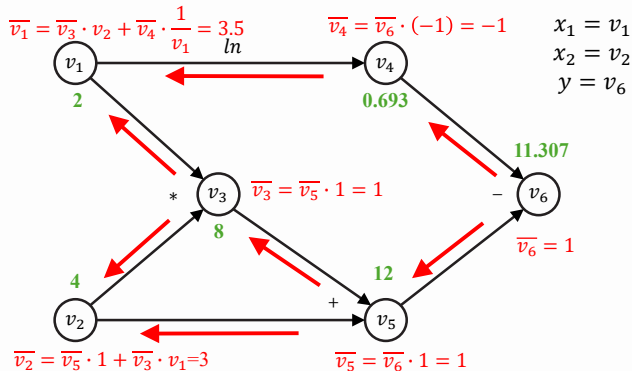


Complexity: $O(|n|) \Rightarrow f : \mathbb{R}^n \rightarrow \mathbb{R}^m, n \ll m$

Reverse-mode AD: backward passes

Derivative $\overline{v}_i^j$: the derivative of output y^j to the internal node v_i .

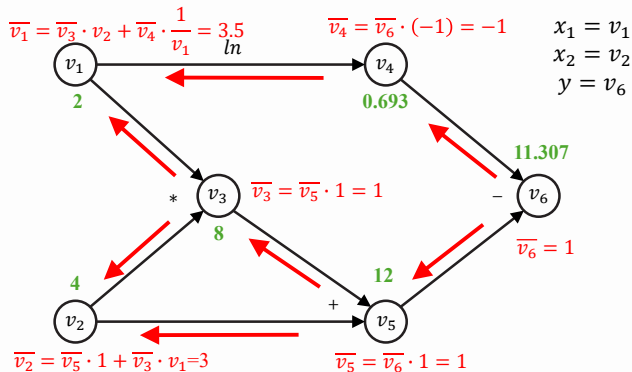
→ Note: we omit the superscript j , since we only have one output in this case.



Reverse-mode AD: backward passes

Derivative $\overline{v}_i^j$: the derivative of output y^j to the internal node v_i .

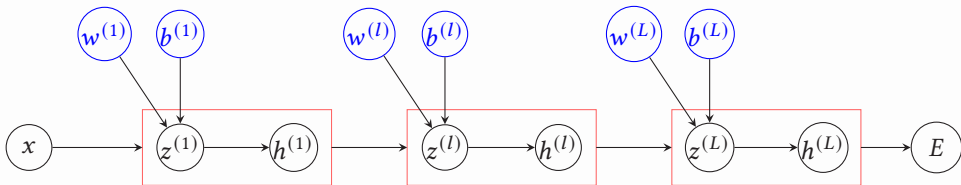
→ Note: we omit the superscript j , since we only have one output in this case.



Complexity: $O(|m|) \Rightarrow f : \mathbb{R}^n \rightarrow \mathbb{R}^m, n \gg m$

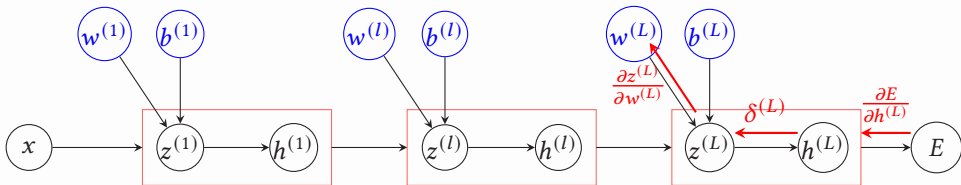
Backpropagation is a special reverse-mode AD algorithm, designed for neural network optimization ($f : \mathbb{R}^n \rightarrow \mathbb{R}^m, n \gg m$).

Backpropagation



Remember $h^{(l)} = a(z^{(l)})$ and $z^{(l)} = w^{(l)} h^{(l-1)} + b^{(l)}$.

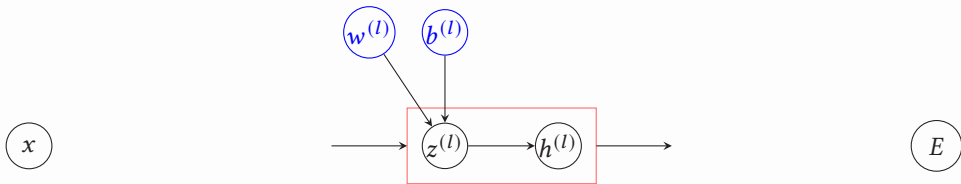
Backpropagation



Remember $h^{(l)} = a(z^{(l)})$ and $z^{(l)} = w^{(l)} h^{(l-1)} + b^{(l)}$.

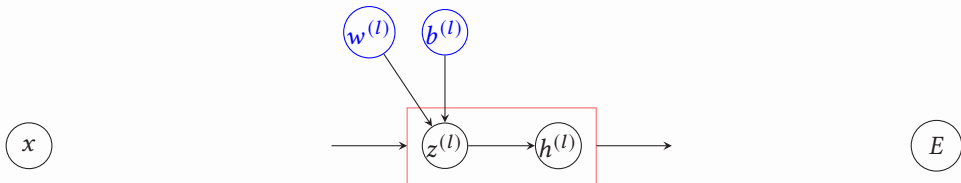
$$\begin{aligned}
 \frac{\partial E}{\partial w^{(L)}} &= \frac{\partial E}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} \\
 &= \underbrace{\frac{\partial E}{\partial h^{(L)}} \cdot a'(z^{(L)}) \cdot h^{(L-1)}}_{\delta^{(L)}} \\
 &= \delta^{(L)} \cdot h^{(L-1)}
 \end{aligned}$$

Backpropagation



Remember $h^{(l)} = a(z^{(l)})$ and $z^{(l)} = w^{(l)} h^{(l-1)} + b^{(l)}$.

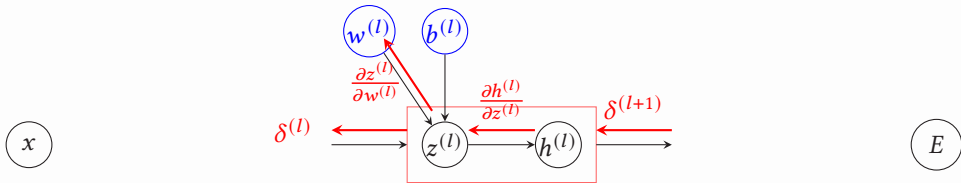
Backpropagation



Remember $h^{(l)} = a(z^{(l)})$ and $z^{(l)} = w^{(l)} h^{(l-1)} + b^{(l)}$.

$$\frac{\partial E}{\partial w^{(l)}} = \frac{\partial E}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \dots \cdot \frac{\partial z^{(l+2)}}{\partial h^{(l+1)}} \cdot \frac{\partial h^{(l+1)}}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial h^{(l)}} \cdot \frac{\partial h^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}}$$

Backpropagation



Remember $h^{(l)} = a(z^{(l)})$ and $z^{(l)} = w^{(l)} h^{(l-1)} + b^{(l)}$.

$$\frac{\partial E}{\partial w^{(l)}} = \underbrace{\frac{\partial E}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdots \frac{\partial z^{(l+2)}}{\partial h^{(l+1)}}}_{\delta^{(l+2)}} \cdot \underbrace{\frac{\partial h^{(l+1)}}{\partial z^{(l+1)}}}_{w^{(l+2)}} \cdot \underbrace{\frac{\partial z^{(l+1)}}{\partial h^{(l)}}}_{a'(z^{(l)})}}_{\delta^{(l+1)}} \cdot \underbrace{\frac{\partial h^{(l)}}{\partial z^{(l)}}}_{w^{(l+1)}} \cdot \underbrace{\frac{\partial z^{(l)}}{\partial w^{(l)}}}_{h^{(l-1)}}_{\delta^{(l)}}$$

Backpropagation key equations

Recursive definition:

$$\frac{\partial E}{\partial w^{(l)}} = \delta^{(l)} \cdot h^{(l-1)} \quad \text{where} \quad \delta^{(l)} = \begin{cases} \frac{\partial E}{\partial h^{(L)}} \cdot a'(z^{(L)}), & \text{if } l = L \\ \delta^{(l+1)} \cdot w^{(l+1)} \cdot a'(z^{(l)}), & \text{if } l = 1, \dots, L-1 \end{cases}$$

Backpropagation algorithm

```
1: procedure BACKPROPAGATION( $x, w, b$ )
2:    $h^{(0)} \leftarrow x$ 
3:   for  $l \leftarrow 1, \dots, L$  do                                     ▶ Forward pass
4:      $z^{(l)} \leftarrow w^{(l)} h^{(l-1)} + b^{(l)}$ 
5:      $h^{(l)} \leftarrow a(z^{(l)})$ 
6:   for  $l \leftarrow L, \dots, 1$  do                                   ▶ Backward pass
7:     if  $l = L$  then
8:        $\delta^{(L)} \leftarrow \frac{\partial E}{\partial h^{(L)}} \odot a'(z^{(L)})$ 
9:     else
10:       $\delta^{(l)} \leftarrow (w^{(l+1)})^T \delta^{(l+1)} \odot a'(z^{(l)})$ 
11:       $\frac{\partial E}{\partial w^{(l)}} \leftarrow \delta^{(l)} h^{(l-1)T}$ 
12:       $\frac{\partial E}{\partial b^{(l)}} \leftarrow \delta^{(l)}$ 
```

Generalisation and Overfitting

Assume that you have a pool of neural networks with different designs which are trained on the same data.

How to choose the best one?

Training / Test / Validation Data

- Partitioning the data...
 - **Training** data – data used for training the network
 - **Validation** data – frequently used to measure the error of a network on “unseen” data *during training* (e.g., after each epoch)
 - **Test** data – “unseen” data, ideally only used once to report performance
- Frequent use of the same test data can indirectly “tune” the network to that data (e.g. by influencing choice of *hyperparameters* such as learning rate, number of hidden units, number of layers, ...)

Measuring generalisation

- Generalisation Error – The predicted error on unseen data. How can the generalisation error be estimated?
 - Error over training set $\mathcal{T}$

$$E_{\text{train}} = - \sum_{n=1}^{|\mathcal{T}|} \sum_{k=1}^K t_k^n \ln y_k^n$$

- Error over validation set $\mathcal{V}$

$$E_{\text{val}} = - \sum_{n=1}^{|\mathcal{V}|} \sum_{k=1}^K t_k^n \ln y_k^n$$

Cross-validation

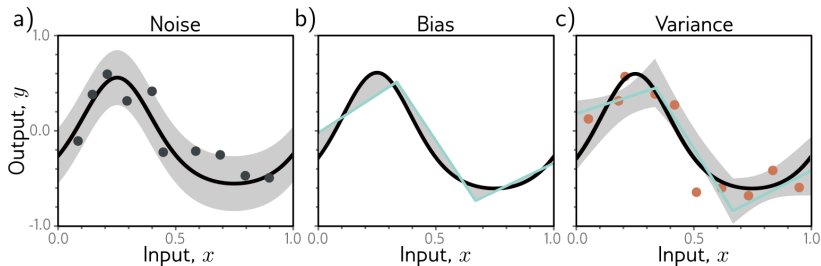
- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalisation performance on this set
 1. Train network in usual way on training data
 2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (**not** the training set)

Cross-validation

- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalisation performance on this set
 1. Train network in usual way on training data
 2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (**not** the training set)
- *n-fold* Cross-validation: divide the data into n partitions; select each partition in turn to be the validation set, and train on the remaining $(n - 1)$ partitions. Estimate generalisation error by averaging over all validation sets.
 - Core ideas: use all the data; obtain a robust generalization evaluation

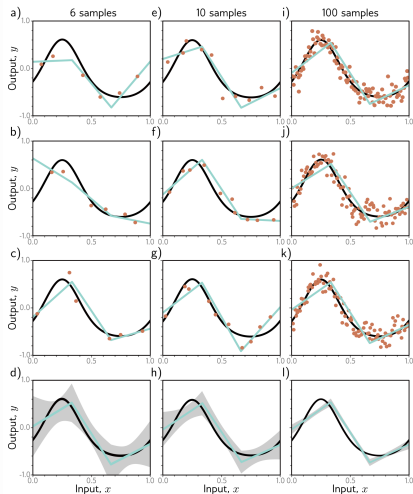
What are the possible sources of error when a model fails to generalize (perfectly)?

Sources of error



- **Noise** – error due to the inherent randomness in the data
- **Bias** – error due to the model's assumptions
- **Variance** – error due to the model's sensitivity to the training data

Reducing variance



- **Variance** – error due to the model's sensitivity to the training data
- Training on more data reduces the variance

Reducing bias

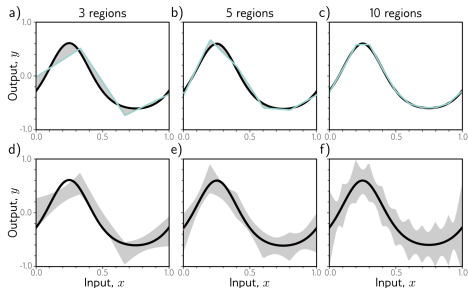
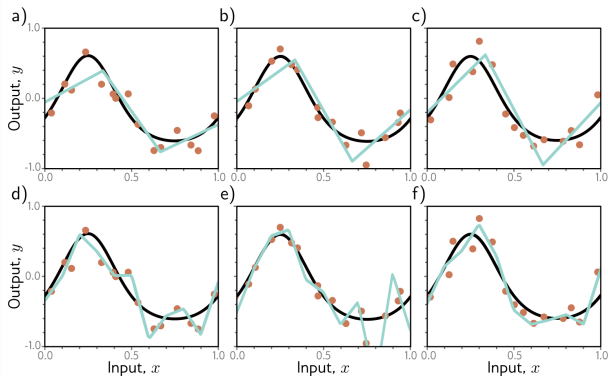


Image credit: Fig 8.7, UDL, Prince

- **Bias** – error due to the model's assumptions
- Increasing the model complexity reduces the bias
- But increasing the model complexity also increases the variance (known as the bias-variance trade-off)

Overfitting



- Underfitting corresponds to a network function not well fit to the training set (too little flexibility)
- Overfitting corresponds to a network function too closely fit to the training set (too much flexibility)

Choosing hyperparameters

- **Hyperparameters** are parameters that are not learned during training but are set before training (*e.g.* model width, depth)
- Don't know bias and variance of a model on unseen data

Choosing hyperparameters

- **Hyperparameters** are parameters that are not learned during training but are set before training (*e.g.* model width, depth)
- Don't know bias and variance of a model on unseen data

How to control the model capacity?

Choosing hyperparameters

- **Hyperparameters** are parameters that are not learned during training but are set before training (*e.g.* model width, depth)
- Don't know bias and variance of a model on unseen data

How to control the model capacity?

- Structural stabilisation – control the number of weights
- Early stopping and regularisation – control the *effective capacity* of the network

Structural Stabilisation

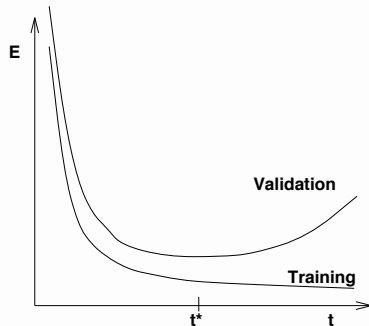
Directly control the number of weights:

- Compare models with different numbers of hidden units
- Start with a large network and reduce the number of weights by pruning individual weights or hidden units
- Weight sharing — use prior knowledge to constrain the weights on a set of connections to be equal.
→ Convolutional Neural Networks

Early Stopping

The strategy to use validation set to decide when to stop training

- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalisation predicted to be at point of minimum validation set error
- “Effective Flexibility” increases as training progresses
- Network weights become more tuned to training data
- Orthogonal to weight regularisation



Summary

- Backpropagation
- Generalisation and overfitting
- Reading from Goodfellow et al, *Deep Learning*
 - sections 5.2–5.3
 - sections 7.1–7.5
 - sections 6.5
- Optional reading from UDL, Prince *Chapter 8*



<https://forms.cloud.microsoft/e/WdeYNv4A5c>