Chapter 6
Multilayer Neural Networks

Instructor: Dr. Wei-Yang Lin

Dept. of CSIE, National Chung Cheng University
mailto:wylin@cs.ccu.edu.tw

Outline

6.1 Introduction

- There are many real-world problems for which linear discriminants are insufficient for minimum error.
- With a clever choice of nonlinear functions, we can obtain arbitrary decision boundaries leading to minimum error.
- The difficulty lies in how to choose nonlinear functions.
- **Multilayer neural networks** is a way to learn the nonlinearity as the linear discriminant.

- The key power provided by multilayer neural networks is that a nonlinear classifier can be learned from training data.
- Neural networks are thus a flexible technique for doing pattern classification with complicated models.
- Neural networks are extremely powerful and apply well to a vast array of real-world applications.
- However, they do not exempt designers from detailed knowledge of the problem domain.

- One of the most popular methods for training neural networks is based on gradient descent.
  - i.e., the **backpropagation algorithm**.
- We will study backpropagation in depth because it is powerful, useful, and relatively easy to understand.
- Many other training methods are modifications of it.
- The backpropagation algorithm has shown success on many real-world problems

- An inevitable problem in the use of neural networks
  - If too many free parameters are used, generalization will be poor.
  - Conversely, if too few parameters are used, the training data cannot be learned adequately.
- **Regularization** is to adjust the complexity of the network.
- In this Chapter, we will explore a number of regularization methods.

Outline

6.1 Introduction

6.2 **Feedforward Operation and Classification**

6.3 Back Propagation Algorithm

6.4 Error Surfaces

6.8 Practical Techniques for Improving Backpropagation

6.2 Feedforward Operation and Classification

- Figure 1 shows a three-layer neural network.
- This one consists of an input layer, a **hidden layer**, and an output layer.
- The layers are interconnected by modifiable weights, represented by links between layers.
- There is a single **bias unit** that is connected to each unit other than the input units.

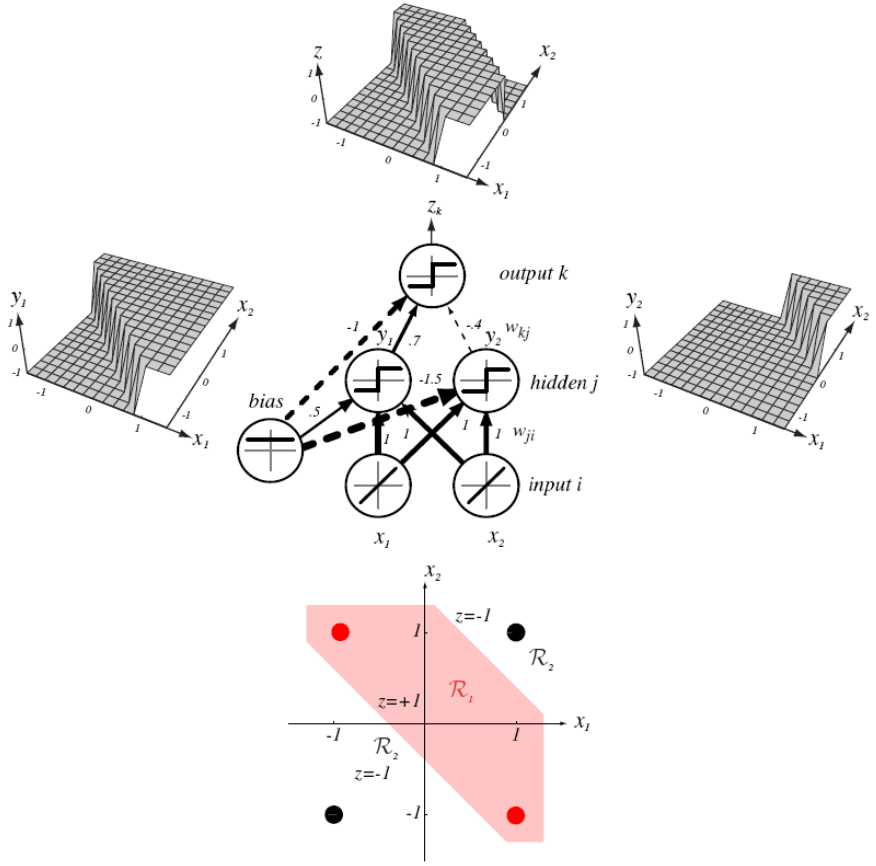

Figure 1: The XOR problem can be solved by a three-layer network.

- The function of units is loosely based on properties of biological neurons, and hence are called **neurons**.
- We are interested in the use of such networks for pattern classification.
- The input units represent the components of a feature vector.
- The signals emitted by output units are the the values of discriminant functions used for classification.

- We can clarify the operation of such a network on the exclusive-OR problem.
- A three-layer network can solve this problem whereas a liner classifier cannot.
- An input vector is presented to the input layer.
- Each hidden unit computes the weighted sum of its inputs to form its net activation.
- For simplicity, we augment both the input vector, by appending a feature value $x_0 = 1$, as well as the weight vector, by appending a value w_0.

- We can then write

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \mathbf{w}_j^t \mathbf{x}, \qquad (1)$$

  where the subscript i indexes units in the input layer, j in the hidden; w_{ji} denotes the weight on the link connecting the input unit i and the hidden unit j.

- In analogy with neurobiology, such connections are called **synapses** and the values of the connections the **synaptic weights**.

- Each hidden unit emits an output that is

$$y_j = f(net_j). \tag{2}$$

- This $f(\cdot)$ is called the **activation function** or merely **nonlinearity** of a unit.
- A simple activation function, the *sign* function, is shown in Figure 1.

$$f(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ -1, & \text{if } net < 0. \end{cases} \tag{3}$$

- There is only one output unit in Figure 1. In general, one can have more than one output unit.
- An output unit computes its net activation as

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}, \qquad (4)$$

  where the subscript k indexes units in the output layer and n_H denotes the number of hidden units.
- Note that the bias unit is equivalent to a hidden unit whose output is always 1.

- An output unit emits

$$z_k = f(net_k). \tag{5}$$

6.2.1 General Feedforward Operation

- Clearly, we can generalize the above discussion to more inputs, other nonlinearities, and arbitrary number of output units.
- For classification, we will have c output units, one for each category. Each output unit is a discriminant function $g_k(\mathbf{x})$.
- An activation function does not have to be a *sign* function.
- We often require an activation function to be differentiable.

Outline

6.1 Introduction

6.2 Feedforward Operation and Classification

6.3 **Back Propagation Algorithm**

6.4 Error Surfaces

6.8 Practical Techniques for Improving Backpropagation

6.3 Back Propagation Algorithm

- Back propagation is one of the simplest methods for supervised learning.
- Supervised learning consists of presenting an input pattern and changing the network parameters to bring the outputs closer to the *target* values.
- Figure 2 shows a three-layer network and the notation we shall use.

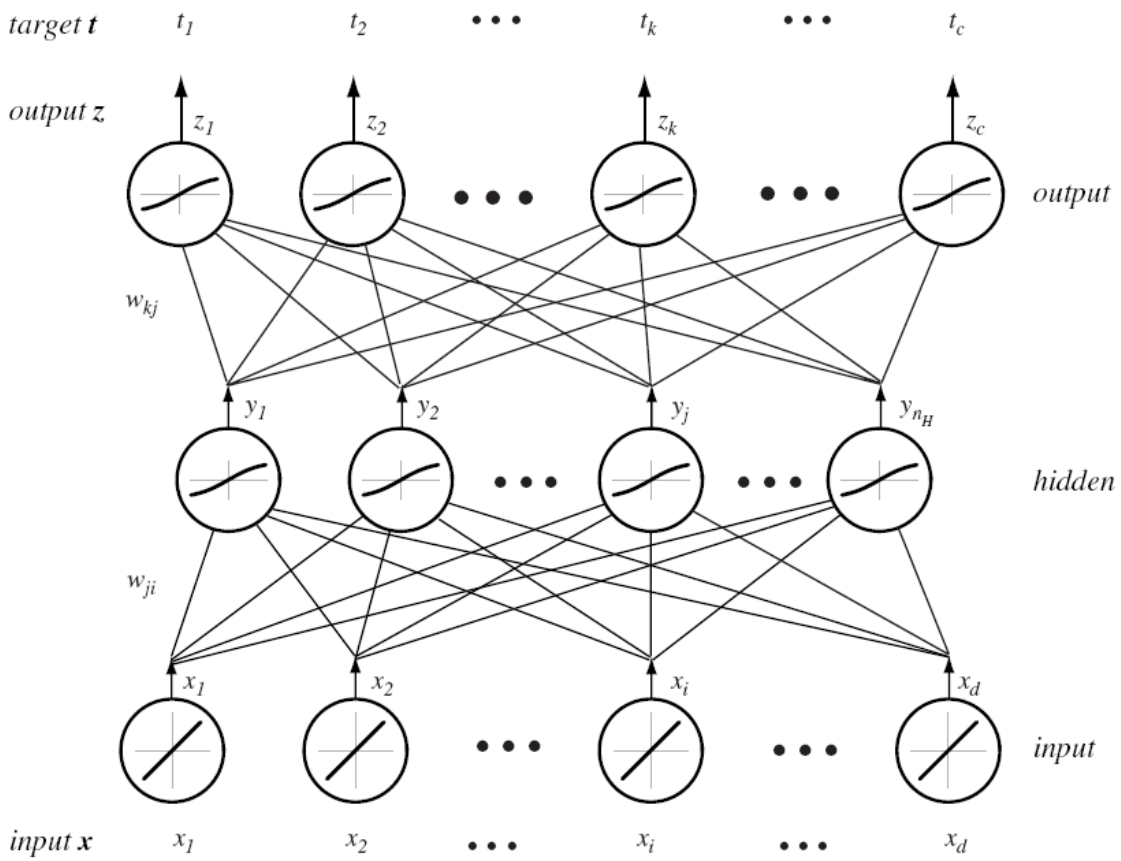

Figure 2: A d-n_H-c fully connected three-layer network.

6.3.1 Network Learning

- Present a training sample and determine the results at the output layer.
- These outputs are compared to the target values; any difference corresponds to an error.
- This error is a function of weights and is minimized when the outputs match the target values.
- Thus, the weights are adjusted to reduce this measure of error.

- We consider the **training error** to be the sum over output units of the squared difference between the desired output t_k and the network output z_k.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2, \tag{6}$$

  where $\mathbf{t}$ and $\mathbf{z}$ are target and network output vectors of length c and $\mathbf{w}$ represents all the weights in a network.

- The backpropagation is based on **gradient descent**.

Gradient Descent

- This video explains how gradient descent works with a simple example.
- Basic intuition and explanation are revealed.
- This explanation should be useful for the beginners of deep learning, machine learning and neural network.

- The weights are initialized with random values, and then they are changed in a direction that will reduce the error:

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}, \tag{7}$$

where η is the **learning rate**, and indicates the relative size of the change in weights.

- The updating rule is given by

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m), \tag{8}$$

where m denotes the iteration index.

- The updating rule can also be written in component form

$$\Delta w_{pq} = -\eta \, \frac{\partial J}{\partial w_{pq}}. \tag{9}$$

- Consider the hidden-to-output weights w_{kj}, we have

$$\frac{\partial J}{\partial \, w_{kj}} = \frac{\partial J}{\partial \, net_k} \cdot \frac{\partial \, net_k}{\partial \, w_{kj}} = -\delta_k \cdot \frac{\partial \, net_k}{\partial \, w_{kj}}, \tag{10}$$

  where the sensitivity of unit k is defined to be

$$\delta_k = -\frac{\partial J}{\partial \, net_k}. \tag{11}$$

- Assuming that the activation function $f(\cdot)$ is differentiable, we differentiate Eq. (6) and find that

$$\delta_k = -\frac{\partial J}{\partial\, net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial\, z_k}{\partial\, net_k} = (t_k - z_k) \cdot f'(net_k).$$

- The last derivative in Eq. (10) is found using Eq. (4):

$$\frac{\partial\, net_k}{\partial\, w_{kj}} = y_j. \tag{12}$$

- These results give the weight update or learning rule for the hidden-to-output weights:

$$\Delta w_{kj} = \eta\, \delta_k\, y_j = \eta\, (t_k - z_k)\, f'(net_k)\, y_j. \tag{13}$$

- The learning rule for the input-to-hidden units is more subtle.
- From Eq. (9), we have

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \tag{14}$$

- The first term requires a bit of care:

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\Big[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\Big] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\cdot\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)\cdot w_{kj}. \qquad (15)
\end{aligned}$$

- The sensitivity for a hidden unit is defined as:

$$\delta_j \equiv f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k. \tag{16}$$

- In plain English, the sensitivity at a hidden unit is the sum of individual sensitivity at the output units weighted by the hidden-to-output weights w_{kj}, all multiplied by $f'(net_j)$.

- Thus, the updating rule for the input-to-hidden weights is

$$\Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i. \quad (17)$$

- Equations (13) and (17) give the backpropagation algorithm.
- During training, an error is propagated from the output layer back to the hidden layer (Fig. 3).

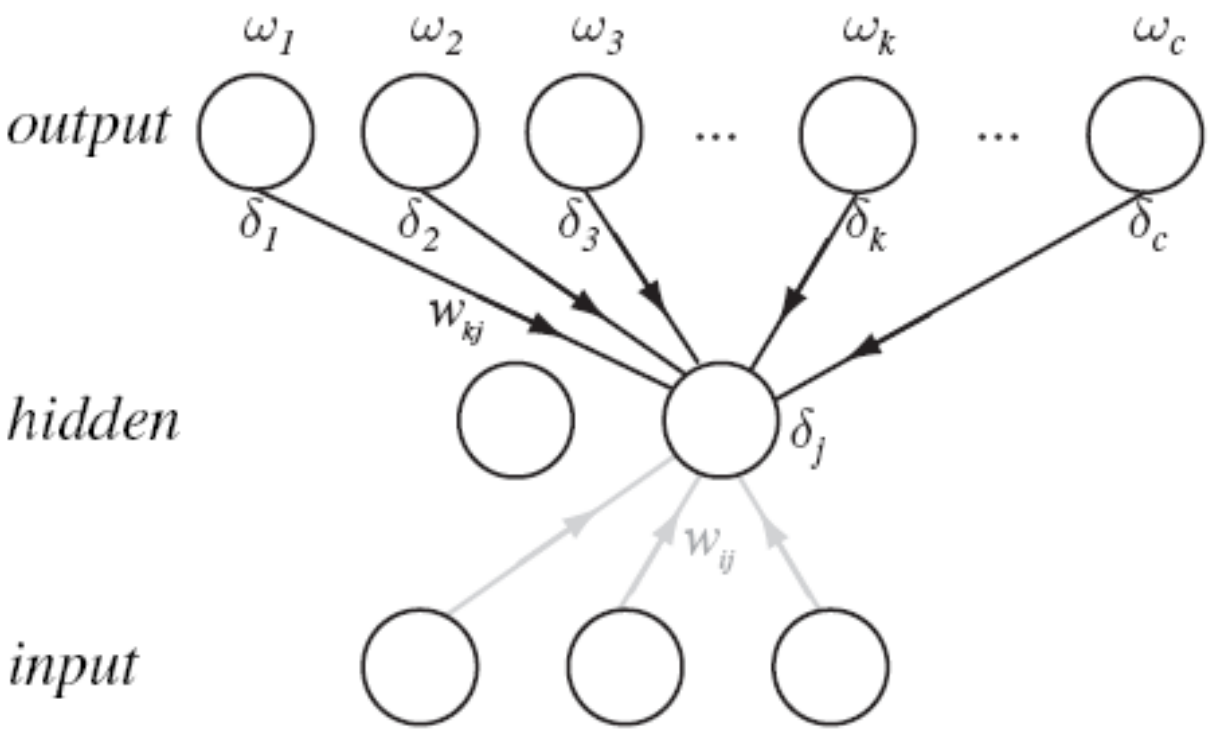

Figure 3: The sensitivities at the output units are propagated back to the hidden units.

- As with all gradient-descent procedures, the behavior of the backpropagation depends on the starting point.
- While it might seem natural to start by zero weights, the Eq. (17) reveals that that has undesirable consequences.
- If w_{kj} were all zero, the input-to-hidden weights would never change.
- This is the reason we start with random values for weights, as discussed in the next section.

- The learning rules make intuitive sense.
- Consider the rule for learning weights at the output units.
- The weight update at unit k should be proportional to $(t_k - z_k)$.
- For the typical sigmoidal $f(\cdot)$ we shall use most often, its derivative is always positive.
- Thus if y_j and $(t_k - z_k)$ are both positive, then the output z_k is too small and the weight w_{kj} must be increased.

6.3.2 Training Protocols

- A supervised training consists of two steps:
  1. Present the patterns with known categories, **the training set**, to a network.
  2. Adjust the weights to make output more like the target values.
- One **epoch** corresponds to presentations of all patterns in the training set.

- The three most popular training protocols are: stochastic, batch, and on-line.
- **Stochastic training**
  - Patterns are chosen randomly from a training set and weights are updated for each pattern presentation.
  - This method is called stochastic because a training pattern is randomly chosen.

- **Batch training**
  - All patterns are presented to the network and then weights are updated once.
  - In virtually every case, we must make several passes through the training set.
- **On-line training**
  - Each pattern is presented once and only once.
  - There is no use of memory for storing the patterns.
  - The notion of epoch does not apply to on-line training.

Algorithm 1. (Stochastic Backpropagation)

1 **initialize** n_H, $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$

2 **do** $m \leftarrow m + 1$

3 $\quad \mathbf{x}^m \leftarrow$ randomly chosen pattern

4 $\quad w_{ji} \leftarrow w_{ji} + \eta\delta_j x_i; \quad w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j$

5 **until** $\|\nabla J(\mathbf{w})\| < \theta$

6 **return** $\mathbf{w}$

Algorithm 2. (Batch Backpropagation)

01 **initialize** n_H, $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$

02 **do** $r \leftarrow r + 1$ (increment epoch)

03 $m \leftarrow 0; \quad \Delta w_{ji} \leftarrow 0; \quad \Delta w_{kj} \leftarrow 0$

04 **do** $m \leftarrow m + 1$

05 $\mathbf{x}^m \leftarrow$ select pattern

06 $\Delta w_{ji} \leftarrow \Delta w_{ji} + \eta \delta_j x_i; \Delta w_{kj} \leftarrow \Delta w_{kj} + \eta \delta_k y_j$

07 **until** $m = n$

08 $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}; \quad w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$

09 **until** $\|\nabla J(\mathbf{w})\| < \theta$

10 **return** $\mathbf{w}$

- So far, we have considered the error on a single pattern.
- But, in fact, we want to consider an error defined over the entire training set.
- We can write this total training error as the sum over the errors on n individual patterns:

$$J = \sum_{p=1}^{n} J_p. \tag{18}$$

- In stochastic training, a weight update may reduce the error on the single pattern being presented, yet increase the error on the full training set.
- Given a large number of such individual updates, however, the total error as given in Eq. (18) decreases.
- In batch backpropagation, we need NOT select patterns randomly, because the weights are updated only after all patterns have been presented once.
- The merits and drawbacks of each algorithm are discussed in Section 6.8.

6.3.3 Learning Curves

- Before training has begun, the error on the training set is typically high.
- The error becomes lower through gradient descent learning on the training set.
- In additional to training set, there are two other sets.
  - **Test set** is used for performance evaluation.
  - **Validation set** is used to decide when to stop training.

- Figure 4 shows the average errors on training, validation, and test sets.
- The validation error and the test error are virtually always higher than the training error.
- The validation set can be used in a stopping criterion in either stochastic training or batch training.
- In some protocols, training is stopped at the first minimum of the validation error (e.g., near epoch 5 in Figure 4).

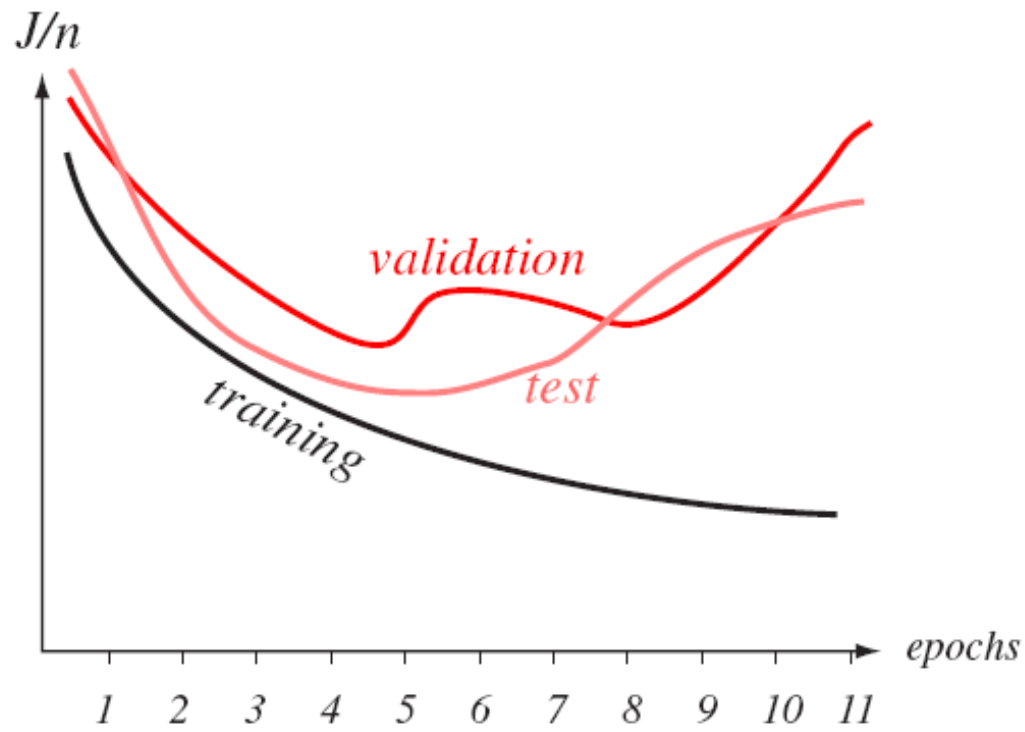

Figure 4: A learning curve shows the criterion function as a function of epoch.

Outline

6.1 Introduction

6.2 Feedforward Operation and Classification

6.3 Back Propagation Algorithm

6.4 **Error Surfaces**

6.8 Practical Techniques for Improving Backpropagation

6.4 Error Surfaces

- Because back propagation is based on gradient descent, we can gain some intuition by studying error surfaces.
- There are some general properties of error surfaces that seem to hold over a broad range of real-world problems.

6.4.1 Some Small Networks

- Consider the simple three-layer network shown in Fig. 5.
- The data shown are linearly separable, and the optimal decision boundary x^* separates the two categories.
- Here, the error surface has a global minimum.
- During learning, the weights descends to the global minimum and the problem is solved.

- In the error surface, a plateau region corresponds to a set of weights that have the same number of misclassified patterns.
- In other words, different plateaus correspond to different numbers of misclassified patterns.
- The maximum number of such misclassified patterns is 8 in this example.

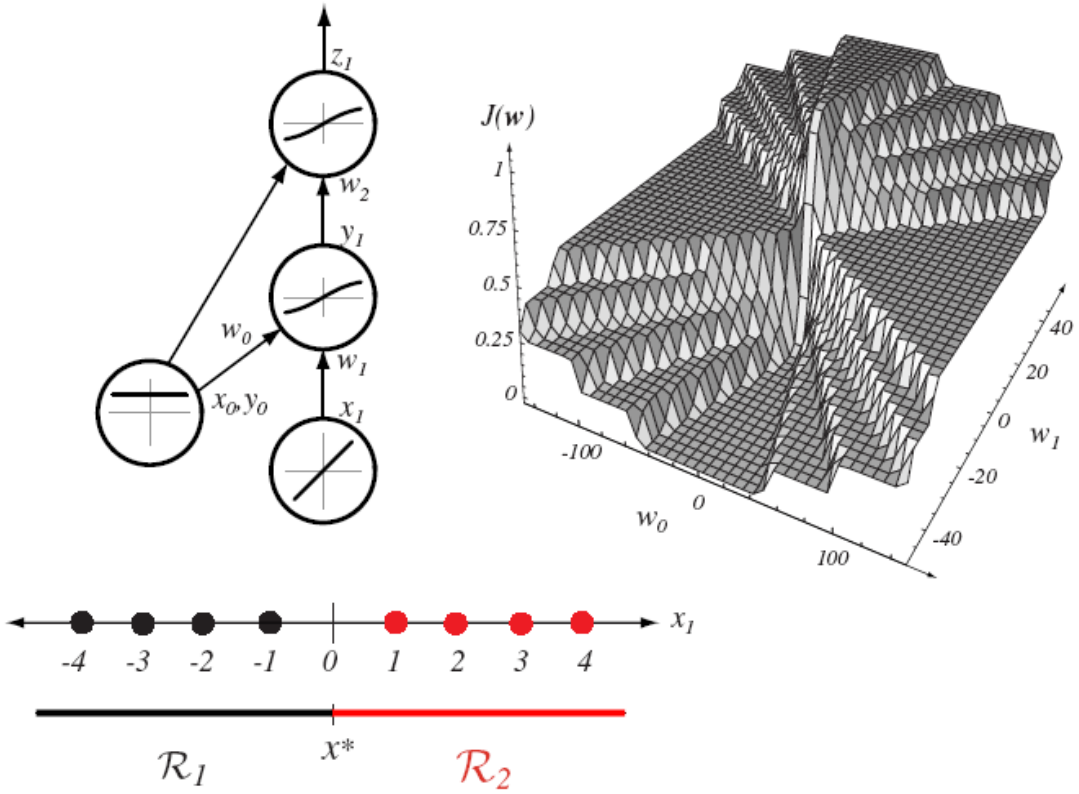

Figure 5: The error surface as a function of w_0 and w_1 is shown.

- Consider the same network applied to another one-dimensional problem which is not linearly separable (Fig. 6).
- The error surface is slightly higher than in Fig. 5 because even the best solution has one pattern been misclassified.
- As before, the different plateaus correspond to different numbers of misclassified patterns.
- A 1-3-1 network can solve this problem but not a 1-2-1 network (Computer exercise 3).

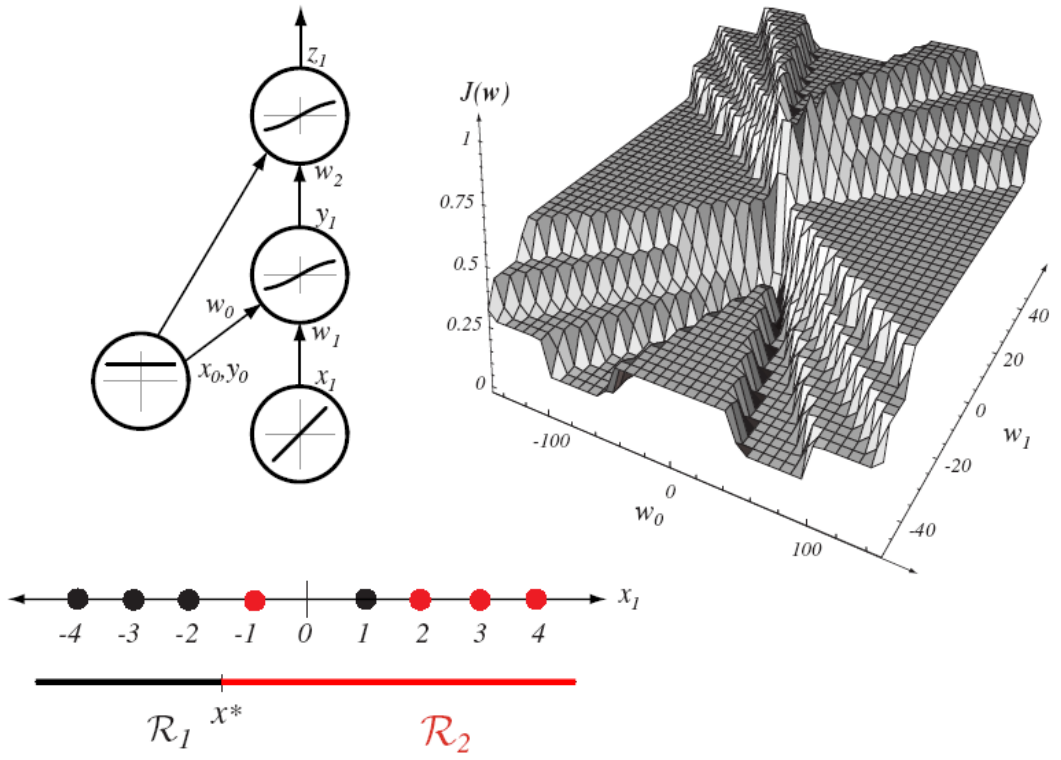

Figure 6: Note that there are two forms of minimum error solution; these correspond to $-2 < x^* < -1$ and $1 < x^* < 2$.

6.4.4 How Important Are Multiple Minima?

- One issue that concerns us is local minima.
  - In such cases, gradient descent is highly unlikely to find the global minimum.
- In practice, convergence to a non-global minimum is often acceptable if the error is fairly low.
- In short, multiple minima do not necessarily present difficulties and some heuristics can help (Section 6.8).

Outline

6.1 Introduction

6.2 Feedforward Operation and Classification

6.3 Back Propagation Algorithm

6.4 Error Surfaces

6.8 **Practical Techniques for Improving Backpropagation**

6.8 Practical Techniques for Improving Backpropagation

- We have ignored a number of practical considerations for the sake of simplicity.
- Although the above analysis is correct, a naive application of the procedures can lead to slow convergence, poor performance or other unsatisfactory results.
- We now turn to a number of practical suggestions for training neural networks.

6.8.1 Activation Function

- There are a number of properties we seek for an activation function.
  1. It is a nonlinear function.
  2. It has some maximum and minimum output values. This will keep activations bounded.
  3. It is a continuous and smooth function, i.e., $f(\cdot)$ and $f'(\cdot)$ be defined throughout their range of the argument.

6.8.2 Parameters for the Sigmoid

- A sigmoid is nonlinear, saturating, and differentiable.
- Given that we use a sigmoidal function, there remain a number of parameters to set.
- It is best to keep an activation function centered on zero and antisymmetric, that is, $f(net) = -f(-net)$.
- Together with the data preprocessing described in Section 6.8.3, antisymmetric sigmoids lead to faster learning.

- Sigmoidal functions are of the form

$$f(net) = a \tanh (b \cdot x) = a \Big[\frac{e^{bx} - e^{-bx}}{e^{bx} + e^{-bx}} \Big]. \qquad (19)$$

- One can choose $a = 1.716$ and $b = 2/3$ so that the linear range is $-1 < net < 1$.

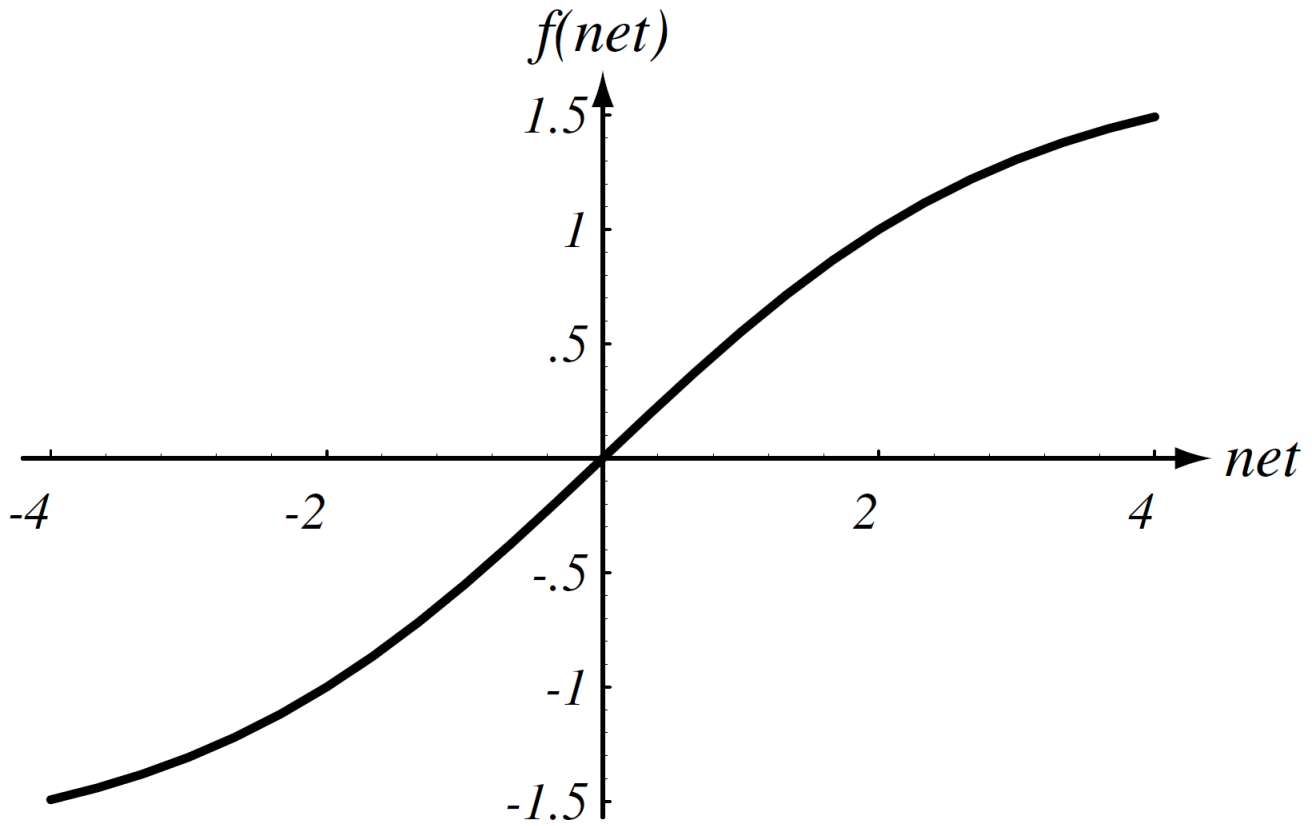

Figure 7: An anti-symmetric sigmoid function.

6.8.3 Scaling Input

- Suppose we design a network to classify fish based on
  - mass (measured in grams)
  - length (measured in meters)
- Such a representation will have serious drawbacks.
  - The numerical value of the mass will be much larger than that for length.
  - The output will largely depend on the mass values.

- Naturally, we do not want our classifier to prefer one of the input features over the others.
- In order to avoid such difficulties, the input patterns should be shifted so that the average of each feature is zero.
- Moreover, the full data set should then be scaled to have the same variance in each feature component.
- This **data standardization** needs to be done once before the start of network training.

6.8.4 Target Values

- We can use teaching values of $+1$ for the target category and -1 for the non-target categories.
- For instance, in a four-category problem if the pattern is in category ω_3, the following target vector should be used:

$$\mathbf{t} = (-1, -1, +1, -1)^t$$

- This target representation yields efficient learning for categorization.

6.8.6 Manufacturing Data (Data Augmentation)

- If we have knowledge about the sources of variation, we can manufacture more training patterns accordingly.
- For instance, in an optical character recognition problem, an input image may be rotated by various amounts.
- Hence, we can take a training pattern and rotate it.
- Likewise, we might scale a pattern, perform simple image processing, and so on.

Figure 8: An image is flipped horizontally and vertically.

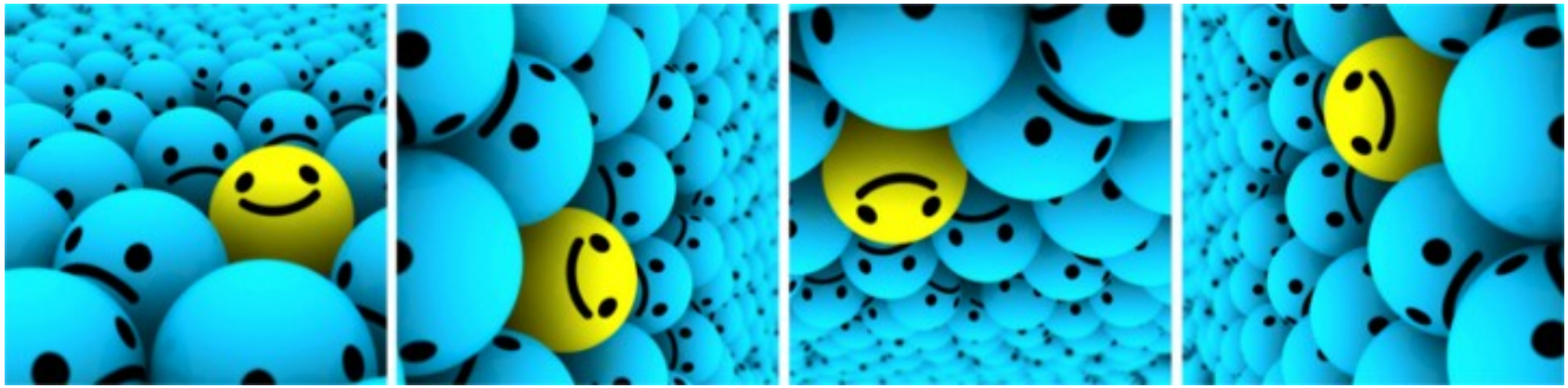

Figure 9: An image is rotated by 90 degrees clockwise.

Figure 10: An image is scaled outward by 10% and 20%.

Figure 11: Crop a portion from the original image and then resize this it to the original image size.

Figure 12: An image is translated to the right and upwards.

Figure 13: An image is added with Gaussian noise, salt and pepper noise.

6.8.8 Initialize Weights

- We now seek to set the initial weight values in order to have **uniform learning**,
  - i.e., weights reach final values at about the same time
- For weights in a given layer, we choose weights randomly from a single distribution to help insure uniform learning.
- Because data standardization gives positive and negative values equally, we want positive and negative weights too.

- Thus, we choose weights from a uniform distribution $-\tilde{w} < w < \tilde{w}$, for some $\tilde{w}$ to be determined.
- In order to calculate $\tilde{w}$, we consider a hidden unit accepting input from d input units.
- On average, the net activation from d inputs through such weights will be $\tilde{w}\sqrt{d}$.
- We would like this net activation to be roughly in the range $-1 < net < +1$.

- This implies that $\tilde{w} = \dfrac{1}{\sqrt{d}}$.
- Thus, input weights should be chosen in the range $-\dfrac{1}{\sqrt{d}} < w_{ji} < \dfrac{1}{\sqrt{d}}$
- The same argument holds for the hidden-to-output weights, where the number of connected units is n_H.
- Thus, hidden-to-output weights should be initialized with values chosen in the range $-\dfrac{1}{\sqrt{n_H}} < w_{kj} < \dfrac{1}{\sqrt{n_H}}$.

6.8.10 Momentum

- Error surfaces often have plateaus
  - regions in which the slope is very small.
- Momentum allows the network to learn more quickly when plateaus exist.
  - Momentum is based on the notion from physics.
  - i.e., moving objects tend to keep moving unless acted upon by outside forces.

- The approach is to alter the learning rule to include some fraction α of the previous weight update.
- Let $\Delta \mathbf{w} = \mathbf{w}(m) - \mathbf{w}(m-1)$.
- Let $\Delta \mathbf{w}_{bp}(m)$ be the change in $\mathbf{w}$ that would be called for by the backpropagation algorithm.
- Then, the learning rule with momenttum is given by

$$\mathbf{w}(m+1) = \mathbf{w}(m) + (1-\alpha)\Delta \mathbf{w}_{bp}(m) + \alpha \Delta \mathbf{w}(m)$$

- If $\alpha = 0$, it is the same as the standard backpropagation.
- If $\alpha = 1$, the change suggested by gradient is ignored.
- The weight changes are sluggish if α is large.
  - Values typically used are $\alpha \simeq 0.9$.
- Thus, the use of momentum can
  - reduce the variation in overall gradient directions
  - and speed up the learning process, especially over plateaus in the error surface.

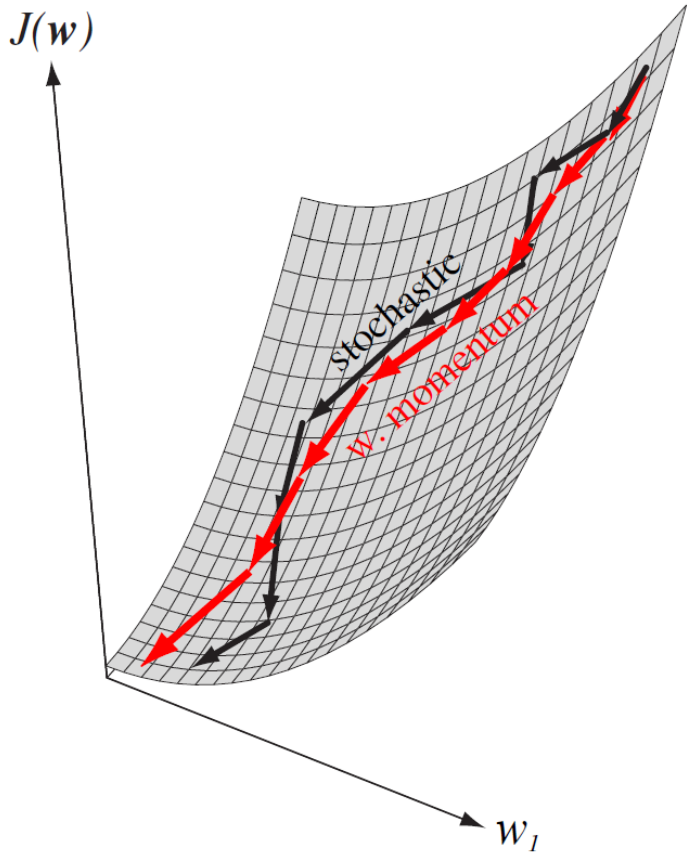

Figure 14: Using momentum averages out stochastic variations.

Algorithm 3. (Backpropagation with Momentum)

1 **initialize** $n_H, \mathbf{w}, \alpha, \theta, \eta, m \leftarrow 0, b_{ji} \leftarrow 0, b_{kj} \leftarrow 0$

2 **do** $m \leftarrow m + 1$

3 $\mathbf{x}^m \leftarrow$ randomly chosen pattern

4 $b_{ji} \leftarrow \eta(1 - \alpha)\delta_j x_i + \alpha b_{ji}$

5 $b_{kj} \leftarrow \eta(1 - \alpha)\delta_k y_j + \alpha b_{kj}$

6 $w_{ji} \leftarrow w_{ji} + b_{ji}; \quad w_{kj} \leftarrow w_{kj} + b_{kj}$

7 **until** $\|\nabla J(\mathbf{w})\| < \theta$

8 **return** w

6.8.11 Weight Decay

- One reason that weight decay is so popular is due to its simplicity.
  - After weight update, every weight is decayed according to $w^{new} = w^{old}(1 - \epsilon)$, where $0 < \epsilon < 1$.
- The weight decay is found in most cases that it helps.
- Indeed, there are occasional cases where it leads to degraded performance.

- In this way, weights that are not needed for reducing the error become smaller and smaller.
- On the other hand, those weights that are needed to solve the problem will not decay indefinitely.
- It can be shown (Problem 42) that the weight decay is equivalent to a new criterion function:

$$J(\mathbf{w}) + \frac{2\epsilon}{\eta}\mathbf{w}^t\mathbf{w}$$

- The second term is called a regularization term.

6.8.13 Stochastic or Batch Training?

- Stochastic
- Batch
- Mini Batch

Stochastic Gradient Descent

- The Stochastic Gradient Descent (SGD) calculates the error for each training sample and adjusts the weights immediately.
- If we have 100 training samples, the SGD adjusts the weights 100 times.
- Figure 15 shows how the weight update of the SGD is related to the entire training samples.

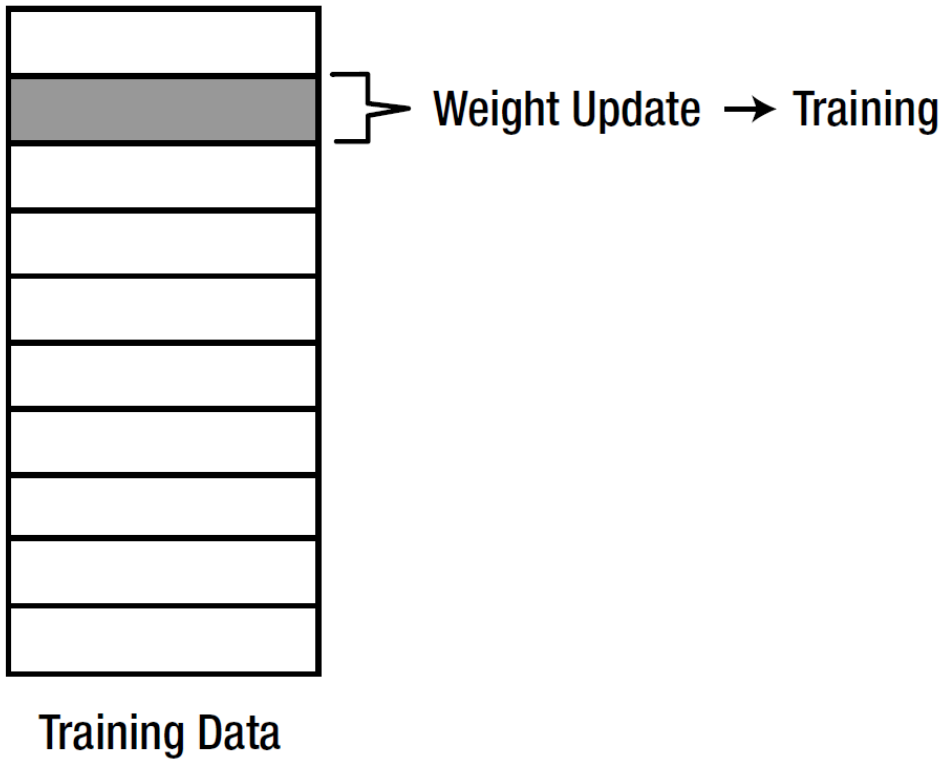

Figure 15: How the weight update of the SGD is related to the entire training data

Batch Method

- In the batch method, each weight update is calculated for all errors of the training data, and the average of the weight updates is used for adjusting the weights.
- This method uses all of the training data and updates only once.
- Figure 16 explains the weight update calculation of the batch method.

- The batch method calculates the weight update as:

$$\Delta w_{ij} = \frac{1}{N} \sum_{k=1}^{N} \Delta w_{ij}(k) \tag{20}$$

  - $\Delta w_{ij}(k)$ is the weight update for the k-th training sample.
  - N is the total number of the training samples.

- Because of the averaged weight update calculation, the batch method consumes a significant amount of time.

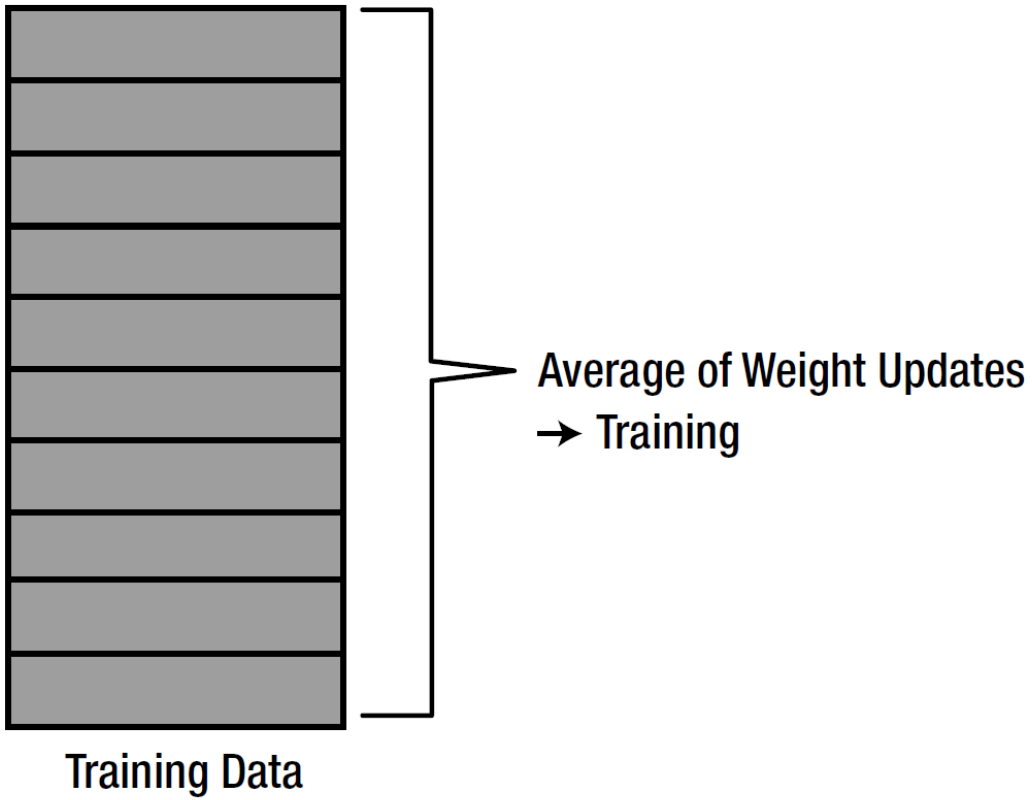

Figure 16: The weight update calculation of the batch method

Mini Batch

- The mini batch method calculates the weight updates from a part of the training dataset.
  - For example, one can select 20 out of 100 training samples, the batch method is applied to the 20 samples.
  - In this case, a total of five weight adjustments are performed to complete the training process for all the training samples.

- Figure 17 shows how the mini batch scheme selects training samples and calculates the weight update.
- The mini batch method obtains the benefits from both methods.
  - Speed from the SGD method
  - Stability from the batch method
- For this reason, it is often utilized in deep learning, which manipulates a significant amount of data.

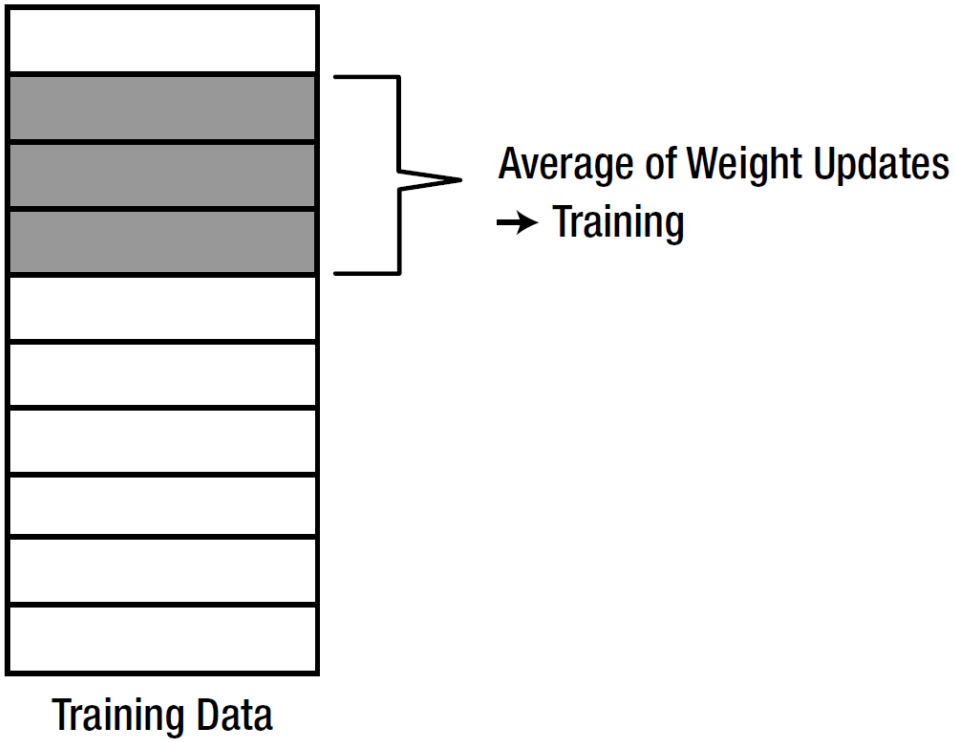

Figure 17: How the mini batch scheme selects training samples and calculates the weight update

6.8.15 Number of Hidden Layers

- We first quote the following statements from the textbook.

  "Since, as we have seen, three layers suffice to implement any arbitrary function, we would need special problem conditions or requirements to recommend the use of more than three layers."

  "In the absence of a problem-specific reason for multiple hidden layers, then, it is simplest to proceed using just a single hidden layer."

Deep Learning

- However, things have changed a lot over the years.
  - It is the time of deep learning!
- Briefly speaking, deep learning is a machine learning technique that employs the deep neural network.
- The deep neural network is a multi-layer neural network that contains two or more hidden layers.

Deep Learning

- Knowing that deep learning is just the use of a deeper neural network, you may want to ask
  - What makes deep learning so attractive?
  - Has anyone ever thought of using more hidden layers?
- In order to answer these questions, we need to look into the history of the neural network.

Deep Learning

- It did not take very long for the single-layer neural network to reveal its fundamental limitations.
- At that time, the researchers knew that the multi-layer neural network would be the next breakthrough.
- However, it took approximately 30 years until another layer was added to the single-layer neural network.

Deep Learning

- The problem of training a multi-layer neural network was solved in 1986 when the backpropagation was introduced.
- However, its performance on practical problems did not meet expectations.
- There were various attempts to overcome the limitations.
- Finally, the neural network was sentenced to having no possibility of improvement and it was forgotten.

Deep Learning

- It remained forgotten for about 20 years until deep learning was introduced.
- It took a while for the deep hidden layer to yield sufficient performance because of the difficulties in training the deep neural network.
- The current technologies in deep learning yield dazzling levels of performance, which outsmarts the other machine learning techniques.

Deep Learning

- In summary, it took 30 years to solve the problems of the single-layer neural network.
  - the back-propagation algorithm
- However, the back-propagation training with the additional hidden layers often resulted in poorer performance.
- Thus, another 20 years has passed until deep learning provides a solution to this problem.

6.8.16 Criterion Function

- cross entropy and its figure

- There are two primary types of criterion functions.

$$J = \sum_{i=1}^{M} \frac{1}{2}(t_i - z_i)^2 \quad (21)$$

$$J = \sum_{i=1}^{M} \{-t_i \ln(z_i) - (1 - t_i) \ln(1 - z_i)\} \quad (22)$$

where z_i is the output from the output node, t_i is the correct output from the training data, and M is the number of output nodes.

- Consider the sum of squared error shown in Equation (21).
- This function measures the square of the difference between the neural network's output z and the correct output t.
- A greater difference between the two values leads to a larger error.
- It is clearly noticeable that the cost function value is proportional to the error.

- The relationship is so intuitive that no further explanation is necessary.
- Most early studies of the neural network employed Equation (21) to derive learning rules.
- Regression problems still use this cost function.

- The following formula, which is inside the curly braces of Equation (22), is called the cross entropy function.

$$E = -t \ln(z) - (1-t) \ln(1-z)$$

- The cross entropy function is the concatenation of the following two equations:

$$E = \begin{cases} -\ln(z) & t = 1 \\ -\ln(1-z) & t = 0 \end{cases}$$

- Due to the definition of a logarithm, the output z should be within 0 and 1.
- Therefore, the cross entropy cost function often teams up with sigmoid and softmax activation functions.
- Now we will see how this function is related to the error.
- Recall that cost functions should be proportional to the output error.

- Figure 18 shows the cross entropy function when $d = 1$.
- When the output $y = 1$, *i.e.*, the error $(d - y)$ is 0, the cost function value is 0 as well.
- In contrast, when the output y approaches 0, *i.e.*, the error grows, the cost function value soars.
- Therefore, this cost function is proportional to the error.

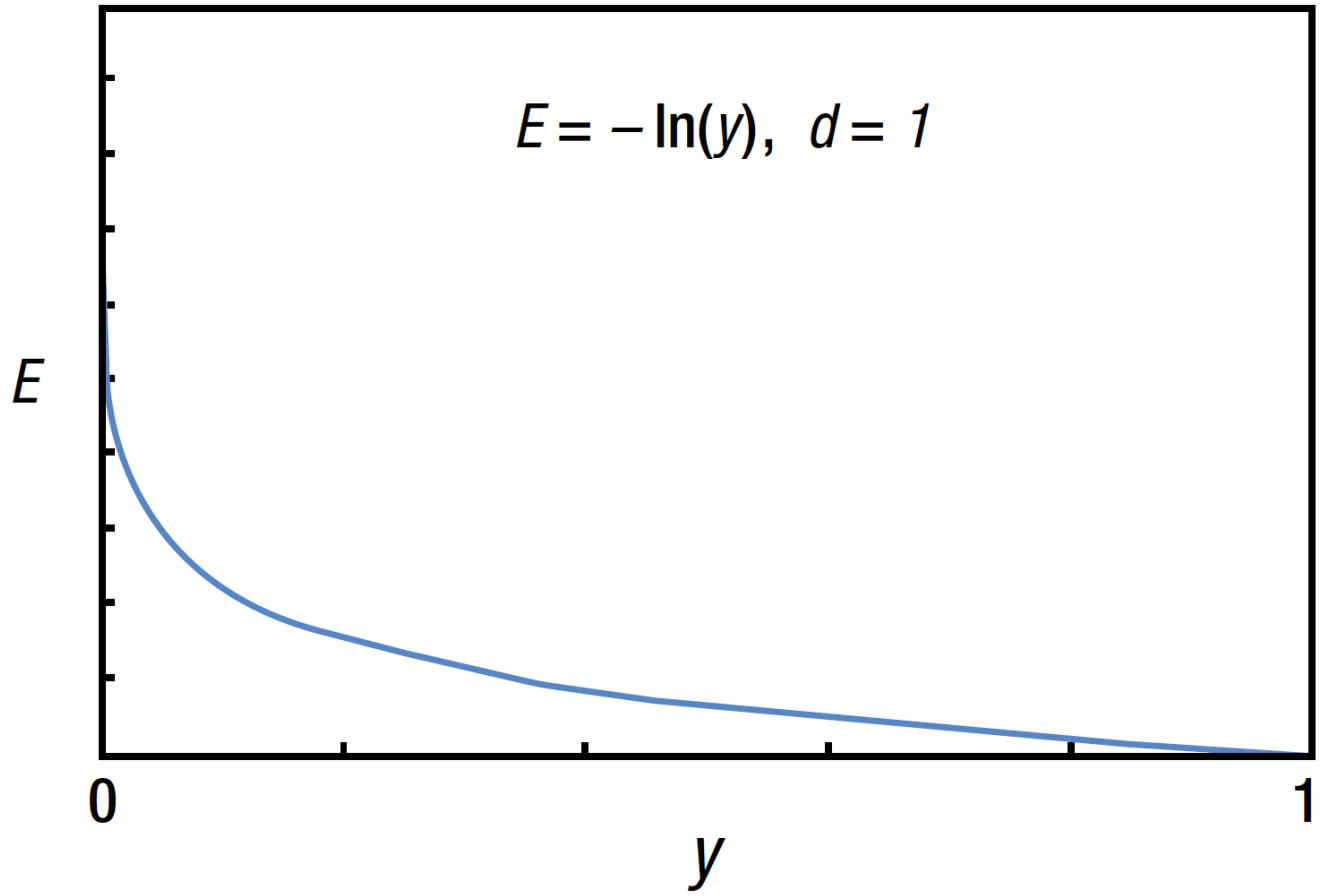

Figure 18: The cross entropy function when $d = 1$

Cost Function and Learning Rule

- Figure 19 shows the cost function when $d = 0$.
- If the output $y = 0$, the error is 0, the cost function yields 0.
- When the output approaches 1, i.e., the error grows, the function value soars.
- Therefore, this cost function in this case is proportional to the error as well.
- These cases confirm that Equation (22) is proportional to the training error.

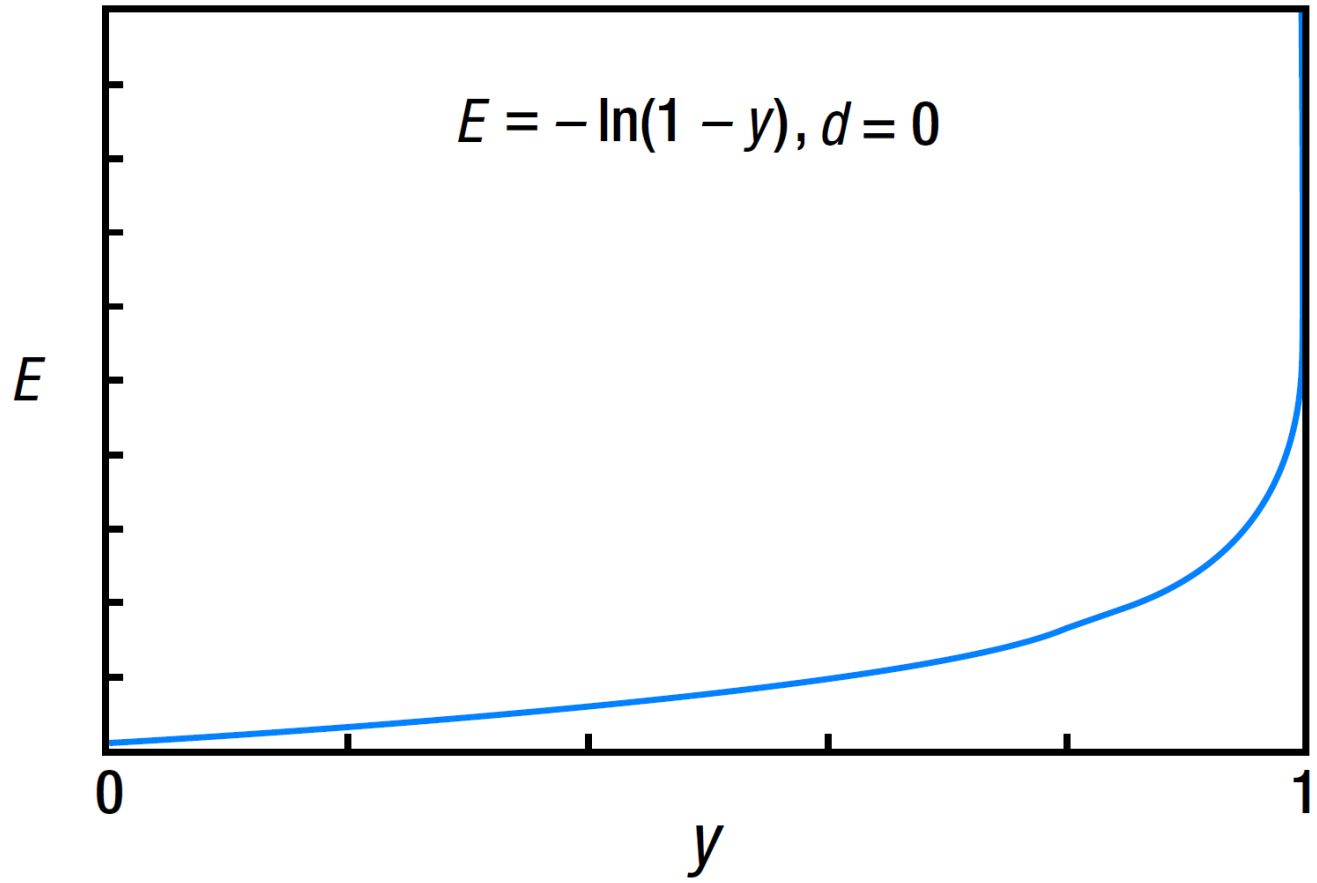

Figure 19: The cross entropy function when $d = 0$