

Single Layer Networks

Terminology

Learn a system which maps an input vector x to an output vector y

- **Runtime:** Compute the output y for each input x
- **Training:** Optimise the parameters of the network such that the correct y is computed for each x
- **Generalisation:** We are most interested in the output accuracy of the system for unseen test data
- **Single Layer Network:** Use a single layer of computation (linear / affine transformation) to map between input and output

Single Layer Networks

Input vector $x = (x_1, \dots, x_d)^T$

Output vector $y = (y_1, \dots, y_K)^T$

Weight matrix W : w_{ki} is the weight from input x_i to output y_k

Bias vector b : b_k is the bias for output k

$$y_k = \sum_{i=1}^d w_{ki}x_i + b_k \quad ; \quad \mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Single Layer Networks

Input vector $x = (x_1, \dots, x_d)^T$

Output vector $y = (y_1, \dots, y_K)^T$

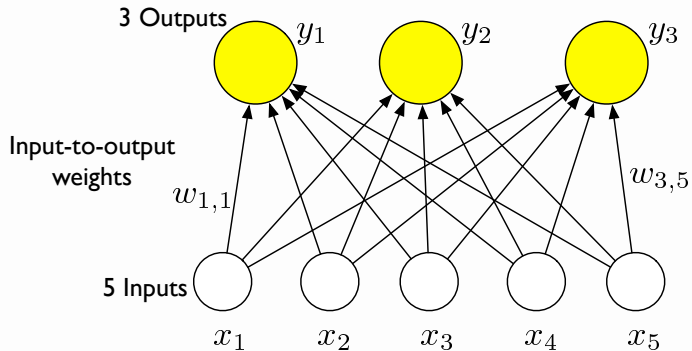
Weight matrix W : w_{ki} is the weight from input x_i to output y_k

Bias vector b : b_k is the bias for output k

$$y_k = \sum_{i=1}^d w_{ki}x_i + b_k \quad ; \quad \mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Also known as Linear Regression

Graph Visualization



$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$y_k = \sum_{i=1}^d W_{ki}x_i + b_k$$

Training Single Layer Networks

Training set N input/output pairs $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \leq n \leq N\}$

Target vector $\mathbf{t}^n = (t_1^n, \dots, t_K^n)^T$ – the target output for input $\mathbf{x}^n$

Output vector $\mathbf{y}^n = \mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$ – the output computed by the network for input $\mathbf{x}^n$

Training Single Layer Networks

Training set N input/output pairs $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \leq n \leq N\}$

Target vector $\mathbf{t}^n = (t_1^n, \dots, t_K^n)^T$ – the target output for input $\mathbf{x}^n$

Output vector $\mathbf{y}^n = \mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$ – the output computed by the network for input $\mathbf{x}^n$

Trainable parameters weight matrix $\mathbf{W}$, bias vector $\mathbf{b}$

Training Single Layer Networks

Training set N input/output pairs $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \leq n \leq N\}$

Target vector $\mathbf{t}^n = (t_1^n, \dots, t_K^n)^T$ – the target output for input $\mathbf{x}^n$

Output vector $\mathbf{y}^n = \mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$ – the output computed by the network for input $\mathbf{x}^n$

Trainable parameters weight matrix $\mathbf{W}$, bias vector $\mathbf{b}$

Supervised learning There is a target output for each input

Training problem Set the values of the weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ such that each input $\mathbf{x}^n$ is mapped to its target $\mathbf{t}^n$

Error function Define the training problem in terms of an error function E ; training corresponds to setting the weights so as to minimise the error

Error function

- Error function should measure how far an output vector is from its target – e.g. (squared) Euclidean distance – *mean square error*.

E^n is the error per example:

$$E^n = \frac{1}{2} \|\mathbf{y}^n - \mathbf{t}^n\|^2 = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2$$

E is the total error averaged over the training set:

$$E = \frac{1}{N} \sum_{n=1}^N E^n = \frac{1}{N} \sum_{n=1}^N \left(\frac{1}{2} \|\mathbf{y}^n - \mathbf{t}^n\|^2 \right)$$

Error function

- Error function should measure how far an output vector is from its target – e.g. (squared) Euclidean distance – *mean square error*.

E^n is the error per example:

$$E^n = \frac{1}{2} \|\mathbf{y}^n - \mathbf{t}^n\|^2 = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2$$

E is the total error averaged over the training set:

$$E = \frac{1}{N} \sum_{n=1}^N E^n = \frac{1}{N} \sum_{n=1}^N \left(\frac{1}{2} \|\mathbf{y}^n - \mathbf{t}^n\|^2 \right)$$

- Training process: set $\mathbf{W}$ and $\mathbf{b}$ to minimise E given the training set

$$\mathbf{W}^*, \mathbf{b}^* = \arg \min_{\mathbf{W}, \mathbf{b}} \underbrace{\frac{1}{N} \sum_{n=1}^N \left(\frac{1}{2} \|\mathbf{y}^n - \mathbf{t}^n\|^2 \right)}_{E(\mathbf{W}, \mathbf{b})}$$

Gradient Descent

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space.
 $E(W)$ is the value of the error at a specific point in weight space (given the training data).

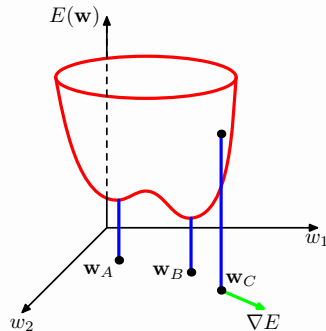


Image credit: 'C. M. Bishop,
H. Bishop, Deep Learning'

Gradient Descent

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space.
 $E(\mathbf{W})$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(\mathbf{W})$ given $\mathbf{W}$ is $\nabla_{\mathbf{W}}E$, the matrix of partial derivatives of E with respect to the elements of $\mathbf{W}$:

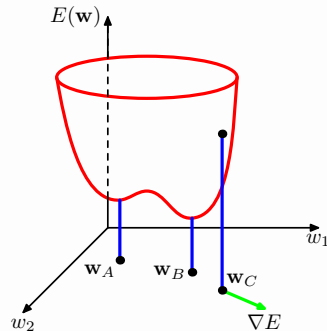


Image credit: 'C. M. Bishop,
H. Bishop, Deep Learning'

Gradient Descent

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space. $E(W)$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(W)$ given W is $\nabla_W E$, the matrix of partial derivatives of E with respect to the elements of W :
- **Gradient Descent Training:** adjust the weight matrix by moving a small direction down the gradient, which is the direction along which E decreases most rapidly.
 - update each weight w_{ki} by adding a factor $-\eta \cdot \partial E / \partial w_{ki}$
 - η is a small constant called the *step size* or *learning rate*.
- Adjust bias vector similarly

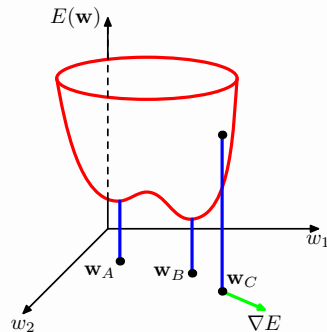


Image credit: 'C. M. Bishop, H. Bishop, Deep Learning'

Gradient Descent Procedure

1. Initialise weights and biases with small random numbers
2. For each epoch (complete pass through the training data)
 - 2.1 Initialise total gradients: $\Delta w_{ki} = 0, \Delta b_k = 0$
 - 2.2 For each training example n :
 - 2.2.1 Compute the error E^n
 - 2.2.2 For all k, i : Compute the gradients $\partial E^n / \partial w_{ki}, \partial E^n / \partial b_k$
 - 2.2.3 Update the total gradients by accumulating the gradients for example n

$$\Delta w_{ki} \leftarrow \Delta w_{ki} + \frac{\partial E^n}{\partial w_{ki}} \quad \forall k, i \qquad \Delta b_k \leftarrow \Delta b_k + \frac{\partial E^n}{\partial b_k} \quad \forall k$$

- 2.3 Update weights:

$$\begin{aligned} \Delta w_{ki} &\leftarrow \Delta w_{ki} / N; & w_{ki} &\leftarrow w_{ki} - \eta \Delta w_{ki} & \forall k, i \\ \Delta b_k &\leftarrow \Delta b_k / N; & b_k &\leftarrow b_k - \eta \Delta b_k & \forall k \end{aligned}$$

3. Terminate: after a number of epochs; or when error stops decreasing (threshold).

Applying gradient descent to a single-layer network

- Error function:

$$E = \frac{1}{N} \sum_{n=1}^N E^n \quad E^n = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2 \quad \text{remember} \quad y_k^n = \sum_{i=1}^d w_{ki} x_i + b_k$$

Applying gradient descent to a single-layer network

- Error function:

$$E = \frac{1}{N} \sum_{n=1}^N E^n \quad E^n = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2 \quad \text{remember} \quad y_k^n = \sum_{i=1}^d w_{ki} x_i + b_k$$

- Gradients:

$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \frac{\partial E^n}{\partial y_r} \cdot \frac{\partial y_r}{\partial w_{rs}} = (y_r^n - t_r^n) \cdot x_s^n$$

$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial E^n}{\partial w_{rs}} = \frac{1}{N} \sum_{n=1}^N (y_r^n - t_r^n) x_s^n$$

Applying gradient descent to a single-layer network

- Error function:

$$E = \frac{1}{N} \sum_{n=1}^N E^n \quad E^n = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2 \quad \text{remember} \quad y_k^n = \sum_{i=1}^d w_{ki} x_i + b_k$$

- Gradients (grads_wrt_params):

$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \underbrace{\frac{\partial E^n}{\partial y_r}}_{\text{error.grad}} \cdot \frac{\partial y_r}{\partial w_{rs}} = \underbrace{(y_r^n - t_r^n)}_{\text{output error}} \cdot \underbrace{x_s^n}_{\text{input}}$$

$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial E^n}{\partial w_{rs}} = \frac{1}{N} \sum_{n=1}^N (y_r^n - t_r^n) \cdot x_s^n$$

Applying gradient descent to a single-layer network

- Error function:

$$E = \frac{1}{N} \sum_{n=1}^N E^n \quad E^n = \frac{1}{2} \sum_{k=1}^K (y_k^n - t_k^n)^2 \quad \text{remember} \quad y_k^n = \sum_{i=1}^d w_{ki} x_i + b_k$$

- Gradients (grads_wrt_params):

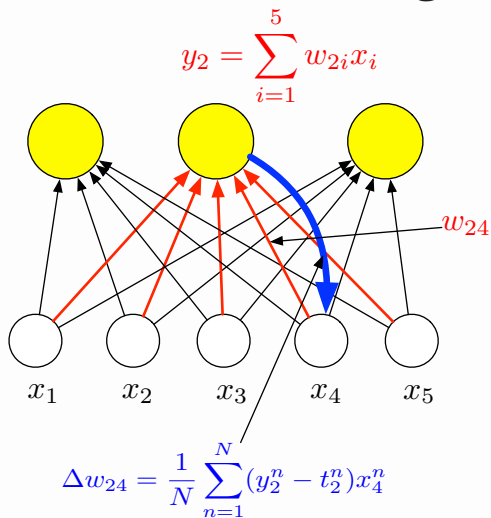
$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \underbrace{\frac{\partial E^n}{\partial y_r}}_{\text{error.grad}} \cdot \frac{\partial y_r}{\partial w_{rs}} = \underbrace{(y_r^n - t_r^n)}_{\text{output error}} \cdot \underbrace{x_s^n}_{\text{input}}$$

$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial E^n}{\partial w_{rs}} = \frac{1}{N} \sum_{n=1}^N (y_r^n - t_r^n) \cdot x_s^n$$

- Weight update

$$w_{rs} \leftarrow w_{rs} - \eta \cdot \frac{1}{N} \sum_{n=1}^N (y_r^n - t_r^n) x_s^n$$

Applying gradient descent to a single-layer network



Summary

Machine Learning Practical Feedback Form

- **Reading** – Goodfellow et al, *Deep Learning* chapter 1; sections 4.3 (p79–83), 5.1, 5.7
- Lab 1: Setup, training data (W2)
- Lab 2: Training single-layer networks (W3)
- **Next lecture:**
 - Stochastic gradient descent and minibatches
 - Classification
 - Sigmoid and softmax



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