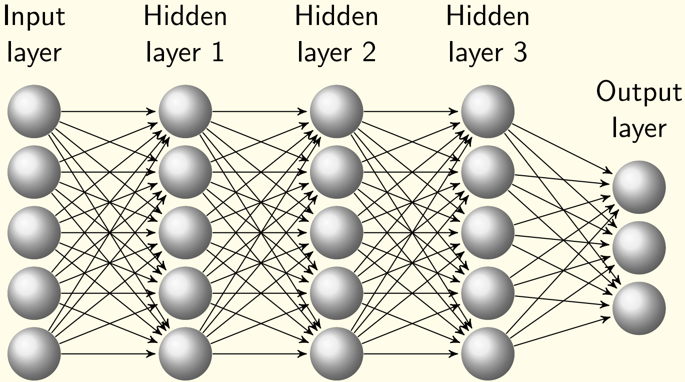
Backprop
How it works (and how it fails)

James V Stone

Sheffield University

Structure

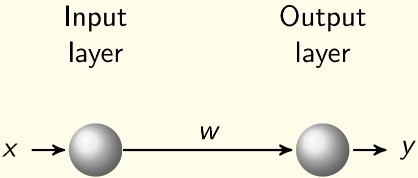
A Deep Neural Network: Credit Assignment



A deep network with three hidden layers.

The most important problem is the *credit assignment problem*. This involves allocating credit (or blame) to each connection weight so that it can be adjusted to increase performance.

The Simplest Neural Network: Definitions



Definitions

x *state* of input unit

w *weight* connecting two units

u total *input* to a unit (e.g. to output unit)

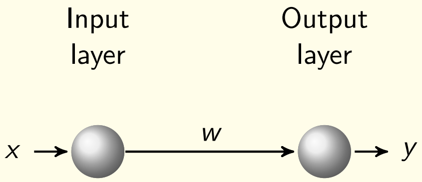
y state of output unit

$y = f(u)$ the *activation function* of a unit is f

τ desired or *target value* of output unit

an *association* is the mapping $x \rightarrow \tau$

The Simplest Neural Network: From input to output



If the input state is x and if the weight of the connection from the input unit to the output unit is w, then the total input u to the output unit is

$$u = wx. \tag{1}$$

In general, the state y of a unit is governed by an *activation function* (i.e. input/output function)

$$y = f(u). \tag{2}$$

The Simplest Neural Network: The Delta Term

Suppose we wish the network to learn to associate an input value of $x = 0.8$ with a target state of $\tau = 0.2$. Usually, we have no idea of the correct value for the weight w, so we may as well begin by choosing its value at random. Suppose we choose a value $w = 0.4$, so the output state is $y = wx = 0.4 \times 0.8 = 0.32$.

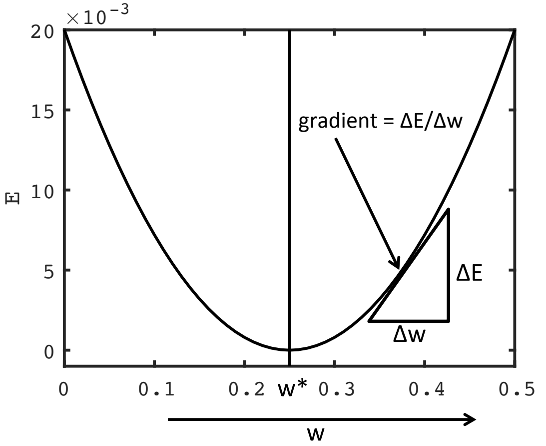
The difference between the output y and the target value τ is defined here as the *delta term*:

$$\delta = y - \tau = 0.32 - 0.2 = 0.12. \tag{3}$$

Ideally, we would like to adjust the weight w so that $\delta = 0$. A standard measure of the error in y is half the squared difference:

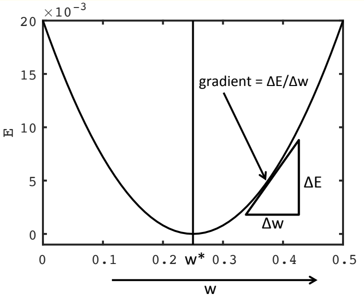
$$\begin{aligned} E &= \tfrac{1}{2}\,(wx - \tau)^2 && (4) \\ &= \tfrac{1}{2}\,(y - \tau)^2 && (5) \\ &= \tfrac{1}{2}\,\delta^2. && (6) \end{aligned}$$

The Simplest Neural Network



The optimal weight is $w^* = 0.25$, because $0.2 = 0.8 \times 0.25$.
The gradient of the error function E at a point w is approximated by $\Delta E / \Delta w$.

The Simplest Neural Network



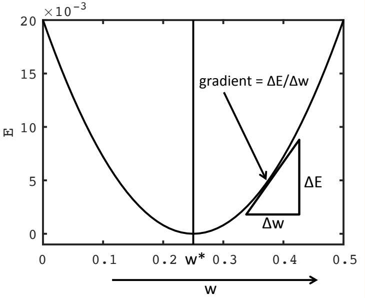
Given that

$$E = \frac{1}{2}(wx - \tau)^2, \tag{7}$$

the gradient of the error function at a point w is approximated by $\Delta E / \Delta w$. An exact measure of the gradient is defined by the derivative of E (Equation 7) with respect to w:

$$\frac{dE}{dw} = (wx - \tau)\, x \approx \frac{\Delta E}{\Delta w}. \tag{8}$$

The Simplest Neural Network

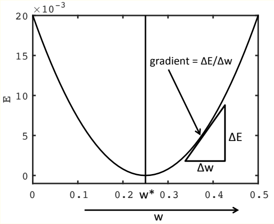


It will prove useful to write Equation 8 in terms of the delta term:

$$\frac{dE}{dw} = \delta \times x. \tag{9}$$

The *magnitude* of the gradient indicates the steepness of the slope at w, and the *sign* of the gradient indicates the direction that increases E.

The Simplest Neural Network



The direction of the gradient measured using calculus points uphill, and is called the *direction of steepest ascent*. This means that in order to reduce E, we should change the value of w by a small amount Δw in the *direction of steepest descent*:

$$\Delta w = -\epsilon \frac{dE}{dw} \tag{10}$$

$$= -\epsilon \, \delta \, x, \tag{11}$$

where the size of the step is defined by a *learning rate parameter* ϵ.

The Simplest Neural Network: Algorithm

Gradient Descent
Learning One Association

initialise network weight w to random value
set input unit states x to training vector x
set learning to true
while *learning* **do**
  get state of output unit $y = wx$
  get delta term $\delta = y - \tau$
  get weight gradient for input vector $dE/dw = \delta x$
  get change in weight $\Delta w = -\epsilon\, dE/dw$
  update weight $w \leftarrow w + \Delta w$
  if *gradient* $dE/dw \approx 0$ **then**
    set learning to false
  end
end

Gradient Descent for a Network with Two Input Units

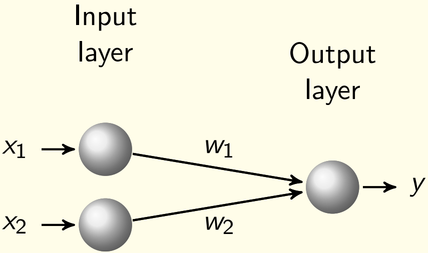


Figure 1: A neural network with two input units and one output unit.

Gradient Descent for a Network with Two Input Units

This network can learn up to two associations. Each association consists of an input, which is a pair of values x_1 and x_2, and each corresponding output is a single value y.

Given one input (x_1, x_2), the output y is found by multiplying each input value by its corresponding weight and then summing the resultant products:

$$y = w_1 x_1 + w_2 x_2. \tag{12}$$

Gradient Descent for a Network with Two Input Units

This can be written succinctly if we represent the weights as a vector, written in bold typeface:

$$\mathbf{w} = (w_1, w_2). \tag{13}$$

Similarly, each pair of input values can be represented as a vector, again in bold typeface:

$$\mathbf{x} = (x_1, x_2). \tag{14}$$

The state y for an input $\mathbf{x}$ is found from the *dot* or *inner* product,

$$y = \mathbf{w} \cdot \mathbf{x}, \tag{15}$$

which is defined by Equation 12.
Notice that scalar variables are in italics, whereas vectors are in bold typeface.

Gradient Descent for a Network with Two Input Units

We use subscripts to denote each association

$$\begin{aligned} y_1 &= \mathbf{w} \cdot \mathbf{x}_1 \\ y_2 &= \mathbf{w} \cdot \mathbf{x}_2. \end{aligned} \tag{16}$$

We will write the problem out in full

$$\begin{aligned} y_1 &= w_1 x_{11} + w_2 x_{21} \\ y_2 &= w_1 x_{12} + w_2 x_{22}. \end{aligned} \tag{17}$$

We can recognise this as two simultaneous equations with two unknowns (w_1 and w_2), so we know that a solution for w_1 and w_2 usually exists.

We could find this solution manually, but because we know that the problems we encounter later will become unrealistic for manual methods, we will stick to using gradient descent.

Gradient Descent for a Network with Two Input Units

To use gradient descent, we first need to write down an error function like Equation 7. The error function for the first association is

$$E_1 \quad = \quad \tfrac{1}{2}\,(\mathbf{w} \cdot \mathbf{x}_1 - \tau_1)^2, \tag{18}$$

and for the second association it is

$$E_2 \quad = \quad \tfrac{1}{2}\,(\mathbf{w} \cdot \mathbf{x}_2 - \tau_2)^2. \tag{19}$$

The error function for the set of two associations is the sum

$$\begin{aligned} E \quad &= \quad E_1 + E_2 && (20) \\ &= \quad \tfrac{1}{2}\left[(\mathbf{w} \cdot \mathbf{x}_1 - \tau_1)^2 + (\mathbf{w} \cdot \mathbf{x}_2 - \tau_2)^2\right], && (21) \end{aligned}$$

which can be written succinctly using the summation convention as

$$E \quad = \quad \frac{1}{2} \sum_{t=1}^{2} (\mathbf{w} \cdot \mathbf{x}_t - \tau_t)^2. \tag{22}$$

Gradient Descent for a Network with Two Input Units

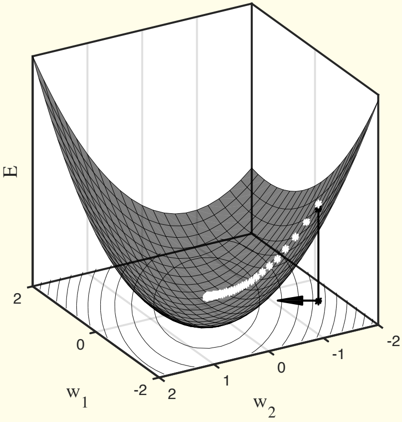


Figure 2: The error surface is obtained by evaluating Equation 22 over a range of values for w_1 and w_2. Given an initial weight vector $\mathbf{w} = (-0.8, -1.6)$, the direction of steepest descent is $-\nabla_{\mathbf{w}} E$ (shown by an arrow on the ground plane). The white dots depict the evolution of weights during learning using Equation 30.

Gradient Descent for a Network with Two Input Units

Using the *chain rule*, the gradient of the error function with respect to w_1 for the tth association ($t = 1$ or 2) is

$$\frac{\partial E_t}{\partial w_1} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial w_1}, \tag{23}$$

where $\partial E_t/\partial y_t = (\mathbf{w}\cdot\mathbf{x}_t - \tau_t)$, $y_t = \mathbf{w}\cdot\mathbf{x}_t$, and $\partial y_t/\partial w_1 = x_{1t}$, so

$$\frac{\partial E_t}{\partial w_1} = (\mathbf{w}\cdot\mathbf{x}_t - \tau_t)\, x_{1t}. \tag{24}$$

Given that the delta term for the tth association is

$$\delta_t = (\mathbf{w}\cdot\mathbf{x}_t - \tau_t), \tag{25}$$

we then have

$$\frac{\partial E_t}{\partial w_1} = \delta_t\, x_{1t}. \tag{26}$$

Gradient Descent for a Network with Two Input Units

When considered over both associations, the gradient of the error function with respect to w_1 is

$$\frac{\partial E}{\partial w_1} = \sum_{t=1}^{2} \delta_t \, x_{1t}. \tag{27}$$

Similarly, the gradient with respect to w_2 is

$$\frac{\partial E}{\partial w_2} = \sum_{t=1}^{2} \delta_t \, x_{2t}. \tag{28}$$

Gradient Descent for a Network with Two Input Units

The direction of steepest ascent is a vector on the ground plane that points in the direction to go in order to increase the value of E as quickly as possible. This direction is represented by the *nabla* symbol (∇), which is a vector of scalar gradients:

$$\nabla_{\mathbf{w}} E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2} \right), \tag{29}$$

where the subscript $\mathbf{w}$ indicates a derivative with respect to $\mathbf{w}$.

If the direction of steepest *ascent* is ∇E then the direction of steepest *descent* is $-\nabla E$, as shown in Figure 2. Accordingly, if the current value of the weight vector is $\mathbf{w}_{\mathrm{old}}$, then

$$\mathbf{w}_{\mathrm{new}} = \mathbf{w}_{\mathrm{old}} - \epsilon \nabla E. \tag{30}$$

Gradient Descent for a Network with Two Input Units

initialise weights $\mathbf{w}$ to random values; set learning to true
while *learning* **do**
  set recorder of weight change vectors $\Delta\mathbf{w}$ to zero
  foreach *association from* $t = 1$ *to* 2 **do**
    set input unit states $\mathbf{x}$ to tth training vector $\mathbf{x}_t$
    get state of output unit $y_t = \mathbf{w} \cdot \mathbf{x}_t$
    get delta term $\delta_t = y_t - \tau_t$
    get weight gradient for tth input vector $\nabla E_t = \delta_t \mathbf{x}_t$
    get change in weights for tth input vector $\Delta\mathbf{w}_t = -\epsilon \nabla E_t$
    accumulate weight changes in $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} + \Delta\mathbf{w}_t$
  end
  update weights $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
  if *gradient* $|\nabla E| \approx 0$ **then**
    set learning to false
  end
end

A Network with Four Input and Output Units

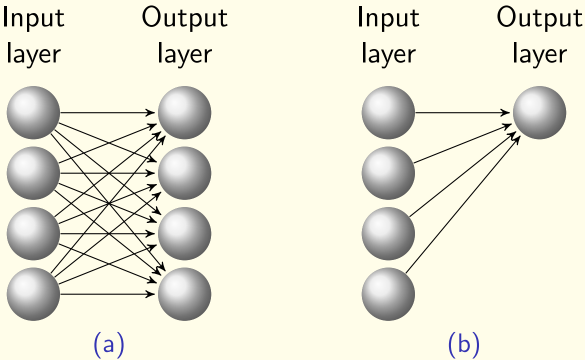


Figure 3: (a) A neural network with four input units and four output units (i.e. a 4-4 network) can be viewed as four neural networks like the one in (b), where each of the four neural networks has the same four input units but a different output unit (i.e. four 4-1 networks).

Learning Photographs

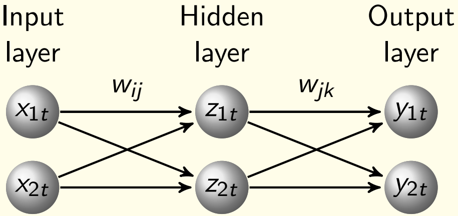
(a) (b) (c) (d) (e) (f)

Learning photographs using a network with the same type of architecture as in Figure 3a. Each photograph $\mathbf{x}_t$ consists of 50×50 pixels. A linear network was used, with an array of 50×50 input units and an array of 50×50 output units. The network was trained to associate each of two training vectors (a and d) with itself.

For example, adding noise to (a) yielded (b), and when (b) was used as input to the network, the output was (c), showing that the network's memory is *content addressable*.

Backprop

A Backprop Neural Network



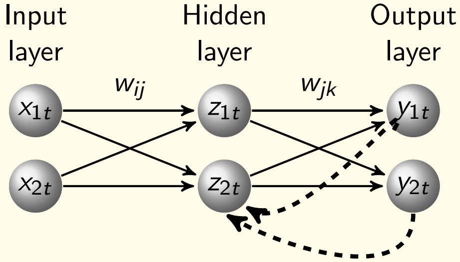
Unit states are indexed by layer ($x =$ input, $z =$ hidden, $y =$ output) and by association number t. We ignore bias terms (thresholds) and bias units.

An *association* is the mapping between input and output vectors $\mathbf{x}_t \rightarrow \mathbf{y}_t$.

A key property of backprop networks is that they can compute nonlinear functions, because most units have nonlinear activation functions.

BUT this also means that the error surface is not convex, so it has *local minima*.

Backward Propagation of Errors: A General Recipe



Note that the delta terms of *all* output units contribute to the delta term of *each* hidden unit.

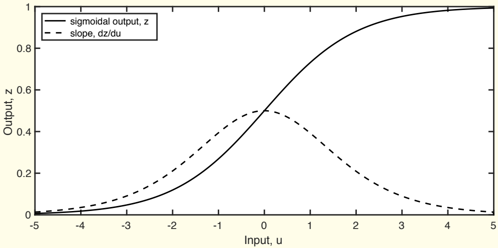
Forward Propagation of Inputs

Each of the $t = 1, \ldots, T$ associations comprises an input vector $\mathbf{x}_{it} = (x_{11}, \ldots, x_{It})$ and a target output τ_{kt}.
For the tth association, the total input to the jth hidden unit is a weighted sum of the I unit states in the input layer:

$$u_{jt} \;\; = \;\; \sum_{i=1}^{I+1} w_{ij}\, x_{it}, \tag{31}$$

where w_{ij} connects the ith input unit to the jth hidden unit.
Note: The upper limit of $I + 1$ (here and below) includes an extra 1 to remind us that we are ignoring inputs from a *bias unit*, which has a state permanently set to one.

Unit Activation Function



Sigmoidal activation function for jth hidden unit and tth association

$$z_{jt} = f(u_{jt}) = (1 + e^{-u_{jt}})^{-1}, \tag{32}$$

where u_{jt} is the total input to a unit.
For later use, the derivative is

$$\frac{dz_{jt}}{du_{jt}} = z_{jt}\,(1 - z_{jt}). \tag{33}$$

Forward Propagation of Inputs

Similarly, the kth output unit has a total input of

$$u_{kt} = \sum_{j=1}^{J+1} w_{jk} \, z_{jt}, \tag{34}$$

where w_{jk} is the weight connecting the jth hidden unit to the kth output unit; the state of the kth output unit is therefore

$$\begin{aligned} y_{kt} &= f_k(u_{kt}) && (35) \\ &= u_{kt}. && (36) \end{aligned}$$

Backward Propagation of Errors: A New Delta Term

For backprop, we need a more general definition of the delta term.
If u_{kt} is the input to the kth output unit for the tth association then the delta term is

$$\delta_{kt} = \frac{\partial E_t}{\partial u_{kt}}. \tag{37}$$

Similarly, if u_{jt} is the input to the jth hidden unit for the tth association then the delta term is

$$\delta_{jt} = \frac{\partial E_t}{\partial u_{jt}}. \tag{38}$$

Notice that this new definition yields the same result as in previous slides for linear output units.

Backward Propagation of Errors: A General Recipe

Given T associations, for now, we consider only the tth association.
And we consider only one weight in each layer w_{ij} and w_{jk}.
Define the learning rate as ϵ (epsilon).
The change in the jth weight of the kth output unit for the tth association is

$$\Delta w_{jkt} = -\epsilon \frac{\partial E_t}{\partial w_{jk}}, \tag{39}$$

where (using the chain rule, and Equation 34)

$$\frac{\partial E_t}{\partial w_{jk}} = \frac{\partial E_t}{\partial u_{kt}} \frac{\partial u_{kt}}{\partial w_{jk}} \tag{40}$$

$$= \delta_{kt} \, z_{jt}. \tag{41}$$

Therefore,

$$\Delta w_{jkt} = -\epsilon \, \delta_{kt} \, z_{jt}. \tag{42}$$

Backward Propagation: Apply Recipe to Hidden Unit

Changing the ith weight of the jth hidden unit for the tth association

$$\Delta w_{ijt} = -\epsilon \frac{\partial E_t}{\partial w_{ij}} \quad (43)$$

$$= -\epsilon \, \delta_{jt} \, x_{it}. \quad (44)$$

This is a general recipe for updating weights in a backprop network.

Once we have the delta term for each unit then we can adjust all the weights between that unit and all the units in the previous layer.

Backward Propagation of Errors: Finding Delta Terms

So, here is the plan. For the tth association:

1. Find the delta term of each output unit, $\delta_{kt} : k = 1, \ldots, K$
2. Use δ_{kt}'s to find the delta term of each hidden unit: $\delta_{jt} : j = 1, \ldots, J$

The delta term of the kth output unit is

$$\delta_{kt} = \frac{\partial E_t}{\partial u_{kt}}. \tag{45}$$

The delta term of the jth hidden unit is

$$\delta_{jt} = \frac{\partial E_t}{\partial u_{jt}}. \tag{46}$$

All(!) we have to do now is to evaluate these delta terms.

The Delta Term of an Output Unit

For the kth output unit, use the chain rule to express the delta term as

$$\delta_{kt} = \frac{\partial E_t}{\partial y_{kt}} \frac{dy_{kt}}{du_{kt}}, \tag{47}$$

where

$$\frac{\partial E_t}{\partial y_{kt}} = (y_t - \tau_t) \tag{48}$$

$$\frac{dy_{kt}}{du_{kt}} = 1. \tag{49}$$

So the delta term for a linear output unit is

$$\delta_{kt} = (y_{kt} - \tau_{kt}). \tag{50}$$

See Equation 44 for using this to obtain weight change.

The Delta Term of a Hidden Unit

This is complicated because it depends on the delta terms of output units, and because hidden units have nonlinear activation functions

$$\delta_{jt} = \frac{\partial E_t}{\partial u_{jt}} = \frac{\partial E_t}{\partial z_{jt}} \frac{dz_{jt}}{du_{jt}}. \tag{51}$$

where

$$\frac{\partial E_t}{\partial z_{jt}} = \sum_{k=1}^{K} \frac{\partial E_t}{\partial u_{kt}} \frac{\partial u_{kt}}{\partial z_{jt}}, \tag{52}$$

where $\partial E_t / \partial u_{kt} = \delta_{kt}$, and $\partial u_{kt} / \partial z_{jt} = w_{jk}$ so that

$$\delta_{jt} = \frac{dz_{jt}}{du_{jt}} \sum_{k=1}^{K} \delta_{kt} \, w_{jk}. \tag{53}$$

Equation 33 implies $dz_{jt}/du_{jt} = z_{jt}(1 - z_{jt})$, so that

$$\delta_{jt} = z_{jt}(1 - z_{jt}) \sum_{k=1}^{K} \delta_{kt} \, w_{jk}. \tag{54}$$

See Equation 44 for using this to obtain weight change

The Backprop Algorithm

For all T associations, the gradient of E with respect to each hidden unit's weight w_{ij} is

$$\Delta w_{ij} = -\epsilon \sum_{t=1}^{T} \Delta w_{ijt}. \tag{55}$$

And, the gradient of E with respect to each output unit's weight w_{jk} is

$$\Delta w_{jk} = -\epsilon \sum_{t=1}^{T} \Delta w_{jkt}. \tag{56}$$

At this point, we have obtained an expression for the weight change applied to every weight in the neural network.

The Backprop Algorithm

Backprop (Short Version)

initialise network weights $\mathbf{w}$ to random values; set learning to true
while *learning* **do**
    set vector of gradients ∇E to zero
    foreach *association from* $t = 1$ *to* N **do**
        set input unit states $\mathbf{x}_{it}$ to tth training vector
        get state of output units $\mathbf{y}_{kt}$
        get delta term δ_{kt} for each output unit
        use output delta terms to get hidden unit delta terms
        use delta terms to get vector of weight gradients ∇E_t
        accumulate gradient $\nabla E \leftarrow \nabla E + \nabla E_t$
    end
    get weight change $\Delta \mathbf{w} = -\epsilon \nabla E$
    update weights $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
    if $|\nabla E| \approx 0$ **then**
        set learning to false
    end
end

Global and Local Minima

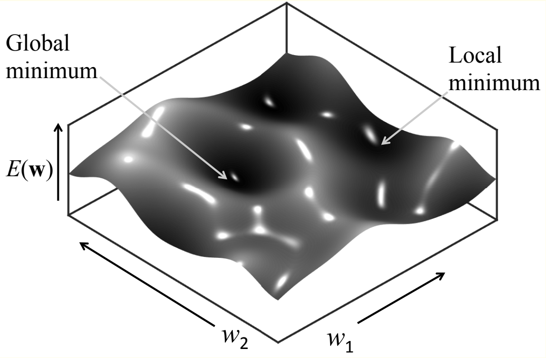


Figure 4: Schematic diagram of local and global minima in the error function $E(\mathbf{w})$ for a network with just two weights.

Global and Local Minima

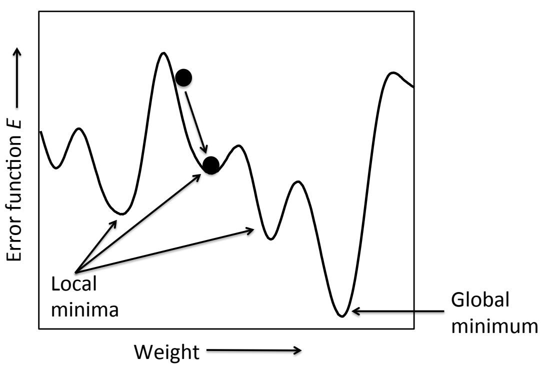
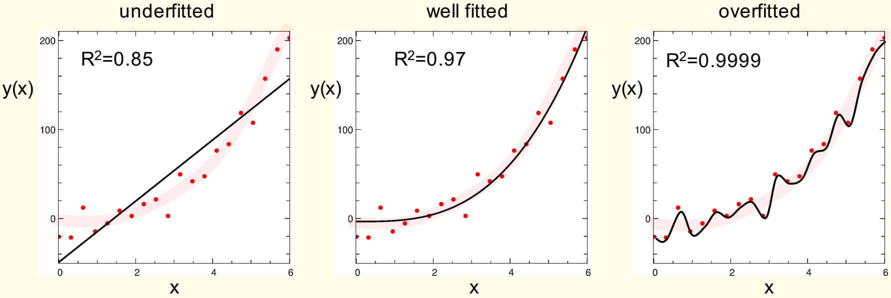


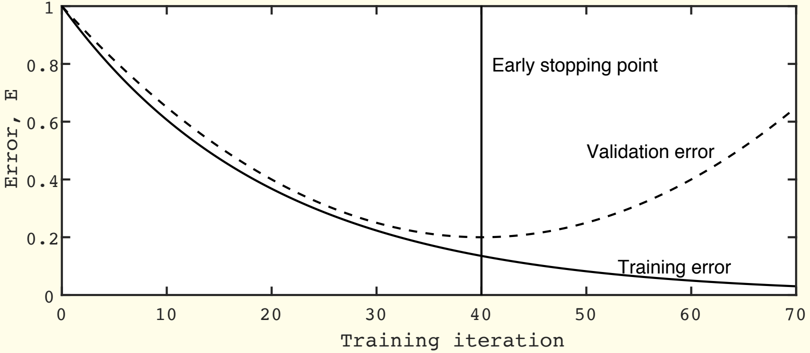
Figure 5: Local and global minima in a cross-section of the error function $E(\mathbf{w})$. At a given initial weight, $E(\mathbf{w})$ may be high, as represented by the black disc. Gradient-based methods always head downhill, but because they can only move downwards, the final weight often corresponds to a local minimum.

Over-fitting, under-fitting, Goldilocks-fitting



The red dots are data points, and the black curve is the fitted function. The quantity R^2 is the proportion of the variance in y that is accounted for by the fitted function.

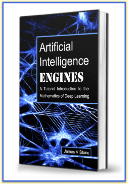
Preventing Over-fitting with Early Stopping



While learning a *training set*, the error on a separate *validation set* is monitored to gauge *generalisation* performance. Training is stopped when the validation error stops decreasing. Other methods include *regularisation*.

Further Reading

- Geoffrey Hinton and Yann LeCun, The Turing Lecture 2019: Comment: An overview from key researchers.
- Nielsen (2015), Neural Networks and Deep Learning is a free online book. Comment: A little dated, but still makes a fine starting point.
- Stone (2019), Artificial Intelligence Engines: A Tutorial Introduction to the Mathematics of Deep Learning.



The End

Thank you.