Backpropagation

— Lecture Notes —

Laurenz Wiskott
Institut für Neuroinformatik
Ruhr-Universität Bochum, Germany, EU

28 January 2017

Contents

Core text and formulas are set in dark red, one can repeat the lecture notes quickly by just reading these; ⧫ marks important formulas or items worth remembering and learning for an exam; ◊ marks less important formulas or items that I would usually also present in a lecture; + marks sections that I would usually skip in a lecture.

More teaching material is available at https://www.ini.rub.de/PEOPLE/wiskott/Teaching/Material/.

1 Supervised learning

1.1 Introduction

In *supervised learning* (D: überwachtes Lernen) **the** learning is done based on ***training data*** (D: Trainingsdaten) that **consist of *input data*** (D: Eingangs-, Eingabedaten) or *patterns* (D: Muster) $\mathbf{x}^\mu$ **and *required output data*** (D: vorgegebene Ausgangs-, Ausgabedaten) or *target values* (D: Zielwerte) s^μ (note that μ is an index here and not an exponent). Both may be vectors, but we assume for simplicity that the output data are scalar values. Generalization to output vectors is straight forward. The individual input-output data-pairs are indexed by $\mu = 1, ..., M$.

The goal of supervised learning is to find an *input-output function* $y(\mathbf{x})$ that generates an output

$$\blacklozenge \quad y^\mu := y(\mathbf{x}^\mu) \tag{1}$$

that is as close as possible to the required output data s^μ for the given input data $\mathbf{x}^\mu$.

To quantify the performance one defines an error function. A simple method to optimize the input-output function to minimize the error is gradient descent. This can be done on all data simultaneously or - which is biologically more plausible - on the individual data points separately, which is then referred to as an online learning rule.

1.2 Error function

An *error function* (D: Fehlerfunktion) has the purpose to quantify the performance of the system. If E^μ is some measure of error made for a single data point, the total error is typically the mean over all data points

$$\lozenge \quad E := \frac{1}{M} \sum_\mu E^\mu \,. \tag{2}$$

A common measure of error for a single data point is half the squared distance between output value and target value, i.e.

$$\lozenge \quad E^\mu := \frac{1}{2}(y^\mu - s^\mu)^2 \,, \tag{3}$$

so that **the total error is often measured by the *mean squared error*** (D: mittlerer quadratischer Fehler)

$$\blacklozenge \quad E = \frac{1}{M} \sum_\mu \frac{1}{2} \left(y^\mu - s^\mu\right)^2 \,, \tag{4}$$

which has to be minimized.

As will become clear later, the factor 1/2 has been introduced for mathematical convenience only and disappears in derivatives.

1.3 Gradient descent

The input-output function is usually parameterized in some way. For instance, we could define $y(\mathbf{x}) := w_1 x_1 + w_2 x_2$. In this case, w_1 and w_2 would be the parameters of the function. To emphasize this we can write $y_\mathbf{w}(\mathbf{x})$.

Learning means adapting the parameters somehow to optimize the function. If an error function is available, a standard method for this is gradient descent, **see figure 1. The idea of *gradient descent*** (D: Gradientenabstieg) **is to start somewhere in parameter space and then walk downhill on the error function until one ends up at the bottom of a valley.** This is a convenient method, because often it is easy to determine the gradient of the error function with respect to the parameters. Formally, the gradient is simply the vector of the derivatives of the error function with respect to the parameters,

$$\lozenge \quad \mathrm{grad}_\mathbf{w} E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_I} \right)^T \,. \tag{5}$$

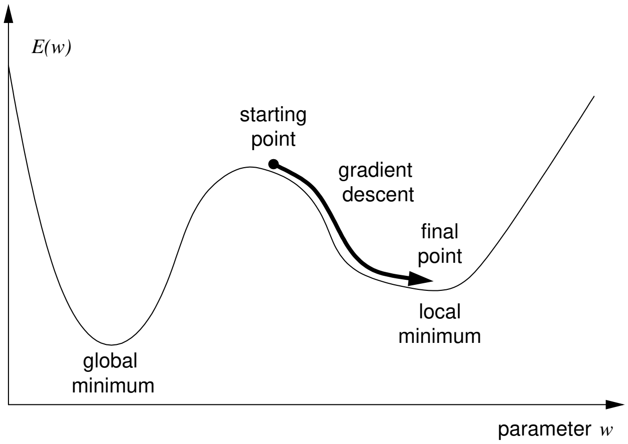


Figure 1: The idea of gradient descent is to start somewhere and then simply walk downhill. In this example there are two minima and the process ends up in the wrong one. If it had started a bit more to the left, it had found the global one.

The change of the weights have to go in the opposite direction of the gradient to go downhill and the steps should be of a reasonable size, so that one moves forward but does not jump too much around. This motivates a negative sign and introducing a *learning rate* (D: Lernrate) η, so that **the learning rule becomes**

$$\lozenge \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \quad \forall i \,, \tag{6}$$

or in vector notation

$$\lozenge \quad \Delta \mathbf{w} = -\eta \, \mathrm{grad}_{\mathbf{w}} E \,. \tag{7}$$

The main problem with gradient descent is that depending on the shape of the error function it can be sensitive to the starting point and is then not guaranteed to find the global minimum, cf. figure 1.

1.4 Online learning rule

From a biological point of view it is not particularly plausible to assume that first all training patterns are collected and then training is applied to all data simultaneously. **It is more reasonable to learn on each training pattern separately,**

$$\blacklozenge \quad \Delta w_i^\mu := -\eta \frac{\partial E^\mu}{\partial w_i} \,, \tag{8}$$

which is referred to as online or incremental learning (D: (inkrementelles) Lernen) in contrast to batch learning seen above. **The two methods are pretty much equivalent,** since

$$\begin{aligned}
\lozenge \quad \Delta w_i &\overset{(6)}{=} -\eta \frac{\partial E}{\partial w_i} && (9)\\
&\overset{(2)}{=} -\eta \frac{\partial}{\partial w_i}\left(\frac{1}{M}\sum_\mu E^\mu\right) && (10)\\
&= \frac{1}{M}\sum_\mu -\eta \frac{\partial E^\mu}{\partial w_i} && (11)\\
\lozenge \quad &\overset{(8)}{=} \frac{1}{M}\sum_\mu \Delta w_i^\mu \,. && (12)
\end{aligned}$$

Thus, it does not matter if we first calculate the mean error and then the gradient (weight change) of the total error or first the gradients (weight changes) for the individual errors and then take the average.

However, it is not quite the same, because when calculating the gradient for a new pattern, the weights have changed already on the previous pattern. Only if the learning rate is small is the change of the weights between two patterns so small that the equivalence between batch and online learning holds.

1.5 Examples

For illustration purposes consider two simple examples, nonlinear regression in one dimension and linear regression in many dimensions.

1.5.1 Nonlinear regression in x

If we choose a polynomial for the input-output function for one-dimensional input x, we can write

$$\Diamond \quad y(\mathbf{x}) \quad := \quad \sum_{i=0}^{I} w_i x^i \,, \tag{13}$$

with w_i indicating the weights of the monomials.
Given some training data x^μ and s^μ with $\mu = 1, ..., M$ **the error**

$$\Diamond \quad E \overset{(4,13)}{=} \frac{1}{M} \sum_\mu \underbrace{\frac{1}{2} \left(\sum_i w_i (x^\mu)^i - s^\mu \right)^2}_{=:E^\mu} \tag{14}$$

can be minimized by standard gradient descent on E with respect to the weights w_i.
By dropping the averaging over μ we get the online learning rule

$$\Diamond \quad \Delta w_i^\mu \quad := \quad -\eta \frac{\partial E^\mu}{\partial w_i} \tag{15}$$

$$\overset{(20)}{=} \quad -\eta \, \frac{\partial \left(\sum_k w_k (x^\mu)^k \right)}{\partial w_i} \left(\sum_j w_j (x^\mu)^j - s^\mu \right) \tag{16}$$

$$= \quad -\eta \, (x^\mu)^i \left(\sum_j w_j (x^\mu)^j - s^\mu \right) \tag{17}$$

$$\Diamond \quad \overset{(19)}{=} \quad -\eta \, (x^\mu)^i \, (y^\mu - s^\mu) \,. \tag{18}$$

Iterating this rule several times over all μ should lead to (locally) optimal weights w_i. Altogether this becomes a gradient descent version of nonlinear regression in x.

1.5.2 Linear regression in x ⋆

For illustration purposes consider linear regression as an example and let us phrase it in the jargon of neural networks. Linear regression can be done with **a simple neural network that has just one linear unit.** It realizes the function

$$\Diamond \quad y(\mathbf{x}) \quad := \quad \sum_i w_i x_i = \mathbf{w}^T \mathbf{x} \,, \tag{19}$$

with w_i indicating the *(synaptic) weights* (D: (synaptischen) Gewichte).
Given some training data $\mathbf{x}^\mu$ and s^μ with $\mu = 1, ..., M$ **the error**

$$\Diamond \quad E \overset{(4,19)}{=} \frac{1}{M} \sum_\mu \underbrace{\frac{1}{2} \left(\sum_i w_i x_i^\mu - s^\mu \right)^2}_{=:E^\mu} \tag{20}$$

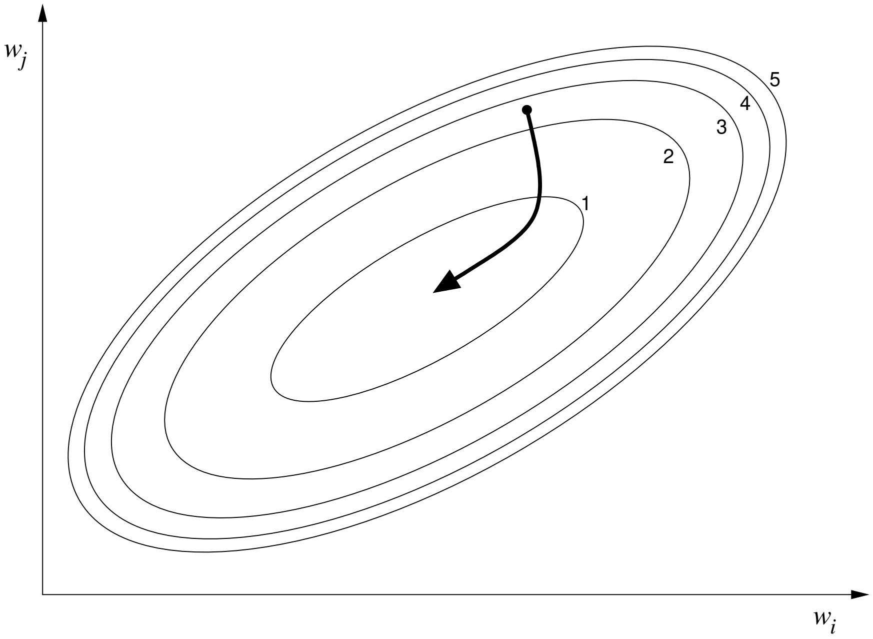


Figure 2: In linear regression the error function is a quadratic function and has only one minimum, and gradient descent is guaranteed to find it.
© CC BY-SA 4.0

can be minimized by standard gradient descent on E with respect to the weights w_i. Since (20) is a quadratic function in the weights, there is only one minimum and gradient descent is guaranteed to find it, see figure 2.

By dropping the averaging over μ we get the online learning rule

$$
\begin{aligned}
\Diamond \quad \Delta w_i^\mu &:= -\eta \frac{\partial E^\mu}{\partial w_i} && (21)\\
&\overset{(20)}{=} -\eta \frac{\partial \left(\sum_k w_k x_k^\mu\right)}{\partial w_i} \left(\sum_j w_j x_j^\mu - s^\mu\right) && (22)\\
&= -\eta\, x_i^\mu \left(\sum_j w_j x_j^\mu - s^\mu\right) && (23)\\
\Diamond \qquad &\overset{(19)}{=} -\eta\, x_i^\mu \left(y^\mu - s^\mu\right) . && (24)
\end{aligned}
$$

Iterating this rule several times over all μ should lead to (locally) optimal weights w_i. Altogether this becomes a gradient descent version of linear regression.

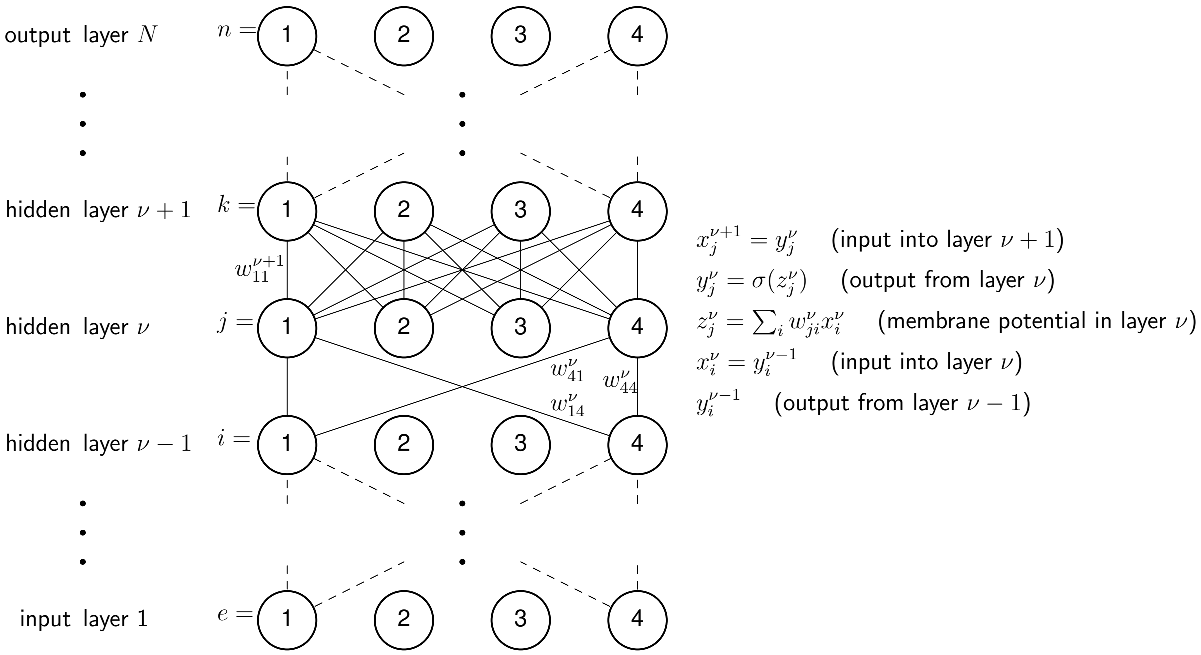
2 Supervised learning in multilayer networks

A neural network with just one linear unit is not particularly powerful. For any serious processing we need a multilayer network. However, with a multilayer network calculating the gradient can quickly become very involved. This makes the strategy sketched above problematic. Fortunately, some clever people came up with a nice algorithm that takes care of these complications - the error backprogation algorithm (Wikipedia, 2017).

2.1 Multilayer networks

A single unit network is obviously very limited. **To realize more complex input-output functions one has to combine several layers of units resulting in a *multilayer network*** (D: mehrschichtiges

Netz(werk)), **see figure 3.** To keep notation simple we drop the pattern index μ altogether. Instead, **we**



output layer N
$n =$
1
2
3
4
hidden layer $\nu + 1$
$k =$
$w_{11}^{\nu+1}$
hidden layer ν
$j =$
w_{41}^{ν}
w_{44}^{ν}
w_{14}^{ν}
hidden layer $\nu - 1$
$i =$
input layer 1
$e =$
$x_j^{\nu+1} = y_j^{\nu}$ (input into layer $\nu + 1$)
$y_j^{\nu} = \sigma(z_j^{\nu})$ (output from layer ν)
$z_j^{\nu} = \sum_i w_{ji}^{\nu} x_i^{\nu}$ (membrane potential in layer ν)
$x_i^{\nu} = y_i^{\nu-1}$ (input into layer ν)
$y_i^{\nu-1}$ (output from layer $\nu - 1$)

Figure 3: A multilayer network and its notation. Processing goes from bottom to top. Full connectivity is only shown between layers ν and $\nu + 1$, but is present between all successive layers.

use a superscript index ν to indicate the layers, with $\nu = 1$ for the input layer and $\nu = N$ for the output layer. Each layer can have several units indicated by $e, ..., i, j, k, ..., n$, with e for the input layer and n for the output layer. In summary **we have the following computation in the network.**

$$\begin{aligned} \blacklozenge \quad x_j^{\nu+1} &:= y_j^{\nu}\,, && (25)\\ \blacklozenge \quad y_j^{\nu} &:= \sigma(z_j^{\nu})\,, && (26)\\ \blacklozenge \quad z_j^{\nu} &:= \sum_i w_{ji}^{\nu} x_i^{\nu}\,. && (27) \end{aligned}$$

The outputs of one layer y_j^{ν} serve as the input $x_j^{\nu+1}$ to the next layer. σ is a sigmoidal nonlinearity, such as

$$\begin{aligned} \sigma(z) &:= \tanh(z) && (28)\\ \text{with} \quad \sigma'(z) &= 1 - \sigma(z)^2 = 1 - y^2\,, && (29)\\ \sigma(-z) &= -\sigma(z)\,. && (30) \end{aligned}$$

2.2 Error backpropagation

Computing the gradient for a multilayer network can quickly become cumbersome and computationally very expensive. However, by backpropagating an error signal, computation can be drastically simplified. **Since now the output layer can have several units we extend the error function** for a single input-output data-pair by summing over all output units

$$\blacklozenge \quad E^{\mu} := \sum_n \frac{1}{2}\left(y_n^{\mu} - s_n^{\mu}\right)^2\,, \qquad (31)$$

but remember that in the following we drop index μ for notational simplicity and use a superscript index ν to indicate the layer instead.

The partial derivative of E (31) with respect to any weight w_{ji}^{ν} is

$$\begin{aligned}
\Diamond \quad \frac{\partial E}{\partial w_{ji}^{\nu}} &\overset{(27)}{=} \frac{\partial E}{\partial z_j^{\nu}} \frac{\partial z_j^{\nu}}{\partial w_{ji}^{\nu}} \quad \text{(chain rule)} && (32)\\
&\overset{(27)}{=} \frac{\partial E}{\partial z_j^{\nu}} x_i^{\nu} && (33)\\
\Diamond \quad &\overset{(35)}{=} \delta_j^{\nu} x_i^{\nu} && (34)\\
\Diamond \quad \text{with} \quad \delta_j^{\nu} &:= \frac{\partial E}{\partial z_j^{\nu}} \,. && (35)
\end{aligned}$$

The x_i^{ν} can be easily computed with (25–27) iteratively from the input layer to the output layer. This leaves the δ_j^{ν} to be determined.

For the output layer we have

$$\begin{aligned}
\Diamond \quad \delta_n^N &\overset{(35)}{=} \frac{\partial E}{\partial z_n^N} && (36)\\
&\overset{(26)}{=} \frac{\partial E}{\partial y_n^N} \frac{\mathrm{d}y_n^N}{\mathrm{d}z_n^N} \quad \text{(chain rule)} && (37)\\
\Diamond \quad &\overset{(26,4)}{=} (y_n^N - s_n)\sigma'(z_n^N)\,. && (38)
\end{aligned}$$

For the hidden layers we get

$$\begin{aligned}
\Diamond \quad \delta_j^{\nu} &\overset{(35)}{=} \frac{\partial E}{\partial z_j^{\nu}} && (39)\\
&\overset{(25\text{–}27)}{=} \sum_k \frac{\partial E}{\partial z_k^{\nu+1}} \frac{\partial z_k^{\nu+1}}{\partial z_j^{\nu}} \quad \text{(chain rule)} && (40)\\
&\overset{(35)}{=} \sum_k \delta_k^{\nu+1} \frac{\partial z_k^{\nu+1}}{\partial z_j^{\nu}} && (41)\\
\Diamond \quad &\overset{(44)}{=} \sigma'(z_j^{\nu}) \sum_k \delta_k^{\nu+1} w_{kj}^{\nu+1}\,, && (42)\\
\text{because} \quad z_k^{\nu+1} &\overset{(25\text{–}27)}{=} \sum_j w_{kj}^{\nu+1} \sigma(z_j^{\nu}) && (43)\\
\Longrightarrow \quad \frac{\partial z_k^{\nu+1}}{\partial z_j^{\nu}} &= w_{kj}^{\nu+1}\,\sigma'(z_j^{\nu})\,. && (44)
\end{aligned}$$

Now, the δ_j^{ν} can be easily computed with the equations (38, 42). This process is referred to as *error backpropagation* (D: Fehlerrückführung) because δ_j^{ν} is related to the error, as can be seen from (38), and it gets computed from the output layer to the input layer (42).

If the x_i^{ν} and δ_j^{ν} are known, the gradient can be computed with (34) and the weight changes with (15). The different phases of the backpropagation algorithm are summarized in figure 4.

The change of weight is computed for each training data pair anew. Several iterations over all training data should lead to (locally) optimal weights. Unfortunately there is no guarantee to find the globally optimal weights.

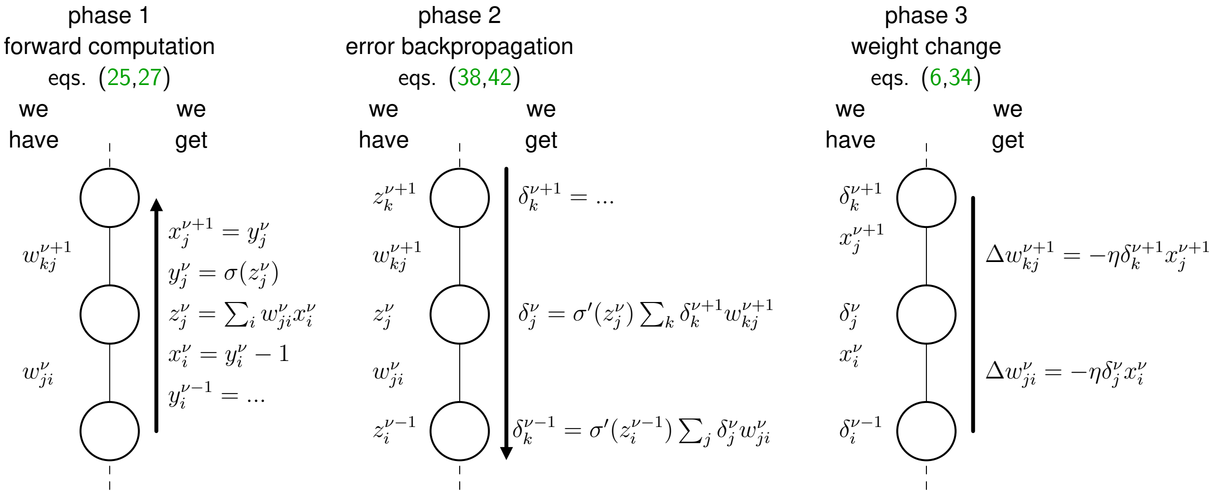
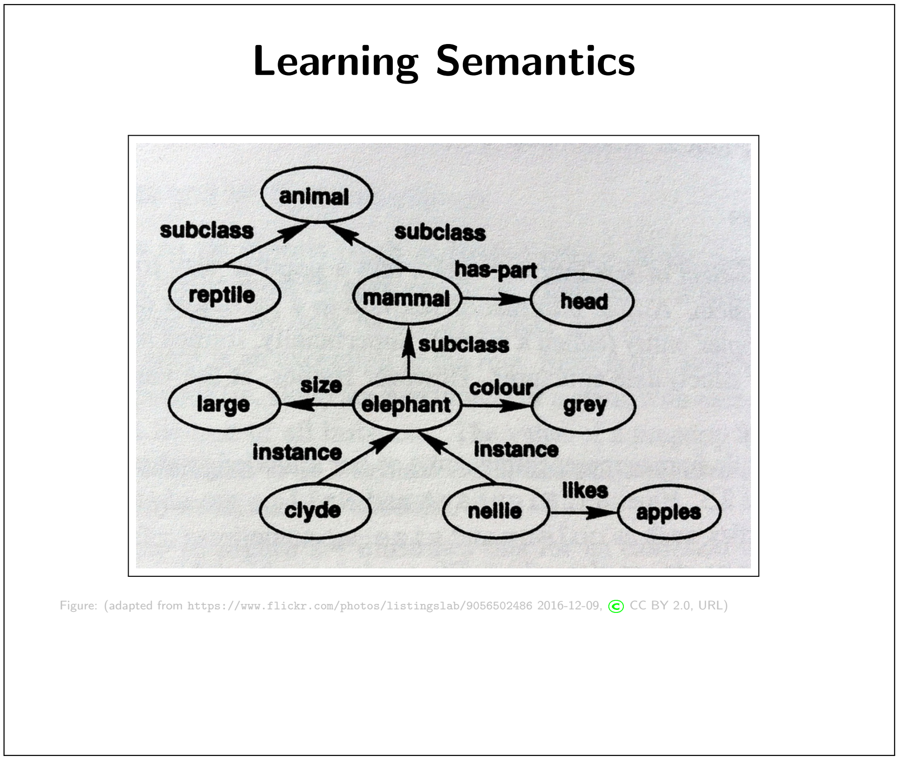

phase 1
forward computation
eqs. (25,27)
we have
we get
$w_{kj}^{\nu+1}$
w_{ji}^{ν}
$x_j^{\nu+1} = y_j^{\nu}$
$y_j^{\nu} = \sigma(z_j^{\nu})$
$z_j^{\nu} = \sum_i w_{ji}^{\nu} x_i^{\nu}$
$x_i^{\nu} = y_i^{\nu} - 1$
$y_i^{\nu-1} = \ldots$
phase 2
error backpropagation
eqs. (38,42)
we have
we get
$z_k^{\nu+1}$
$w_{kj}^{\nu+1}$
z_j^{ν}
w_{ji}^{ν}
$z_i^{\nu-1}$
$\delta_k^{\nu+1} = \ldots$
$\delta_j^{\nu} = \sigma'(z_j^{\nu}) \sum_k \delta_k^{\nu+1} w_{kj}^{\nu+1}$
$\delta_k^{\nu-1} = \sigma'(z_i^{\nu-1}) \sum_j \delta_j^{\nu} w_{ji}^{\nu}$
phase 3
weight change
eqs. (6,34)
we have
we get
$\delta_k^{\nu+1}$
$x_j^{\nu+1}$
δ_j^{ν}
x_i^{ν}
$\delta_i^{\nu-1}$
$\Delta w_{kj}^{\nu+1} = -\eta \delta_k^{\nu+1} x_j^{\nu+1}$
$\Delta w_{ji}^{\nu} = -\eta \delta_j^{\nu} x_i^{\nu}$

Figure 4: The phases of the backpropagation algorithm.
© CC BY-SA 4.0

3 Sample applications (→ slides)

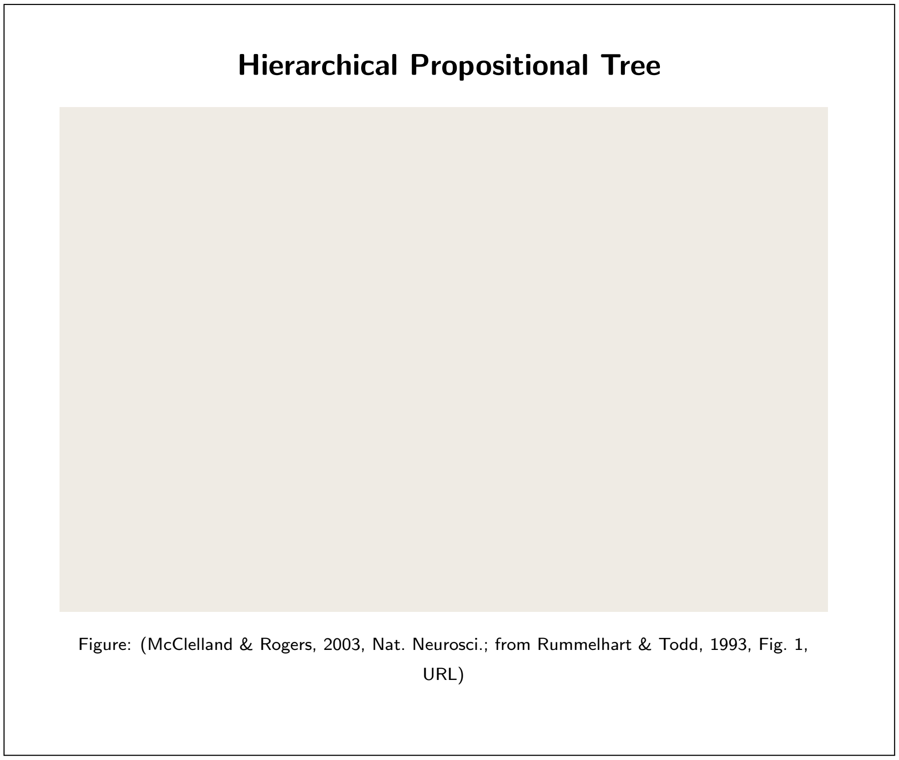


Learning Semantics

Figure: (adapted from https://www.flickr.com/photos/listingslab/9056502486 2016-12-09, © CC BY 2.0, URL)

Learning a semantic representation with a backpropagation network is an example where in some sense a rather high-level cognitive task is solved with a rather simple network. It illustrates that a clever representation can make a complicated task simple.

Figure: (adapted from https://www.flickr.com/photos/listingslab/9056502486 2016-12-09, © CC BY 2.0, URL)[1]

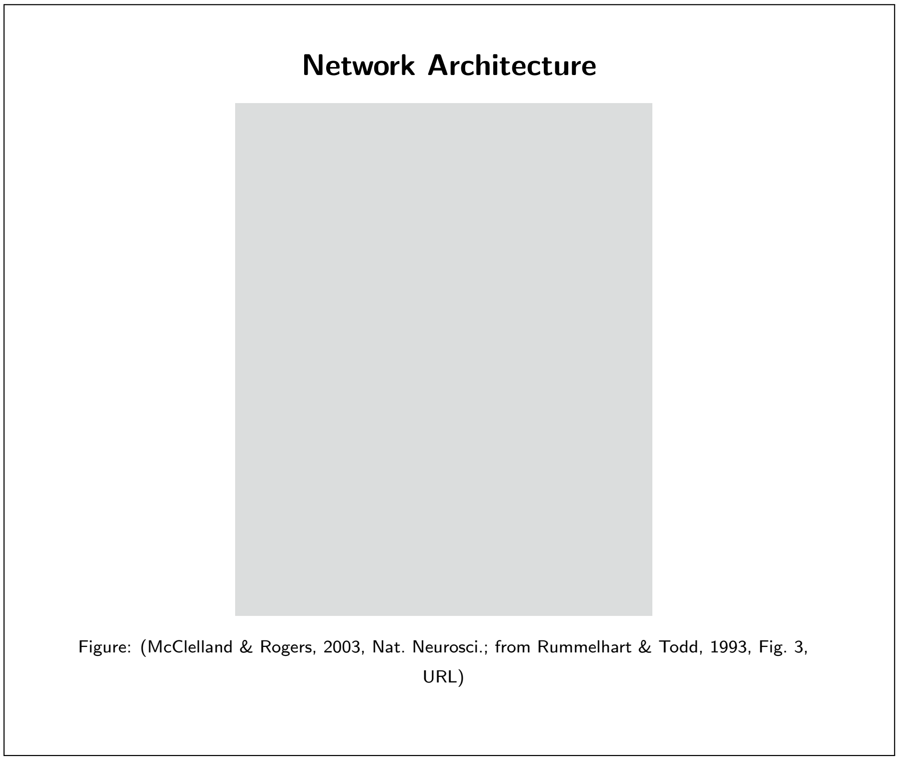


Hierarchical Propositional Tree

Figure: (McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 1, URL)

Quillian (1968, Semantic Information Processing, MIT Press) proposed that semantic knowledge is represented in a hierarchical propositional tree. The more abstract concepts are are at the root and the more specific concepts are at the leaves. Properties can be stored at different locations depending on how general they are. Rummelhart & Todd (1993, Attention and Performance XIV, MIT Press) applied this to a simple example of living things to create a structured training set for training a backpropagation network to learn semantics.

Figure: (McClelland and Rogers, 2003, Fig. 1, URL)[2]



Network Architecture

Figure: (McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 3, URL)

A network considered by (Rumelhart and Todd, 1993) for semantic learning consists of one input layer indicating a specific plant or animal, one input layer indicating a propositional realtionship, such as 'is a' or 'can', two hidden layers, and one output layer indicating higher order concepts, such as 'tree' or 'living thing', or properties, such as 'big' or 'feathers'. The units highlighted are active and indicate a training example teaching the network that a robin can grow, move, and fly. The network is trained with many of such examples using the backpropagation algorithm.

Figure: (McClelland and Rogers, 2003, Fig. 3, URL)[3]

Hidden Layer Representation

Figure: (McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 4a, URL)

This graph shows how the first hidden layer representation develops with training. Each graph shows the activity of the eight first hidden layer units when one of the first input layer units is active. After 25 epoches the representation is still fairly unstructured and undifferentiated. One epoche corresponds to the presentation of all training examples. After 500 epoches the representation is much more structured, i.e. has more strongly active or inactive units, and is more differentiated, i.e. different input leads to different hidden layer activities. It is interesting that similar input classes, such as salmon and sunfish, lead to similar hidden layer activities. This is the key to the generalization capability of the network.
Figure: (McClelland and Rogers, 2003, Fig. 4a, URL)[4]

Discovered Similarities

Figure: (McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 4b, URL)

This graph illustrates the similarities between the hidden layer activities for different inputs more clearly. The Euclidean distance between the hidden layer activity vectors become large for dissimilar input classes, such as ’canary’ and ’oak’, but remains small for similar ones, such as ’canary’ and ’robin’. For instance, the network makes a natural distinction between plants and animals because of their different properties.
Figure: (McClelland and Rogers, 2003, Fig. 4b, URL)[5]

Generalization

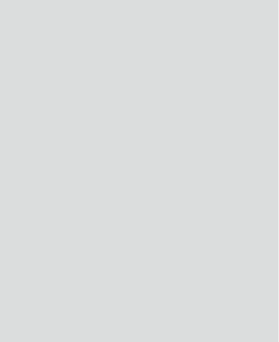


Figure: (McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 3, URL)

Trained only with the 'sparrow ISA animal & bird' input-output pair the network learns the correct reponse also to the 'sparrow is / can / has' inputs.

This does not work for a penguin.

The structured hidden layer representation allows the network to generalize to new input classess, such as 'sparrow'. If one continues to train the network with 'sparrow ISA animal & bird', the network generalizes the correct answers also for the 'sparrow is / can / has' questions.

It is obvious that this cannot work for penguins. The network will not know that a penguine cannot fly until it is explicitely taught so.

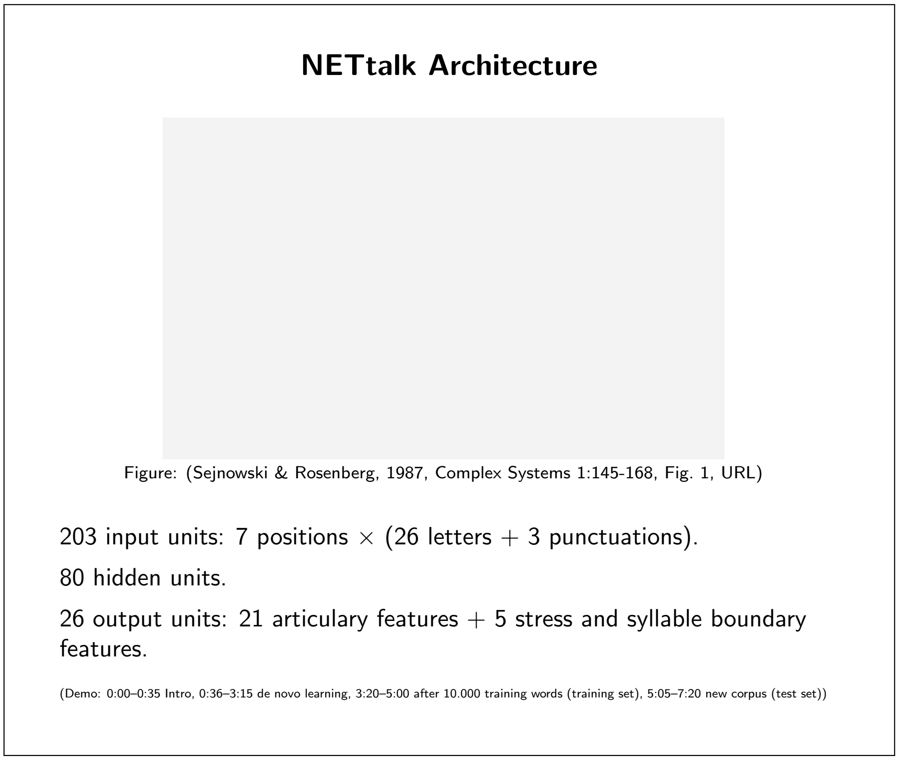
Figure: (McClelland and Rogers, 2003, Fig. 3, URL)[6]

NETtalk

Image: (http://www.cartoonstock.com/lowres/hkh00461.jpg 2005-10-27, URL)

Although quite old, the NETtalk example is still a very popular application where a backpropagation network was trained to read text and actually produce speach through a speach synthesizer.

Image: (http://www.cartoonstock.com/lowres/hkh00461.jpg 2005-10-27, URL)[7]



NETtalk Architecture

Figure: (Sejnowski & Rosenberg, 1987, Complex Systems 1:145-168, Fig. 1, URL)

203 input units: 7 positions × (26 letters + 3 punctuations).

80 hidden units.

26 output units: 21 articulary features + 5 stress and syllable boundary features.

(Demo: 0:00–0:35 Intro, 0:36–3:15 de novo learning, 3:20–5:00 after 10.000 training words (training set), 5:05–7:20 new corpus (test set))

The network receives text as input and produces phonemes as output. The phonemes can then be fed into a speach synthesizer to produce sound. The input layer represents seven consecutive characters of the text. For each character position there are 26 units for the letters plus 3 units for punctuation. One of these 29 units is active and indicates the character at hand. This makes $203 = 7 \times (26 + 3)$ input units. Text is shifted through the character positions from right to left, which corresponds to reading from left to right. The hidden layer has 80 units. The output layer has 26 units, 21 of which represent phonemes (articulary features) and 5 represent stress and syllable boundaries.

The whole network has been trained with the backpropagation algorithm on a large corpus of hand labeled text. Creating this training corpus was actually the main work.

The audio example shows how the reading performance improves over time and that the network is able to generalize to new text. It is interesting that in some cases the network makes similar errors as children as they learn reading, for example in the way it over-generalizes. However, one should keep in mind that children learn reading very differently. For instance, they first learn talking and then reading while the network learns both simultaneously.

Figure: (Sejnowski and Rosenberg, 1987, Fig. 1, URL)[8]

NETtalk Learning Curves

Figure: (Sejnowski & Rosenberg, 1987, Complex Systems 1:145-168, Fig. 4, URL)

This graph shows how the performance improves with training. It is no surprise that the network performs better on the stress features than on the phonemes, because there are fewer of the former (5, chance level 1/5) than of the latter (21, chance level 1/21). The exponential learning curve is also rather typical, with fast learning in the beginning and much slower learning in the end.

Figure: (Sejnowski and Rosenberg, 1987, Fig. 4, URL)[9]

References

McClelland, J. L. and Rogers, T. T. (2003). The parallel distributed processing approach to semantic cognition. *Nat. Rev. Neurosci.*, 4(4):310–322. 9, 10, 11

Rumelhart, D. E. and Todd, P. M. (1993). Learning and connectionist representations. In Meyer, D. E. and Kornblum, S., editors, *Attention and performance XIV: Synergies in experimental psychology, artificial intelligence, and cognitive neuroscience*, pages 3–30. MIT Press, Cambridge, Massachusetts. 9

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Complex systems*, 1(1):145–168. 12, 13

Wikipedia (2017). Backpropagation. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Backpropagation accessed 28-January-2017. 5

Notes

[1] adapted from https://www.flickr.com/photos/listingslab/9056502486 2016-12-09, © CC BY 2.0, https://www.flickr.com/photos/listingslab/9056502486

[2] McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 1, https://cnbc.cmu.edu/%7eplaut/IntroPDP/papers/McClellandRogers03NatNeuRev.semCog.pdf

[3] McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 3, https://cnbc.cmu.edu/%7eplaut/IntroPDP/papers/McClellandRogers03NatNeuRev.semCog.pdf

[4] McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 4a, https://cnbc.cmu.edu/%7eplaut/IntroPDP/papers/McClellandRogers03NatNeuRev.semCog.pdf

[5] McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 4b, https://cnbc.cmu.edu/%7eplaut/IntroPDP/papers/McClellandRogers03NatNeuRev.semCog.pdf

[6] McClelland & Rogers, 2003, Nat. Neurosci.; from Rummelhart & Todd, 1993, Fig. 3, https://cnbc.cmu.edu/%7eplaut/IntroPDP/papers/McClellandRogers03NatNeuRev.semCog.pdf

[7] http://www.cartoonstock.com/lowres/hkh0046l.jpg 2005-10-27, http://www.cartoonstock.com/lowres/hkh0046l.jpg

[8] Sejnowski & Rosenberg, 1987, Complex Systems 1:145-168, Fig. 1, https://www.researchgate.net/profile/Terrence_Sejnowski/publication/237111620_Parallel_Networks_that_Learn_to_Pronounce/links/54a4b0150cf257a63607270d.pdf

[9] Sejnowski & Rosenberg, 1987, Complex Systems 1:145-168, Fig. 4, https://www.researchgate.net/profile/Terrence_Sejnowski/publication/237111620_Parallel_Networks_that_Learn_to_Pronounce/links/54a4b0150cf257a63607270d.pdf

Copyrightprotectionlevel: 2/ 2