Neural networks

Göran Bergqvist

AI Competence for Sweden

Neural network = function (of several variables) $f(\bar{x})$, which is a composition of several "simple" functions

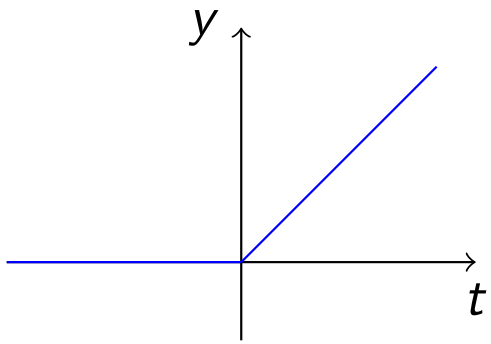
$$y = f(\bar{x}) = f_L(\dots \bar{f}_2(\bar{f}_1(\bar{x}))\dots) = (f_L \circ \cdots \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x})$$

$$\bar{x} \overset{\bar{f}_1}{\mapsto} \bar{f}_1(\bar{x}) \overset{\bar{f}_2}{\mapsto} \bar{f}_2(\bar{f}_1(\bar{x})) \overset{\bar{f}_3}{\mapsto} \cdots \overset{f_L}{\mapsto} f_L(\dots \bar{f}_2(\bar{f}_1(\bar{x}))\dots) = y$$

L = number of layers in the neural network

$\bar{f}_j$ "simple":
each $\bar{f}_j$ usually matrix multiplication (linear), then maybe adding a constant vector (affine), followed by σ (nonlinear but simple)

A nonlinear function and its derivative:



$$\sigma(t) = (t)_+ = \begin{cases} t\ , & t \geq 0 \\ 0\ , & t < 0 \end{cases} \quad \text{and} \quad H(t) = \sigma'(t) = \begin{cases} 1\ , & t > 0 \\ 0\ , & t < 0 \\ \mathit{undef}, & t = 0 \end{cases}$$

For vectors:

$$\bar{\sigma}(\bar{t}) = (\bar{t})_+ = \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix}_+ = \begin{pmatrix} (t_1)_+ \\ \vdots \\ (t_n)_+ \end{pmatrix}, \text{ e.g. } \bar{\sigma} \begin{pmatrix} -1 \\ 2 \\ 3 \\ -4 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \\ -4 \end{pmatrix}_+ = \begin{pmatrix} 0 \\ 2 \\ 3 \\ 0 \end{pmatrix}$$

$\bar{\sigma}$ zeros negative components

Example of neural network with 3 layers, $f(\bar{x}) = (f_3 \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x})$:

$$A_1 = \begin{pmatrix} 2 & -1 \\ 1 & 1 \end{pmatrix}, B_1 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & 1 \\ 2 & 0 \\ -3 & 1 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & -1 & 1 \end{pmatrix}$$

$\bar{x} \in \mathbb{R}^2$, $\bar{x}_1 = \bar{f}_1(\bar{x}) = \bar{\sigma}(A_1\bar{x} + B_1) = (A_1\bar{x} + B_1)_+ \in \mathbb{R}^2$

$\bar{x}_2 = \bar{f}_2(\bar{x}_1) = \bar{\sigma}(A_2\bar{x}_1) = (A_2\bar{x}_1)_+ \in \mathbb{R}^3, \quad y = f_3(\bar{x}_2) = A_3\bar{x}_2 \in \mathbb{R}$

E.g., $\bar{x} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \Rightarrow \bar{x}_1 = (A_1 \begin{pmatrix} 2 \\ 1 \end{pmatrix} + B_1)_+ = \begin{pmatrix} 2 \\ 5 \end{pmatrix}_+ = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \Rightarrow$

$\Rightarrow \bar{x}_2 = (A_2 \begin{pmatrix} 2 \\ 5 \end{pmatrix})_+ = \begin{pmatrix} 7 \\ 4 \\ -1 \end{pmatrix}_+ = \begin{pmatrix} 7 \\ 4 \\ 0 \end{pmatrix} \Rightarrow y = A_3 \begin{pmatrix} 7 \\ 4 \\ 0 \end{pmatrix} = 3$

Hence, $f(\begin{pmatrix} 2 \\ 1 \end{pmatrix}) = (f_3 \circ \bar{f}_2 \circ \bar{f}_1)(\begin{pmatrix} 2 \\ 1 \end{pmatrix}) = (f_3 \circ \bar{f}_2)(\begin{pmatrix} 2 \\ 5 \end{pmatrix}) = f_3(\begin{pmatrix} 7 \\ 4 \\ 0 \end{pmatrix}) = 3$

Neural network $\quad y = f(\bar{x}) = (f_L \circ \cdots \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x})$

Layer j : $\quad \bar{f}_j(\bar{x}_{j-1}) = \bar{\sigma}(A_j\bar{x}_{j-1} + B_j) = (A_j\bar{x}_{j-1} + B_j)_+$

How do we find a neural network for a certain application?
An input $\bar{x}$ should give a good output y.
How do we determine all the matrices in the network?

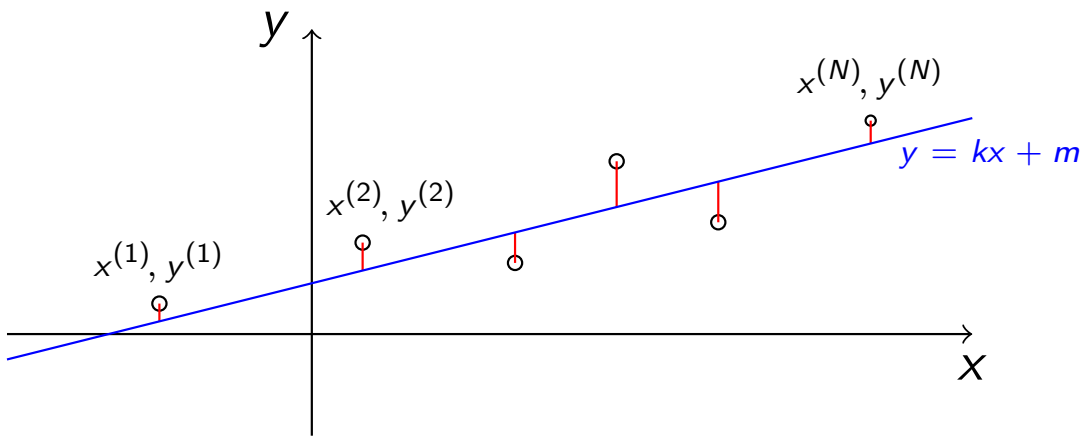
Use that we have training data:

for N points $\bar{x}^{(1)}, \ldots, \bar{x}^{(N)}$ (input data),
we know the output values $y^{(1)}, \ldots, y^{(N)}$

Fit the matrices A_j and B_j in all $\bar{f}_j$ to our training data.

Called supervised learning. We "train" the network.
Deep learning if many layers.

Remember: Least-square fit of line $y = kx + m$ (find k and m) in $\mathbb{R}^2$ to N data points :



Find k and m such that $\sum_{r=1}^{N}((kx^{(r)} + m) - y^{(r)})^2 = \ell(k, m)$ is minimized:

Solve

$$\nabla\ell(k,m) = (\ell'_k, \ell'_m) \underset{\substack{\nabla(g^2)\\ =2g\nabla g}}{=} 2\sum_{r=1}^{N}(kx^{(r)}+m-y^{(r)}) \underbrace{\nabla(kx^{(r)} + m - y^{(r)})}_{=(x^{(r)},1)} = (0,0)$$

Alt.: $\ell(k, m) = ||A\bar{x} - \bar{b}||^2$, where $A = \begin{pmatrix} x^{(1)} & 1 \\ \vdots & \vdots \\ x^{(N)} & 1 \end{pmatrix}, \bar{x} = \begin{pmatrix} k \\ m \end{pmatrix}, \bar{b} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}$

Linear algebra: solve the normal equations $A^T A\bar{x} = A^T\bar{b}$ to obtain k and m (least-square solution to the unsolvable system $A\bar{x} = \bar{b}$).

$$y = f(\bar{x}) = (f_L \circ \cdots \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x}) \; ; \quad \bar{f}_j(\bar{x}_{j-1}) = (A_j \bar{x}_{j-1} + B_j)_+$$

Determine the matrices A_j and B_j in all $\bar{f}_j$ such that $f(\bar{x}^{(k)}) \approx y^{(k)}$ for the training data $\{\bar{x}^{(k)}, y^{(k)}\}, k = 1, \ldots, N$.

Denote all elements in $A_1, B_1, \ldots, A_L, B_L$ by $w_1, \ldots, w_M$.

Least-square fit of the matrices to training data:

Find $\bar{w} = (w_1, \ldots, w_M)$ that minimizes $\ell(\bar{w}) = \frac{1}{N} \sum_{k=1}^{N} (f(\bar{x}^{(k)}; \bar{w}) - y^{(k)})^2$

Write $f(\bar{x}^{(k)}; \bar{w})$ since the output f depends on the matrix elements $\bar{w}$.

Min-problem in $\mathbb{R}^M$: $\nabla \ell(\bar{w}) = \frac{2}{N} \sum_{k=1}^{N} (f(\bar{x}^{(k)}; \bar{w}) - y^{(k)}) \nabla_w f(\bar{x}^{(k)}; \bar{w}) = \bar{0}$

gives stationary points, but in practice impossible to solve exactly (M very large). Find an algorithm for minimizing $\ell(\bar{w})$.

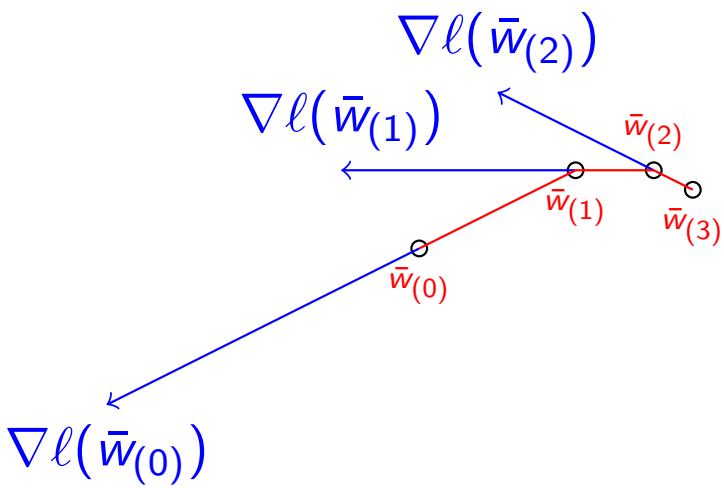
Note: ∇_w means gradient w.r.t. the $\bar{w}$-variables (not w.r.t. $\bar{x}$). The constant factor $\frac{1}{N}$ in $\ell(\bar{w})$ has no impact on the result but is usually included.

Minimize $\ell(\bar{w}) = \frac{1}{N}\sum_{k=1}^{N}(f(\bar{x}^{(k)};\bar{w}) - y^{(k)})^2$ on $\mathbb{R}^M$

$$\nabla\ell(\bar{w}) = \frac{2}{N}\sum_{k=1}^{N}(f(\bar{x}^{(k)};\bar{w}) - y^{(k)})\nabla_w f(\bar{x}^{(k)};\bar{w})$$

Algorithm:

Pick initial point $\bar{w}_{(0)} \in \mathbb{R}^M$, $\ell(\bar{w})$ decreases fastest in direction $-\nabla\ell(\bar{w}_{(0)})$ $\Rightarrow$ step in this direction to new point $\bar{w}_{(1)} = \bar{w}_{(0)} - \eta\nabla\ell(\bar{w}_{(0)}) \in \mathbb{R}^M$. Continue in direction $-\nabla\ell(\bar{w}_{(1)})$ to new point $\bar{w}_{(2)}$ etc etc. Check $\ell(\bar{w}_{(k)})$ at each step and stop the iteration when it is acceptably small. Then $\bar{w}$ is determined, and hence all matrices are determined, and the neural network is ready to be used.



This method is called "steepest descent" in optimization. The best length η of the steps can be difficult to decide.

How do we calculate $\nabla\ell(\bar{w})$?

How to calculate $\nabla_w f(\bar{x}^{(k)};\bar{w})$?

The Jacobian matrix of $\bar{f}(\bar{x}) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$ is

$$\bar{f}'(\bar{x}) = \frac{\partial(\bar{f})}{\partial(\bar{x})} = \frac{\partial(f_1, \ldots, f_m)}{\partial(x_1, \ldots, x_n)} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \quad m \times n \text{ matrix}$$

If $\bar{g}(\bar{x}) = A\bar{x}$ (linear) or $\bar{g}(\bar{x}) = A\bar{x} + B$ (affine), then $\bar{g}'(\bar{x}) = \frac{\partial(\bar{g})}{\partial(\bar{x})} = A$

$$\bar{\sigma}(\bar{t}) = \begin{pmatrix} \sigma(t_1) \\ \vdots \\ \sigma(t_n) \end{pmatrix} \Rightarrow \bar{\sigma}'(\bar{t}) = \frac{\partial(\bar{\sigma})}{\partial(\bar{t})} = \begin{pmatrix} H(t_1) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & H(t_n) \end{pmatrix} = H(\bar{t})$$

If $\bar{x}$ constant and the elements of A and B variables in $A\bar{x} + B$, write e.g.

$$\bar{g}(w_1, \ldots, w_6) = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} w_5 \\ w_6 \end{pmatrix} = \begin{pmatrix} w_1 x_1 + w_2 x_2 + w_5 \\ w_3 x_1 + w_4 x_2 + w_6 \end{pmatrix} \Rightarrow$$

$$\frac{\partial(g_1, g_2)}{\partial(w_1, \ldots, w_6)} = \underbrace{\begin{pmatrix} x_1 & x_2 & 0 & 0 \\ 0 & 0 & x_1 & x_2 \end{pmatrix}}_{\text{from } A} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\text{from } B} = \underbrace{\begin{pmatrix} \bar{x}^T & \bar{0}^T & 1 & 0 \\ \bar{0}^T & \bar{x}^T & 0 & 1 \end{pmatrix}}_{\text{block notation}} = M(\bar{x})$$

Chain rule: The composition $(\bar{f} \circ \bar{g})(\bar{x}) = \bar{f}(\bar{g}(\bar{x}))$ has Jacobian

$$(\bar{f} \circ \bar{g})'(\bar{x}) = \bar{f}'(\bar{g}(\bar{x}))\bar{g}'(\bar{x}) \quad \text{or} \quad \frac{\partial(\bar{f})}{\partial(\bar{x})} = \frac{\partial(\bar{f})}{\partial(\bar{g})}\frac{\partial(\bar{g})}{\partial(\bar{x})} \quad \text{(matrix mult.)}$$

One layer $\bar{f}_j(\bar{x}_{j-1}) = \bar{\sigma}(\underbrace{A_j\bar{x}_{j-1}+B_j}_{=\bar{t}_j})$, where $\bar{x}_{j-1} = \bar{f}_{j-1}(\bar{x}_{j-2})$, has matrix

$$\frac{\partial(\bar{f}_j)}{\partial(\bar{f}_{j-1})} = H(\bar{t}_j)A_j = H_jA_j \qquad \text{(usually no } \bar{\sigma} \text{ or } H_L \text{ in last } f_L)$$

Neural network $y = f(\bar{x}) = (f_L \circ \cdots \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x}) = (f_L \circ \cdots \circ \bar{f}_{j+1})(\bar{x}_j) = (f_L \circ \cdots \circ \bar{f}_{j+1})[\bar{f}_j(\bar{x}_{j-1})] = (f_L \circ \cdots \circ \bar{f}_{j+1})[\bar{\sigma}(A_j\bar{x}_{j-1} + B_j)]$

If A_j and B_j contain elements $w_p, \ldots, w_q$, repeated use of chain rule gives

$$\frac{\partial(y)}{\partial(w_p, \ldots, w_q)} = \frac{\partial(f_L)}{\partial(\bar{f}_{L-1})}\frac{\partial(\bar{f}_{L-1})}{\partial(\bar{f}_{L-2})} \cdots \frac{\partial(\bar{f}_{j+1})}{\partial(\bar{f}_j)}\frac{\partial(\bar{f}_j)}{\partial(w_p, \ldots, w_q)} =$$

$$= A_L(H_{L-1}A_{L-1}) \ldots (H_{j+1}A_{j+1})(H_jM_j) \quad \text{where } M_j = M(\bar{x}_{j-1})$$

For each $\bar{x}^{(k)}$ we calculate $\nabla_w f(\bar{x}^{(k)}; \bar{w}) = \dfrac{\partial(y)}{\partial(w_1, \ldots, w_M)} =$

$$= (\underbrace{A_L H_{L-1} \ldots A_2 H_1 M_1}_{\text{derivatives on elements in } A_1, B_1}, \underbrace{A_L H_{L-1} \ldots H_2 M_2}_{\text{on elements in } A_2, B_2}, \ldots, \underbrace{A_L H_{L-1} M_{L-1}}_{\text{in } A_{L-1}, B_{L-1}}, \underbrace{M_L}_{\text{in } A_L}),$$

and the difference $f(\bar{x}^{(k)}; \bar{w}) - y^{(k)}$. Add $\Rightarrow$

$$\nabla \ell(\bar{w}) = \frac{2}{N} \sum_{k=1}^{N} (f(\bar{x}^{(k)}; \bar{w}) - y^{(k)}) \nabla_w f(\bar{x}^{(k)}; \bar{w})$$

is now computed, and can be used for steepest descent calculations.

Example

Find a small neural network $f(\bar{x}) = (f_3 \circ \bar{f}_2 \circ \bar{f}_1)(\bar{x}) = f_3(\bar{f}_2(\bar{f}_1(\bar{x})))$ that fits $N = 5$ training data points $(\bar{x}^{(1)}, y^{(1)}), \ldots, (\bar{x}^{(5)}, y^{(5)})$:

$$\left(\begin{pmatrix}-0.5\\1.1\end{pmatrix}, 3\right),\ \left(\begin{pmatrix}0.6\\-0.7\end{pmatrix}, 1.5\right),\ \left(\begin{pmatrix}1.4\\0.6\end{pmatrix}, 2\right),\ \left(\begin{pmatrix}1.1\\-0.4\end{pmatrix}, 1.7\right),\ \left(\begin{pmatrix}-0.3\\-1.1\end{pmatrix}, 2.5\right)$$

The network structure should be

$$\bar{x}_1 = \bar{f}_1(\bar{x}) = (A_1\bar{x} + B_1)_+ \ , \quad A_1 = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix}, \ B_1 = \begin{pmatrix} w_5 \\ w_6 \end{pmatrix}$$

$$\bar{x}_2 = \bar{f}_2(\bar{x}_1) = (A_2\bar{x}_1 + B_2)_+ \ , \quad A_2 = \begin{pmatrix} w_7 & w_8 \\ w_9 & w_{10} \end{pmatrix}, \ B_2 = \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix}$$

$$y = f_3(\bar{x}_2) = A_3\bar{x}_2 \ , \quad A_3 = \begin{pmatrix} w_{13} & w_{14} \end{pmatrix}$$

Minimize $\quad \ell(\bar{w}) = \frac{1}{5}\sum_{k=1}^{5}(f(\bar{x}^{(k)}; \bar{w}) - y^{(k)})^2 \quad$ on $\mathbb{R}^{14}$

Step along minus $\nabla\ell(\bar{w}) = \frac{2}{5}\sum_{k=1}^{5}(f(\bar{x}^{(k)}; \bar{w}) - y^{(k)})\nabla_w f(\bar{x}^{(k)}; \bar{w})$

$$\nabla_w f(\bar{x}^{(k)}; \bar{w}) = \frac{\partial(y)}{\partial(w_1, \ldots, w_{14})} = (\underbrace{A_3H_2A_2H_1M_1}_{\text{deriv. on } w_1,..,w_6}, \underbrace{A_3H_2M_2}_{w_7,..,w_{12}}, \underbrace{M_3}_{w_{13},w_{14}})$$

Try starting at $\bar{w}_{(0)} = (1, 2, 0, -1, -1, 1, 0, 2, 1, -1, 1, 2, 1, 1)$, with steps $\eta = 0.05$. Initially, the matrices are

$$A_1 = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}, B_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 2 \\ 1 & -1 \end{pmatrix}, B_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

Training data point $\bar{x}^{(1)} = \begin{pmatrix} -0.5 \\ 1.1 \end{pmatrix}, y^{(1)} = 3$ gives

$$\bar{x}_1^{(1)} = (A_1\bar{x}^{(1)} + B_1)_+ = \begin{pmatrix} 0.7 \\ -0.1 \end{pmatrix}_+ = \begin{pmatrix} 0.7 \\ 0 \end{pmatrix} \Rightarrow \bar{x}_2^{(1)} = (A_2\bar{x}_1^{(1)} + B_2)_+ = \begin{pmatrix} 1 \\ 2.7 \end{pmatrix}_+ = \begin{pmatrix} 1 \\ 2.7 \end{pmatrix}$$

$$\Rightarrow \quad f(\bar{x}^{(1)}; \bar{w}) = A_3\bar{x}_2^{(1)} = 3.7 \quad \text{and} \quad f(\bar{x}^{(1)}; \bar{w}) - y^{(1)} = 3.7 - 3 = 0.7,$$

and $H_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, H_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, M_1 = \begin{pmatrix} -0.5 & 1.1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -0.5 & 1.1 & 0 & 1 \end{pmatrix}$,

$M_2 = \begin{pmatrix} 0.7 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.7 & 0 & 0 & 1 \end{pmatrix}, M_3 = \begin{pmatrix} 1 & 2.7 \end{pmatrix}$, which gives

$A_3H_2M_2 = (0.7, 0, 0.7, 0, 1, 1), A_3H_2A_2H_1M_1 = (-0.5, 1.1, 0, 0, 1, 0)$ so

$\nabla_w f(\bar{x}^{(1)}; \bar{w}_{(0)}) = (-0.5, 1.1, 0, 0, 1, 0, 0.7, 0, 0.7, 0, 1, 1, 1, 2.7) \quad \circledast$

$\Rightarrow$ term $k=1$ in $\nabla\ell(\bar{w}) = \frac{2}{5}\sum_{k=1}^{5}(\underbrace{f(\bar{x}^{(k)}; \bar{w}) - y^{(k)}}_{=0.7 \text{ för } k=1})\underbrace{\nabla_w f(\bar{x}^{(k)}; \bar{w})}_{=\circledast \text{ för } k=1}$ is ready

$$A_1 = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}, B_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 2 \\ 1 & -1 \end{pmatrix}, B_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

Similar calculations for the 4 other training points $\bar{x}^{(2)}, \ldots, \bar{x}^{(5)}$ gives

$$\ell(\bar{w}_{(0)}) = \frac{1}{5}\sum_{k=1}^{5}(f(\bar{x}^{(k)}; \bar{w}_{(0)}) - y^{(k)})^2 = 6.862 \quad \text{and}$$

$$\nabla\ell(\bar{w}_{(0)}) = \frac{2}{5}\sum_{k=1}^{5}(f(\bar{x}^{(k)}; \bar{w}_{(0)}) - y^{(k)})\nabla_w f(\bar{x}^{(k)}; \bar{w}_{(0)}) \approx$$

$$\approx (1.54, 1.03, 2.99, -2.98, 1.48, 5.72, 2.12, 6.44, 2.12, 4.17, 4.92, 3.84, 17.8, 5.63)$$

Take the step to the new point, $\bar{w}_{(1)} = \bar{w}_{(0)} - 0.05\nabla\ell(\bar{w}_{(0)}) \approx$
$(0.92, 1.95, -0.15, -0.85, -1.07, 0.71, -0.11, 1.68, 0.89, -1.21, 0.75, 1.81, 0.11, 0.72)$
New matrix values are therefore

$$A_1 \approx \begin{pmatrix} 0.92 & 1.95 \\ -0.15 & -0.85 \end{pmatrix}, B_1 \approx \begin{pmatrix} -1.07 \\ 0.71 \end{pmatrix}, A_2 \approx \begin{pmatrix} -0.11 & 1.68 \\ 0.89 & -1.21 \end{pmatrix}, B_2 \approx \begin{pmatrix} 0.75 \\ 1.81 \end{pmatrix}, A_3 \approx \begin{pmatrix} 0.11 & 0.72 \end{pmatrix}$$

Repeat the process with these matrices. Gives $\ell(\bar{w}_{(1)}) \approx 1.55$, $\nabla\ell(\bar{w}_{(1)}) \approx$
$(0.25, -0.30, 0.48, -0.11, -0.25, 0.36, -0.02, -0.24, -0.11, -0.57, -0.22, -0.82, -5.20, -1.24)$
and $\bar{w}_{(2)} = \bar{w}_{(1)} - 0.05\nabla\ell(\bar{w}_{(1)}) \approx$
$(0.91, 1.96, -0.17, -0.85, -1.06, 0.70, -0.10, 1.69, 0.90, -1.18, 0.76, 1.85, 0.37, 0.78)$

Steps with the gradient method gives (approximative values)

Point	A_1	B_1	A_2	B_2	A_3	error $\ell(\bar{w}_{(k)})$
$\bar{w}_{(0)}$	$\begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 \\ 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$	$(1 \quad 1)$	6.86
$\bar{w}_{(1)}$	$\begin{pmatrix} 0.92 & 1.95 \\ -0.15 & -0.85 \end{pmatrix}$	$\begin{pmatrix} -1.07 \\ 0.71 \end{pmatrix}$	$\begin{pmatrix} -0.11 & 1.68 \\ 0.89 & -1.21 \end{pmatrix}$	$\begin{pmatrix} 0.75 \\ 1.81 \end{pmatrix}$	$(0.11 \quad 0.72)$	1.55
$\bar{w}_{(2)}$	$\begin{pmatrix} 0.91 & 1.96 \\ -0.17 & -0.85 \end{pmatrix}$	$\begin{pmatrix} -1.06 \\ 0.70 \end{pmatrix}$	$\begin{pmatrix} -0.10 & 1.69 \\ 0.90 & -1.18 \end{pmatrix}$	$\begin{pmatrix} 0.76 \\ 1.85 \end{pmatrix}$	$(0.37 \quad 0.78)$	0.51
...						
...						
$\bar{w}_{(5)}$	$\begin{pmatrix} 0.86 & 1.98 \\ -0.19 & -0.88 \end{pmatrix}$	$\begin{pmatrix} -1.06 \\ 0.73 \end{pmatrix}$	$\begin{pmatrix} -0.12 & 1.72 \\ 0.88 & -1.19 \end{pmatrix}$	$\begin{pmatrix} 0.78 \\ 1.85 \end{pmatrix}$	$(0.54 \quad 0.76)$	0.26
...						
...						
$\bar{w}_{(10)}$	$\begin{pmatrix} 0.79 & 2.01 \\ -0.20 & -0.92 \end{pmatrix}$	$\begin{pmatrix} -1.05 \\ 0.76 \end{pmatrix}$	$\begin{pmatrix} -0.13 & 1.73 \\ 0.86 & -1.23 \end{pmatrix}$	$\begin{pmatrix} 0.79 \\ 1.82 \end{pmatrix}$	$(0.55 \quad 0.75)$	0.22
...						
...						
$\bar{w}_{(20)}$	$\begin{pmatrix} 0.66 & 2.06 \\ -0.21 & -0.98 \end{pmatrix}$	$\begin{pmatrix} -1.03 \\ 0.81 \end{pmatrix}$	$\begin{pmatrix} -0.13 & 1.72 \\ 0.85 & -1.31 \end{pmatrix}$	$\begin{pmatrix} 0.79 \\ 1.79 \end{pmatrix}$	$(0.55 \quad 0.78)$	0.15
...						
...						
$\bar{w}_{(40)}$	$\begin{pmatrix} 0.43 & 2.13 \\ -0.28 & -1.00 \end{pmatrix}$	$\begin{pmatrix} -1.02 \\ 0.81 \end{pmatrix}$	$\begin{pmatrix} -0.13 & 1.73 \\ 0.87 & -1.33 \end{pmatrix}$	$\begin{pmatrix} 0.78 \\ 1.78 \end{pmatrix}$	$(0.55 \quad 0.81)$	0.087
...						

When we consider ℓ sufficiently small, we have found our neural network and it is ready to be used!

Comments

Number of training data points might be $N \sim 10^3 - 10^6$
Number of elements in all matrices (length of $\bar{w}$) can be $M \sim 10^6 - 10^9$

Lots of computing time is needed to calculate the direction $\nabla \ell(\bar{w})$ of the next step as a sum from N training points. Usually one randomly chooses a smaller set of these ($\sim 10 - 10^2$), and approximates $\nabla \ell(\bar{w})$ with the sum over those. This is called a stochastic gradient method and saves lots of time. Huge neural networks may take weeks to train on supercomputers.

The step length η is called "learning rate". Choosing η is a big problem. For smaller optimization problems there are ways of finding the best η. For neural networks they require too much time, and one decides on a value from the start (sometimes decreaseing values in each iteration).

The number of iterations with the gradient method before one stops is called epochs. Choice of initial point $\bar{w}_{(0)}$ is difficult, but techniques for good choices exist. One may try several $\bar{w}_{(0)}$, but it takes time.

The nonlinear function $\sigma(t)$ is called activation function. There are many choices but $(t)_+$ is now the most common, called ReLU(t) by AI people.

More comments

The last f_L is often different depending on the type of output wanted (scalar/vector, continuous/discrete etc).

Different choices of loss function $\ell(\bar{w})$ can be made, depending on what error measure one wants for training data.

Elements in $\bar{w}$ (all matrix elements) are called weights. The number of layers, sizes of matrices, choice of η, etc are called the hyperparameters of the net. One can try different choices of hyperparameters but it costs. The networks are called deep if the have many layers, examples with $\sim 10^2 - 10^3$ layers exist.

Some training data points (often 10-30 %) are not used when training the net, but are kept as test data for a test if the ready net seems OK. This is fundamental for checking that one has not over- or underfitted the size of the net. Overfit $\Rightarrow$ generalizes poorly to new input data $\bar{x}$. Underfit $\Rightarrow$ cannot describe the function one looks for.

The computation of $\nabla\ell(\bar{w})$ using the chain rule is called the back propagation algorithm, and some claim it was discovered in the 1980's. Other sources claim the chain rule was known before that ....

More comments (2)

The min-problem in $\mathbb{R}^M$ is very difficult to analyze analytically, probably a huge number of stationary points exist, and many local minima. That the above methods converge to networks that work incredibly well in applications is a mathematical mystery. There are almost certainly many different nets that work equally well for a given application. Also in other mathematical contexts functions of many variables seem to have many almost equally good local optima. To design and train neural networks has become an art for skilled engineers and mathematicians!

Many applications utilizes matrices (and tensors) with special structure. Although the matrices are big, many elements are 0 and the others have only a few different values $\Rightarrow$ fewer weights $\Rightarrow$ much faster to train. Used with great success in, e.g., computer vision, where input data points $\bar{x}$ really are three matrices (or a 3-tensor); these nets are called convolutional neural networks (CNN's).

There's lots of ready software for building neural networks, using them does not require knowing how they work or how they are trained. Like a black box. Test yourselves on playground.tensorflow.org !

Everything fine ?

Neural networks don't give explanations, difficult to know what in the input $\bar{x}$ that decide $f(\bar{x})$. Better than humans in analyzing images or playing games, but has nothing to do with logical thinking or ability to follow rules. Who's responsible if something goes wrong in,e.g., medicine?

Ethical and juridical problem:

Bias is built into the nets if it is present in training data, many examples of discrimination exist. Who is responsible? How can fairness be guaranteed? Neural networks are used by banks to decide who can get a loan (and they don't know themselves what information about us that decides), in medicine to make diagnoses, in some countries to decide if convicted need to spend time in prison.

Individuals may have legal right to get decisions about them motivated and explained, difficult if neural networks were used.

Mass survaillence with face recognition - integrity?

Language translating programs can have discriminating translations built in (Google Translate is a neural network since a couple of years).

Neural networks can be manipulated by those who know how they are constructed, and training data can be manipulated.

Finally

Are (artificial) neural networks similar to the net of neurons in our brain? The neurons in our brains are incredibly many, and their link structure complicated.

One has started designing hardware with special architecture that should imitate the brain and lead to much faster training of (artificial) neural networks.

Neural networks are now used everywhere, and within many areas, including biology, chemistry and medicine, their use is increasing dramatically. They can be used in many positive ways, but one must be awary of the risks. Then it is good to know how they ("the algorithms") work. That's your job!

Remember:

A neural network is a function $y = f(\bar{x})$ which is a composition of several simpler functions.
It is trained by minimizing a loss function for a large amount of training data, then gradient methods and the chain rule are used.