Machine Learning

Lecture 16: Neural networks

Stefan Harmeling

6. December 2017

Was haben Sie nicht verstanden?

- Ich habe heute komplett den Anschluss verloren.
- Complete data? Incomplete data? Complete: x, z; incomplete x
- Wieso statistische Größen von Daten ermitteln, die man gar nicht hat?! wenn man ein Gaussian mixture model fitten möchten, gibt es bis jetzt keine andere Möglichkeit.
- Über was für *theta* und $theta_0$ maximieren wir überhaupt? die Parameter the Gaussian mixture, siehe 15.10 (10. Folie in der 15. Vorlesung)

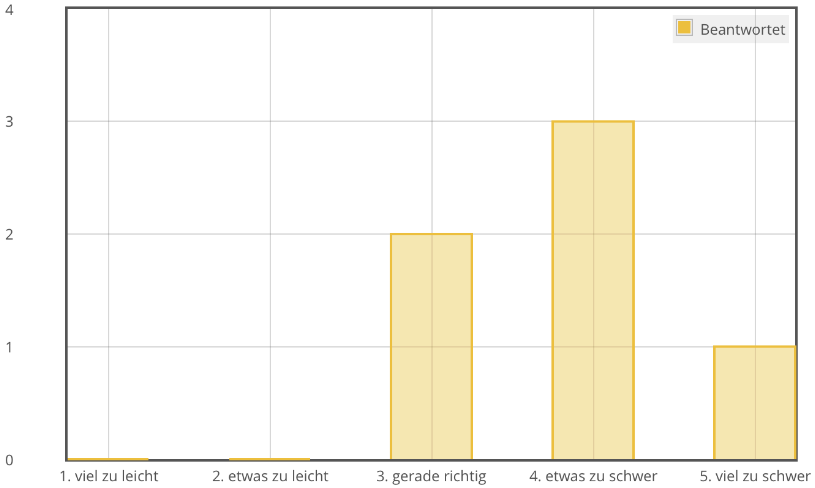
Was hat Ihnen gefallen?

- Die Erklärung zur Entropie
- Das Droppen der englischen Terms showt, dass wir international people am been sind

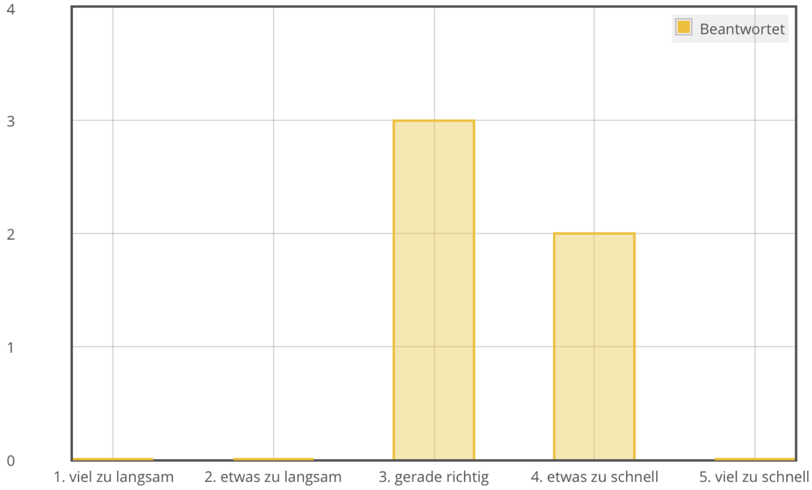
Was hat Ihnen nicht gefallen?

- Vorlesung war heute zu abstrakt

War die Vorlesung zu leicht oder zu schwer?



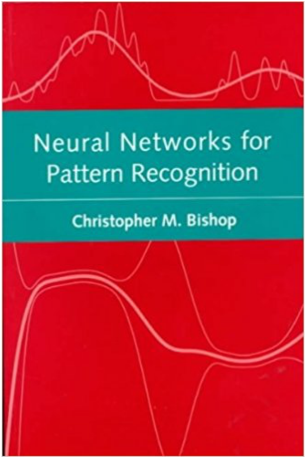
War die Vorlesung zu schnell oder zu langsam?



Haben Sie weitere Anmerkungen?

- Die Jeans stehen Ihnen
- Some good jokes for parties : Deep Belief Nets actually believe deeply in Geoff Hinton. Geoff Hinton never takes the plane. He doesn’t even take the hyperplane. He prefers to ride on a quasi-spherical Riemannian manifold. Geoff Hinton can make you regret without bounds. :DDD
- Könnten Sie die Literaturempfehlung, die Sie heute im Zusammenhang mit Entropie beim Spiel "Wer bin ich" in einem Nebensatz erwähnten, in der nächsten Vorlesung nennen? Das Buch möchte ich wirklich gerne lesen!
- Buch: “Elements of Information Theory” von Cover/Thomas

We follow Chapters 3 and 4 of the book:
Neural Networks and Pattern Recognition
by Chris Bishop



Single-layer networks

Two category classification problem

- linear discriminant function

$$y(x) = w^T x + w_0$$

  with weights w and threshold w_0 (sometimes called bias)
- if $y(x) > 0$ assign x to class 1 otherwise to class 2
- our first neural network! (see board)
- creates a linear decision boundary
- the bias can be merged into the weights:

$$\begin{aligned}\tilde{x} &= [1, x^T]^T \\ \tilde{w} &= [w_0, w^T]^T \\ y(x) &= \tilde{w}^T \tilde{(}x)\end{aligned}$$

  (see board, getting from 1D to 2D, or from 2D to 3D)

Single-layer networks

Several category classification problem

- linear discriminant function for k classes

$$y_k(x) = w_k^T x + w_{k0}$$

  with per-class weights w_k and threshold w_{k0} (sometimes called bias)
- if $y_k(x) > y_j(x)$ for all $k \neq j$ assign x to class k
- neural network representation (see board)
- consider x assigned to class k, i.e. for $j \neq k$:

$$\begin{aligned} y_k(x) - y_j(x) &> 0 \\ (w_k - w_j)^T x + w_{k0} - w_{j0} &= 0 \end{aligned}$$

- creates also linear decision boundaries (see board, similar to Voronoi cells)

Single-layer networks

Logistic discriminants

- generalize linear discriminant function for two classes by applying monotonic non-linear function g called *activation function*:

$$y(x) = g(w^T x + w_0)$$

- still linear decision boundary (monotonicity), so what's the point? are there useful choices for g?

Single-layer networks

Consider two class problem

- assume the locations x sampled from Gaussians with class-dependent means μ_1 and μ_2 and covariance matrix Σ:

$$p(x|C_k) = \frac{1}{(2\pi)^d/2|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma(x-\mu_k)\right)$$

- Bayes theorem implies:

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \frac{1}{1 + \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)}}$$

$$= \frac{1}{1 + \exp(-a)} = g(a) \quad \textit{the logistic sigmoid}$$

  with $a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} = w^T x + w_0$. Can be solved for w and w_0.

- Thus: the output for the logistic sigmoid can be interpreted as posterior probabilities.

Single-layer networks

The expressions for w and w_0 from the previous slide are:

$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$w_0 = -\tfrac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 + \tfrac{1}{2}\mu_2^T \Sigma^{-1} \mu_2 + \ln \frac{p(C_1)}{p(C_2)}$$

Single-layer networks

Linear discriminats

- discriminant function for two classes by applying sigmoid:

$$y(x) = g(w^T x + w_0)$$

- how powerful is this? any limitations? e.g. limitations in 2D?
- XOR-problem can not be solved! (see board)
- this is related to the VC dimension of linear classifier in two dimensions which is 3
- from the perspective of statistical learning theory, failing at the XOR problem is not a bad thing, but bounds the capacity of the single-layer networks which leads to regularization...
- nonetheless: how can we make it more powerful?

Single-layer networks

Generalized linear discriminants

- generalized discriminant function for several classes:

$$y_k(x) = \sum_{j=0}^{M} w_{kj}\phi_j(x)$$

  with pre-defined *basis functions* ϕ_j (aka features) and fixing $\phi_0(x) = 1$ to omit the theshold/bias
- how powerful is this? any limitations?
- one can show that for suitable choice of basis functions *any* continuous function can be arbitrarily well approximated
- so there is also a choice that solves the XOR-problem, e.g. in 2D:

$$y(x) = w\phi(x)$$

  with a single basis function $\phi(x) = x_1 x_2$ for $x = [x_1, x_2]^T$.

Single-layer networks

Learn weights from data:

- given N training data points $(x^1, t^1), \ldots, (x^N, t^N)$ consisting of locations and targets $t^n = [t_1^n, \ldots, t_c^n]^T$ for c classes
- $t_k^n = 1$ if x^n is in class k otherwise $t_k^n = 0$

Least-squares techniques

- Minimize sum-of-squares error function:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(y_k(x^n) - t_k^n)^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\left(\sum_{j=0}^{M} w_{kj}\phi_j(x) - t_k^n\right)^2$$

- derivative of E wrt. w:

$$\frac{\partial}{\partial w_{kj}}E(w) = \sum_{n=1}^{N}\left(\sum_{j'=0}^{M} w_{kj'}\phi_{j'}(x) - t_k^n\right)\phi_j(x) = \sum_{n=1}^{N} r_k^n \phi_j(x)$$

where residuals $r_k^n = y_k(x^n) - t_k^n$

Single-layer networks

Derivative for a single location:

$$\frac{\partial}{\partial w_{kj}} E^n(w) = r_k^n \phi_j(x)$$

Stochastic gradient descent

- iterate over all data points and update the weights using some learning rate, i.e. for $n = 1$ to N update w:

$$w_{kj} \leftarrow w_{kj} - \eta r_k^n \phi_j(x)$$

with learning rate η

Goal: given training examples $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, \theta))^2$$

Algorithm 16.1 (Batch gradient descent)

1. *Initialize θ_0.*
2. *For several epochs update the parameters with the overall gradient:*

$$\theta_{t+1} = \theta_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta} f(x_i, \theta_t)$$

Algorithm 16.2 (Stochastic gradient descent)

1. *Initialize θ_0.*
2. *For several epochs update the parameters with a local gradient at a randomly chosen location (e.g. for point x_i):*

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} f(x_i, \theta_t)$$

Single-layer networks

The perceptron for the two class problem (Rosenblatt, 1962)

- output of the perceptron:

$$y(x) = g\left(\sum_{j=0}^{M} w_j \phi_j(x)\right) = g(w^t \phi(x))$$

  with activation function $g(a) = \text{sign}(a)$ being 1 if $a \geq 0$ and -1 otherwise
- given N training data points $(x^1, t^1), \dots, (x^N, t^N)$ consisting of locations and targets $t^n \in \{-1, +1\}$
- perceptron criterion: minimize

$$E^{\text{perc}}(w) = -\sum_{y(x^n) \neq t^n} w^T \phi(x^n) t^n$$

  where the summation is over all miss-classified training examples
- essentially the same as *adalines* (ADAptive LINear Element, Widrow and Hoff 1960)

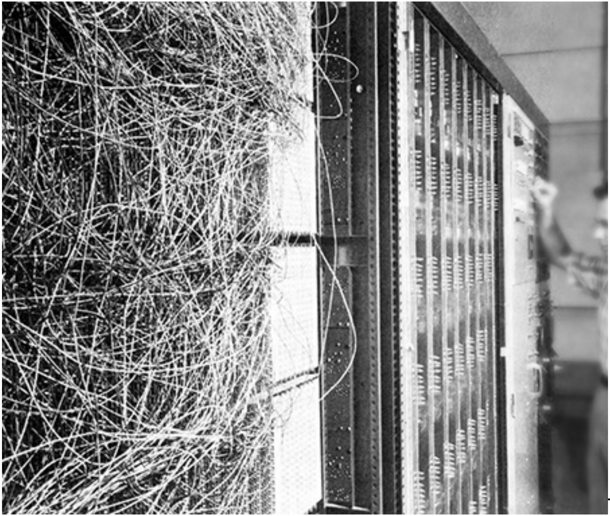
Single-layer networks

Pioneer of connectionism:

Frank Rosenblatt, with the image sensor of the Mark I Perceptron, 1962

Single-layer networks

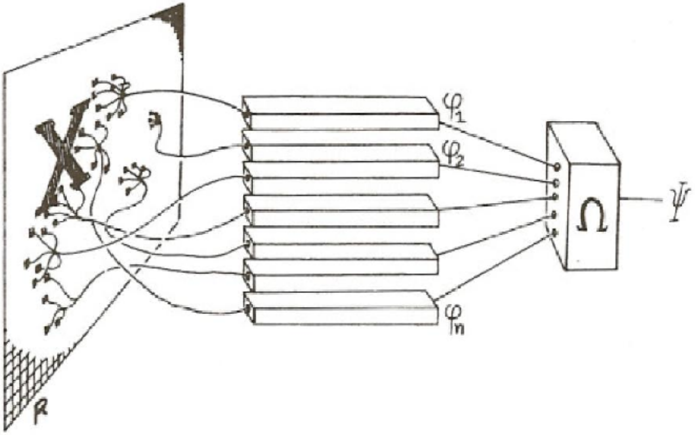
The perceptron (Rosenblatt, 1962)



The Mark I Perceptron

Single-layer networks

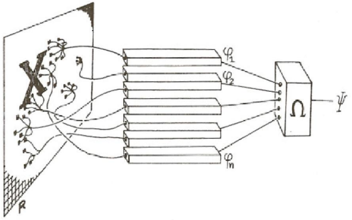
The perceptron (Rosenblatt, 1962)



From Perceptrons by M. L Minsky and S. Papert, Copyright 1969 by MIT Press

Single-layer networks

The perceptron (Rosenblatt, 1962)



From Perceptrons by M. L Minsky and S. Papert, Copyright 1969 by MIT Press

Limits?

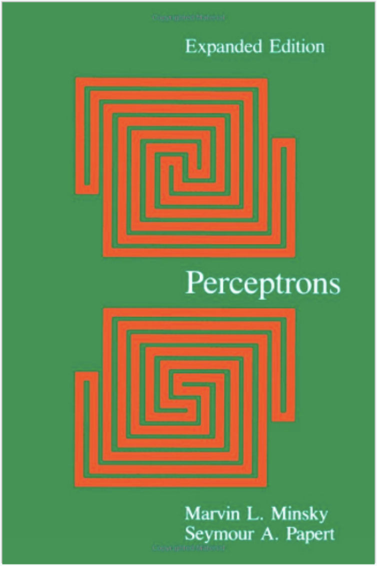
- can solve XOR problem with appropriate choice of ϕ
- however, when limiting the *receptive fields* of the single $\phi_j(x)$ certain connectivity problems can not be solved (book "Perceptron", Minsky and Papert, 1969)

Spot the tiger!


© Poonam Ohri

Single-layer networks

The diameter-limited perceptron can not solve the connectivity problem



Can you?

From Single-layer to Multi-layer networks

How to generalize the single-layer network?

$$y(x) = g\left(\sum_{j=0}^{M} w_j \phi_j(x)\right) = g(w^t \phi(x))$$

Make $\phi_j(x)$ adaptive as well! Make it another single-layer network!

Multi-layer network

- with one hidden layer:

$$y(x) = b_2 + W_2 \tanh(b_1 + W_1 x)$$

  with activation function tanh, vector-valued input x and thresholds b_1 and b_2, output y, and matrix-valued weights W_1 and W_2.

- with two hidden layers:

$$y(x) = b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x))$$

Multi-layer networks

Multi-layer network:

$$y(x) = b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x))$$

Forward computation:

```
z1 = x;                       # input layer
z2 = tanh(b1 + W1*z1);        # second layer (hidden)
z3 = tanh(b2 + W2*z2);        # third layer (hidden)
y  =      b3 + W3*z3 ;        # output layer
r  = y-t;                     # compare to truth
E  = 0.5 * r' * r;            # squared error
```

Multi-layer networks

Calculating the differentials

$$
\begin{aligned}
\mathrm{d}E &= r^{\mathsf{T}} \mathrm{d}r \\
&= r^{\mathsf{T}} \mathrm{d}y \\
&= r^{\mathsf{T}} (\mathrm{d}b_3 + (\mathrm{d}W_3) z_3 + W_3 \mathrm{d}z_3) \qquad \text{read off } \mathrm{D}_{b_3} E \text{ and } \mathrm{D}_{W_3} E \\
&= r^{\mathsf{T}} W_3 \mathrm{d}z_3 \qquad \text{continuing with } z_3 \\
&= r^{\mathsf{T}} W_3 \mathrm{d} \tanh(b_2 + W_2 z_2) \\
&= r^{\mathsf{T}} W_3 \operatorname{Diag}(1 - z_3 \odot z_3)(\mathrm{d}b_2 + (\mathrm{d}W_2) z_2 + W_2 \mathrm{d}z_2) \\
&= \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{d}\tanh(a) &= \mathrm{d}\frac{e^a - e^{-a}}{e^a + e^{-a}} \\
&= \frac{(e^a + e^{-a})(e^a + e^{-a})\mathrm{d}a - (e^a - e^{-a})(e^a - e^{-a})\mathrm{d}a}{(e^a + e^{-a})^2} \\
&= (1 - \tanh(a)^2)\mathrm{d}a
\end{aligned}
$$

Multi-layer networks

Forward computation:

```
z1 = x;                      # input layer
z2 = tanh(b1 + W1*z1);       # second layer (hidden)
z3 = tanh(b2 + W2*z2);       # third layer (hidden)
y  =      b3 + W3*z3 ;       # output layer
r  = y-t;                    # compare to truth
E  = 0.5 * r' * r;           # squared error
```

Backward computation (calculate gradients):

```
b3E = r;
W3E = b3E * z3';
b2E = (1-z3.^2) .* (W3' * b3E)
W2E = b2E * z2';
b1E = (1-z2.^2) .* (W2' * b2E)
W1E = b1E * z1';
```

Multi-layer networks

Backward computation (calculate gradients):

```
b3E = r;
W3E = b3E * z3';
b2E = (1-z3.^2) .* (W3' * b3E)
W2E = b2E * z2';
b1E = (1-z2.^2) .* (W2' * b2E)
W1E = b1E * z1';
```

Gradient descent:

```
eta = 0.1;
W1  = W1 - eta * W1E;
b1  = b1 - eta * b1E;
W2  = W2 - eta * W2E;
b2  = b2 - eta * b2E;
W3  = W3 - eta * W3E;
b3  = b3 - eta * b3E;
```

What is back propagation?

- Consider simple neural network with scalars

$$y = f_3(w_3, f_2(w_2, f_1(w_1, x)))$$

  x and y are scalars, parameters w_3, w_2, w_1 are also scalars. We don't specify f_1, f_2, f_3.
- For a data point x with target value t we define the loss function

$$E = \tfrac{1}{2}(y - t)^2$$

- Goal: learn parameters that minimize the loss

$$\min_{w_3, w_2, w_1} E$$

- *Back-propagation* is an efficient way to calculate the derivatives of the the loss wrt the parameters:

$$\frac{\partial E}{\partial w_3} \qquad \frac{\partial E}{\partial w_2} \qquad \frac{\partial E}{\partial w_1}$$

Rewrite the network into layers

$$\begin{aligned} z_1 &= x && \text{input layer} \\ z_2 &= f_1(w_1, z_1) && \text{second layer (hidden)} \\ z_3 &= f_2(w_2, z_2) && \text{third layer (hidden)} \\ y = z_4 &= f_3(w_3, z_3) && \text{output layer} \\ E &= \tfrac{1}{2}(y - t)^2 && \text{squared error (the loss)} \end{aligned}$$

Applying chain rule

$$\begin{aligned} \frac{\partial E}{\partial w_3} &= \frac{\partial E}{\partial z_4} \frac{\partial z_4}{\partial w_3} \\ \frac{\partial E}{\partial w_2} &= \frac{\partial E}{\partial z_4} \frac{\partial z_4}{\partial z_3} \frac{\partial z_3}{\partial w_2} \\ \frac{\partial E}{\partial w_1} &= \frac{\partial E}{\partial z_4} \frac{\partial z_4}{\partial z_3} \frac{\partial z_3}{\partial z_2} \frac{\partial z_2}{\partial w_1} \end{aligned}$$

Propagation (feed forward)

$$\begin{tikzcd}
x = z_1 \arrow[r] & \boxed{f_1} \arrow[r] & z_2 \arrow[r] & \boxed{f_2} \arrow[r] & z_3 \arrow[r] & \boxed{f_3} \arrow[r] & z_4 = y \\
& w_1 \arrow[u] & & w_2 \arrow[u] & & w_3 \arrow[u] &
\end{tikzcd}$$

Back propagation

$$\begin{tikzcd}
\boxed{f_1} \arrow[d] & \frac{\partial E}{\partial z_2} \arrow[l] & \boxed{f_2} \arrow[l] \arrow[d] & \frac{\partial E}{\partial z_3} \arrow[l] & \boxed{f_3} \arrow[l] \arrow[d] & \frac{\partial E}{\partial z_4} \arrow[l] \\
\frac{\partial E}{\partial w_1} & & \frac{\partial E}{\partial w_2} & & \frac{\partial E}{\partial w_3} &
\end{tikzcd}$$

Back-propagation step by step

1. At the loss:
   - sends $\frac{\partial E}{\partial z_4}$ to f_3
2. At f_3:
   - receives $\frac{\partial E}{\partial z_4}$
   - calculates $\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial z_4} \frac{\partial z_4}{\partial w_3}$ with $\frac{\partial z_4}{\partial w_3} = \frac{\partial}{\partial w_3} f_3(w_3, z_3)$
   - updates $w_3 = w_3 - \eta \frac{\partial E}{\partial w_3}$
   - sends $\frac{\partial E}{\partial z_3} = \frac{\partial E}{\partial z_4} \frac{\partial z_4}{\partial z_3}$ to f_2 with $\frac{\partial z_4}{\partial z_3} = \frac{\partial}{\partial z_3} f_3(w_3, z_3)$
3. At f_2:
   - receives $\frac{\partial E}{\partial z_3}$
   - calculates $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z_3} \frac{\partial z_3}{\partial w_2}$ with $\frac{\partial z_3}{\partial w_2} = \frac{\partial}{\partial w_2} f_2(w_2, z_2)$
   - updates $w_2 = w_2 - \eta \frac{\partial E}{\partial w_2}$
   - sends $\frac{\partial E}{\partial z_2} = \frac{\partial E}{\partial z_3} \frac{\partial z_3}{\partial z_2}$ to f_1 with $\frac{\partial z_3}{\partial z_2} = \frac{\partial}{\partial z_2} f_2(w_2, z_2)$
4. At f_1:
   - receives $\frac{\partial E}{\partial z_2}$
   - calculates $\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial z_2} \frac{\partial z_2}{\partial w_1}$ with $\frac{\partial z_2}{\partial w_1} = \frac{\partial}{\partial w_1} f_1(w_1, z_1)$
   - updates $w_1 = w_1 - \eta \frac{\partial E}{\partial w_1}$

Notation!

- Scalars only (EASY)

$$\underbrace{\frac{\partial E}{\partial w}}_{\in \mathbb{R}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}} \underbrace{\frac{\partial y}{\partial w}}_{\in \mathbb{R}}$$

- Scalars and vectors (BE CAREFUL)

$$\underbrace{\frac{\partial E}{\partial x}}_{\in \mathbb{R}^{1\times n}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}^{1\times m}} \underbrace{\frac{\partial y}{\partial x}}_{\in \mathbb{R}^{m\times n}}$$

  where $E \in \mathbb{R}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$.

- Scalars, vectors, and matrices (DANGER!)

$$\underbrace{\frac{\partial E}{\partial W}}_{\in \mathbb{R}^{n\times m}} = \underbrace{\frac{\partial E}{\partial y}}_{\in \mathbb{R}^{1\times m}} \underbrace{\frac{\partial y}{\partial W}}_{\in \mathbb{R}^{m\times m\times n}}$$

  Here, good notation is difficult!

  where $E \in \mathbb{R}$, $W \in \mathbb{R}^{m\times n}$, $y \in \mathbb{R}^m$.