So you think you know backpropagation?

A little talk about the *most obvious* algorithm in deep learning

---

Nikolaj van Omme

February 21, 2018

Funartech

Table of contents

Basic ideas

Backpropagation: easy!

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$

end for

source: section 6.5, algorithm 6.4 in [4]

History

Reddit[1]: Dr. Schmidhuber, your know a lot about the history of neural networks. Who actually invented backpropagation?

> *You are handing that one to me on a silver platter! I even have a little web site on this.*
>
> *The continuous form of backpropagation was derived in the early 1960s (Bryson, 1961; Kelley, 1960; Bryson and Ho, 1969). Dreyfus (1962) published the elegant derivation based on the chain rule only. Back then, computers were ten billion times slower than today, and many researchers did not even have access to computers. The modern efficient version for discrete sparse networks was published in 1970 (Linnainmaa, 1970, 1976). Linnainmaa also published FORTRAN code. Dreyfus used backpropagation to adapt control parameters (1973). By 1980, automatic differentiation could derive backpropagation for any differentiable graph (Speelpenning, 1980). Werbos (1982) published the first application of backpropagation to neural networks (extending thoughts in his 1974 thesis). Computer experiments demonstrated that this can yield useful internal representations (Rumelhart et al.,1986). LeCun et al. (1989) applied backpropagation to Fukushimas convolutional architecture (1979). Reference details as well as many additional relevant citations can be found in Sec. 5.5. of the survey (Sec. 5.5.1 also has compact pseudo code for backpropagation in recurrent or feedforward weight-sharing NNs).*

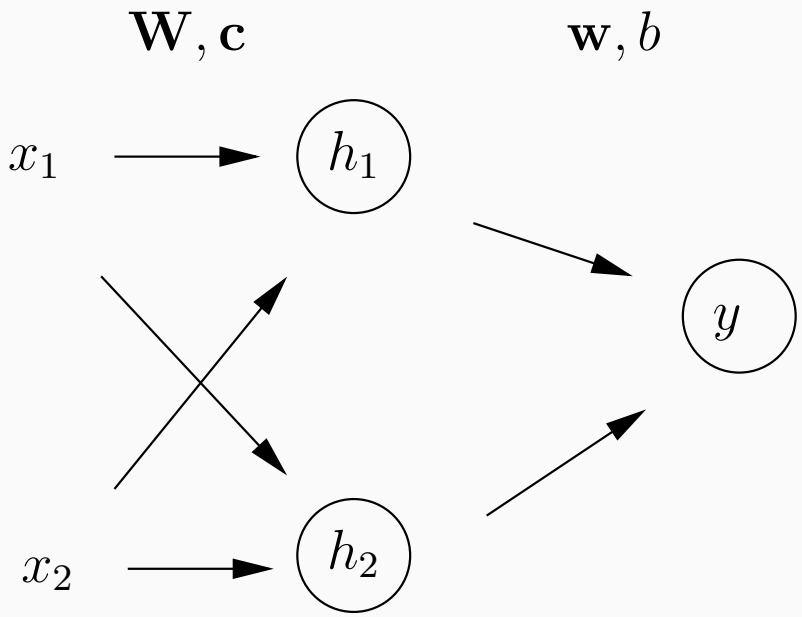
- Website: http://people.idsia.ch/ juergen/who-invented-backpropagation.html
- Paper: http://people.idsia.ch/ juergen/deep-learning-overview.html. See [12].

[1] https://www.reddit.com/r/MachineLearning/comments/2xcyrl/i_am_jügen_schmidhuber_ama/cp7adxn/

Feedforward Neural Networks: XOR I

Universal function approximator.

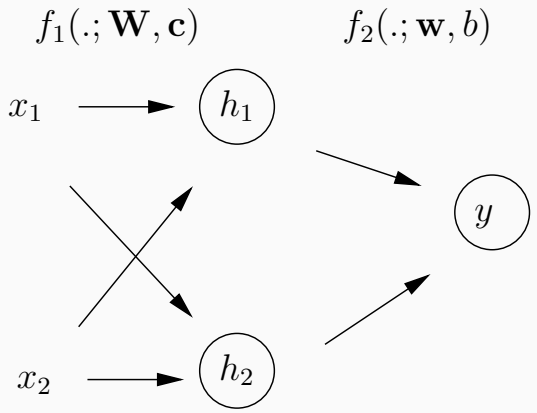
Example: XOR



x_1	x_1	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_1 \end{bmatrix} \xrightarrow{f} y = f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b)$$

Feedforward Neural Networks: XOR II



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_1 \end{bmatrix} \xrightarrow{f = f_2 \circ f_1} y = f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b)$$

$$f_1 : \mathbb{R}^2 \to \mathbb{R}^2 : \mathbf{x} = (x_1, x_2) \mapsto \max\left\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\right\}$$

$$f_2 : \mathbb{R}^2 \to \mathbb{R} : \mathbf{x} = (x_1, x_2) \mapsto \mathbf{w}^T\mathbf{x} + b$$

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

Feedforward Neural Networks: XOR training

Supervised learning:

$$\begin{aligned} \mathbf{x} &\mapsto \hat{y} \\ (0,0) &\mapsto 0 \\ (0,1) &\mapsto 1 \\ (0,1) &\mapsto 1 \\ (0,1) &\mapsto 1,1 \\ &\dots \end{aligned}$$

Loss function (MSE):

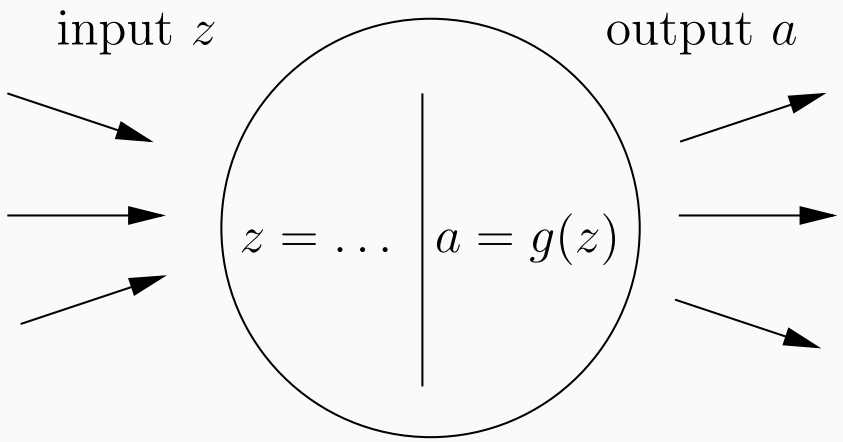
$$\mathbb{R} \ni L(y, \hat{y}) = \frac{1}{|\mathbb{X}|} \sum_{\mathbf{x} \in \mathbb{X}} \|y - \hat{y}\|_2^2$$

Optimization (training):

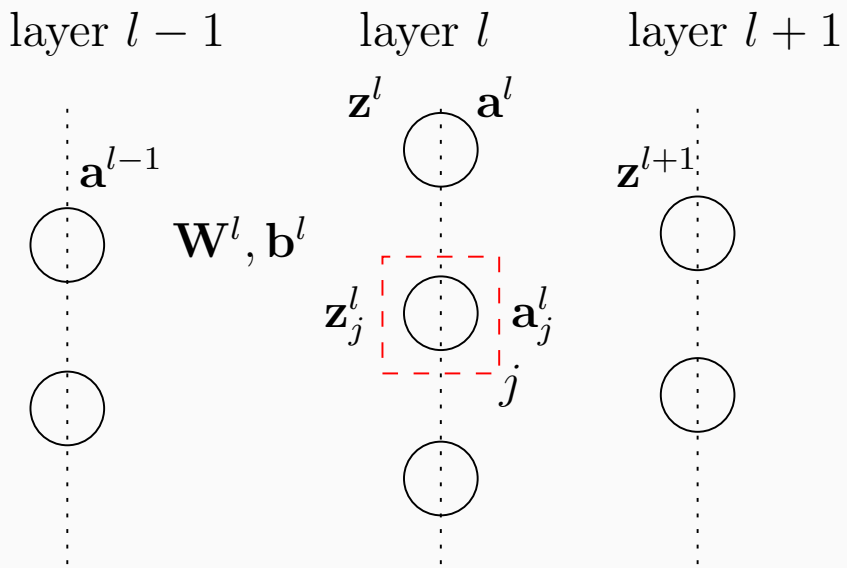
$$\mathbf{W}^*, \mathbf{c}^*, \mathbf{w}^*, \mathbf{b}^* = \underset{\mathbf{W}, \mathbf{c}, \mathbf{w}, \mathbf{b}}{\arg\min}\, L(y, \hat{y})$$

Little detour: the activation function I

- Linear regression: only $\mathbf{W}, \mathbf{b}$ (no activation function)
- XOR (and problems in general): not linear
  - need to introduce non-linearity
  - one possibility: non linear activation function g
- One neuron j:

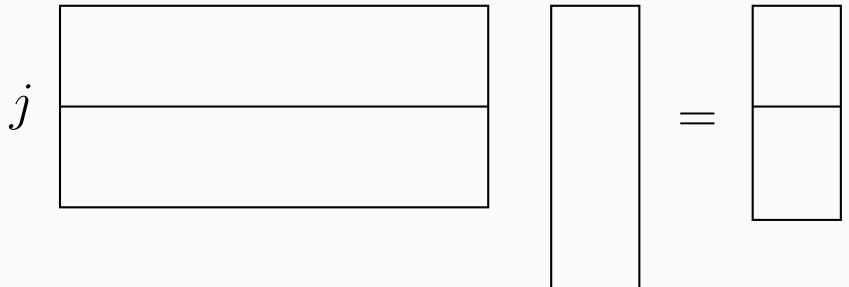


Little detour: the activation function II



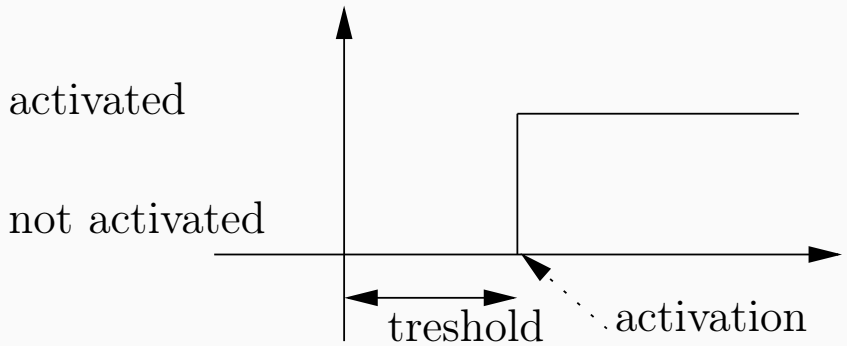
$$\mathbf{a}_j^l = g(\mathbf{z}_j^l)$$

$$\mathbf{z_j^l} = \left[\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l\right]_j = \sum_k \mathbf{W}_{jk}^l \mathbf{a}_k^{l-1} + \mathbf{b}_j^l$$



Little detour: the activation function III

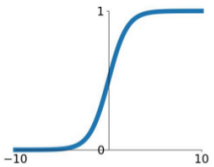
Idea: a neuron should only be activated after a certain **treshold**.



Activation Functions

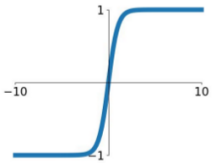
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



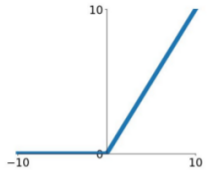
tanh

$\tanh(x)$



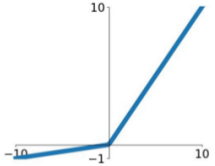
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

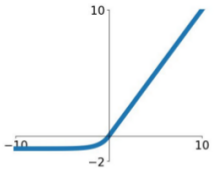


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



source: https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual-dc2a200f10f2

Model training

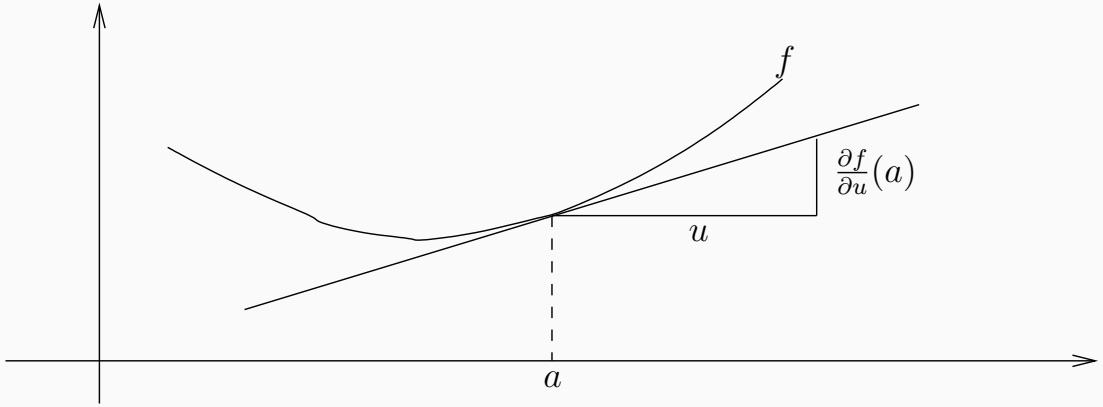
Optimization (training):

$$\mathbf{W}^*, \mathbf{c}^*, \mathbf{w}^*, \mathbf{b}^* = \underset{\mathbf{W},\mathbf{c},\mathbf{w},\mathbf{b}}{\arg\min}\, L(y, \hat{y})$$

How do we do that?

- **Gradient descent method**:
  - First order method
  - Local search
  - Converges towards a local optimum
  - Generic
  - Converges slowly "at the end"
  - Initialization: very important!
  - Really . . . bad but "best" method in town . . .
- **Backpropagation**:
  - Method to compute (pseudo) gradients
  - Generic
  - Quite fast (à la dynamic programming (memoization)) but memory hungry

Little detour: First order derivatives I



$$f : \mathbb{R} \to \mathbb{R} : x \mapsto f(x)$$

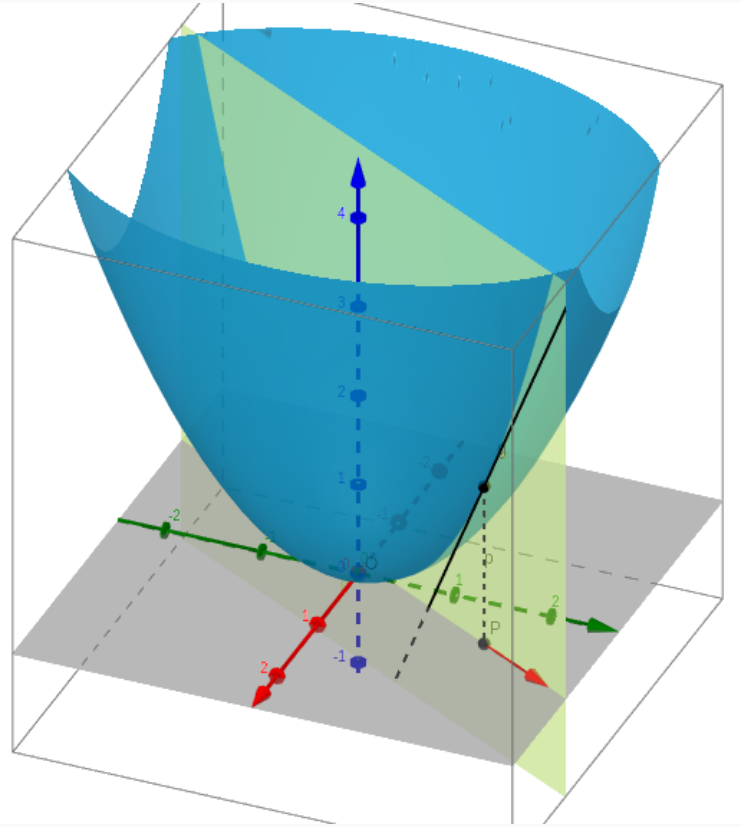
$$df|_{x=a} = df_a : \mathbb{R} \to \mathbb{R} : u \mapsto df_a(u) = \frac{\partial f}{\partial u}(a)$$

$$df_a(u) = f'(a)u$$

Best linear approximation:

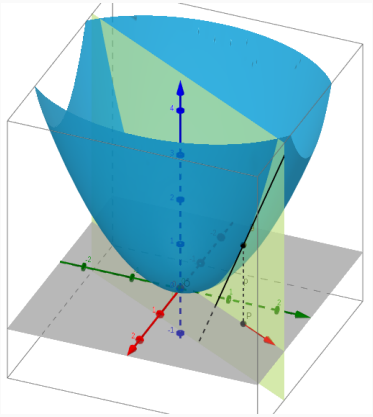
$$\lim_{u \mapsto 0} \frac{f(a+u) - f(a) - df_a(u)}{u} = 0$$

Little detour: First order derivatives II



Visualization: https://www.geogebra.org/m/Bx8nFMNc

Little detour: First order derivatives III



$$f : \mathbb{R}^2 \to \mathbb{R} : \mathbf{x} = (x_1, x_2) \mapsto f(x_1, x_2)$$

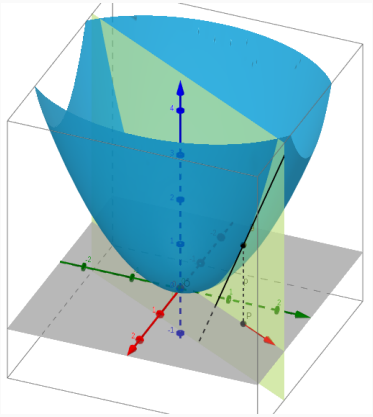
$$df|_{\mathbf{x}=\mathbf{a}} = df_{\mathbf{a}} : \mathbb{R}^2 \to \mathbb{R} : \mathbf{u} \mapsto df_{\mathbf{a}}(\mathbf{u}) = \frac{\partial f}{\partial \mathbf{u}}(\mathbf{a})$$

$$df_{\mathbf{a}}(\mathbf{u}) = \nabla f(\mathbf{a})^T \cdot \mathbf{u}$$

with:

$$\nabla f(\mathbf{a}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{a}) \\ \frac{\partial f}{\partial x_2}(\mathbf{a}) \end{bmatrix}$$

Little detour: First order derivatives IV



$$f : \mathbb{R}^2 \to \mathbb{R} : \mathbf{x} = (x_1, x_2) \mapsto f(x_1, x_2)$$

$$df|_{\mathbf{x}=\mathbf{a}} = df_{\mathbf{a}} : \mathbb{R}^2 \to \mathbb{R} : \mathbf{u} \mapsto df_{\mathbf{a}}(\mathbf{u}) = \frac{\partial f}{\partial \mathbf{u}}(\mathbf{a}) = \nabla f(\mathbf{a})^T \cdot \mathbf{u}$$

Best linear approximation:

$$\lim_{\alpha \mapsto 0} \frac{\|f(\mathbf{a} + \alpha\mathbf{u}) - f(\mathbf{a}) - df_{\mathbf{a}}(\alpha\mathbf{u})\|}{\alpha \|\mathbf{u}\|} = 0$$

$$f(\mathbf{a} + \mathbf{u}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a})^T \cdot \mathbf{u}$$

Gradient descent: main idea

At point $\mathbf{a}$, find a direction $\mathbf{u}$ (with $\|u\| = 1$) s.t.

$$f(\mathbf{a} + \mathbf{u}) < f(\mathbf{a})$$

$$f(\mathbf{a} + \mathbf{u}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a})^T \cdot \mathbf{u}$$

$$\nabla f(\mathbf{a})^T \cdot \mathbf{u} < 0$$

$$\min_{\mathbf{u}} \nabla f(\mathbf{a})^T \cdot \mathbf{u} = \min_{\mathbf{u}} \|u\| \, \|\nabla f(\mathbf{a})\| \cos\theta = \min_{\theta} \cos\theta$$

$$\mathbf{u}^* = -\nabla f(\mathbf{a})$$

Algo:

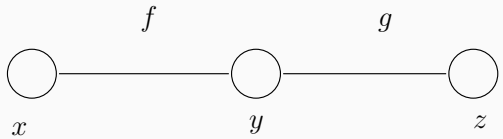
$$\mathbf{x}' = \mathbf{x} - \alpha \nabla f(\mathbf{a})$$

Gradient descent: main ideas

Visualization: http://web.mit.edu/jorloff/www/jmoapplets/ddm-mathinsight/ddm-jmo.html

Little detour: The Chaine Rule and Paths I

Basic composition:



$$g : \mathbb{R} \to \mathbb{R} : y \mapsto g(y)$$

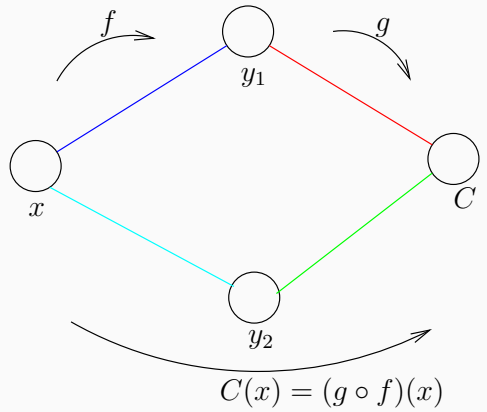
$$f : \mathbb{R} \to \mathbb{R} : x \mapsto f(x)$$

$$C : \mathbb{R} \to \mathbb{R} : x \mapsto C(x) = (g \circ f)(x)$$

$$\left.\frac{dC}{dx}\right|_{x=x_0} = \left.\frac{dg}{dy}\right|_{y=f(x_0)} \left.\frac{df}{dx}\right|_{x=x_0}$$

Little detour: The Chaine Rule and Paths II

Basic composition:



$$g : \mathbb{R}^2 \to \mathbb{R} : \mathbf{y} = (y_1, y_2) \mapsto g(y_1, y_2)$$

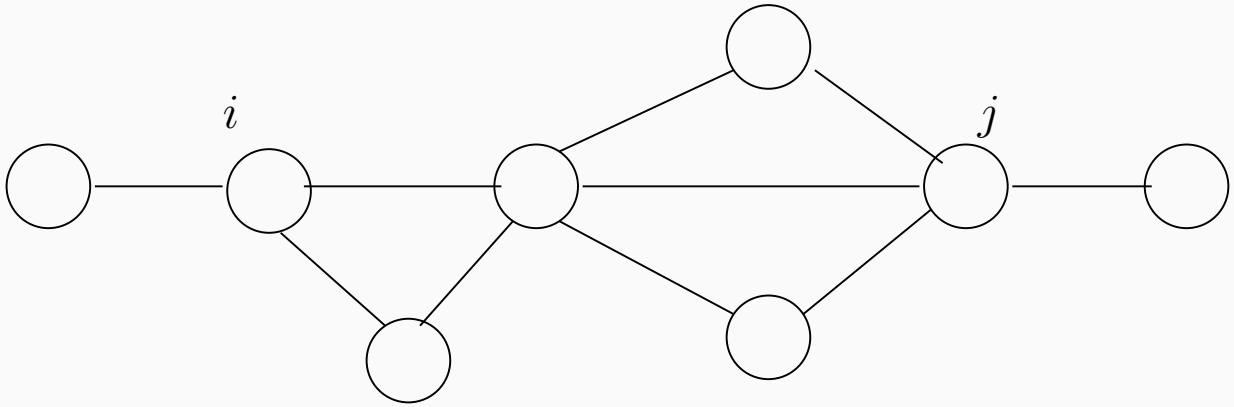
$$f : \mathbb{R} \to \mathbb{R}^2 : x \mapsto \mathbf{f}(x) = (f_1(x), f_2(x)) = (y_1, y_2)$$

$$C : \mathbb{R} \to \mathbb{R} : x \mapsto C(x) = (g \circ f)(x)$$

$$\left.\frac{dC}{dx}\right|_{x=x_0} = \left.\frac{\partial g}{\partial y_1}\right|_{y_1=f_1(x_0)} \left.\frac{\partial f_1}{\partial x}\right|_{x=x_0} + \left.\frac{\partial g}{\partial y_2}\right|_{y_2=f_2(x_0)} \left.\frac{\partial f_2}{\partial x}\right|_{x=x_0}$$

Little detour: The Chaine Rule and Paths III

Full composition:



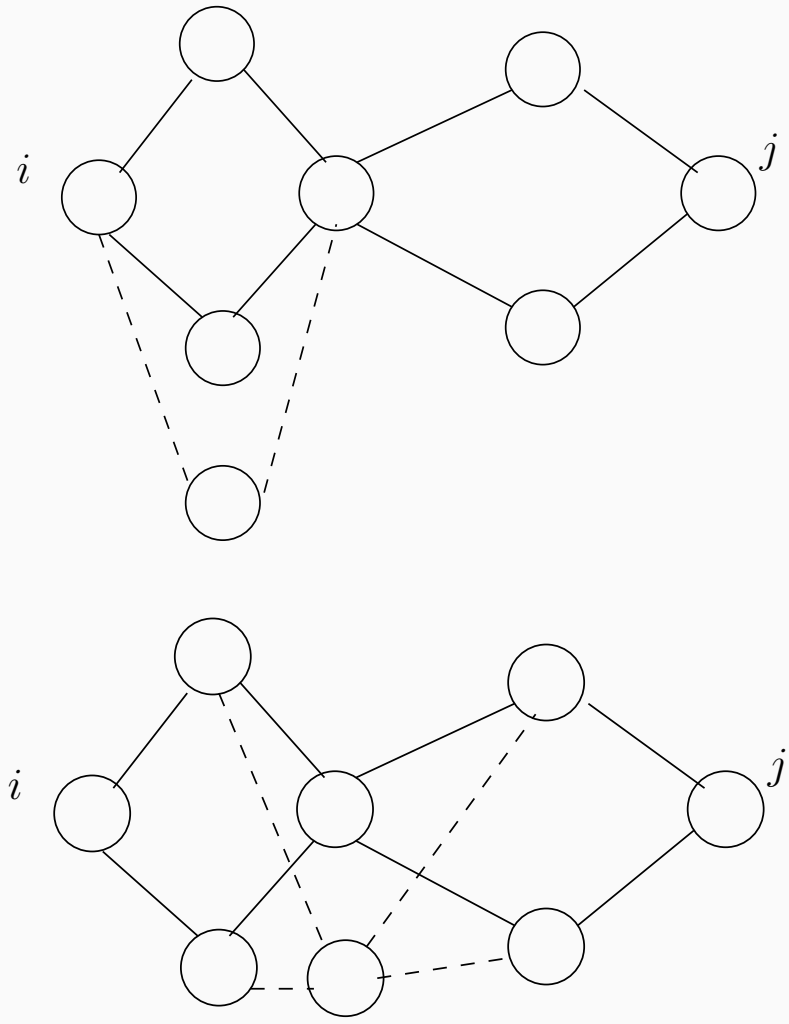
$$\frac{\partial y_i}{\partial x_i} = \sum_{[i \to j]} \prod_{(k,l) \in [i \to j]} c_{l,k}$$

Observations:

1. Lots of derivations are the same;
2. Lots of subpaths are the same;

Little detour: The Chaine Rule and Paths IV

Network: what happens if you add a node?



Backpropagation: equations

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation: algo

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

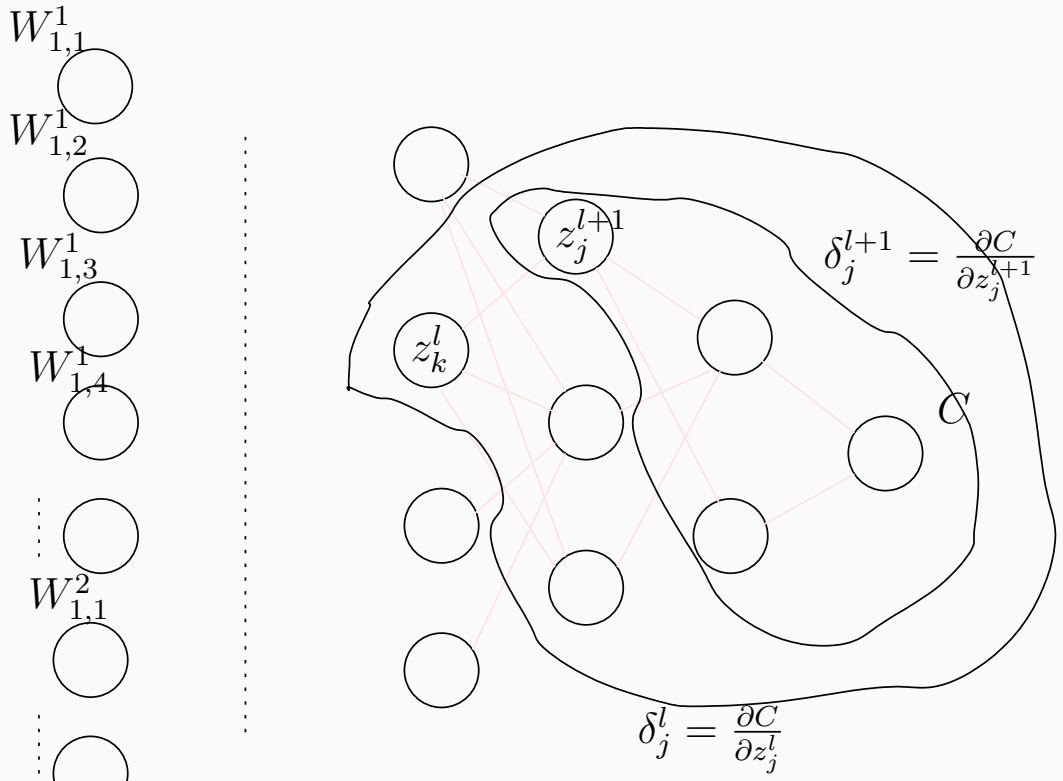
3. **Output error** δ^L**:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation: intuition



l-DAG

$$\delta^l = F(\delta^{l+1})$$

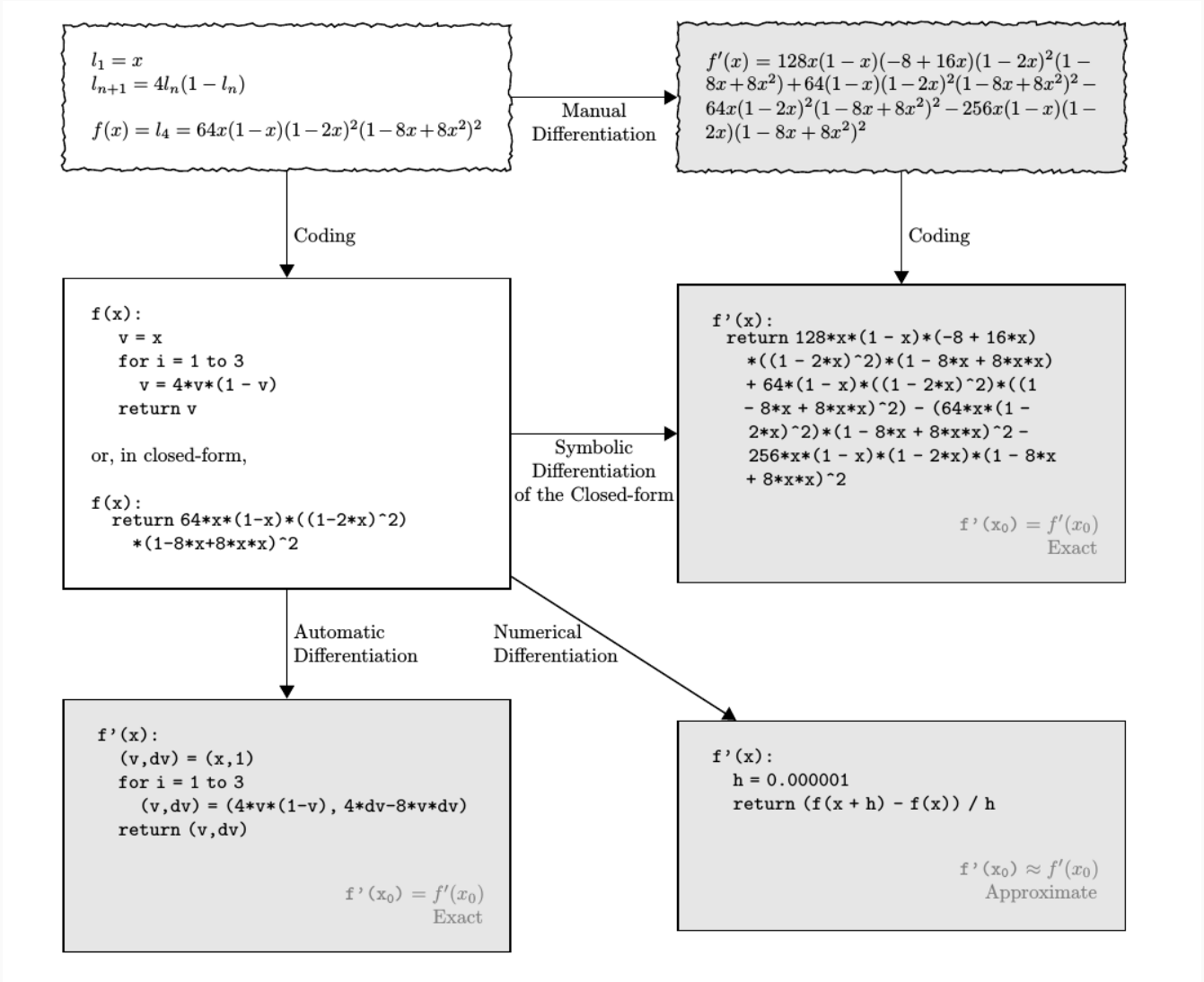
Variationen

Little detour: Automatic differentiation

Automatic differentiation

- $\neq$ numerical differentiation
- $\neq$ symbolic differentiation
- is a combination (uses) numerical and symbolic differentiations but is a lot more!

Little detour: Automatic differentiation



From [1]: *Automatic differentiation in machine learning: a survey*, 2015.

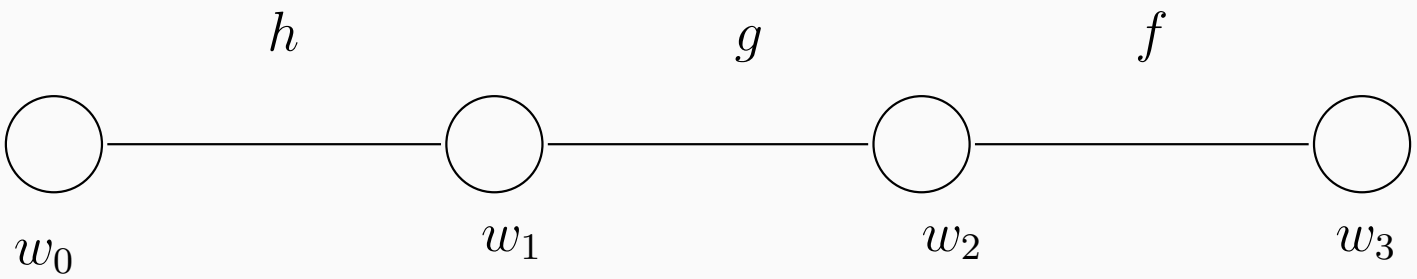
Forward and backward modes

Example:

$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx}$$

1. **forward accumulation** computes the recursive relation: $\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}}\frac{dw_{i-1}}{dx}$ with $w_3 = y$, and,
2. **reverse accumulation** computes the recursive relation: $\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}}\frac{dw_{i+1}}{dw_i}$ with $w_0 = x$.

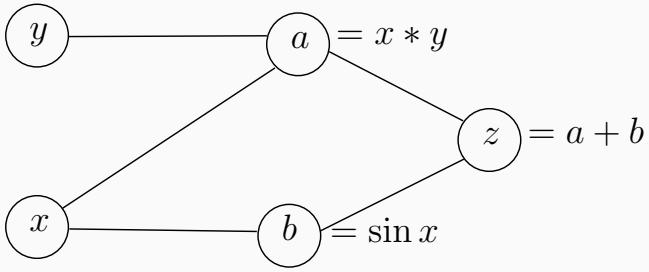


From https://en.wikipedia.org/wiki/Automatic_differentiation

Forward and backward modes: example I

https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation

Forward and backward modes: example II



$$\frac{\partial s}{\partial z} = ? \tag{1}$$

$$\frac{\partial s}{\partial b} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial s}{\partial z}1 \tag{2}$$

$$\frac{\partial s}{\partial a} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial a} = \frac{\partial s}{\partial z}1 \tag{3}$$

$$\frac{\partial s}{\partial y} = \frac{\partial s}{\partial a}\frac{\partial a}{\partial y} = \frac{\partial s}{\partial a}x \tag{4}$$

$$\frac{\partial s}{\partial x} = \frac{\partial s}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial s}{\partial b}\frac{\partial b}{\partial x} \tag{5}$$

$$= \frac{\partial s}{\partial a}y + \frac{\partial s}{\partial b}\cos x \tag{6}$$

Forward and backward modes: computational costs

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

- forward: $O(n \cdot \text{cost}(f))$
- backward: $O(m \cdot \text{cost}(f))$ but ...
  - we need to keep the intermediate results for each node ($O(V)$)
  - we need to keep track of the function in reverse mode (Wengert list)

Little detour: the Optimal Jacobian Accumulation problem I

Definition:

minimizing the number of scalar multiplications and addition[2]
performed by a Jacobian accumulation code

We are interested in minimizing the number of *elementary* operations to compute the gradients.

This problem is intractable:

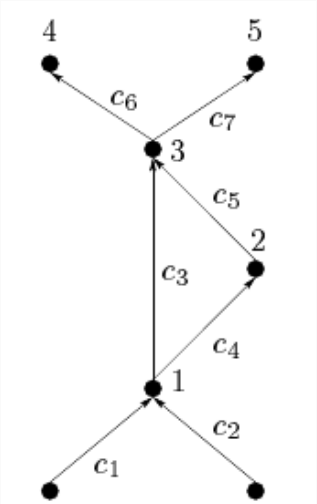
Optimal Jacobian accumulation is NP-complete

2008, Naumann in [10]

[2] **fma**: *Fused multiply-add operations:* $c_k + c_j c_i$.

Little detour: the Optimal Jacobian Accumulation problem II

Example:



$$v_1 = v_{-1}v_0;\ v_2 = \sin(v_1);\ v_3 = v_1v_2;\ v_4 = \cos(v_3);\ v_5 = \exp(v_3)$$

$$F' = \begin{pmatrix} c_1c_3c_6 + c_1c_4c_5c_6 & c_2c_3c_6 + c_2c_4c_5c_6 \\ c_1c_3c_7 + c_1c_4c_5c_7 & c_2c_3c_7 + c_2c_4c_5c_7 \end{pmatrix}$$

$$F' = \begin{pmatrix} c_6s & c_6t \\ c_7s & c_7t \end{pmatrix}$$

$$r = c_3 + c_5c_4,\ s = rc_1,\ \text{and}\ t = rc_2$$

From [9]: Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph, 2004.

Little detour: Gradient Checking I

In a now famous post from Justin Domke dated February 17, 2009 and titled[3]:

> *Automatic Differentiation: The most criminally underused tool in the potential machine learning toolbox?*

he stated:

> *A lot of papers in machine learning (including many papers I like) go like this:*
>
> 1. *Invent a new type of model or a new loss function*
> 2. *Manually crank out the derivatives. (The main technical difficulty of the paper.)*
> 3. *Learn by plugging the derivatives into an optimization procedure. (Usually L-BFGS or stochastic gradient.)*
> 4. *Experimental results.*

[3] https://justindomke.wordpress.com/2009/02/17/automatic-differentiation-the-most-criminally-underused-tool-in-the-potential-machine-learning-toolbox/.

Little detour: Gradient Checking II

Question:

How did researchers become confident about their gradient computations before the use of automatic differentiation?

Anwer:

$$\frac{\partial L}{\partial x_i}(\mathbf{a}) = \lim_{\varepsilon \mapsto 0} \frac{L(\mathbf{a} + \varepsilon \mathbf{e}_i) - L(\mathbf{a} - \varepsilon \mathbf{e}_i)}{2\varepsilon}$$

Improve derivation

Sigmoid:

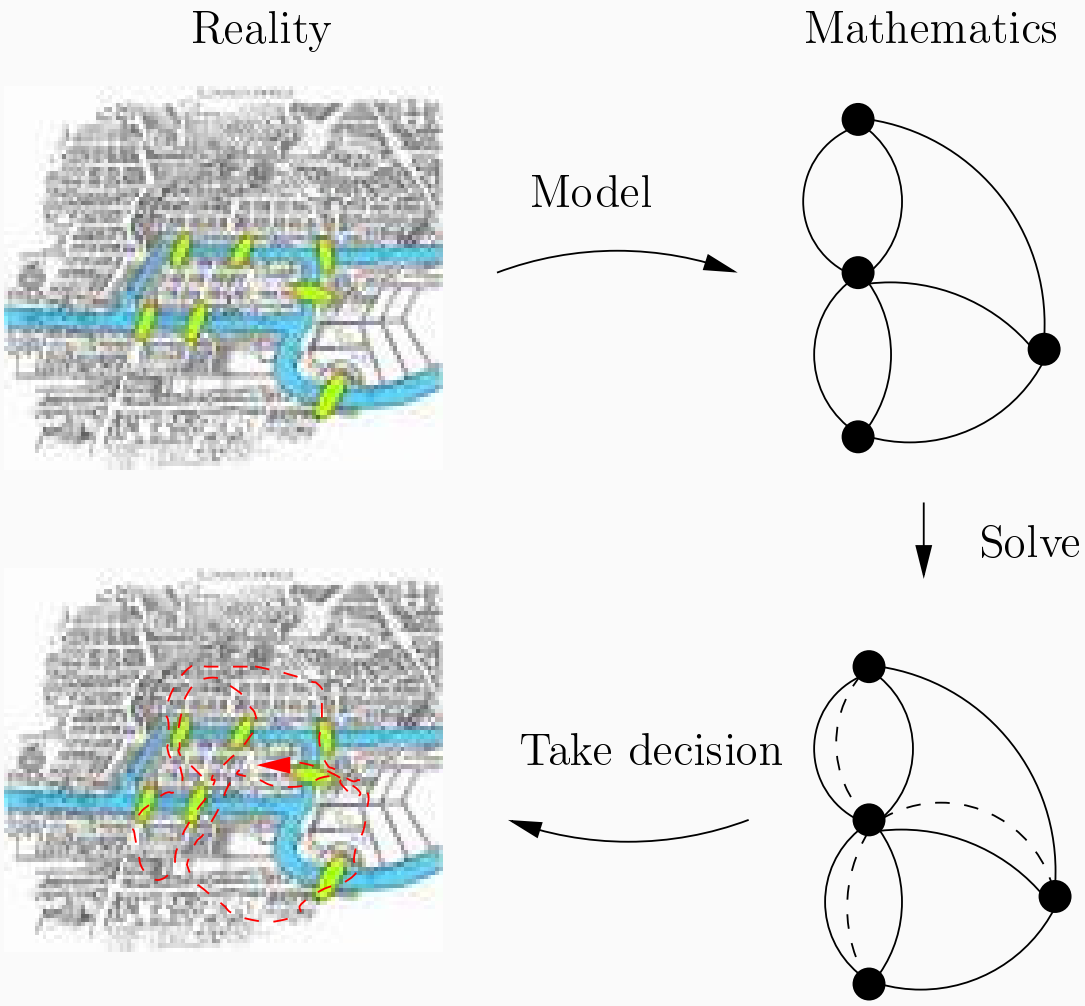
$$g(z) = \frac{1}{1 - e^{-z}} \tag{7}$$

Derivative:

$$\begin{aligned} \frac{df(z)}{dz}(z) &= \frac{0 \cdot (1 - e^{-z}) - (-e^{-z})}{(1 + e^{-z})^2} \\ &= \frac{1}{1 + e^{-z}} \left(\frac{e^{-z}}{1 + e^{-z}} \right) \\ &= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right) \\ &= f(z)(1 - f(z)) \end{aligned} \tag{8}$$

Questions?

What is Operations Research?



Operations Research: Results

- 30% improvements
- 0,1% = ...
- Extensively used in very competitive sectors:
  - Transportation (air, sea, land)
  - Internet
  - Army
  - Finance
  - ...
- Heavy tendancy: "soon" you will be outcompeted by your competitors if you don't use OR...
- Hot: Machine Learning "revolution":
  - Watch out for the OR revolution!
  - Gartner predicts 2018 will be the year of Operations Research!

Operations Research and Machine Learning

1000 feet view:

- **Machine Learning** (aka curve fitting)
  What are my points?
- **Operations Research** (aka space search)
  What are my best points?

OR + ML: example 1

Detecting faulty tracks:

- **Problem:** You are a train company and you own kilometers of tracks that you need to repair from time to time (**before** an accident). You want to do this for a minimum cost. How do you do this?
- **Solution:**
  1. You take photographs in quick succession of the tracks and with machine learning you detect possible defectuosities;
  2. You define (optimal) repair/replacement policies with operations research.

OR + ML: example 2

Construction team scheduling on (dangerous) construction sites:

- **Problem:** You are a construction company with hunderds of construction sites. You want to schedule your teams accross them and be sure that the right persons are going to the right construction sites and do exactly as told. How do you do this?
- **Solution:**
  1. You put a gate on each construction site and with facial recognition software (machine learning) you only allow the right people in and track their whereabouts so you know exactly where and when they are.
  2. Once you have the information about where your teams are operating, you can schedule them to do some tasks in an optimal way (operations research).

References i


A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind.
Automatic differentiation in machine learning: a survey.
arXiv preprint arXiv:1502.05767, 2015.


T. Chen, B. Xu, C. Zhang, and C. Guestrin.
Training deep nets with sublinear memory cost.
arXiv preprint arXiv:1604.06174, 2016.


W. M. Czarnecki, G. Swirszcz, M. Jaderberg, S. Osindero,
O. Vinyals, and K. Kavukcuoglu.
Understanding synthetic gradients and decoupled neural interfaces.
CoRR, abs/1703.00522, 2017.

References ii

I. Goodfellow, Y. Bengio, and A. Courville.
Deep Learning.
MIT Press, 2016.
`http://www.deeplearningbook.org`.

A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves.
Memory-efficient backpropagation through time.
In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4125–4133. Curran Associates, Inc., 2016.

References iii


J. M. Hernandez-Lobato and R. Adams.
Probabilistic backpropagation for scalable learning of bayesian neural networks.
In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1861–1869, Lille, France, 07–09 Jul 2015. PMLR.


M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, and K. Kavukcuoglu.
Decoupled neural interfaces using synthetic gradients.
CoRR, abs/1608.05343, 2016.

References iv


J. Martens and I. Sutskever.
Training deep and recurrent networks with hessian-free optimization.
In *Neural networks: Tricks of the trade*, pages 479–535. Springer, 2012.


U. Naumann.
Optimal accumulation of jacobian matrices by elimination methods on the dual computational graph.
Mathematical Programming, 99(3):399–421, 2004.


U. Naumann.
Optimal jacobian accumulation is np-complete.
Mathematical Programming, 112(2):427–441, 2008.

References v

D. E. Rumelhart, G. E. Hinton, and R. J. Williams.
Learning representations by back-propagating errors.
nature, 323(6088):533, 1986.

J. Schmidhuber.
Deep learning in neural networks: An overview.
Neural Networks, 61:85–117, 2015.
Published online 2014; based on TR arXiv:1404.7828 [cs.NE].