ENM 3600: Data-driven Modeling and Probabilistic Scientific Computing

Lecture #12: Multi-layer perceptrons

University of Pennsylvania

Feed-forward neural networks

Pros:

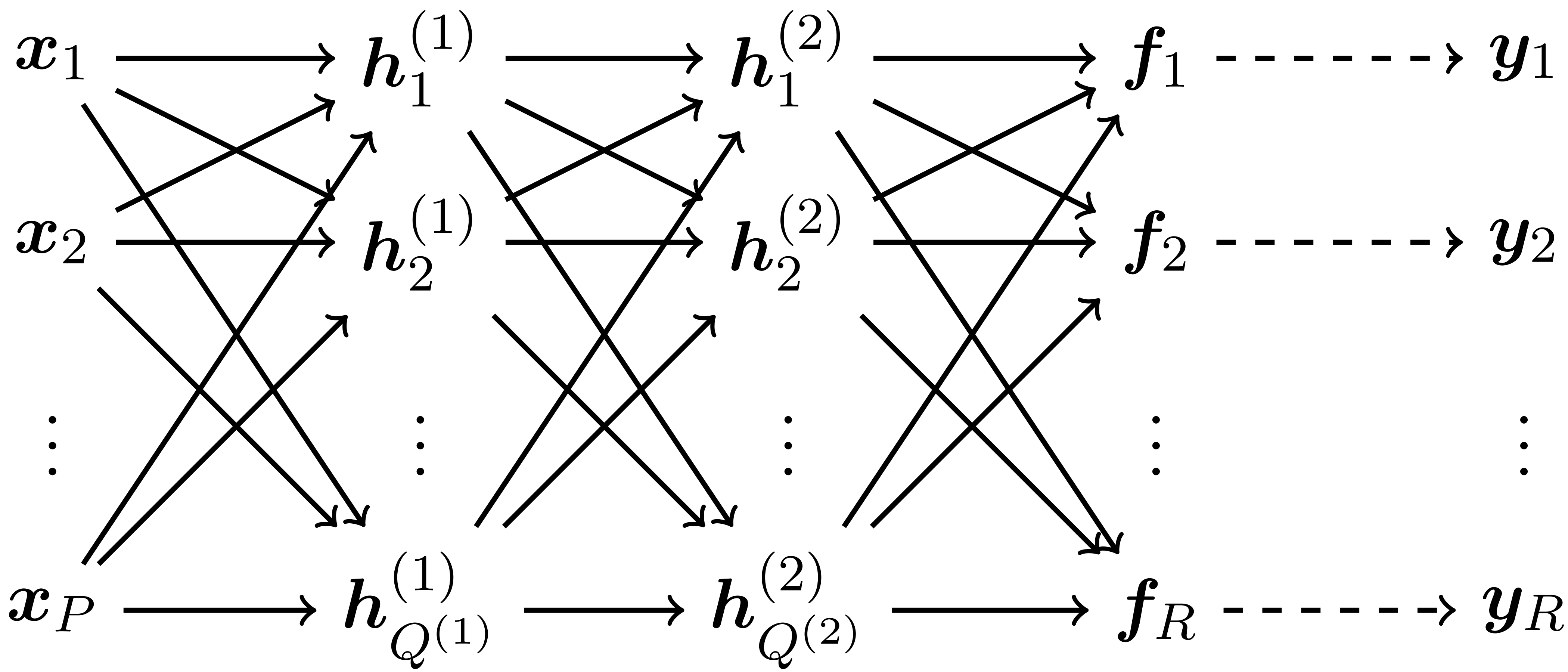
- Adaptive features/basis functions (parametric)
- Flexible non-linear regression models that can approximate any function.
- Scalability to high dimensions.

Cons:

- The likelihood function is no longer a convex function of the model parameters.
- Over-fitting in data-scarce scenarios.
- Results are hard to interpret.

University of Illinois at Urbana-Champaign. Center for Supercomputing Research and Development, & Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient.

Feed-forward neural networks



Universal approximation theorem

Theorem 1. *Let σ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^{\mathsf{T}} x + \theta_j) \tag{2}$$

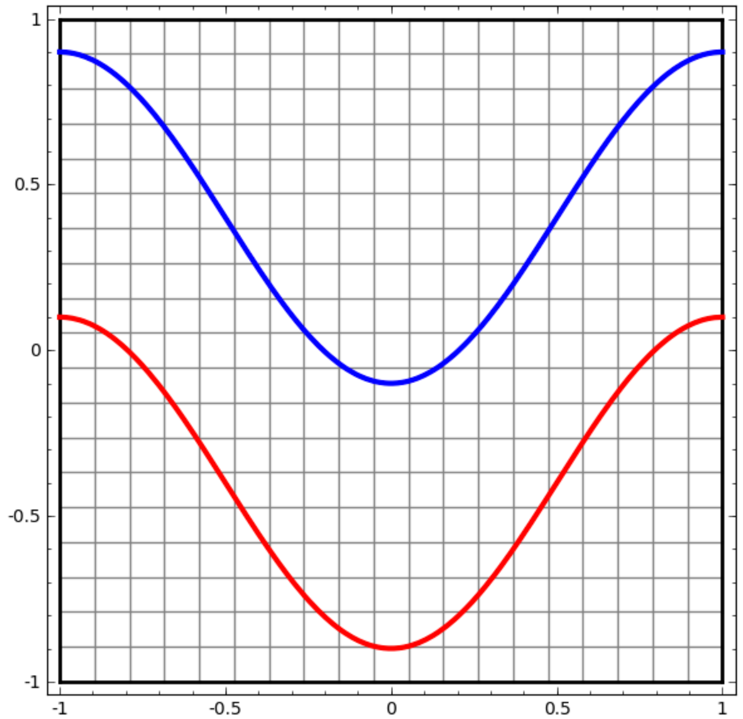
are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \textit{for all} \quad x \in I_n.$$

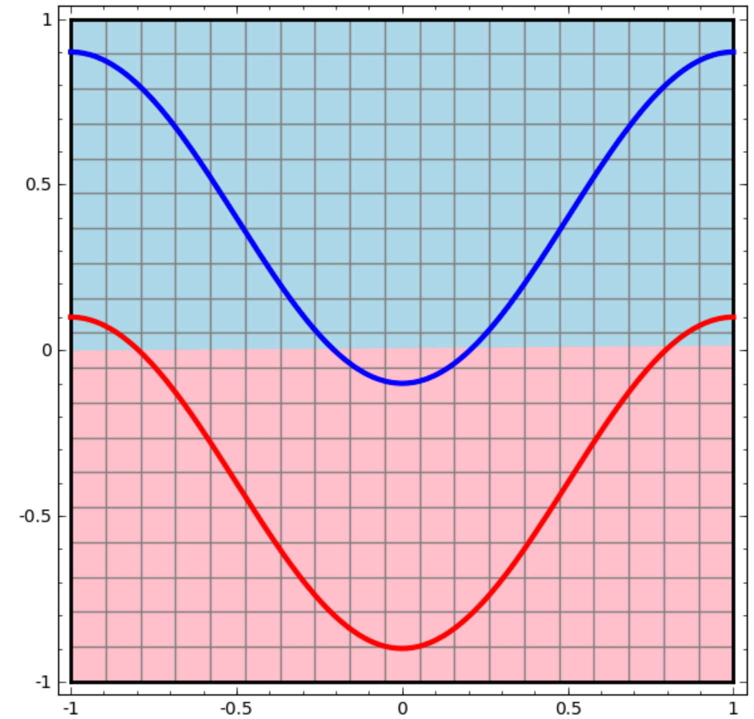
Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

Some intuition

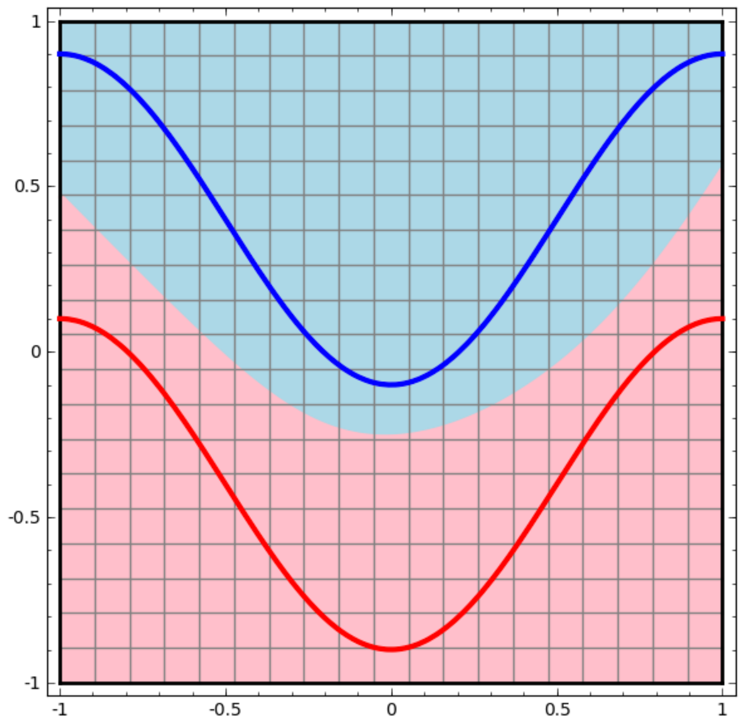
Data to classify



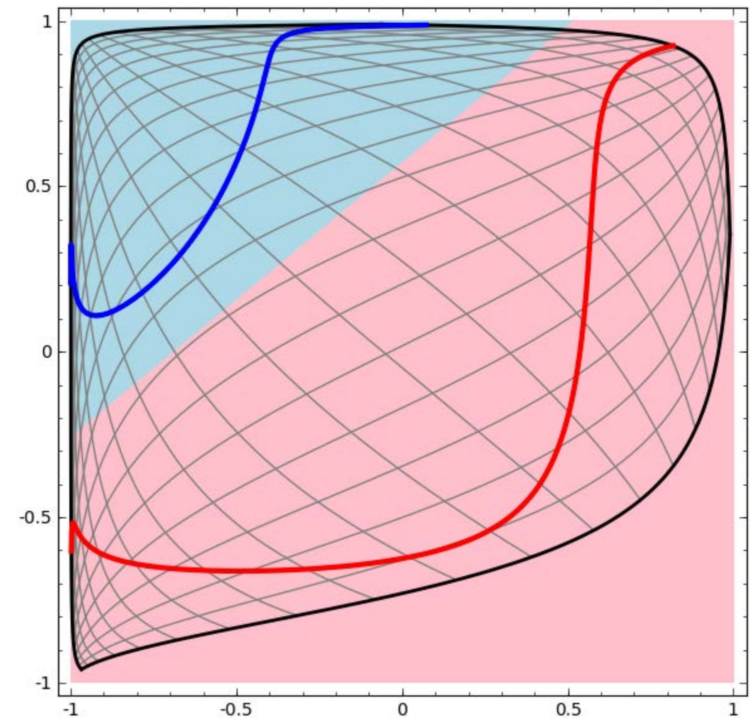
One input, one output layer



One input, one hidden, one output layer

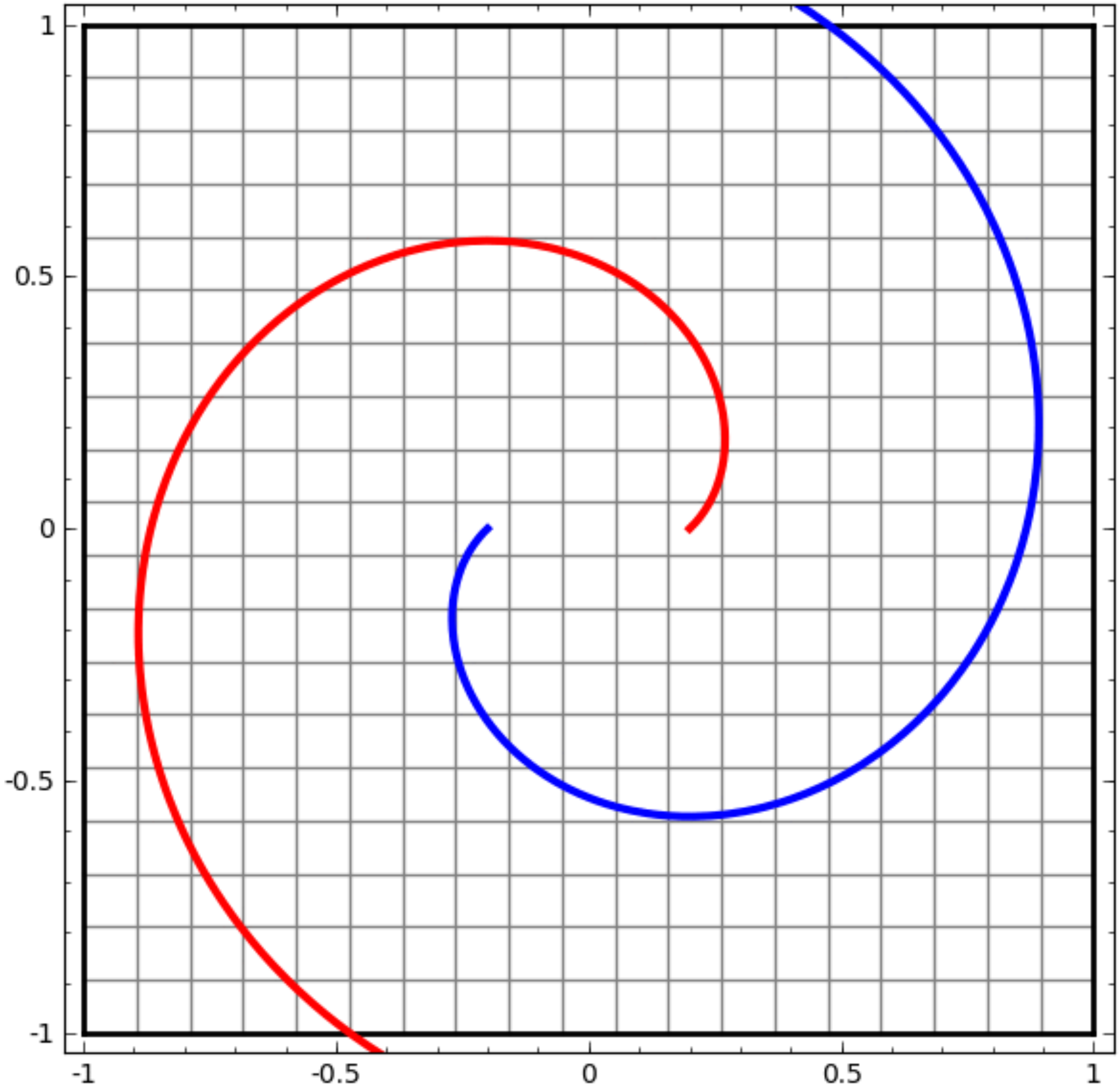


Visualizing the hidden layer



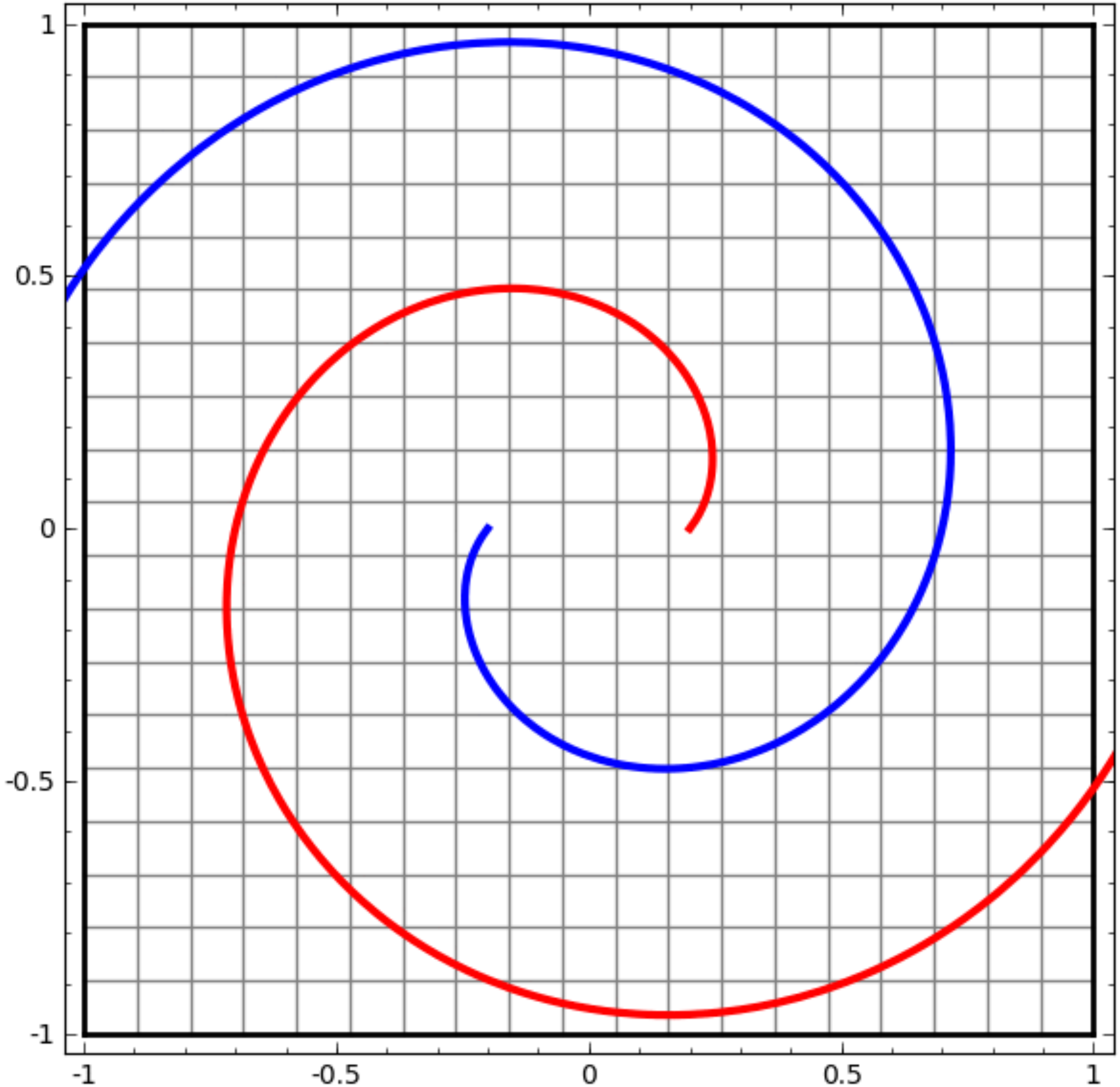
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Some intuition



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Some intuition



http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

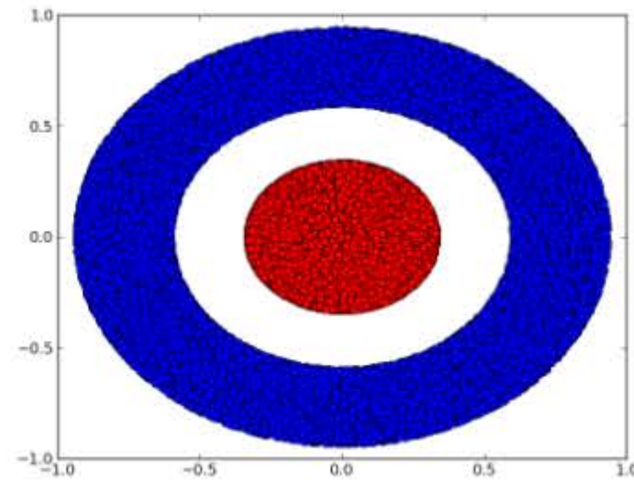
Some intuition

Topology and Classification

Consider a two dimensional dataset with two classes $A, B \subset \mathbb{R}^2$:

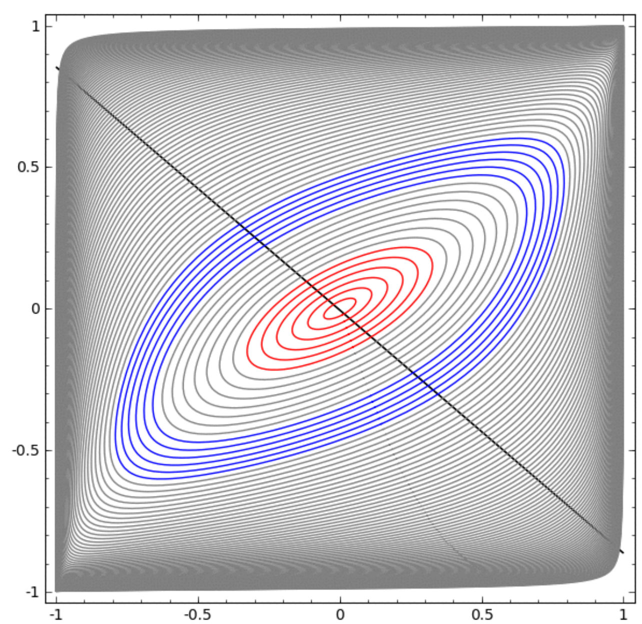
$$A = \{x | d(x, 0) < 1/3\}$$

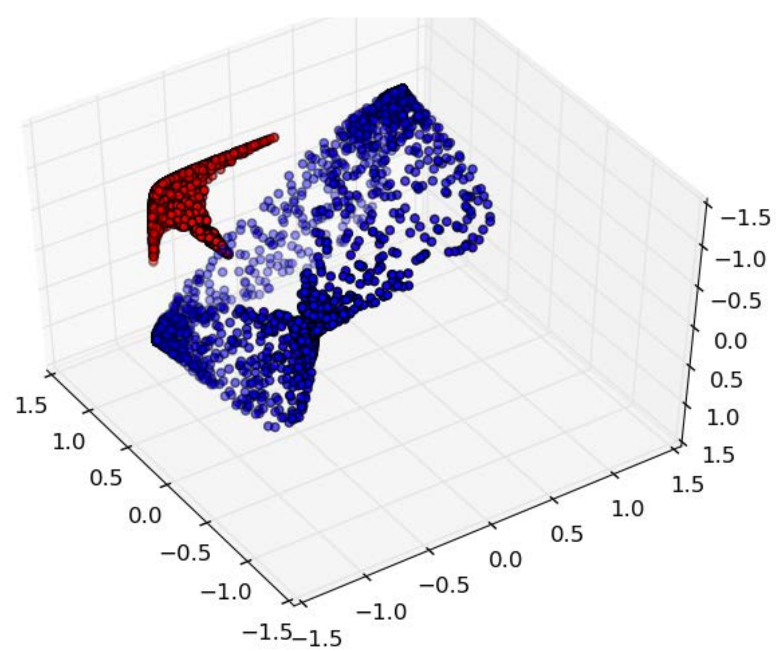
$$B = \{x | 2/3 < d(x, 0) < 1\}$$



A is red, B is blue

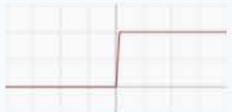
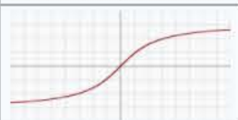
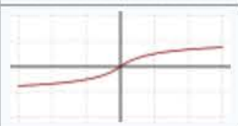
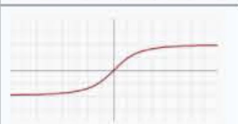
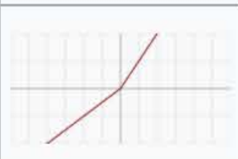
Claim: It is impossible for a neural network to classify this dataset without having a layer that has 3 or more hidden units, regardless of depth.





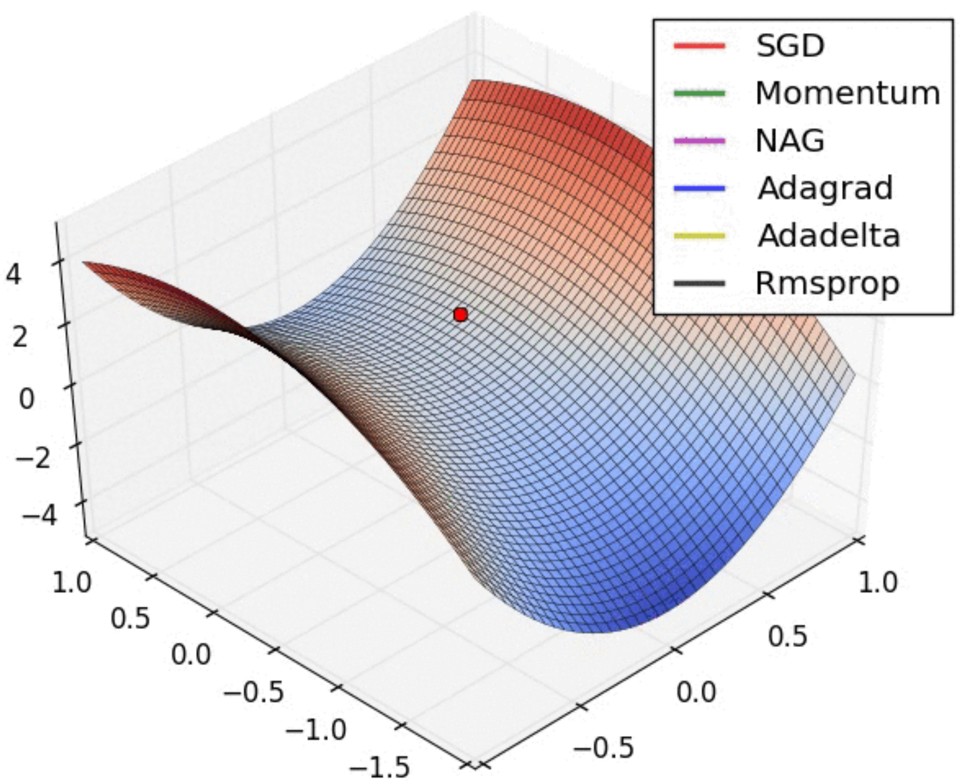
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

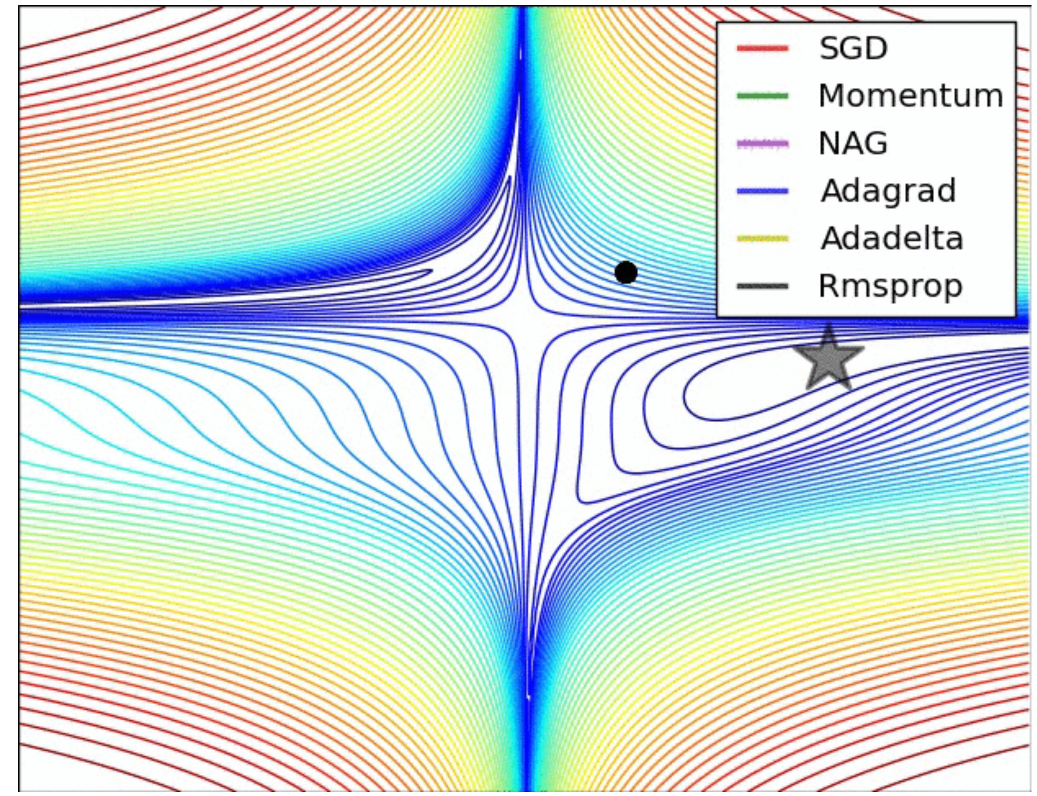
Activation functions

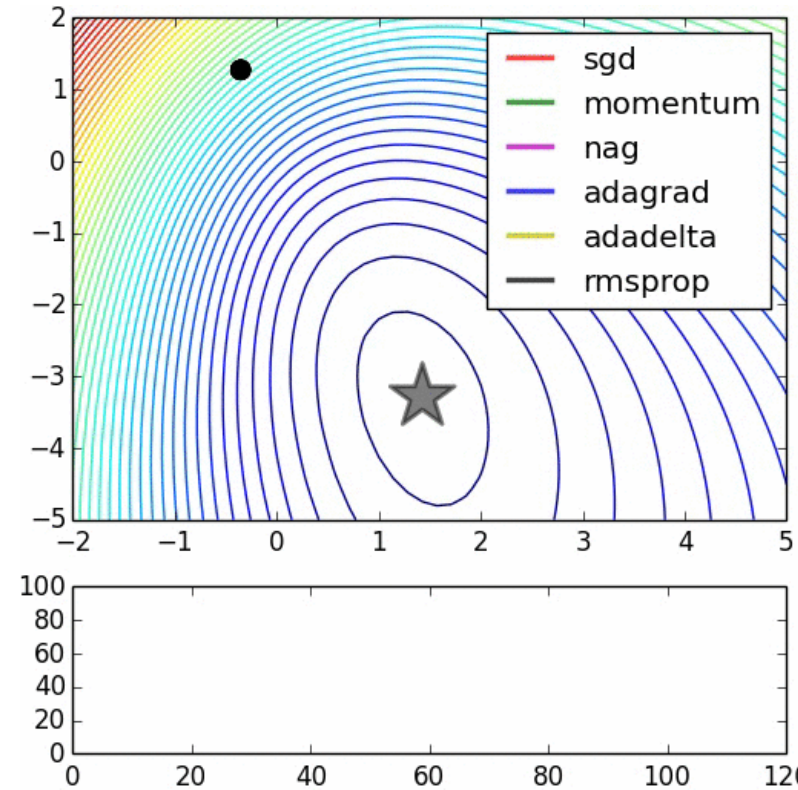
Name	Plot	Equation	Derivative (with respect to *x*)	Range	Order of continuity	Monotonic	Derivative Monotonic	Approximates identity near the origin
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No	No
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No	No
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes	No	Yes
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No	Yes
Softsign [7][8]		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$f'(x) = \frac{1}{(1 + \lvert x \rvert)^2}$	$(-1, 1)$	C^1	Yes	No	Yes
Inverse square root unit (ISRU)[9]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$	C^∞	Yes	No	Yes
Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes	No
Leaky rectified linear unit (Leaky ReLU)[11]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes iff $\alpha \geq 0$	Yes	Yes iff $\alpha = 1$

⋮

Modern SGD variants







http://ruder.io/optimizing-gradient-descent/

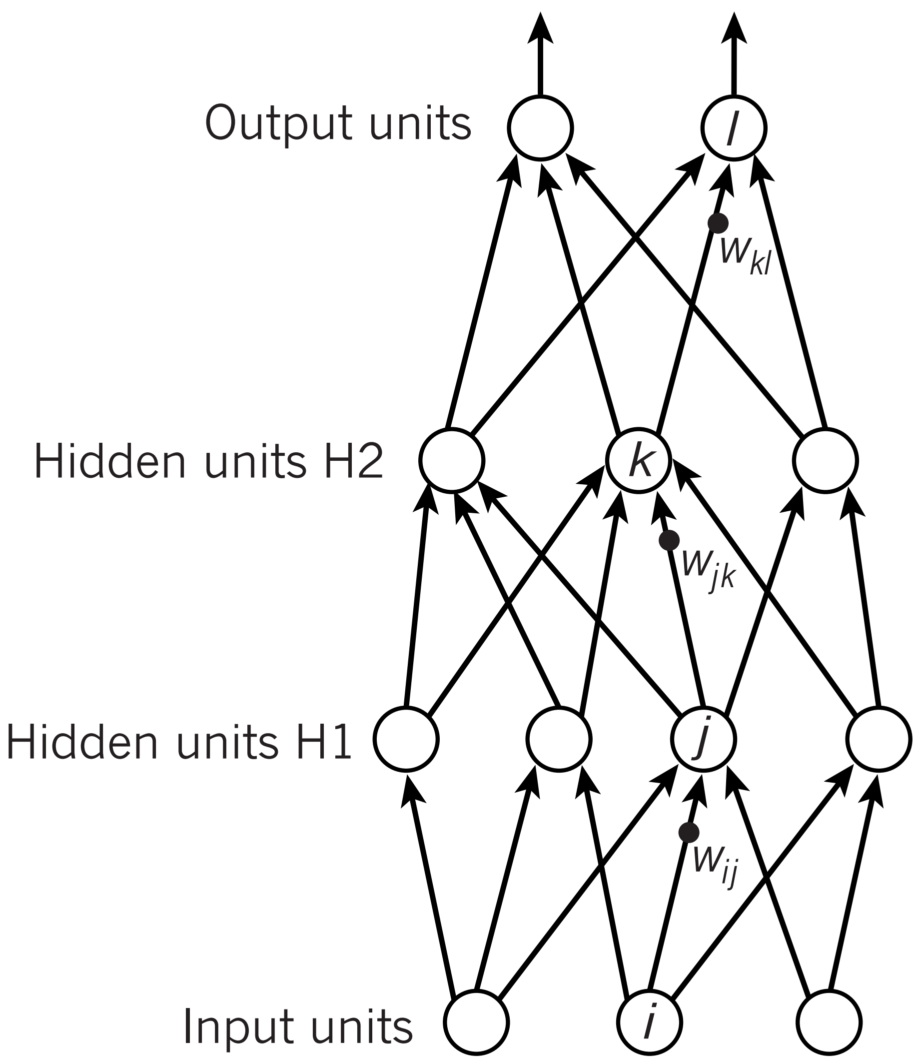
https://distill.pub/2017/momentum/

http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/

*animation credit: Alec Redford

Back-propagation

Forward pass



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, \mathrm{H2}} w_{kl}\, y_k$$

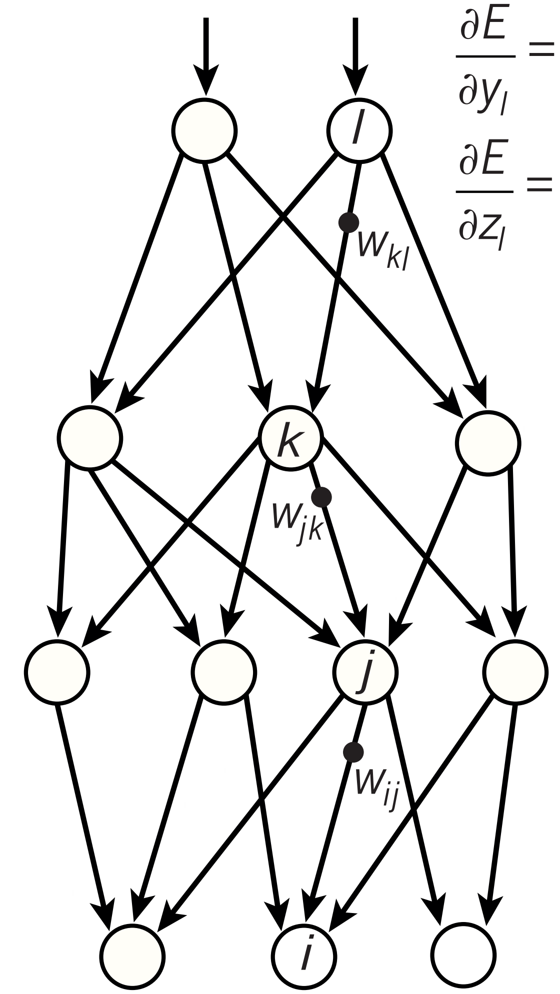
$$y_k = f(z_k)$$

$$z_k = \sum_{j \,\varepsilon\, \mathrm{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \mathrm{Input}} w_{ij}\, x_i$$

Backward pass



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \,\varepsilon\, \mathrm{out}} w_{kl}\frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \,\varepsilon\, \mathrm{H2}} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

The forward pass

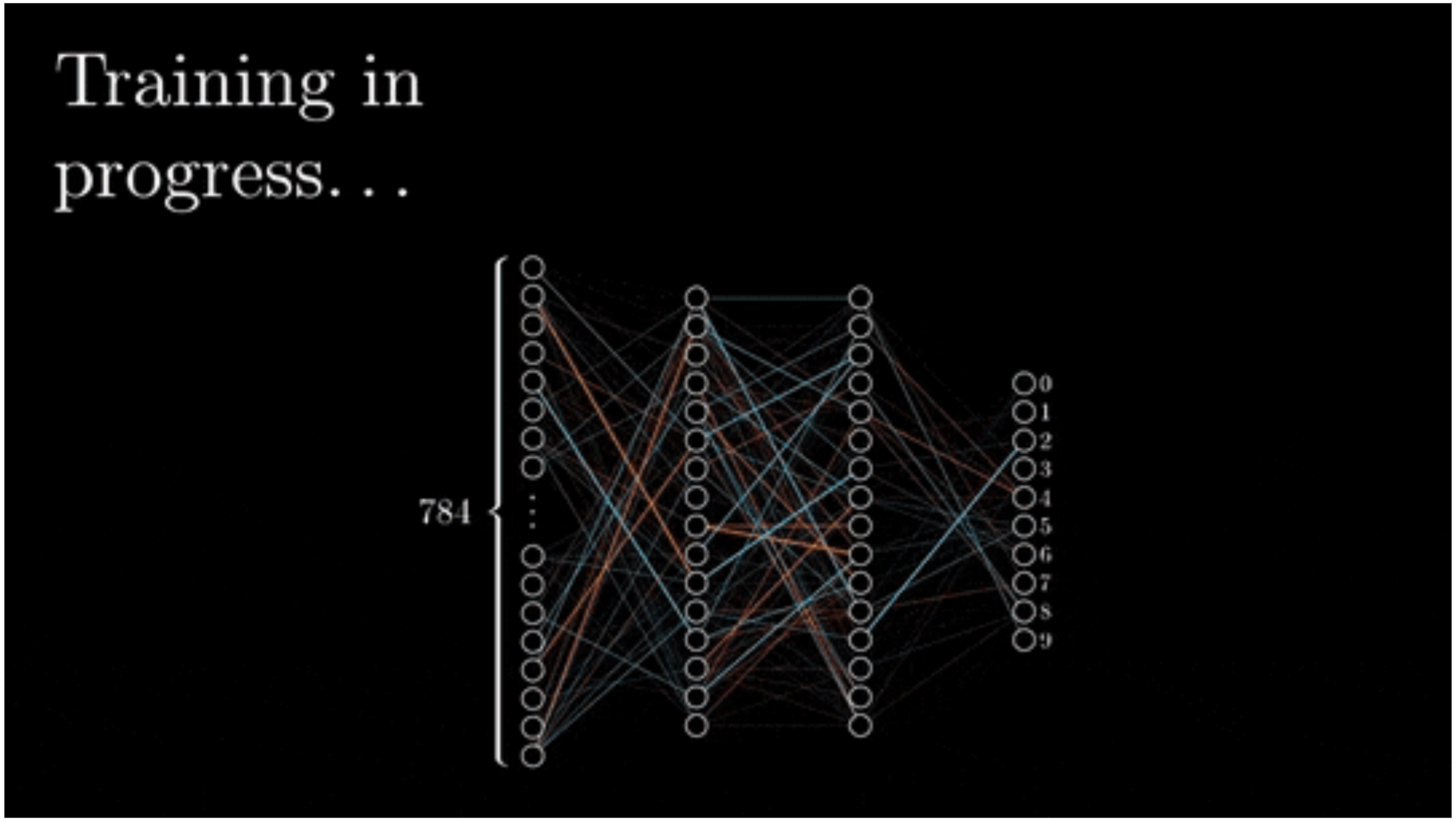
$$
\begin{array}{ccccc}
F = A_\ell W_\ell + b_\ell, & F \in \mathbb{R}^{n\times p_{\ell+1}}, & A_\ell \in \mathbb{R}^{n\times p_\ell}, & W_\ell \in \mathbb{R}^{p_\ell\times p_{\ell+1}}, & b_\ell \in \mathbb{R}^{1\times p_{\ell+1}}, \\
A_\ell = \tanh(H_\ell), & A_\ell \in \mathbb{R}^{n\times p_\ell}, & H_\ell \in \mathbb{R}^{n\times p_\ell}, & & \\
H_\ell = A_{\ell-1} W_{\ell-1} + b_{\ell-1}, & H_\ell \in \mathbb{R}^{n\times p_\ell}, & A_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & W_{\ell-1} \in \mathbb{R}^{p_{\ell-1}\times p_\ell}, & b_{\ell-1} \in \mathbb{R}^{1\times p_\ell}, \\
A_{\ell-1} = \tanh(H_{\ell-1}), & A_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & H_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & & \\
H_{\ell-1} = A_{\ell-2} W_{\ell-2} + b_{\ell-2}, & H_{\ell-1} \in \mathbb{R}^{n\times p_{\ell-1}}, & A_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & W_{\ell-2} \in \mathbb{R}^{p_{\ell-2}\times p_{\ell-1}}, & b_{\ell-2} \in \mathbb{R}^{1\times p_{\ell-1}}, \\
A_{\ell-2} = \tanh(H_{\ell-2}), & A_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & H_{\ell-2} \in \mathbb{R}^{n\times p_{\ell-2}}, & & \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
H_2 = A_1 W_1 + b_1, & H_2 \in \mathbb{R}^{n\times p_2}, & A_1 \in \mathbb{R}^{n\times p_1}, & W_1 \in \mathbb{R}^{p_1\times p_2}, & b_1 \in \mathbb{R}^{1\times p_2}, \\
A_1 = \tanh(H_1), & A_1 \in \mathbb{R}^{n\times p_1}, & H_1 \in \mathbb{R}^{n\times p_1}, & & \\
H_1 = X W_0 + b_0, & H_1 \in \mathbb{R}^{n\times p_1}, & X \in \mathbb{R}^{n\times p_0}, & W_0 \in \mathbb{R}^{p_0\times p_1}, & b_0 \in \mathbb{R}^{1\times p_1},
\end{array}
$$

The backward pass

$$\mathcal{L} := \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p_{\ell+1}}(F_{i,j} - Y_{i,j})^2 \longrightarrow \nabla_\theta \mathcal{L}(\theta)$$

$$\begin{aligned}
G_\ell &= F - Y \in \mathbb{R}^{n \times p_{\ell+1}}, & \tfrac{\partial \mathcal{L}}{\partial W_\ell} &= A_\ell^T G_\ell \in \mathbb{R}^{p_\ell \times p_{\ell+1}}, & \tfrac{\partial \mathcal{L}}{\partial b_\ell} &= \mathbb{1}^T G_\ell \in \mathbb{R}^{1 \times p_{\ell+1}}, \\
G_{\ell-1} &= (1 - A_\ell \odot A_\ell) \odot (G_\ell W_\ell^T) \in \mathbb{R}^{n \times p_\ell}, & \tfrac{\partial \mathcal{L}}{\partial W_{\ell-1}} &= A_{\ell-1}^T G_{\ell-1} \in \mathbb{R}^{p_{\ell-1} \times p_\ell}, & \tfrac{\partial \mathcal{L}}{\partial b_{\ell-1}} &= \mathbb{1}^T G_{\ell-1} \in \mathbb{R}^{1 \times p_\ell}, \\
G_{\ell-2} &= (1 - A_{\ell-1} \odot A_{\ell-1}) \odot (G_{\ell-1} W_{\ell-1}^T) \in \mathbb{R}^{n \times p_{\ell-1}}, & \tfrac{\partial \mathcal{L}}{\partial W_{\ell-2}} &= A_{\ell-2}^T G_{\ell-2} \in \mathbb{R}^{p_{\ell-2} \times p_{\ell-1}}, & \tfrac{\partial \mathcal{L}}{\partial b_{\ell-2}} &= \mathbb{1}^T G_{\ell-2} \in \mathbb{R}^{1 \times p_{\ell-1}}, \\
&\vdots & &\vdots & &\vdots \\
G_1 &= (1 - A_2 \odot A_2) \odot (G_2 W_2^T) \in \mathbb{R}^{n \times p_2}, & \tfrac{\partial \mathcal{L}}{\partial W_1} &= A_1^T G_1 \in \mathbb{R}^{p_1 \times p_2}, & \tfrac{\partial \mathcal{L}}{\partial b_1} &= \mathbb{1}^T G_1 \in \mathbb{R}^{1 \times p_2}, \\
G_0 &= (1 - A_1 \odot A_1) \odot (G_1 W_1^T) \in \mathbb{R}^{n \times p_1}, & \tfrac{\partial \mathcal{L}}{\partial W_0} &= X^T G_0 \in \mathbb{R}^{p_0 \times p_1}, & \tfrac{\partial \mathcal{L}}{\partial b_0} &= \mathbb{1}^T G_0 \in \mathbb{R}^{1 \times p_1}.
\end{aligned}$$

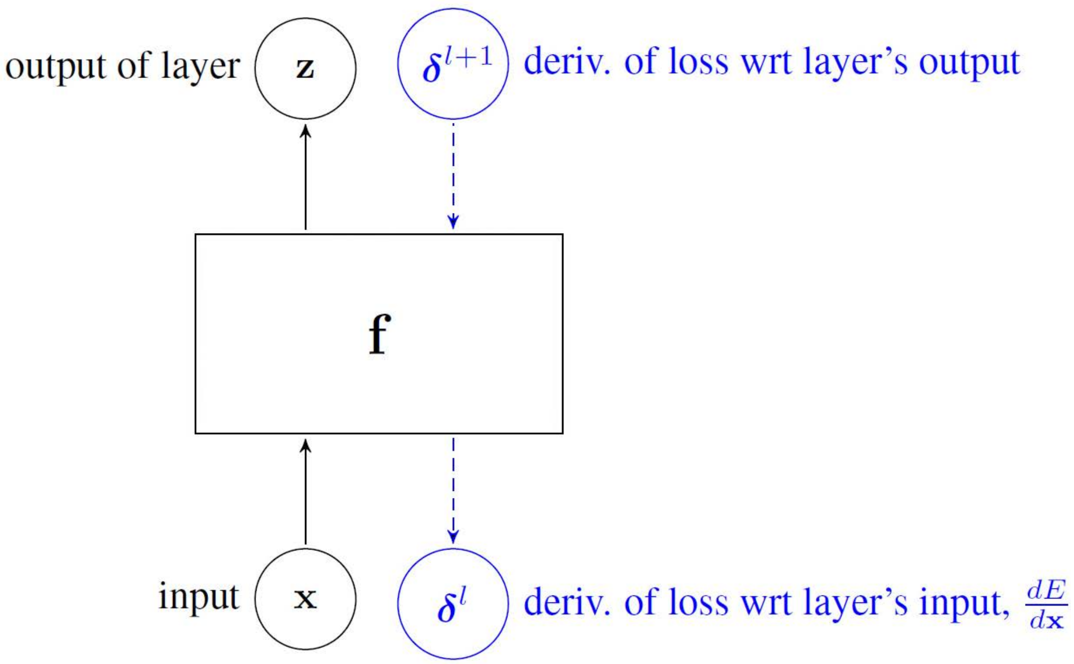
Backpropagation

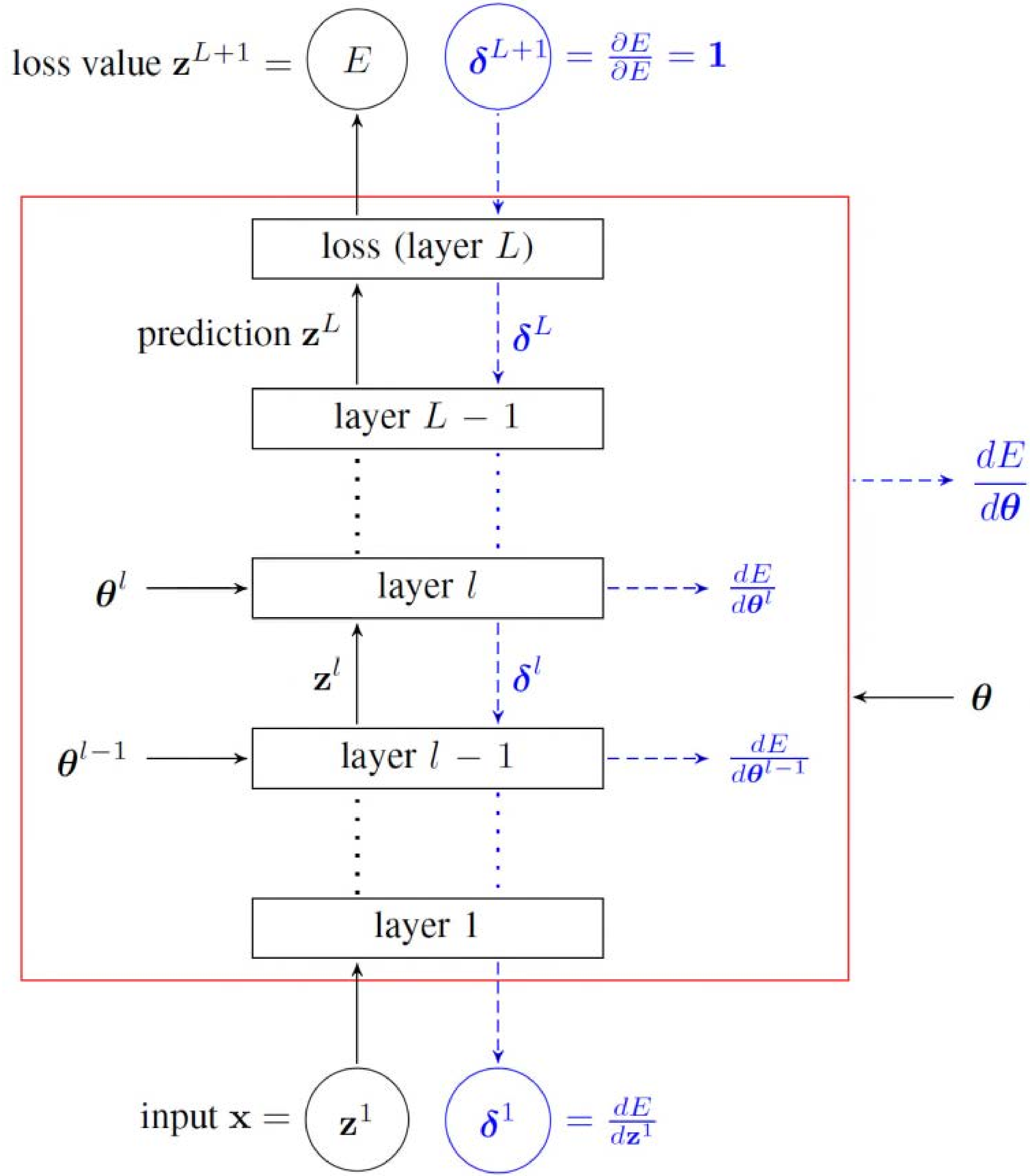


$$
\begin{aligned}
G_\ell &= F - Y \in \mathbb{R}^{n \times p_{\ell+1}}, & \frac{\partial \mathcal{L}}{\partial W_\ell} &= A_\ell^T G_\ell \in \mathbb{R}^{p_\ell \times p_{\ell+1}}, & \frac{\partial \mathcal{L}}{\partial b_\ell} &= \mathbb{1}^T G_\ell \in \mathbb{R}^{1 \times p_{\ell+1}}, \\
G_{\ell-1} &= (1 - A_\ell \odot A_\ell) \odot (G_\ell W_\ell^T) \in \mathbb{R}^{n \times p_\ell}, & \frac{\partial \mathcal{L}}{\partial W_{\ell-1}} &= A_{\ell-1}^T G_{\ell-1} \in \mathbb{R}^{p_{\ell-1} \times p_\ell}, & \frac{\partial \mathcal{L}}{\partial b_{\ell-1}} &= \mathbb{1}^T G_{\ell-1} \in \mathbb{R}^{1 \times p_\ell}, \\
G_{\ell-2} &= (1 - A_{\ell-1} \odot A_{\ell-1}) \odot (G_{\ell-1} W_{\ell-1}^T) \in \mathbb{R}^{n \times p_{\ell-1}}, & \frac{\partial \mathcal{L}}{\partial W_{\ell-2}} &= A_{\ell-2}^T G_{\ell-2} \in \mathbb{R}^{p_{\ell-2} \times p_{\ell-1}}, & \frac{\partial \mathcal{L}}{\partial b_{\ell-2}} &= \mathbb{1}^T G_{\ell-2} \in \mathbb{R}^{1 \times p_{\ell-1}}, \\
&\vdots & &\vdots & &\vdots \\
G_1 &= (1 - A_2 \odot A_2) \odot (G_2 W_2^T) \in \mathbb{R}^{n \times p_2}, & \frac{\partial \mathcal{L}}{\partial W_1} &= A_1^T G_1 \in \mathbb{R}^{p_1 \times p_2}, & \frac{\partial \mathcal{L}}{\partial b_1} &= \mathbb{1}^T G_1 \in \mathbb{R}^{1 \times p_2}, \\
G_0 &= (1 - A_1 \odot A_1) \odot (G_1 W_1^T) \in \mathbb{R}^{n \times p_1}, & \frac{\partial \mathcal{L}}{\partial W_0} &= X^T G_0 \in \mathbb{R}^{p_0 \times p_1}, & \frac{\partial \mathcal{L}}{\partial b_0} &= \mathbb{1}^T G_0 \in \mathbb{R}^{1 \times p_1}.
\end{aligned}
$$

https://maziarraissi.github.io/teaching/3_backpropagation/

Modular implementation





Automatic differentiation

The chain rule, forward and reverse accumulation [edit]

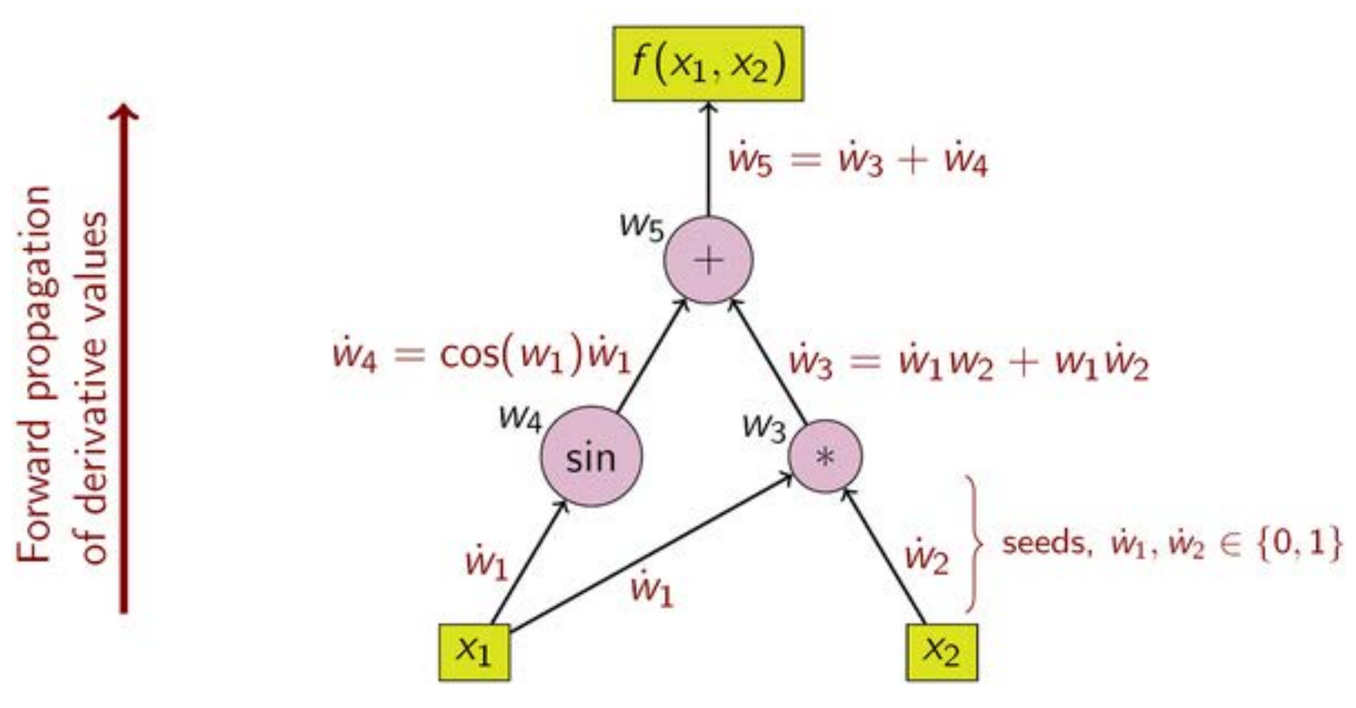
Fundamental to AD is the decomposition of differentials provided by the chain rule. For the simple composition $y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$ the chain rule gives

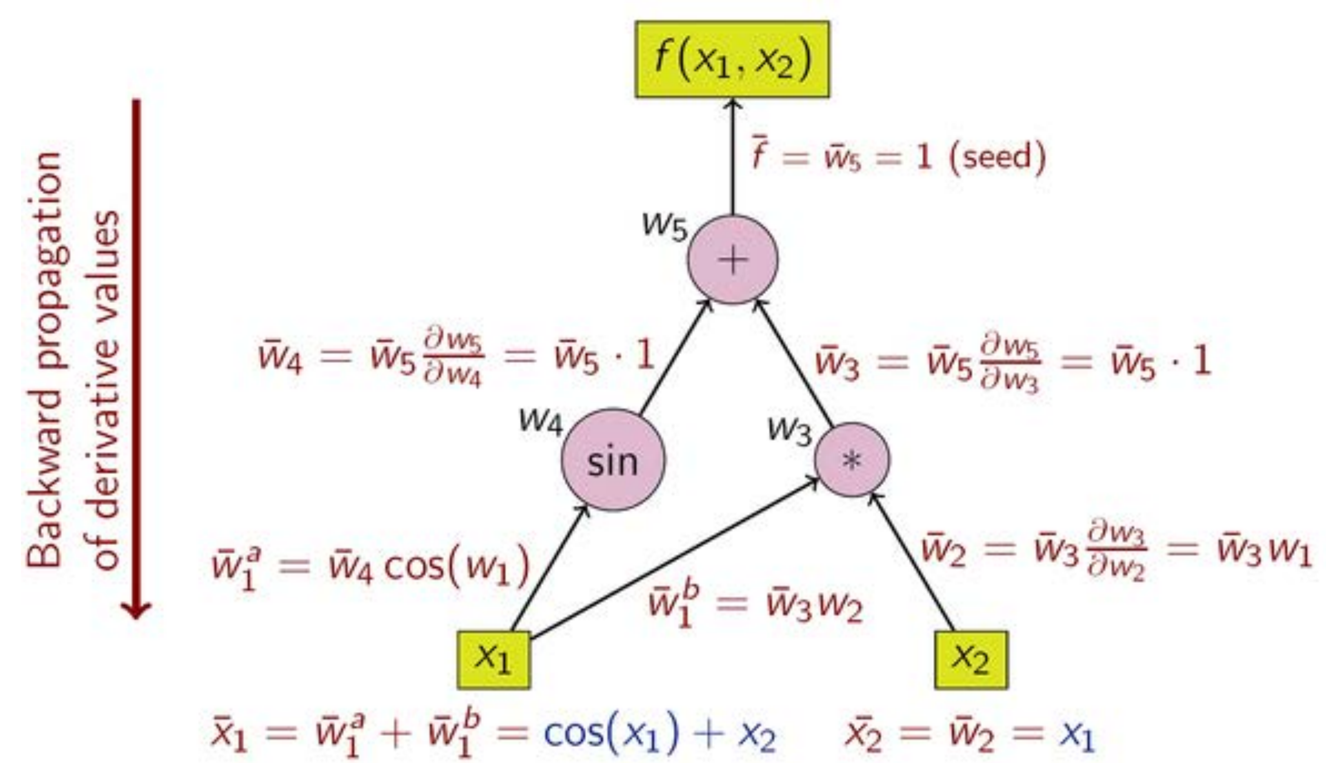
$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx}$$

Usually, two distinct modes of AD are presented, **forward accumulation** (or **forward mode**) and **reverse accumulation** (or **reverse mode**). Forward accumulation specifies that one traverses the chain rule from inside to outside (that is, first compute dw_1/dx and then dw_2/dx and at last dy/dx), while reverse accumulation has the traversal from outside to inside (first compute dy/dw_2 and then dy/dw_1 and at last dy/dx). More succinctly,

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 $w_0 = x$.

Example: $z = f(x_1, x_2) = x_1 x_2 + \sin x_1$

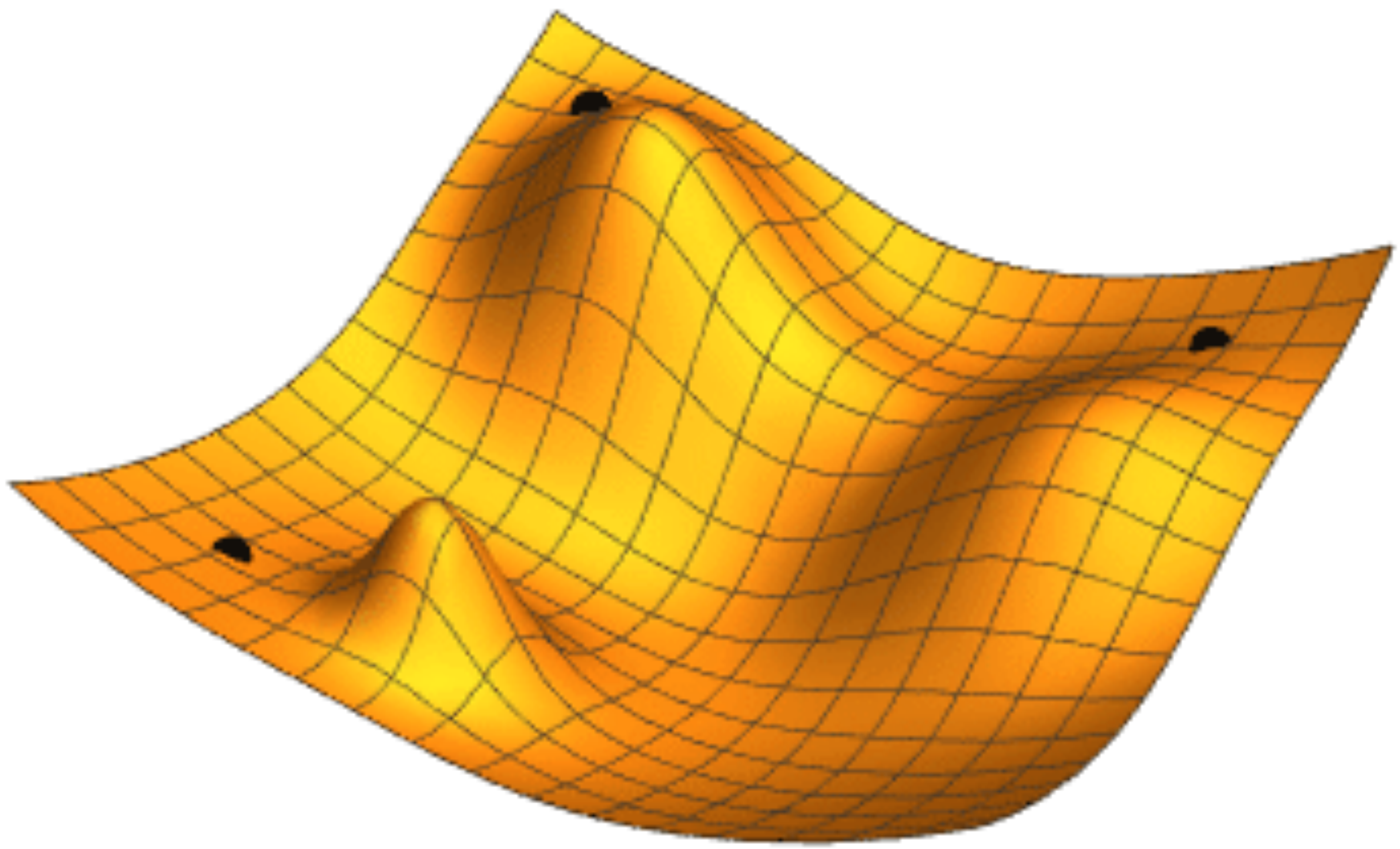




Gradient descent

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} J(\boldsymbol{\theta})$$

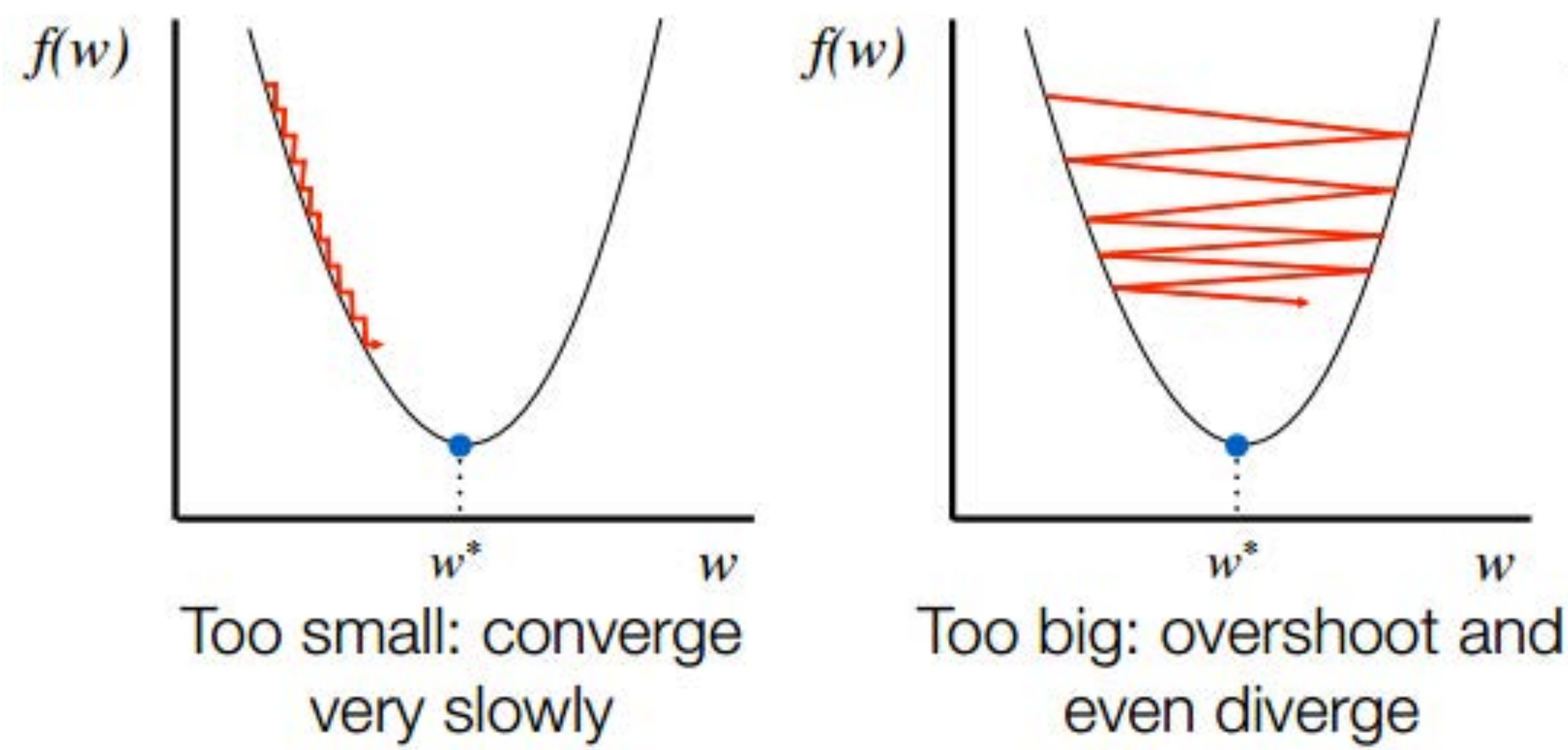
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$



Gradient descent

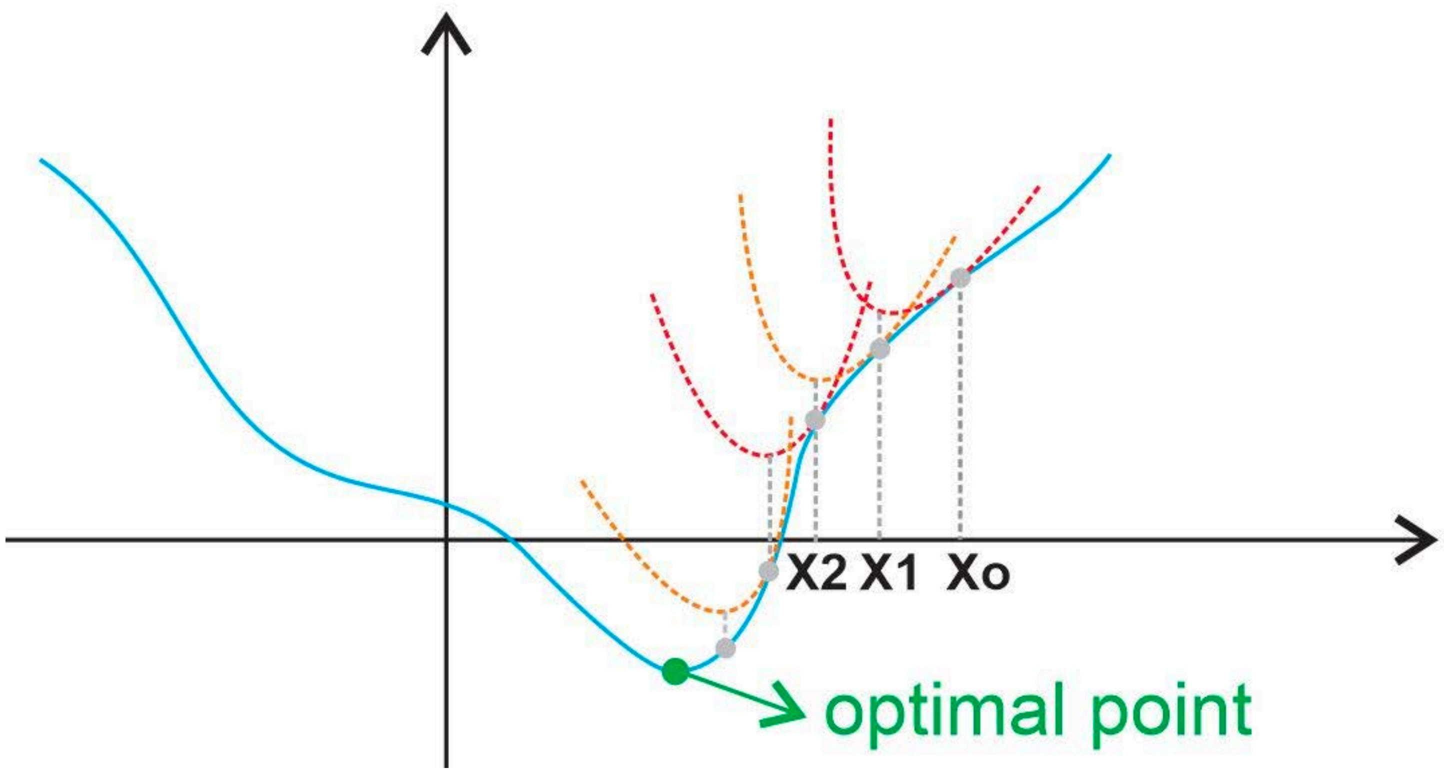
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Effect of the learning rate

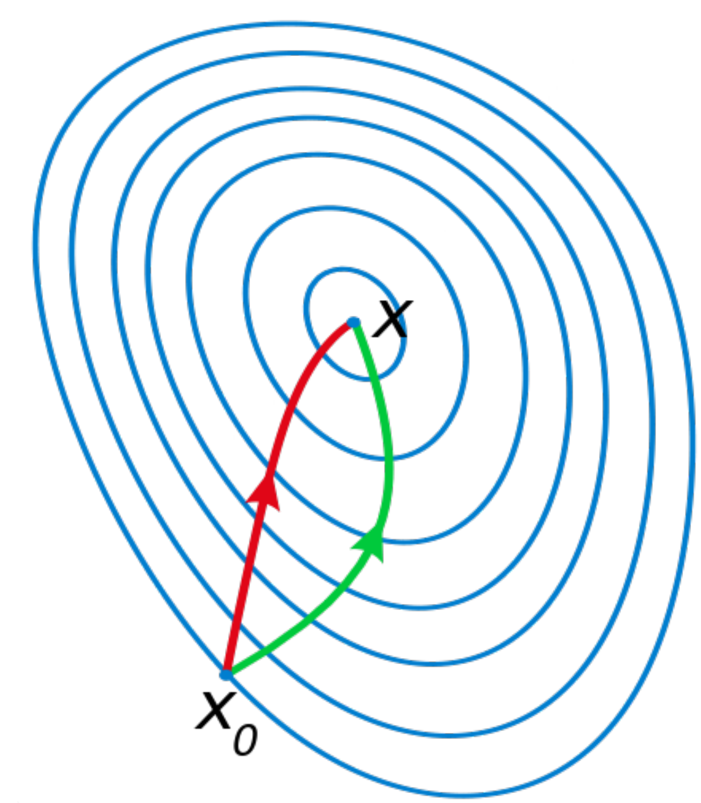


Hessian

$$\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1^2} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d^2} \end{bmatrix}$$



Local quadratic approximation
of the loss == Newton's method



Gradient descent vs Newton

BFGS

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item

Broyden–Fletcher–Goldfarb–Shanno algorithm

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. [hide]

(Learn how and when to remove these template messages)

- This article **may be too technical for most readers to understand**. Please help improve it to make it understandable to non-experts, without removing the technical details. *(September 2010)*
- This article **needs additional citations for verification**. *(March 2016)*

In numerical optimization, the **Broyden–Fletcher–Goldfarb–Shanno** (**BFGS**) **algorithm** is an iterative method for solving unconstrained nonlinear optimization problems.[1]

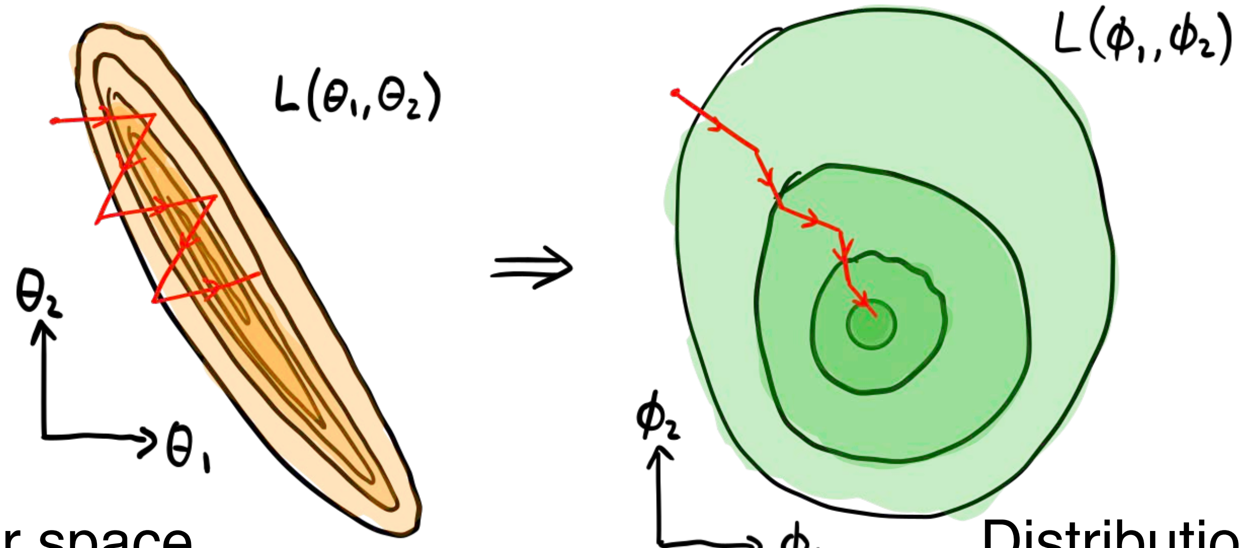
The BFGS method belongs to quasi-Newton methods, a class of hill-climbing optimization techniques that seek a stationary point of a (preferably twice continuously differentiable) function. For such problems, a necessary condition for optimality is that the gradient be zero. Newton's method and the BFGS methods are not guaranteed to converge unless the function has a quadratic Taylor expansion near an optimum. However, BFGS has proven to have good performance even for non-smooth optimizations.[2]

In quasi-Newton methods, the Hessian matrix of second derivatives doesn't need to be evaluated directly. Instead, the Hessian matrix is approximated using updates specified by gradient evaluations (or approximate gradient evaluations). Quasi-Newton methods are generalizations of the secant method to find the root of the first derivative for multidimensional problems. In multi-dimensional problems, the secant equation does not specify a unique solution, and quasi-Newton methods differ in how they constrain the solution. The BFGS method is one of the most popular members of this class.[3] Also in common use is L-BFGS, which is a limited-memory version of BFGS that is particularly suited to problems with very large numbers of variables (e.g., >1000). The BFGS-B[4] variant handles simple box constraints.

The algorithm is named after Charles George Broyden, Roger Fletcher, Donald Goldfarb and David Shanno.

Gradient descent vs natural gradient descent

Motivation: If our objective is to minimize the loss function (maximizing the likelihood), then it is natural that we taking step in the space of all possible likelihoods, realizable by the parameters θ. As the likelihood function itself is a probability distribution, we call this space distribution space. Thus it makes sense to take the steepest descent direction in this distribution space instead of parameter space.



Parameter space

Distribution space

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

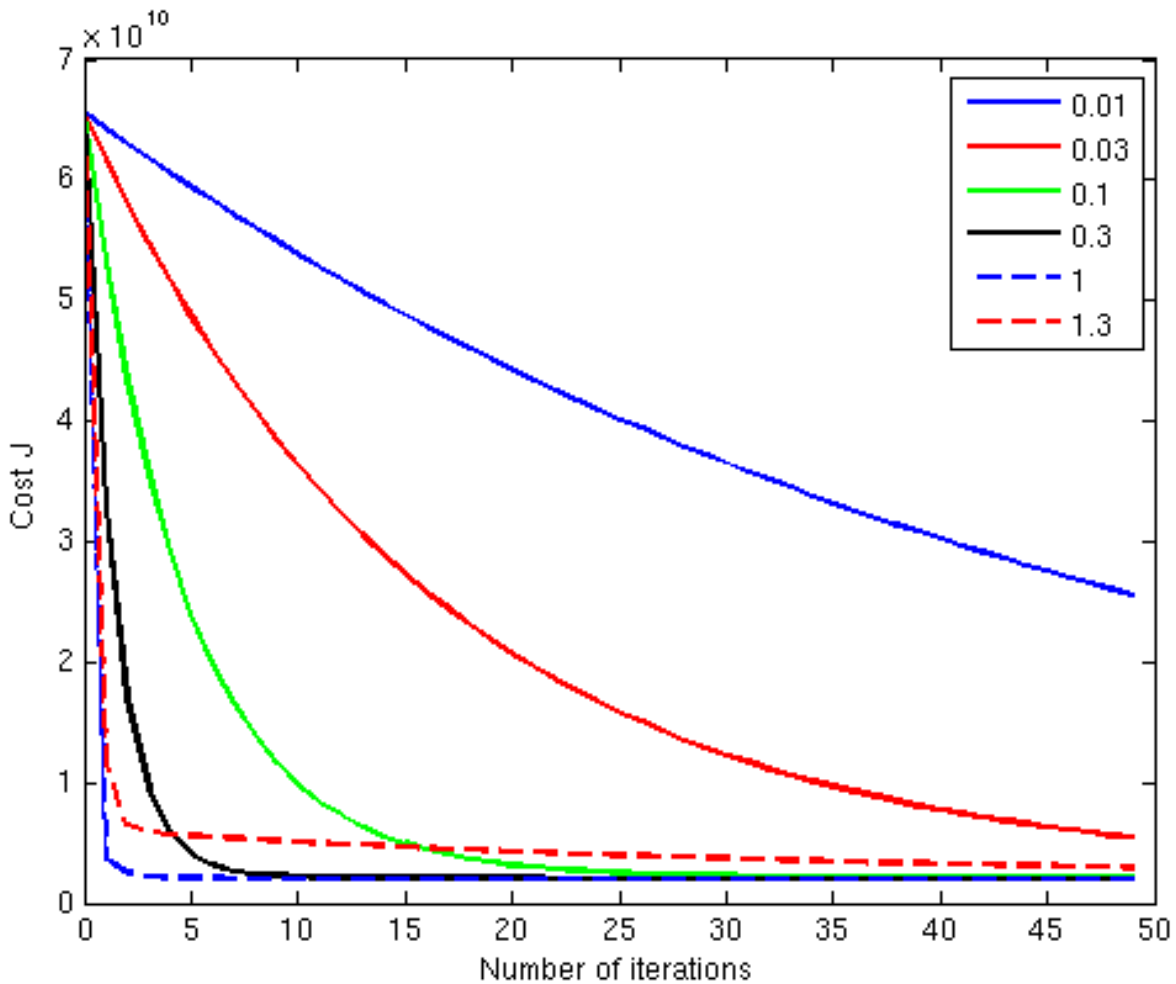
$$\theta_{n+1} = \theta_n - \eta F^{-1} \nabla \mathscr{L}(\theta)$$

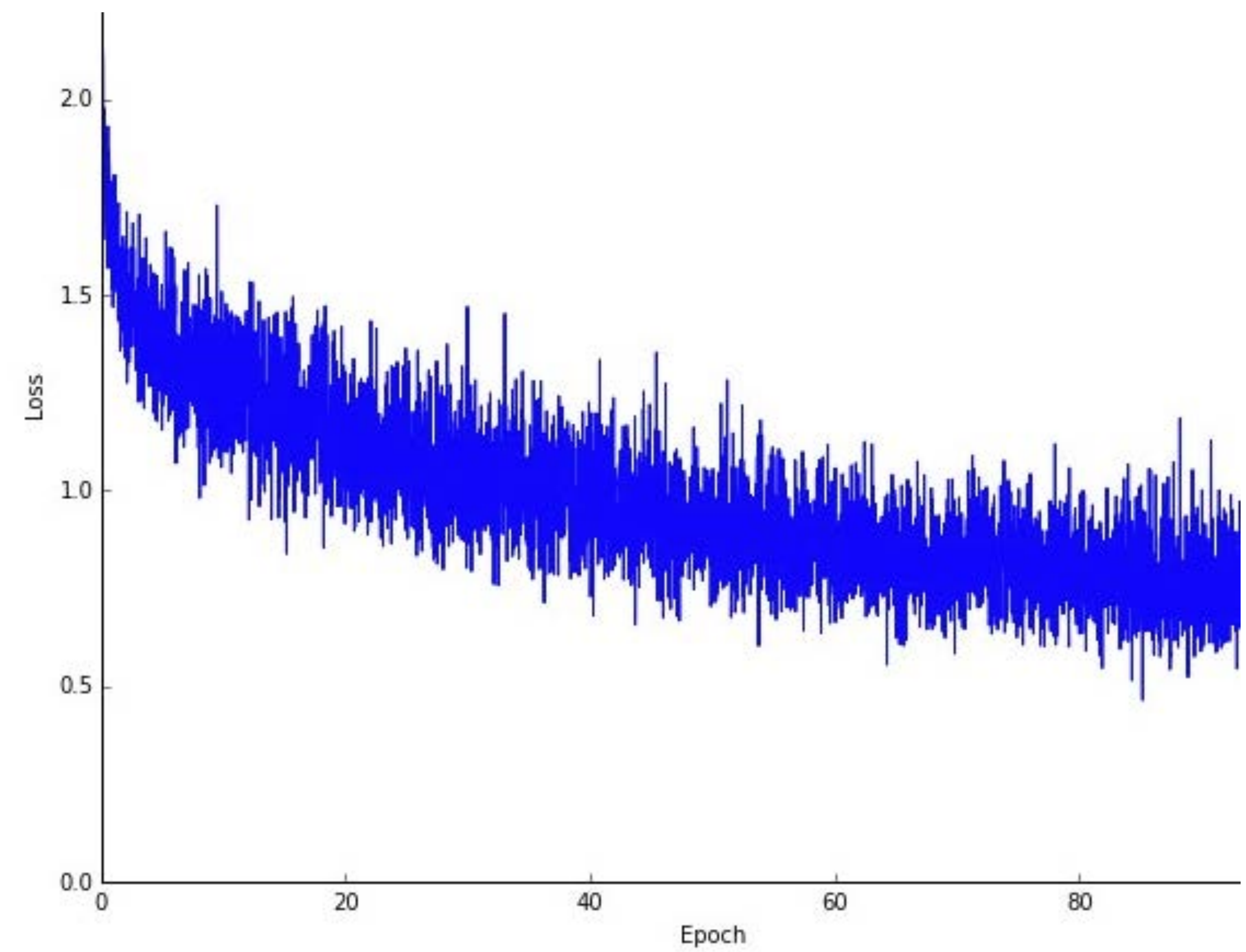
Cramer-Rao bound: The inverse of the Fisher information is a lower bound on the variance of any unbiased estimator of θ

Fisher Information Matrix == negative expected Hessian of log likelihood

$$\begin{aligned}
\mathrm{H}_{\mathrm{KL}[p(x|\theta)\,\|\,p(x|\theta')]} &= -\int p(x|\theta)\ \nabla^2_{\theta'} \log p(x|\theta')\big|_{\theta'=\theta}\,\mathrm{d}x \\
&= -\int p(x|\theta)\,\mathrm{H}_{\log p(x|\theta)}\,\mathrm{d}x \\
&= -\mathop{\mathbb{E}}_{p(x|\theta)}[\mathrm{H}_{\log p(x|\theta)}] \\
&= \mathbf{F}\,.
\end{aligned}$$

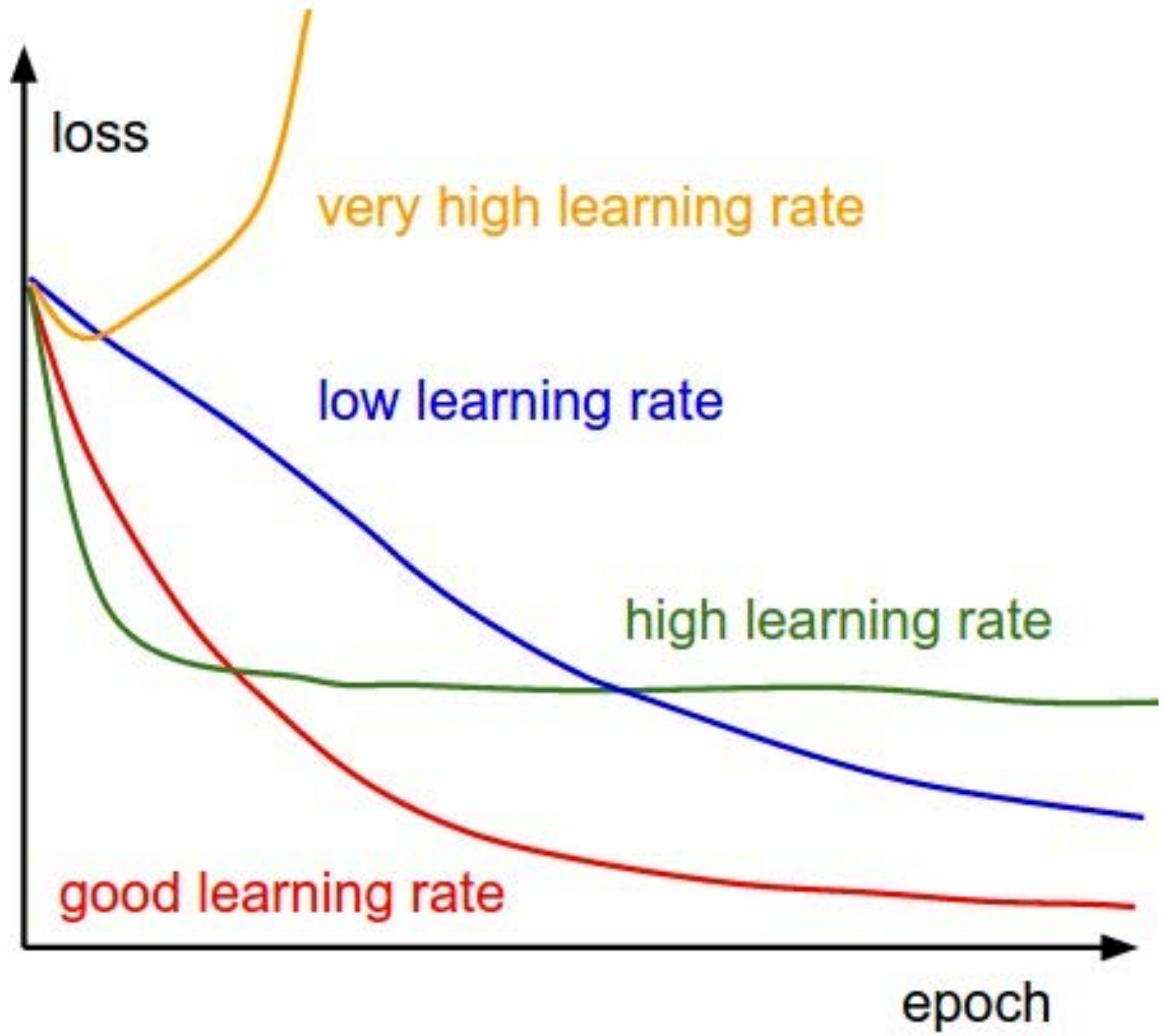
Gradient descent vs SGD





https://lossfunctions.tumblr.com

Gradient descent vs SGD



Remarks

Vanilla mini-batch gradient descent, however, does not guarantee good convergence, but offers a few challenges that need to be addressed:

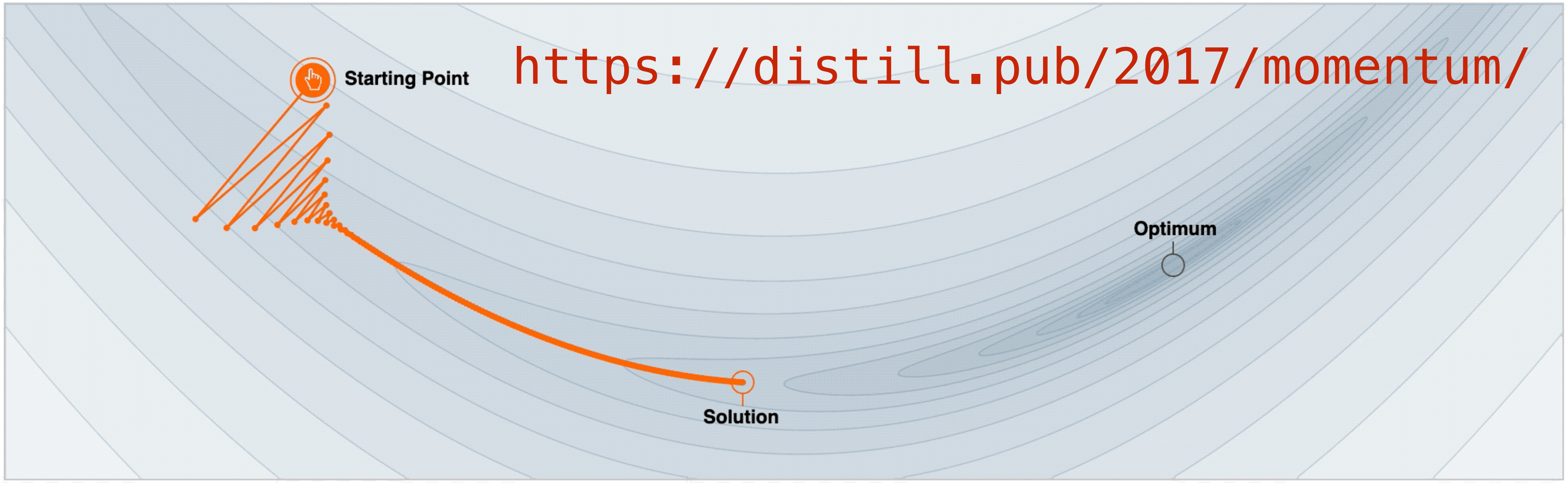
- Choosing a proper learning rate can be difficult.
- Learning rate schedules (i.e., adjusting the learning rate during training) has to be defined in advance and it is thus unable to adapt to a dataset's characteristics.
- The same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima. Dauphin et al. argue that the difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

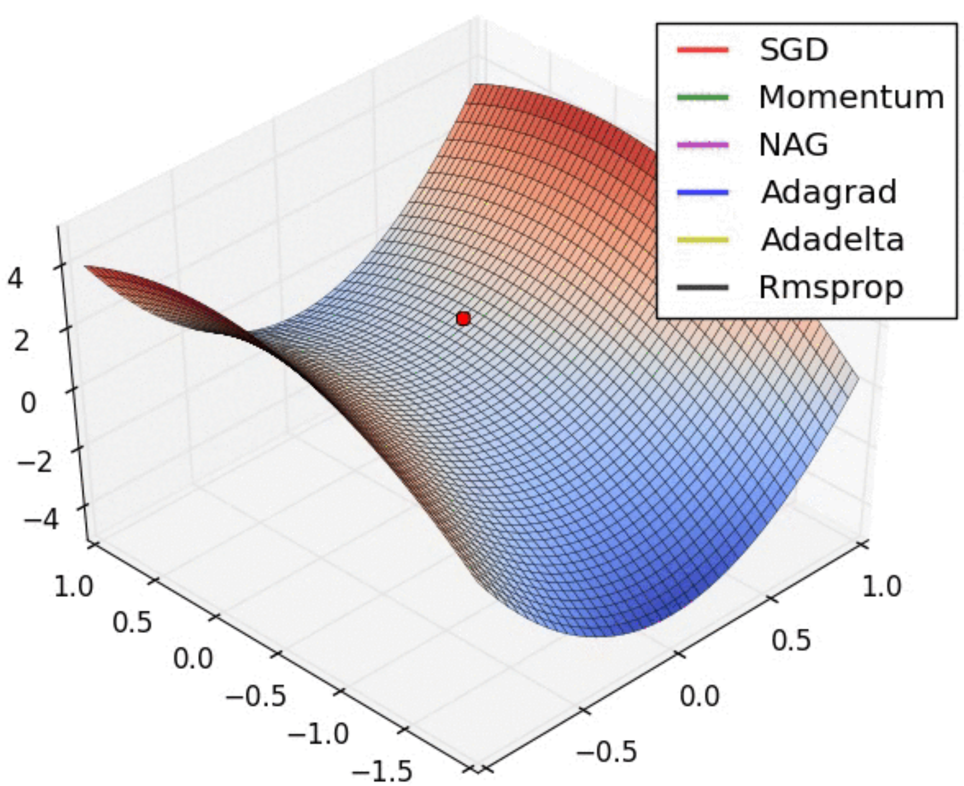
Dauphin, Y., et. al. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv, 1–14.

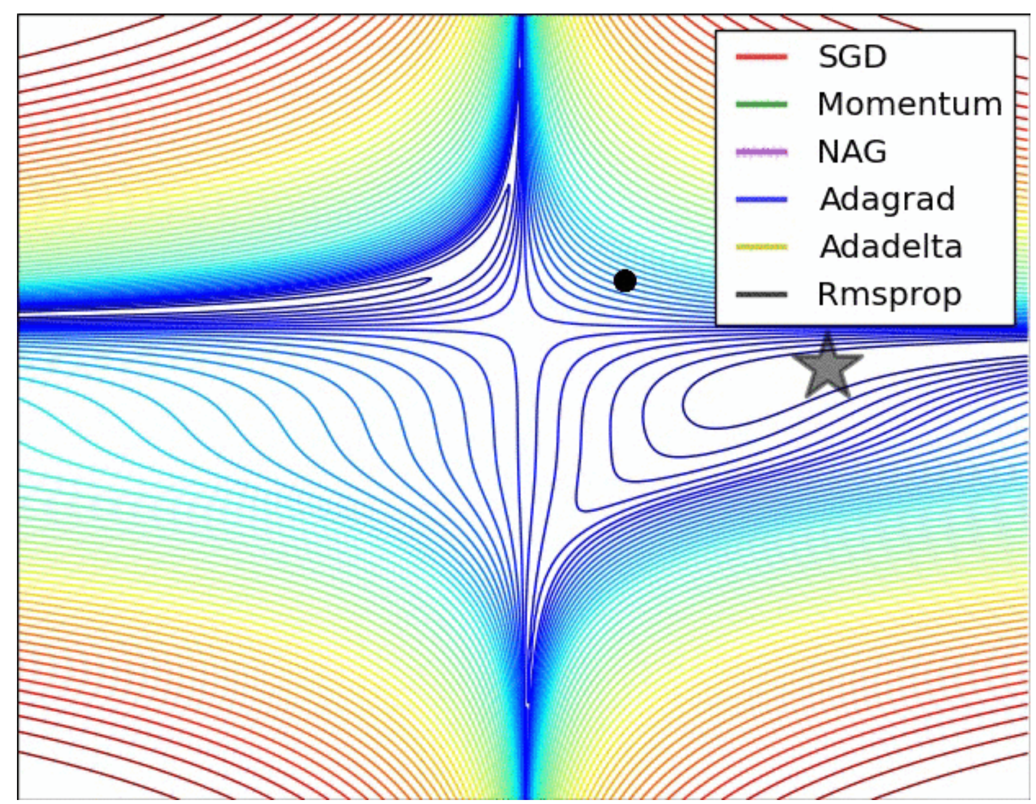
Modern SGD variants

https://distill.pub/2017/momentum/



We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?





http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/

http://ruder.io/optimizing-gradient-descent/

*animation credit: Alec Redford