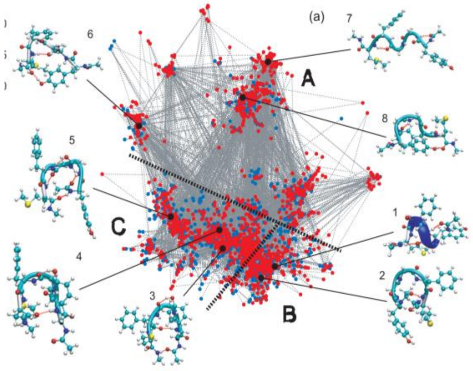
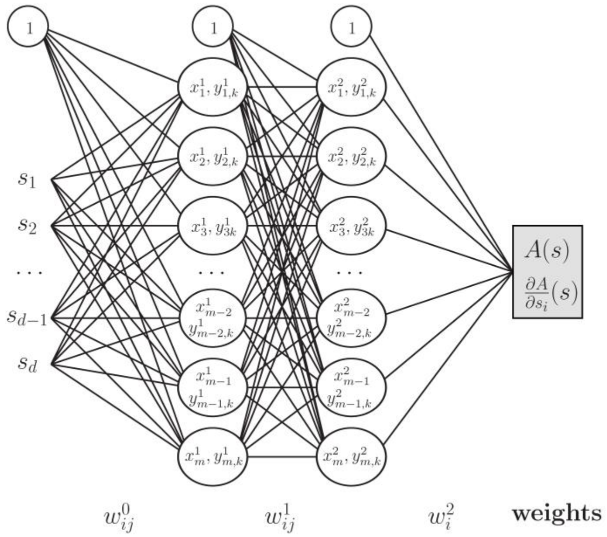
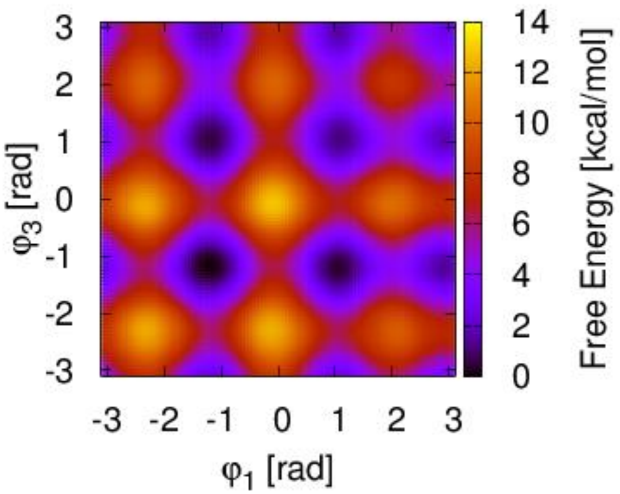
An Introduction to Neural Networks







Mark E. Tuckerman
Dept. of Chemistry and Courant Institute of Mathematical Sciences
New York University, 100 Washington Square East, NY 10003
NYU-ECNU Center for Computational Chemistry at NYU Shanghai 200062, China
华 东 师 范 大 学-纽约大学 计算化学联合研究中心

NEW YORK UNIVERSITY MRSEC

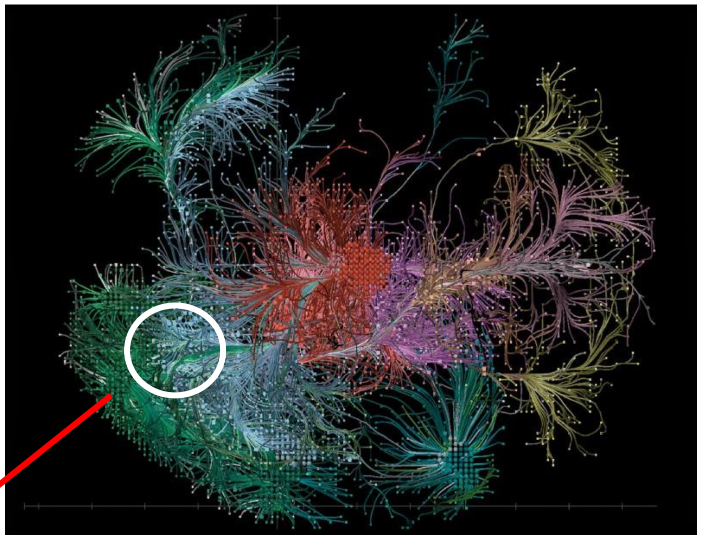
Neural networks in everyday life…

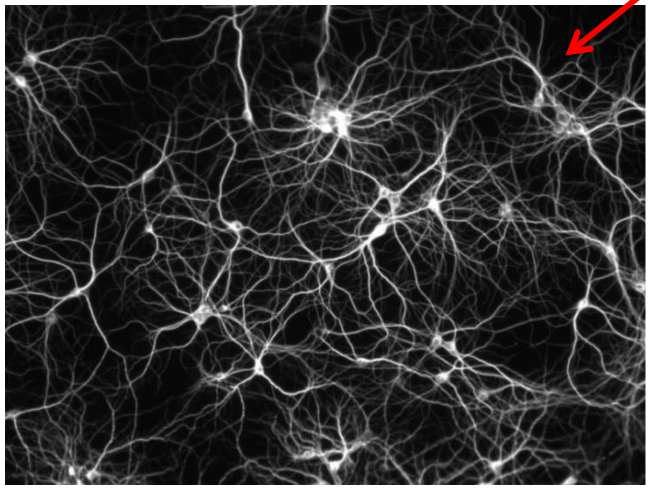
IBM's Watson computer plays and wins "Jeopardy" in 2011.

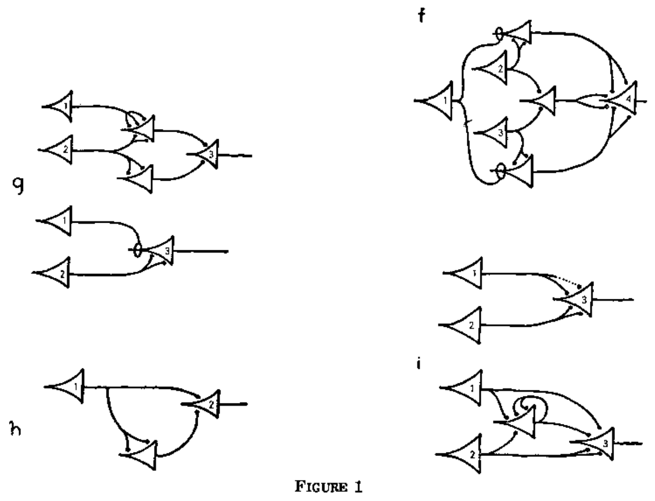
…..and after?

Eterni.mi – Create a chatbot of yourself after you're gone from your digital footprint?

Try to “mimic” the brain’s neuronal connections







First neuron-based computational model: McCulloch and Pitts (1943), “A logical calculus of the ideas immanent in nervous activity”, *Bull. Math. Biophys.*

Kolmogorov Superposition Theorem (1957) – Answers Hilbert's 13th problem (1900)

Given a function $f(x_1,...,x_n)$ of n variables $x_1,...,x_n,\ x_p \in [0,1]$,

f can be represented (Sprecher form) as

$$\boxed{f(x_1,...,x_n) = \sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n} \lambda_p \varphi_q(x_p)\right)}$$

$\lambda_1,...,\lambda_n > 0,\ \varphi_q(x)$ is monotonically increasing, and $\varphi : [0.1] \rightarrow [0,1]$.

The function $g(y)$ is continuous and $g : R^1 \rightarrow R^1$

Proof (existence but NOT constructive):

Let ε and δ be numbers such that $0 < \varepsilon, \delta < 1$,

consider a set of functions $\varphi_1,...,\varphi_{2n+1},$ such that $\exists \gamma : R^1 \rightarrow R^1$

such that $\|\gamma\| \leq \| f \|$ and such that, for the given set of $\varphi_1,...,\varphi_{2n+1},$

$$\left\| f(x_1,...,x_n) - \sum_{q=1}^{2n+1} \gamma\left(\sum_{p=1}^{n} \lambda_p \varphi_q(x_p)\right)\right\| \leq (1-\varepsilon)\| f \|, \qquad \|\gamma\| = \delta \| f \|$$

We now define a series of functions

$$\gamma_j,\ h_j,\ j = 1, 2, \ldots\infty$$

$$h_j(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \gamma_j \left(\sum_{p=1}^{n} \lambda_p \varphi_q(x_p) \right)$$

Applying the above result, in an inductive fashion, we have the following series:

$$\| f - h_1 \| \le (1-\varepsilon) \| f \|, \qquad \| \gamma_1 \| = \delta \| f \|$$

$$\| (f - h_1) - h_2 \| \le (1-\varepsilon) \| f - h_1 \| \le (1-\varepsilon)^2 \| f \|, \qquad \| \gamma_2 \| = \delta \| f - h_1 \| = \delta (1-\varepsilon) \| f \|$$

$$\ldots$$

$$\left\| f - \sum_{j=1}^{r} h_j \right\| \le (1-\varepsilon)^r \| f \|, \qquad \| \gamma_r \| = \delta (1-\varepsilon)^{r-1} \| f \|$$

Let $r \to \infty$,

$$\lim_{r\to\infty}\left\| f - \sum_{j=1}^{r} h_j \right\| \le \lim_{r\to\infty}(1-\varepsilon)^r \| f \| = 0, \qquad \lim_{r\to\infty}\|\gamma_r\| = \lim_{r\to\infty}\delta(1-\varepsilon)^{r-1}\| f \| = 0$$

$$f(x_1,\dots,x_n) = \sum_{j=1}^{\infty} h_j(x_1,\dots,x_n)$$

$$= \sum_{j=1}^{\infty}\sum_{q=1}^{2n+1}\gamma_j\left(\sum_{p=1}^{n}\lambda_p\varphi_q(x_p)\right)$$

$$= \sum_{q=1}^{2n+1}\sum_{j=1}^{\infty}\gamma_j\left(\sum_{p=1}^{n}\lambda_p\varphi_q(x_p)\right)$$

$$\equiv \sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n}\lambda_p\varphi_q(x_p)\right)$$

Q.E.D.

Iterating the Kolmogorov theorem:

$$f(x_1,...,x_n)=\sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n}\lambda_p\varphi_q(x_p)\right)$$

$$h_q(x_1,...,x_n)=\sum_{p=1}^{n}\lambda_p\varphi_q(x_p)$$

$$h_q(x_1,...,x_n)=\sum_{r=1}^{2n+1}\gamma_q\left(\sum_{p=1}^{n}\lambda_p\psi_r(x_p)\right)$$

$$h_r(x_1,...,x_n)=\sum_{p=1}^{n}\lambda_p\psi_r(x_p)$$

$$h_r(x_1,...,x_n)=\sum_{s=1}^{2n+1}\gamma_r\left(\sum_{p=1}^{n}\lambda_p\omega_s(x_p)\right)$$

$$f(x_1,...,x_n)=\sum_{q=1}^{2n+1} g\left(\sum_{r=1}^{2n+1}\gamma_q\left(\sum_{s=1}^{2n+1}\gamma_r\left(\sum_{p=1}^{n}\lambda_p\omega_s(x_p)\right)\right)\right)$$ etc.

Kurkova’s Theorem (1991)

V. Kurkova *Neural Computation* (1991)

Let m be an integer such that $m \geq 2n+1$, and let $w_{pq}, p=1,...,n, \; q=1,...,m$ be a set of parameters. Then, Kolmogorov's theorem can be restated as

$$\boxed{f(x_1,...,x_n) = \sum_{q=1}^{m} g\left(\sum_{p=1}^{n} w_{pq}\varphi_q(x_p)\right)}$$

Kurkova’s theorem can also be iterated:

$$f(x_1,...,x_n) = \sum_{q=1}^{m} g\left(\sum_{r=1}^{m'} \gamma_q\left(\sum_{s=1}^{m''} \gamma_r\left(\sum_{p=1}^{n} w_{ps}\omega_s(x_p)\right)\right)\right)$$

Concrete example of a neural network:

$$\omega_s(x_p) = x_p + c_s$$

$$a_s^{(1)} = \sum_{p=1}^{n} w_{sp}^{(0)} x_p + w_{s0}^{(0)}, \qquad z_s^{(1)} = h\left(a_s^{(1)}\right)$$

$$a_r^{(2)} = \sum_{s=1}^{m''} \left(w_{rs}^{(1)} z_s^{(1)} + w_{r0}^{(1)} \right), \qquad z_r^{(2)} = h\left(a_r^{(2)}\right)$$

$$a_q^{(3)} = \sum_{r=1}^{m'} \left(w_{qr}^{(2)} z_r^{(2)} + w_{q0}^{(2)} \right)$$

$$f(x_1,\ldots,x_n) = \sum_{q=1}^{m} h\left(a_q^{(3)}\right)$$

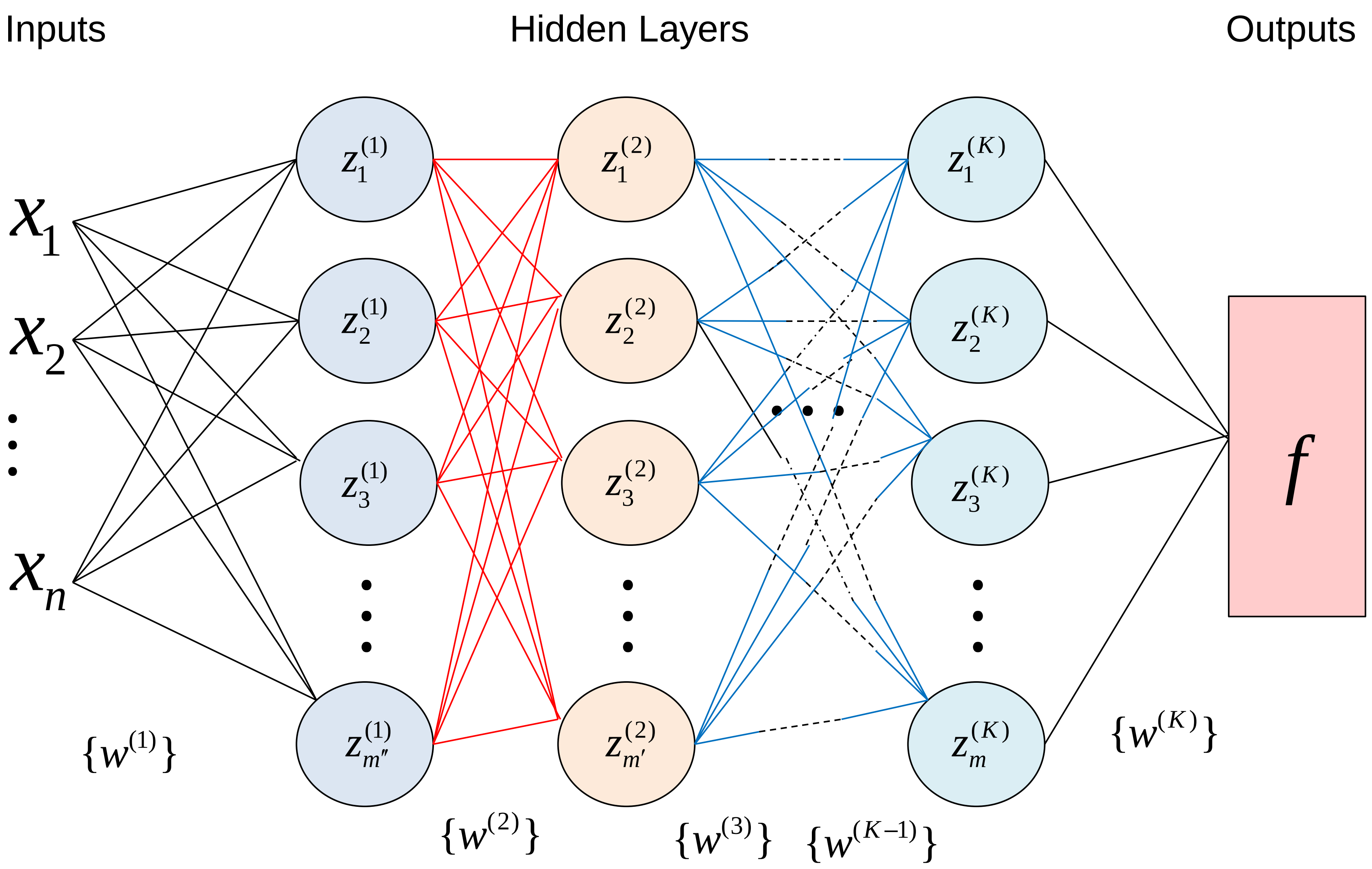
$h(y)$ known as an "activation function"

Expanded out and generalized to an arbitrary number K of nestings.

$$f_{\mathrm{NN}}(x_1,\ldots,x_n,\mathbf{w}) \equiv f_{\mathrm{NN}}(\mathbf{x},\mathbf{w})$$

$$= \sum_{j_K=1}^{M} h\left(\sum_{j_{K-1}=1}^{M} h\left(\cdots \sum_{j_1=1}^{M} h\left(\sum_{p=1}^{n} x_p w_{pj_1}^{0} + w_{0\,j_1}^{0} \right) w_{j_1 j_2}^{1} + w_{0\,j_2}^{1} \cdots \right) w_{j_{K-2} j_{K-1}}^{K-1} + w_{0\,j_{K-1}}^{K-1} \right) w_{j_{K-1} j_K}^{K} + w_{0\,j_K}^{K}$$

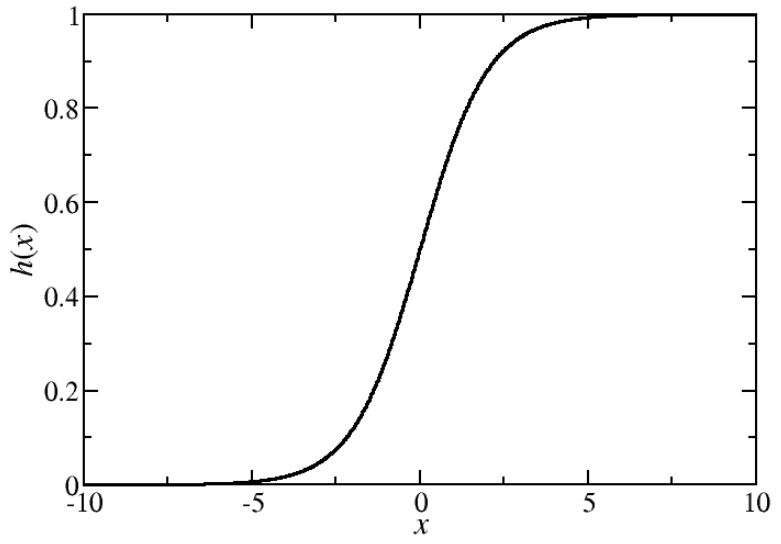
Schematic/graph representation of a neural network



Examples of activation functions

Sigmoid function:

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

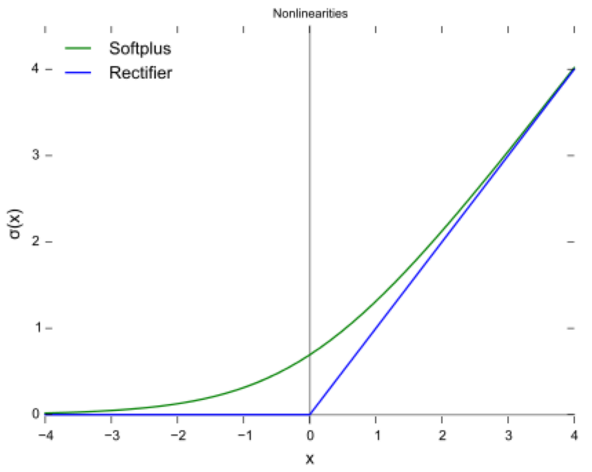


Rectified linear unit ("ReLu") function:

$$h(x) = \max(0, x) \qquad \text{(not differentiable)}$$

Soft form called "softplus" function:

$$h(x) = \ln(1+e^{x}) \qquad \text{(soft, differentiable)}$$



Many other types: see stats.stackexchange.com

Network training

Given M specific values of the function f_λ, $\lambda = 1,\ldots,M$ at specific values $x^\lambda \equiv x_1^\lambda,\ldots,x_n^\lambda$ training consists in using this data to fit a set of connection parameters $\mathbf{w}$.

In order to perform this training, we first set up a regression *cost function* or *error function*:

$$\boxed{E(\mathbf{w}) = \frac{1}{2M}\sum_{\lambda=1}^{M}\left| f_{NN}\left(\mathbf{x}^\lambda;\mathbf{w}\right) - f_\lambda\right|^2}$$

The error function can also include a regularization term:

$$E(\mathbf{w}) = \frac{1}{2M}\sum_{\lambda=1}^{M}\left| f_{NN}\left(\mathbf{x}^\lambda;\mathbf{w}\right) - f_\lambda\right|^2 + \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

The cost function must then be minimized with respect to $\mathbf{w}$:

$$\frac{\partial E}{\partial \mathbf{w}} = 0$$

Calculation of derivatives needed for network training

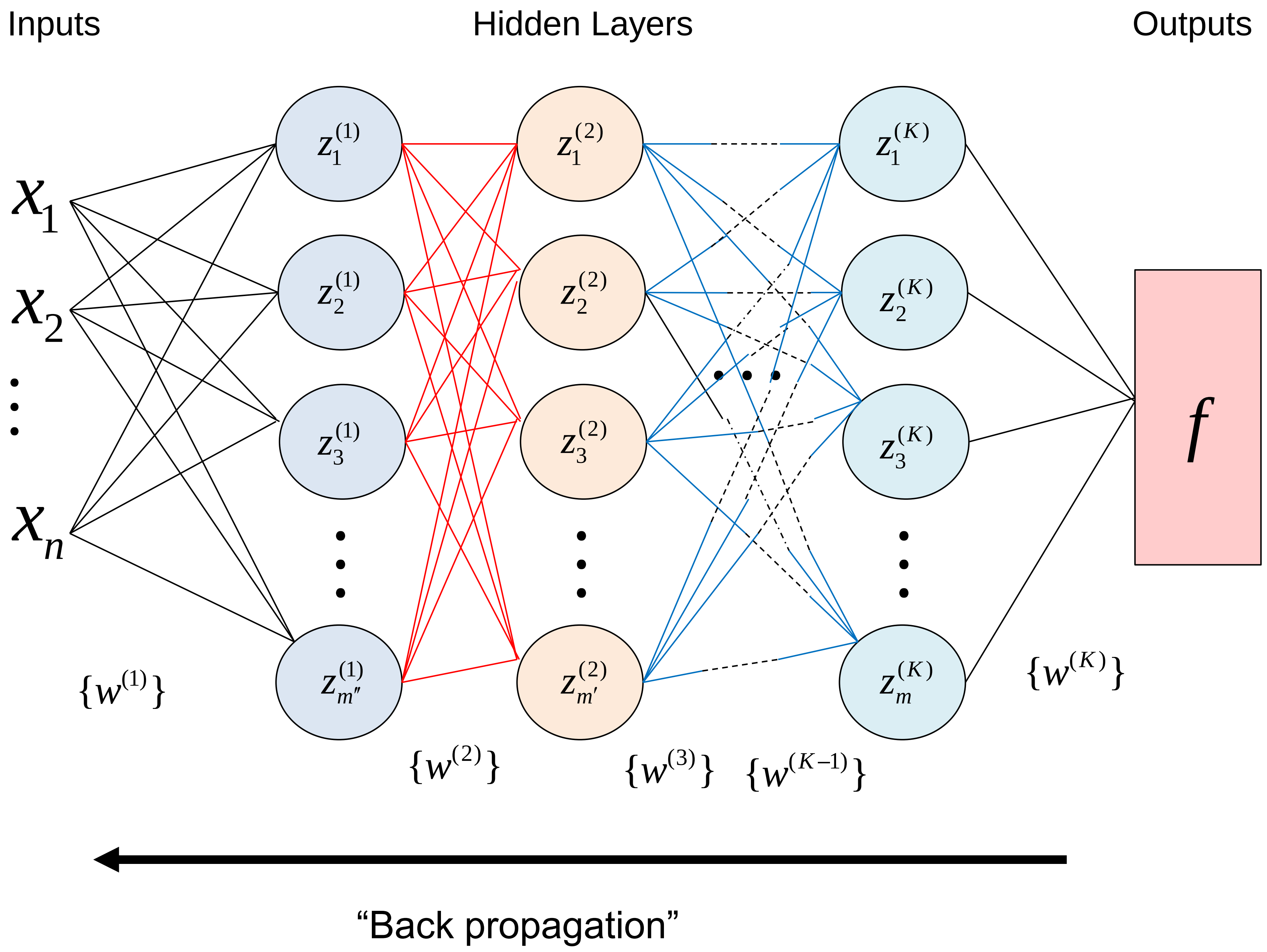
$$\frac{\partial E}{\partial w_{jr}^{(l)}} = \sum_{\lambda=1}^{M} \sum_{s=1}^{m^{(l+1)}} \frac{\partial E}{\partial a_s^{(l+1)}(\mathbf{x}^\lambda)} \frac{\partial a_s^{(l+1)}(\mathbf{x}^\lambda)}{\partial w_{jr}^{(l)}}$$

$$= \begin{cases} \displaystyle\sum_{\lambda=1}^{M} \frac{\partial E}{\partial a_r^{(l+1)}(\mathbf{x}^\lambda)} x_j^\lambda, & l=0, \quad j=1,\dots,n \\ \displaystyle\sum_{\lambda=1}^{M} \frac{\partial E}{\partial a_r^{(l+1)}(\mathbf{x}^\lambda)} h\left(a_j^{(l)}(\mathbf{x}^\lambda)\right), & 0<l\le K \end{cases}$$

We can express this as a "backwards propagation" iterative scheme through the layers of the network, using the following rules for the above derivatives:

$$\frac{\partial E}{\partial a^{(K+1)}\left(\mathbf{x}^\lambda\right)} \equiv \frac{\partial E}{\partial f_{NN}\left(\mathbf{x}^\lambda;\mathbf{w}\right)} = \frac{1}{M}\left(f_{NN}(\mathbf{x}^\lambda;\mathbf{w}) - f_\lambda \right)$$

$$\frac{\partial E}{\partial a_r^{(l)}(\mathbf{x}^\lambda)} = \begin{cases} \displaystyle\sum_{j=1}^{m^{(l+1)}} \frac{\partial E}{\partial a_j^{(l+1)}\left(\mathbf{x}^\lambda\right)} w_{rj}^{(l)}, & l=0 \\ \displaystyle\sum_{j=1}^{m^{(l+1)}} \frac{\partial E}{\partial a_j^{(l+1)}\left(\mathbf{x}^\lambda\right)} w_{rj}^{(l)} h'\left(a_r^{(l)}(\mathbf{x}^\lambda)\right), & 1\le l\le K \end{cases}$$


Inputs
Hidden Layers
Outputs
x_1
x_2
x_n
$z_1^{(1)}$
$z_2^{(1)}$
$z_3^{(1)}$
$z_{m''}^{(1)}$
$z_1^{(2)}$
$z_2^{(2)}$
$z_3^{(2)}$
$z_{m'}^{(2)}$
$z_1^{(K)}$
$z_2^{(K)}$
$z_3^{(K)}$
$z_m^{(K)}$
f
$\{w^{(1)}\}$
$\{w^{(2)}\}$
$\{w^{(3)}\}$
$\{w^{(K-1)}\}$
$\{w^{(K)}\}$
"Back propagation"

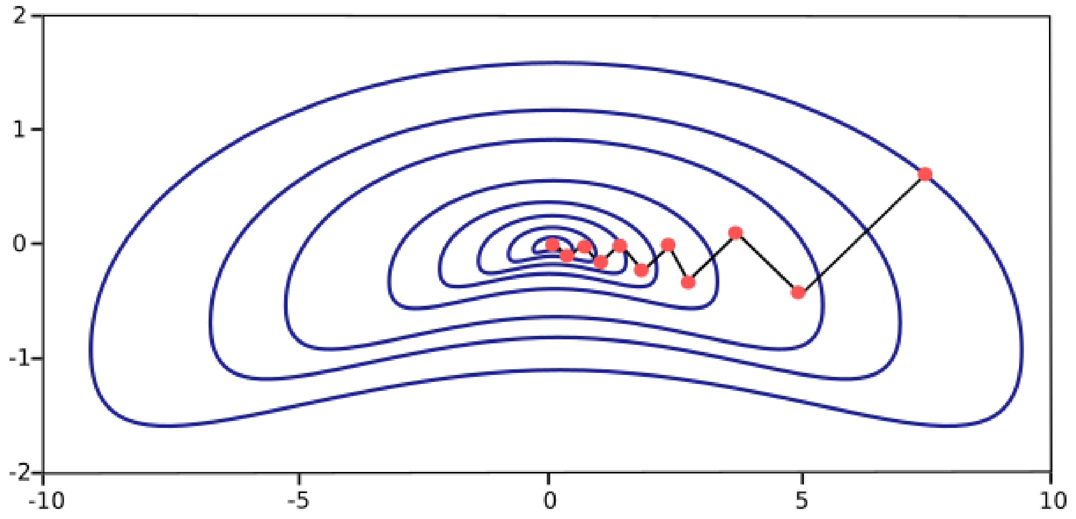
Optimization algorithms

Steepest descent (a.k.a. gradient descent) is based on a first-order ODE:

$$\frac{d\mathbf{w}}{d\tau} = -\frac{\partial E}{\partial \mathbf{w}}$$

Discretize in "time" τ:

$$\mathbf{w}(\tau + \delta\tau) = \mathbf{w}(\tau) - \delta\tau \left.\frac{\partial E}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(\tau)}$$

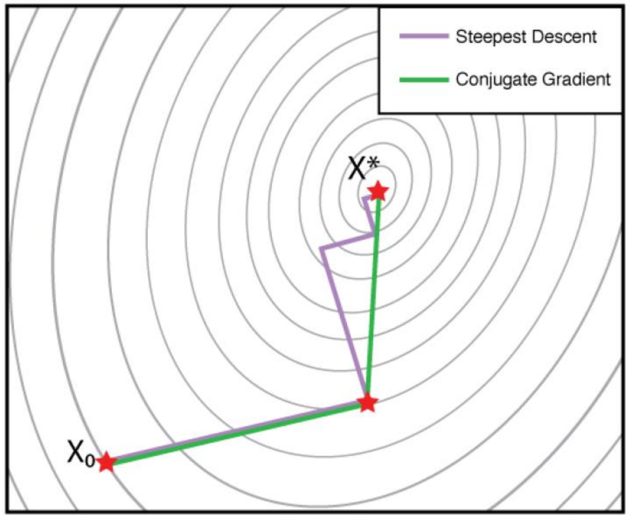


Conjugate gradient descent:

Local quadratic expansion of $E(\mathbf{w})$ about 0

$$E(\mathbf{w}) \approx \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathrm{H}\mathbf{w} - \mathbf{w}^{\mathrm{T}}\mathbf{F}$$

$$\nabla_{\mathrm{w}} E = 0 \quad \Rightarrow \quad \mathrm{H}\mathbf{w} = \mathbf{F}$$



Let $\mathbf{w}_*$ be the solution vector. Let $\mathbf{b}_k$ be vectors such that $\mathbf{b}_i^{\mathrm{T}}\mathrm{H}\mathbf{b}_j = \mathbf{b}_i^{\mathrm{T}}\mathrm{H}\mathbf{b}_i\delta_{ij}$

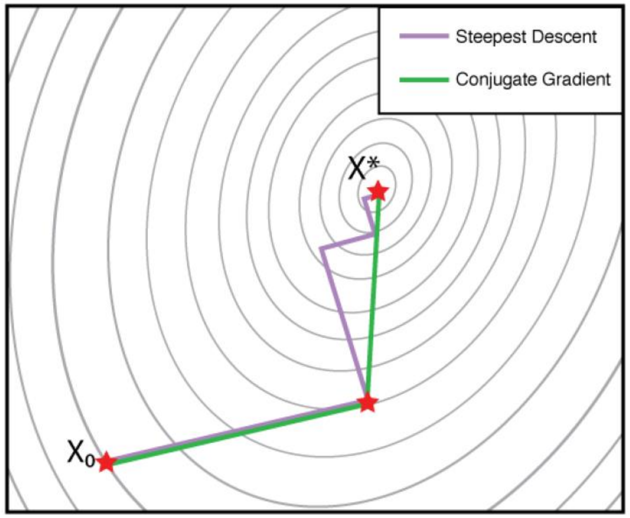
$$\mathbf{w}_* = \sum_k \beta_k \mathbf{b}_k$$

$$\mathrm{H}\mathbf{w}_* = \sum_k \beta_k \mathrm{H}\mathbf{b}_k$$

$$\mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{w}_* = \sum_k \beta_k \mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{b}_k = \beta_j \mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{b}_j$$

$$\beta_j = \frac{\mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{w}_*}{\mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{b}_j} = \frac{\mathbf{b}_j^{\mathrm{T}}\mathbf{F}}{\mathbf{b}_j^{\mathrm{T}}\mathrm{H}\mathbf{b}_j}$$

Conjugate gradient descent:



$$\mathbf{w}(i+1) = \mathbf{w}(i) + \alpha \frac{\mathbf{d}(i)}{|\mathbf{d}(i)|}$$

$$\mathbf{d}(0) = -\nabla_{\mathbf{w}} E\Big|_{\mathbf{w}(0)}$$

$$\mathbf{d}(i) = -\nabla_{\mathbf{w}} E\Big|_{\mathbf{w}(i)} + \beta \mathbf{d}(i-1)$$

$$\beta = \frac{\mathbf{d}^{\mathrm{T}}(i-1)\cdot\mathbf{H}\cdot\left(\nabla_{\mathbf{w}} E\Big|_{\mathbf{w}(i)}\right)}{\mathbf{d}^{\mathrm{T}}(i-1)\cdot\mathbf{H}\cdot\mathbf{d}(i-1)}$$

$$\mathbf{H}\cdot\mathbf{d}(i-1) \approx \frac{1}{\varepsilon}\left[\nabla_{\mathbf{w}} E\Big|_{\mathbf{w}(i)+\varepsilon\mathbf{d}(i-1)} - \nabla_{\mathbf{w}} E\Big|_{\mathbf{w}(i)}\right]$$

Using Langevin dynamics:

Create a probability distribution function of the $\mathbf{w}$ parameters:

$$P(\mathbf{w}) = \mathcal{N}^{-1} e^{-\beta E(\mathbf{w})}, \qquad \mathcal{N} = \int d\mathbf{w}\, e^{-\beta E(\mathbf{w})}$$

Sample using overdamped Langevin dynamics with a "temperature" β^{-1}

$$\gamma \mathrm{d}\mathbf{w} = -\nabla_{\mathbf{w}} E \mathrm{d}\tau + \sqrt{2\beta^{-1}\gamma}\,\mathrm{d}\boldsymbol{\eta}$$

$\boldsymbol{\eta}$ is a vector of Gaussian random numbers with distribution of zero-mean and unit-width.

Low-error algorithm [Matthews and Leimkuhler]:

$$\boxed{\mathbf{w}(\tau+\delta\tau) = \mathbf{w}(\tau) - \nabla_{\mathbf{w}} E(\mathbf{w}(\tau))\delta\tau + \sqrt{2\beta^{-1}\gamma\delta\tau}\left[\frac{\mathbf{r}(\tau)+\mathbf{r}(\tau+\delta\tau)}{2}\right]}$$

$\mathbf{r}(\tau)$ and $\mathbf{r}(\tau+\delta\tau)$ are vectors of Gaussian random numbers drawn from a distribution of zero mean and unit width.

Can also write this as a second-order dynamical system:

$$\mathrm{d}\mathbf{w} = \mathrm{M}^{-1}\mathbf{p}\,\mathrm{d}t$$

$$\mathrm{d}\mathbf{p} = -\nabla_{\mathbf{w}} E(\mathbf{w})\mathrm{d}t - \gamma\mathbf{p}\,\mathrm{d}t + \sqrt{2\beta^{-1}\gamma}\,\mathrm{M}^{1/2}\mathrm{d}\boldsymbol{\eta}$$

Let $\mathbf{F} = -\nabla_{\mathbf{w}} E(\mathbf{w}),\qquad \Sigma = \sqrt{2\beta^{-1}\gamma}\,\mathrm{M}^{1/2}$

<u>Low-error numerical integrator [Matthews and Leimkuhler (2012)]:</u>

```
$\mathbf{p} \leftarrow \mathbf{p} + 0.5 * \Delta t * \mathbf{F};$
$\mathbf{w} \leftarrow \mathbf{w} + 0.5 * \Delta t * \mathrm{M}^{-1}\mathbf{p};$
$\mathbf{p} \leftarrow \mathbf{p} * e^{-\gamma\Delta t} + \Sigma * \mathbf{R} * \sqrt{(1 - e^{-2\gamma\Delta t})/2\gamma};$
$\mathbf{w} \leftarrow \mathbf{w} + 0.5 * \Delta t * \mathrm{M}^{-1}\mathbf{p};$
Update Gradients;
$\mathbf{p} \leftarrow \mathbf{p} + 0.5 * \Delta t * \mathbf{F};$
```

Use of minibatches:

If the dataset if large, the evaluation of $\mathbf{F}(\mathbf{w}) = -\nabla_{\mathbf{w}} E(\mathbf{w})$ can be very expensive.

Define a minibatch of size $m < M$ and a “noisy” cost function and gradient

$$\tilde{E}(\mathbf{w}) = \frac{1}{2m} \sum_{\lambda=1}^{m} \left| f_{NN}\left(\mathbf{x}^{\lambda}; \mathbf{w}\right) - f_{\lambda} \right|^{2}, \qquad \tilde{\mathbf{F}}(\mathbf{w}) = -\nabla_{\mathbf{w}} \tilde{E}(\mathbf{w})$$

Assume the noisy gradient can be written as

$$\tilde{\mathbf{F}}(\mathbf{w}) = \mathbf{F}(\mathbf{w}) + \sqrt{\Sigma(\mathbf{w})}\mathrm{M}^{1/2}\mathbf{R}$$

where $\Sigma(\mathbf{w})$ is the (unknown) covariance matrix of the noisy gradient

Rewrite the stochastic sampling scheme as

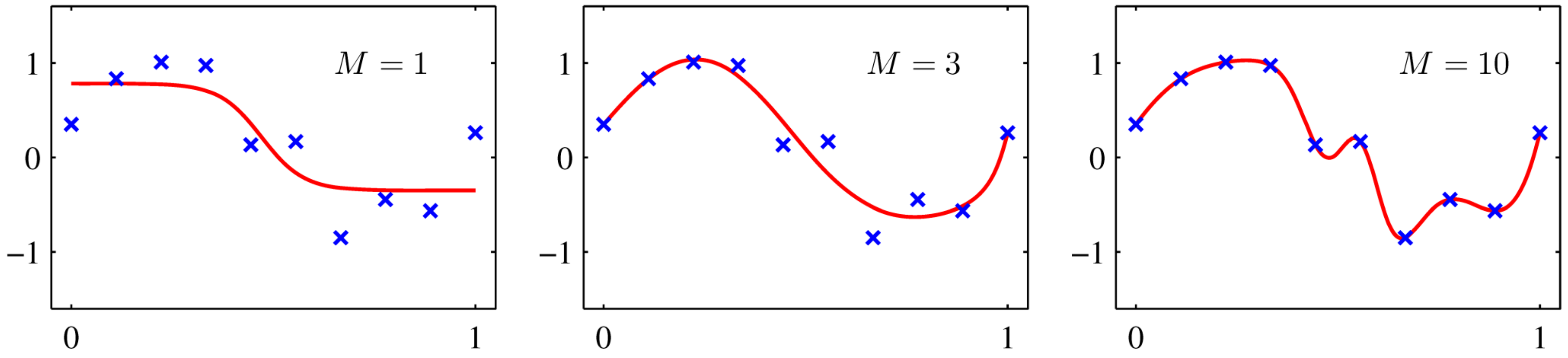
$$\mathrm{d}\mathbf{w} = \mathrm{M}^{-1}\mathbf{p}\,\mathrm{d}t$$

$$\mathrm{d}\mathbf{p} = \tilde{\mathbf{F}}(\mathbf{w})\mathrm{d}t - \sqrt{\Sigma(\mathbf{w})}\mathrm{M}^{1/2}\mathrm{d}\boldsymbol{\eta} - \gamma\mathbf{p}\,\mathrm{d}t + \sqrt{2\beta^{-1}\gamma}\,\mathrm{M}_{\mathrm{A}}^{1/2}\mathrm{d}\boldsymbol{\eta}_{\mathrm{A}}$$

1. Model $\Sigma(\mathbf{w})$, e.g., $\Sigma(\mathbf{w}) = \sigma\mathbf{I}$
2. Run minibatches in parallel and estimate $\Sigma(\mathbf{w})$ from the parallel runs.
3. Approximate update algorithms for $\Sigma(\mathbf{w})$ [Leimkuhler etal. *NIPS* (2015)].

Simple Example

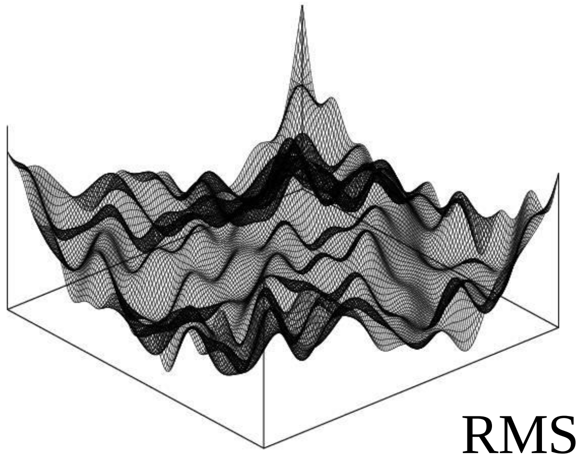
We take 10 “synthetic” data points drawn from the function $f(x) = \sin(2\pi x)$ to which random noise is added. We use these to train a network with a single hidden later having M units/nodes:



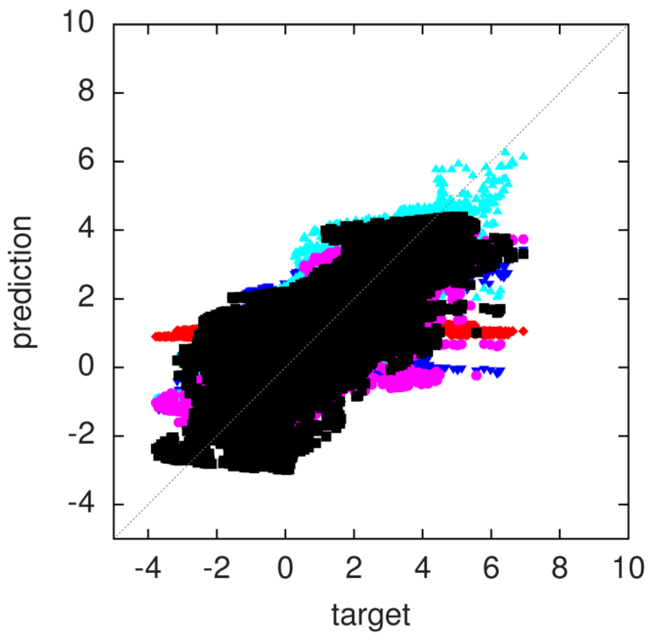
- $M = 1$: Insufficient to represent $f(x)$
- $M = 3$: Close approximation to $f(x)$
- $M = 10$: Overfitting of $f(x)$.

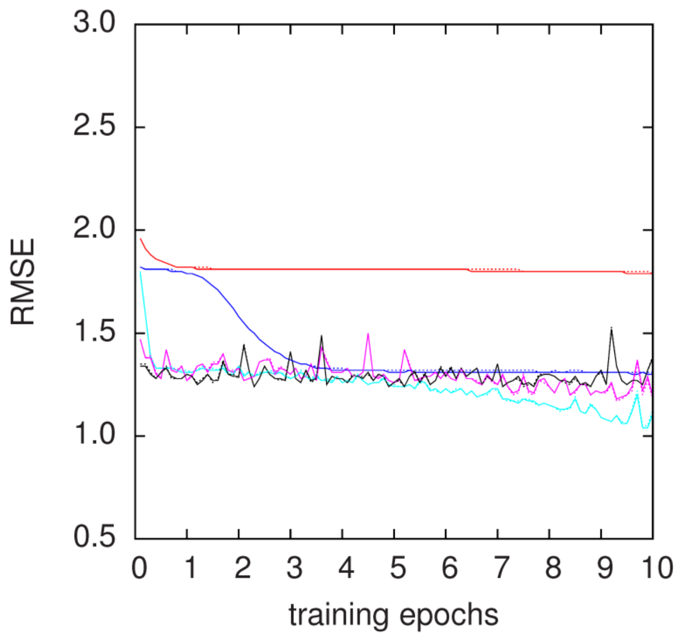
More complicated two-dimensional example

$f(x, y) =$



$$\text{RMSE} = \sqrt{\frac{1}{N_g}\sum_{\lambda=1}^{N_g}\left[f_{NN}(x^\lambda, y^\lambda; \mathbf{w}) - f(x^\lambda, y^\lambda)\right]^2}$$





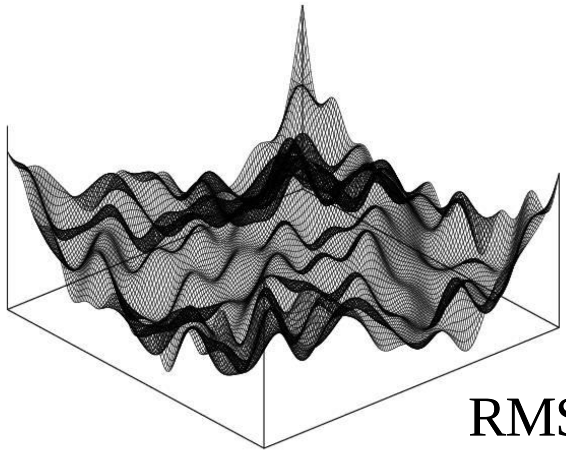
- One layer
- 10 nodes
- 10,000 training pts
- 41 parameters
- Steepest descent
- 1,000 validation pts

Minibatch sizes

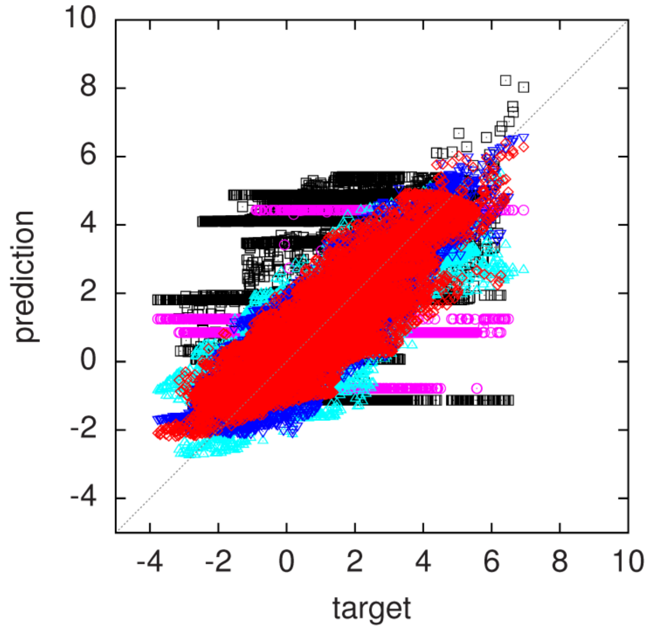
- Black = 1
- Pink = 10
- Turquoise = 100
- Blue = 1000
- Red = 10,000

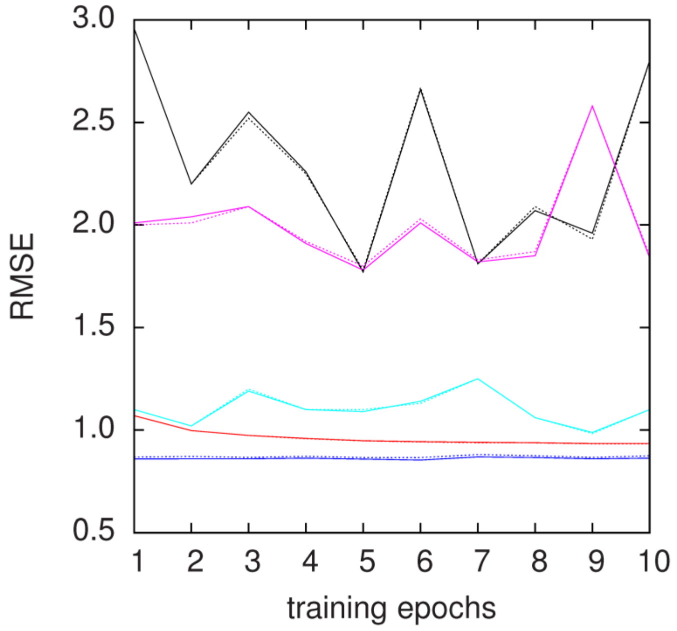
More complicated two-dimensional example

$$f(x,y) =$$



$$\mathrm{RMSE} = \sqrt{\frac{1}{N_g}\sum_{\lambda=1}^{N_g}\left[f_{NN}(x^\lambda, y^\lambda; \mathbf{w}) - f(x^\lambda, y^\lambda)\right]^2}$$





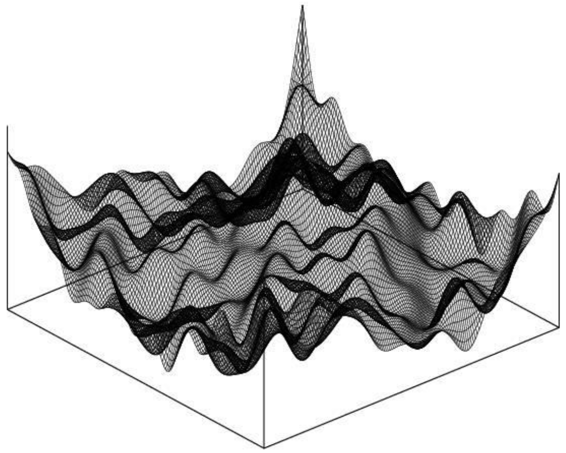
- One layer
- 10 nodes
- 10,000 training pts
- 41 parameters
- Conjugate gradient
- 1,000 validation pts

Minibatch sizes

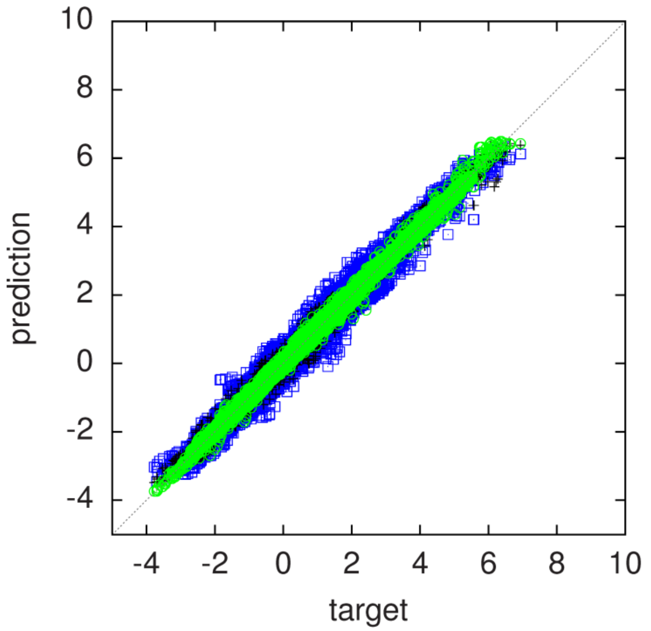
- Black = 1
- Pink = 10
- Turquoise = 100
- Blue = 1000
- Red = 10,000

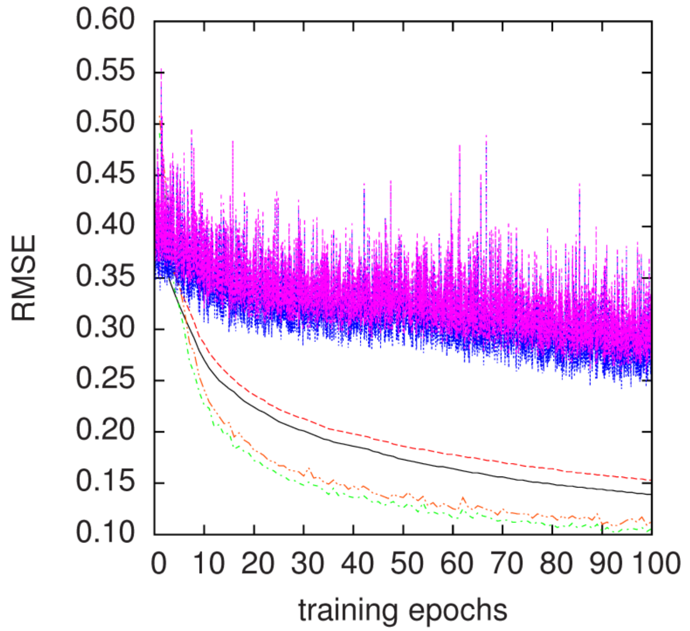
More complicated two-dimensional example

$f(x, y) =$



$$\text{RMSE} = \sqrt{\frac{1}{N_g}\sum_{\lambda=1}^{N_g}\left[f_{NN}(x^\lambda, y^\lambda; \mathbf{w}) - f(x^\lambda, y^\lambda)\right]^2}$$





- Two layers
- 20 nodes/layer
- 10,000 training pts
- 501 parameters
- 1,000 validation pts

<u>Minibatch sizes</u>

- Black = 10,000 CG
- Green = 1,000 CG
- Turquoise = 100
- Blue = 10,000 SD

Network training using gradients

Given M specific values of the function ∇f_λ, $\lambda = 1,\ldots, M$ at specific values $\mathrm{x}^\lambda \equiv x_1^\lambda,\ldots, x_n^\lambda$ training consists in using this data to fit the connection parameters $\mathbf{w}$.

In order to perform this training, we set up a gradient-based regression cost function:

$$\boxed{E_G(\mathbf{w}) = \frac{1}{2M}\sum_{\lambda=1}^{M}\left|\nabla f_{NN}\left(\mathbf{x}^\lambda;\mathbf{w}\right) - \nabla f_\lambda\right|^2}$$

The error function can also include a regularization term:

$$E_G(\mathbf{w}) = \frac{1}{2M}\sum_{\lambda=1}^{M}\left|\nabla f_{NN}\left(\mathbf{x}^\lambda;\mathbf{w}\right) - \nabla f_\lambda\right|^2 + \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

The cost function must then be minimized with respect to $\mathbf{w}$:

$$\nabla_{\mathbf{w}} E_G(\mathbf{w}) = 0$$

Analogous iterative back-propagation schemes can be derived for gradient training

Calculation of input derivatives

Recall our general definition

$$a_j^{(l)} = \begin{cases} \sum_{s=1}^{n} w_{js}^{(0)} x_s + w_{j0}^{(0)}, & l = 1 \\ \sum_{s=1}^{m^{(l-1)}} w_{js}^{(l-1)} h\left(a_s^{(l-1)}\right) + w_{j0}^{(l-1)}, & l = 2,\ldots,K \end{cases}$$

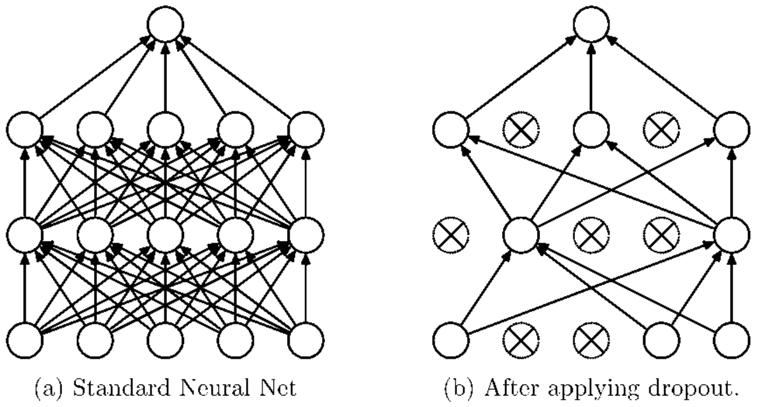
$$f(x_1,\ldots,x_n) = \sum_{s=1}^{m^{(K)}} h\left(a_s^{(K)}\right)$$

$$\frac{\partial a_j^{(l)}}{\partial x_r} \equiv y_{jr}^{(l)} = \begin{cases} w_{jr}^{(0)}, & l = 1 \\ \sum_{s=1}^{m^{(l-1)}} w_{js}^{(l-1)} h'\left(a_s^{(l-1)}\right) y_{jr}^{(l-1)}, & l > 1 \end{cases}$$

Final output: $$\frac{\partial f}{\partial x_r} = \sum_{j=1}^{m^{(K)}} h'\left(a_j^{(K)}\right) w_j^{(K)} y_{jr}^{(K-1)}$$

Dropout regulation of neural networks

Srivastava *et al. J. Mach. Learning Res.* (2014)



Let $r^{(l)}$ be a Bernoulli random number

$$\omega_s(x_p) = r^{(0)} x_p + c_s$$

$$a_s^{(1)} = \sum_{p=1}^{n} w_{sp}^{(0)} x_p + w_{s0}^{(0)}, \qquad z_s^{(1)} = r^{(1)} h\left(a_s^{(1)}\right)$$

$$a_r^{(2)} = \sum_{s=1}^{m''} \left(w_{rs}^{(1)} z_s^{(1)} + w_{r0}^{(1)}\right), \qquad z_r^{(2)} = r^{(2)} h\left(a_r^{(2)}\right)$$

$$a_q^{(3)} = \sum_{r=1}^{m'} \left(w_{qr}^{(2)} z_r^{(2)} + w_{q0}^{(2)}\right)$$

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{m} h\left(a_q^{(3)}\right)$$

Connection to Bayes' Theorem

Bayes' Theorem:

$$P(\mathbf{w}\,|\,\mathcal{D}) = \frac{P(\mathcal{D}\,|\,\mathbf{w})\,p(\mathbf{w})}{p(\mathcal{D})}$$

$$\text{posterior} \;\propto\; \text{likelihood} \times \text{prior}$$

Take the prior to be a Gaussian of width σ

$$p(\mathbf{w},\sigma) = \left(\frac{1}{2\pi\sigma^2}\right)^{D/2} e^{-\mathbf{w}^{\mathrm{T}}\mathbf{w}/2\sigma^2}$$

Take the likelihood function to be the Boltzmann distribution of the cost function:

$$P(\mathcal{D}\,|\,\mathbf{w},\beta) = \mathcal{N}^{-1}(\beta)e^{-\beta E(\mathbf{w};\mathcal{D})}, \qquad \mathcal{N}(\beta) = \int d\mathbf{w}\; e^{-\beta E(\mathbf{w};\mathcal{D})}$$

Connection to Bayes' Theorem

The posterior probability becomes:

$$P(\mathbf{w},\sigma,\beta\,|\,\mathcal{D}) = \mathcal{N}^{-1}(\beta)\left(\frac{1}{2\pi\sigma^2}\right)^{D/2} e^{-\frac{\beta}{2M}\sum_{\lambda=1}^{M}\left|f_{NN}\left(\mathbf{x}^{\lambda};\mathbf{w}\right)-f_{\lambda}\right|^2} e^{-\mathbf{w}^{\mathrm{T}}\mathbf{w}/2\sigma^2}$$

If we fix σ and β and take the −log of the posterior, we obtain

$$-\ln P(\mathbf{w},\sigma,\beta\,|\,\mathcal{D}) = \frac{\beta}{2M}\sum_{\lambda=1}^{M}\left|f_{NN}\left(\mathbf{x}^{\lambda};\mathbf{w}\right)-f_{\lambda}\right|^2 + \frac{1}{2\sigma^2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \text{const} \quad (1)$$

which is just the regularized cost function

We can also consider σ and β as additional parameters to be optimized, in which case, we have

$$-\ln P(\mathbf{w},\sigma,\beta\,|\,\mathcal{D}) = \frac{\beta}{2M}\sum_{\lambda=1}^{M}\left|f_{NN}\left(\mathbf{x}^{\lambda};\mathbf{w}\right)-f_{\lambda}\right|^2 + \frac{1}{2\sigma^2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \quad (2)$$

$$+ D\ln(\sigma) + \ln\mathcal{N}(\beta)$$

We do not know $\mathcal{N}(\beta)$ analytically, so the optimization cannot be performed directly.

The following procedure is, therefore, used:

1. Begin with an estimate of σ and β.
2. Optimize (1) to obtain a solution for $\mathbf{w}$, denoted $\mathbf{w}_{\text{opt}}$.
3. Use $\mathbf{w}_{\text{opt}}$ to expand the unregularized cost function to second order:

$$E(\mathbf{w}) = \frac{1}{2M}\sum_{\lambda=1}^{M}\left| f_{NN}\left(\mathbf{x}^{\lambda};\mathbf{w}\right) - f_{\lambda}\right|^2$$

$$\approx E_0 + \mathbf{b}^{\mathrm{T}}(\mathbf{w}-\mathbf{w}_{\text{opt}}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_{\text{opt}})^{\mathrm{T}}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\text{opt}})$$

4. Compute $\mathcal{N}(\beta)$ analytically from the second-order expansion.

$$\mathcal{N}(\beta) = \left(\frac{2\pi}{\beta}\right)^{D/2}[\det(\mathbf{A})]^{-1/2}e^{\beta^2\mathbf{b}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{b}/2}$$

5. Substitute into (2) and optimize with respect to σ and β

$$-\ln P(\mathbf{w},\sigma,\beta\,|\,\mathcal{D}) = \frac{\beta}{2M}\sum_{\lambda=1}^{M}\left| f_{NN}\left(\mathbf{x}^{\lambda};\mathbf{w}_{\mathrm{opt}}\right) - f_{\lambda}\right|^{2} + \frac{1}{2\sigma^{2}}\mathbf{w}_{\mathrm{opt}}{}^{\mathrm{T}}\mathbf{w}_{\mathrm{opt}}$$
$$+ D\ln(\sigma) + \ln\mathcal{N}(\beta)$$

6. Repeat from (2) until convergence is reached.

Such an approach is known as a *Bayesian neural network*.

Conclusions

1. Kolmorogov’s superposition theorem and Kurkova’s corollary form a rigorous basis for the neural network scheme

2. The nesting of functions and large data sets make the optimization of neural networks with respect to their parameters computationally expensive.

3. The iterative “back propagation” scheme improves the computational efficiency.

4. The data can also be processed in minibatches to improve the efficiency further.

5. In using neural networks, care must be taken to avoid overfitting, as with any other ML approach.

6. Gradient information can also be used to optimized a neural network.

7. Connecting neural networks to Bayes’ theorem leads to the Bayesian neural network approach.