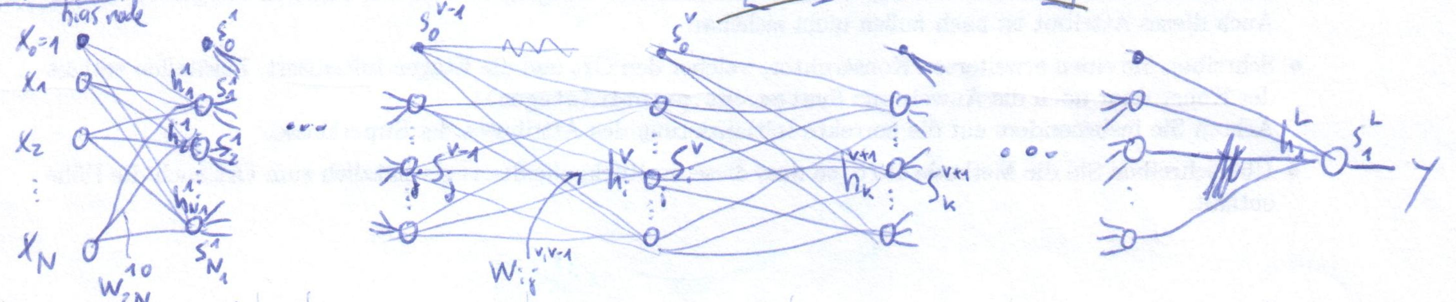
MLP: cont., backpropagation & symmetries

~~"will be discussed at the end"~~

a) Derivation of backpropagation (incl. architecture of MLP and gradient descent)

feed forward MLP: $y(\underline{x}) \in \mathbb{R}$ $\left[\text{or } \cancel{y(\underline{x}) \in \mathbb{R}^M}\right]$ for $\underline{x} \in \mathbb{R}^N$

1) Architecture



layer:	input	first hidden	...	v-1-th hidden	v-th hidden	v+1-th hidden ...	L-1-th hidden	output
index:	0	1		v-1	v	v+1	L-1	L
size (no. neurons)	N	N_1		N_{v-1}	N_v	N_{v+1}	N_{L-1}	
transfer function		f_1		f_{v-1}	f_v	f_{v+1}	f_{L-1}	f_L

often $f_v = f_{\text{hidden}}$ for $v = 1, \ldots, L-1$ ("tanh or rectLU" typical examples)

f_{output} (often: "linear, logistic sigmoid, softmax")

2) Evaluation by forward propagation

input → 1st layer:

$$S^1_2 = f_1(h^1_2) \quad \text{with} \quad h^1_2 = \sum_{k=0}^{N} x_k W^{10}_{2k}$$

from layer 0 to 1

output of neuron 2 in layer 1, input to that neuron

from neuron k (of layer 0) to neuron 2 (of layer 1)

In general: for each layer $v = 1, \ldots, L$:

$$\boxed{S^v_i = f_v(h^v_i) \quad \text{with} \quad h^v_i = \sum_{k=0}^{N_{v-1}} S^{v-1}_k W^{v,v-1}_{ik}, \quad i = 1, \ldots, N_v}$$

output of neuron k in layer v-1

including the input → 1st layer by $S^0_k = x_k$, $k = 0, \ldots, N = N_0$; bias nodes: $S^v_0 = 1 \ \forall v$

Output: $\underline{y = S^L_1 = f_L(h^L_1)}$ with $h^L_1 = \sum_{k=0}^{N_{L-1}} S^{L-1}_k W^{L,L-1}_{1k}$

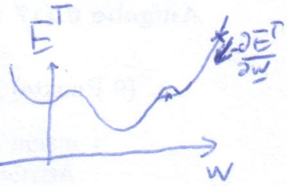
→ MLP: function $y(\underline{x}; \underline{w})$ parametrized by $\underline{w} = \left\{ W^{v,v-1}_{ik} \right\}_{k=0,\ldots,N_{v-1}}^{v=1,\ldots,L;\ i=1,\ldots,N_v}$

next: how to learn parameter values $\underline{w}$ from data $\left\{ \left(\underline{x}^{(\alpha)}, y_T^{(\alpha)}\right) \right\}_{\alpha=1}^{p}$

MLP: ~~st~~ arch., backprop., symmetries (cont'd)

3) Parameter optimization through gradient descent

Aim: find optimal parameter values $\underline{w}$ by minimizing the training cost:



$$E^T(\underline{w}) = \frac{1}{p}\sum_{\alpha=1}^{p} e^{(\alpha)}(\underline{w}) \overset{!}{=} \min_{\underline{w}}$$

Typical (in neural networks) approach: descent the gradient, stopping at stationary point $\frac{\partial E^T}{\partial \underline{w}}(\underline{w}) = \underline{0}$:

initialize weights with $\underline{w}^{(0)}$

learning step

until convergence: $\underline{w}^{(n+1)} = \underline{w}^{(n)} - \eta \frac{\partial E^T}{\partial \underline{w}}(\underline{w}^{(n)})$

descent

linearity of gradient

$= \underline{w}^{(n)} - \eta \frac{1}{p} \sum_{\alpha=1}^{p} \frac{\partial e^{(\alpha)}}{\partial \underline{w}}(\underline{w}^{(n)})$

gradient of error of datapt. α

→ per iteration n, per data point α required:

$$\cancel{\frac{\partial e^{(\alpha)}}{\partial \underline{w}} = \frac{\partial e^{(\alpha)}}{\partial y}\frac{\partial y}{\partial \underline{w}}}$$

:)

$$\frac{\partial e^{(\alpha)}}{\partial \underline{w}}\left(y(\underline{x}^{(\alpha)};\underline{w}), y_T^{(\alpha)}\right) = \frac{\partial e^{(\alpha)}}{\partial y} \cdot \frac{\partial y}{\partial \underline{w}}(\underline{x}^{(\alpha)};\underline{w})$$

dep. on cost fct., e.g. :)

$$\frac{\partial e^{(\alpha)}}{\partial y} = \frac{\partial}{\partial y}\left(\frac{1}{2}\left[y(\underline{x}^{(\alpha)};\underline{w}) - y_T^{(\alpha)}\right]^2\right) = y(\underline{x}^{(\alpha)};\underline{w}) - y_T^{(\alpha)}$$

cross entropy: homework

dep. on neural network

Backpropagation algorithm: efficient evaluation of ↑ using the chain rule: runtime complexity $O(\#\text{weights})$ instead of $O(\#\text{weights}^2)$!

$$\#\text{weights} = N_1(N+1) + \sum_{v=2}^{L} N_v(N_{v-1}+1) = 10N^2 + 10N$$

e.g. $L=10$, $N_v = N$

MLP: arch., backprop, sym. (cont'd)

4) Backpropagation algorithm

3 cases: - "normal" weight
- input weight
- bias node weight

component-wise (simplest):

$$\frac{\partial y}{\partial w_{ij}^{v,v-1}} = \underbrace{\frac{\partial y}{\partial h_i^v}}_{\delta_i^v :=} \underbrace{\frac{\partial h_i^v}{\partial w_{ij}^{v,v-1}}}_{= s_j^{v-1}} = \delta_i^v s_j^{v-1} = \begin{cases} \delta_i^v f_{v-1}(h_j^{v-1}) & , v=2,\dots,L \\ \delta_i^v x_j & , v=1 \\ \delta_i^v & , j=0 \end{cases} = \begin{cases} \delta_i^v & , j=0 \\ \delta_i^v f_{v-1}(h_j^{v-1}) & \text{else} \end{cases}$$

by def $h_j^0 := x_j$ and $f_0 := id$

identity function $id(x)=x$

"local error of neuron (i,v)":

$$\delta_i^v = \frac{\partial y}{\partial h_i^v} = \frac{\partial y}{\partial s_i^v} \underbrace{\frac{\partial s_i^v}{\partial h_i^v}}_{=f_v'(h_i^v)} = f_v'(h_i^v) \cdot \sum_{k=1}^{N_{v+1}} \underbrace{\frac{\partial y}{\partial h_k^{v+1}}}_{=\delta_k^{v+1}} \frac{\partial h_k^{v+1}}{\partial s_i^v} = f_v'(h_i^v) \sum_{k=1}^{N_{v+1}} \delta_k^{v+1} w_{ki}^{v+1,v}$$

$y\left(h_1^{v+1}(s_i^v),\dots,h_{N_{v+1}}^{v+1}(s_i^v)\right)$ general chain rule

$\rightarrow$ recursive relation of local errors: $\{\delta_k^{v+1}\}_{k=1}^{N_{v+1}}$ known $\Rightarrow \{\delta_i^v\}_{i=1}^{N_v}$ can be calculated (i.e. backpropagation of local errors)

backprop algorithm:

given: $\underline{x}, y_T, \underline{w}$

define: $h_k^0 = x_k$ for $k=0,\dots,N$; $f_0 = id$; $N_0 = N$

forward propagation:

for $v = 1,2,\dots,L$ do

$\cancel{h_i^v = \sum_{k=0}^{N_{v-1}} w_{ik}^{v,v-1} f_{v-1}(h_k^{v-1})}$, $\cancel{i=1,\dots,N_v}$, $\cancel{=1 \text{ for } k=0}$

more precisely

$$h_i^v = \sum_{k=1}^{N_{v-1}} f_{v-1}(h_k^{v-1}) w_{ik}^{v,v-1} + w_{i0}^{v,v-1}$$

local error at output neuron & output

$$\delta_1^L = f_L'(h_1^L)$$
$$y = f_L(h_1^L)$$

backward propagation:

for $v = L-1,\dots,1$ do

$$\delta_i^v = f_v'(h_i^v) \sum_{k=1}^{N_{v+1}} \delta_k^{v+1} w_{ki}^{v+1,v} \quad , i=1,\dots,N_v$$

gradient of network output:

$$\boxed{\frac{\partial y}{\partial w_{ij}^{v,v-1}} = \cancel{\delta_i^v f_{v-1}(h_j^{v-1})} \quad , v=1,\dots,L \quad , i=1,\dots,N_v \quad , j=0,\dots,N_{v-1}}$$

more precisely

$$\frac{\partial y}{\partial w_{ij}^{v,v-1}} = \begin{cases} \delta_i^v f_{v-1}(h_j^{v-1}) & , \text{else} \\ \delta_i^v & , j=0 \end{cases}$$

MLP: acts, backprop, symmetries (cont'd)

b) Consequences of parameterspace symmetries

i) permutation of ~~weight~~ $\boxed{\text{neuron}}$ indices within a layer

Let $\underline{w}$ be given parameter vector, then $\underline{\tilde{w}}$ will make the same cost E^T as $\underline{w}$

if $\quad \tilde{w}^{v,v-1}_{ij} = w^{v,v-1}_{\pi(i),j}$ for some permutation π, and $\tilde{w}^{v+1,v}_{k,i} = w^{v+1,v}_{k,\pi(i)}$

$\rightarrow$ neuron $\pi(i)$ $(\underline{w})$ $\rightarrow$ i $(\underline{\tilde{w}})$

$\Rightarrow$ $N_v!$ permutations per hidden layer v

ii) sign reversal across layers

let $\underline{w}$ be given parameters, then $\underline{\tilde{w}}$ will produce a model with same cost.

if $\quad \tilde{w}^{v,v-1}_{i,j} = -w^{v,v-1}_{i,j}$ and $\tilde{w}^{v+1,v}_{ki} = -w^{v+1,v}_{ki}$ and $f_{\text{hidden}} = \tanh$

(follows from $\tanh(-h) = -\tanh(h)$)

$\Rightarrow$ 2^{N_v} combinations per hidden layer v

$\Rightarrow$ overall $\prod_{v=1}^{L} N_v!\, 2^{N_v}$ equivalent solution

(also for global or local minima true)

$\rightarrow$ no unique optimal parameters! but at least $\prod_{v=1}^{L} N_v!\, 2^{N_v}$ equivalent solutions (same cost)