Lecture 16

Neural networks

Idea: make basis functions weight-dependent and fit those weights.

Previously, we considered

$$y(\vec{x}, \vec{w}) = f\Big(\sum_{j=1}^{M} w_j \varphi_j(\vec{x})\Big) \text{, where}$$

$f(\cdot)$ is identity for regression and non-linear activation f'n for classification.

Now, consider $(x_1, \dots, \overset{\swarrow \text{ input vector}}{x_D})$

$$\underset{\substack{\uparrow\\ \text{activation}}}{a_j} = \sum_{i=1}^{D} \underset{\substack{\uparrow\\ \text{weights}}}{w_{ji}^{(1)}} x_i + \underset{\substack{\uparrow\\ \text{bias}}}{w_{j0}^{(1)}} \qquad j = 1, \dots, M$$

Then, $z_j = \underset{\substack{\uparrow\\ \text{non-linear activation f'n}}}{h}(a_j)$

$h(\cdot)$ can be $\sigma(\cdot)$ or $\tanh(\cdot)$

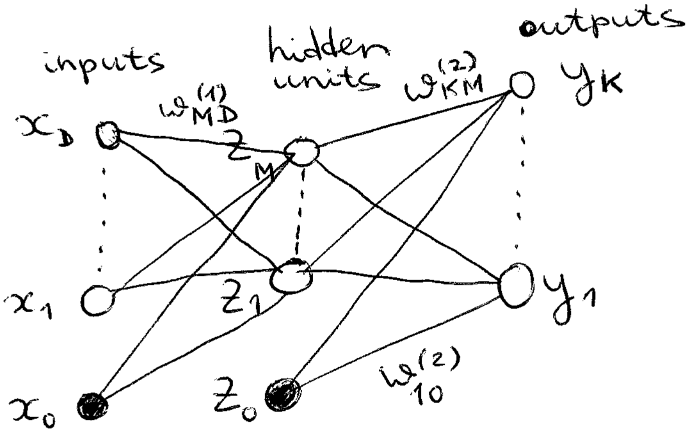
Next, $$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad k = 1, \dots, \underset{\substack{\uparrow\\ \text{total \# outputs}}}{K}$$

Finally, $\underset{\substack{\nearrow\\ \text{output vector}}}{y_k} = \tilde{h}(a_k)$, where $\tilde{h}$ may be $\overset{\substack{K=1\\ \text{or } K>1\\ \downarrow}}{}$ identity for regression, $\sigma(\cdot)$ for binary classification $(K=2)$, etc.

We have:

$$y_k(\vec{x},\vec{w}) = \sigma\Big(\sum_{j=1}^{M} w_{kj}^{(2)} \overbrace{h\Big(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\Big)}^{z_j,\ j=1,\dots,M} + w_{k0}^{(2)}\Big)$$

Note that it does not make sense to have $h(\cdot)$ as identity, since then the argument of the σ-function is just a linear model with various products of weights.



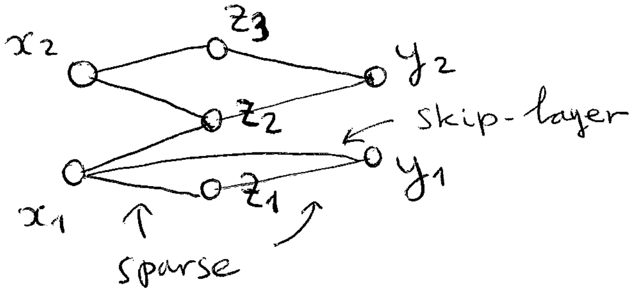
Define $x_0 = 1$ & $z_0 = 1$, then

$$y_k(\vec{x},\vec{w}) = \sigma\Big(\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{z_j}_{=\begin{cases}1, & j=0\\ h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right), & j=1,\dots,M\end{cases}}\Big)$$

<u>Generalizations</u>: (1) multiple layers of hidden units

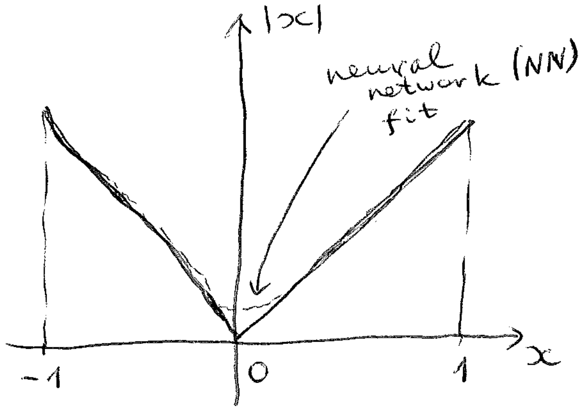
(2) sparse network architectures

(3) skip-layer connections:

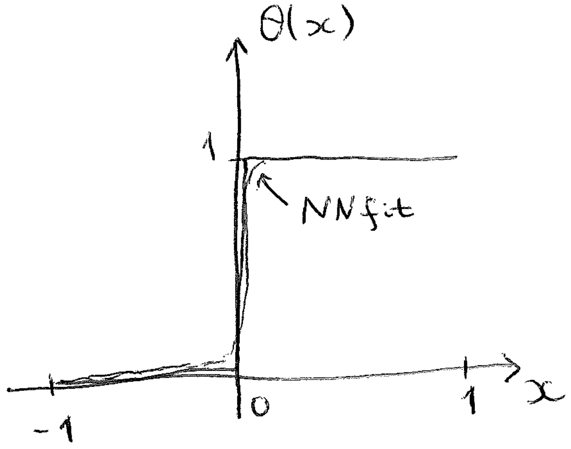


Note: feed-forward architectures <u>only</u>

Performance: can fit various functions fairly accurately



Sample $N = 50$ datapoints uniformly in $[-1, 1]$ interval, fit a two-layer network (input layer + hidden layer) discussed above:
3 hidden units,
$\tanh(\cdot)$ activation f'n,
linear output units.



Weight-space symmetries:

with $\tanh(\cdot)$ activation f's,

$\tanh(-a) = -\tanh(a)$, and

changing the sign of all weights (and the bias) leading into a unit can be compensated by the change in sign of all weights leading out of that unit.

M hidden units → 2^M equivalent weight vectors.

Similarly, can exchange weight values leading in and out of a hidden unit with another hidden unit ⇒ $M!$ permutations

So we have $2^M M!$ symmetries for a two-layer network (can be easily generalized to other architectures and activation functions)

Network training

$n=1,\ldots,N$: $\{\vec{x}_n\}$ $\{\vec{t}_n\}$

↑ inputs ↑ targets

Regression: (consider scalar targets $\{t_n\}$ for simplicity)

Assume $p(t|\vec{x},\vec{w},\beta) = \mathcal{N}(t|y(\vec{x},\vec{w}),\beta^{-1})$

β = precision $\left(=\frac{1}{\sigma^2}\right)$

Output unit activation function = identity ; single output unit.

As before,

$$\mathcal{L} = \prod_{n=1}^{N} p(t_n|\vec{x}_n,\vec{w},\beta) \quad , \text{ or}$$

$$\underset{\underset{\text{error f'n}}{\|}}{-\log\mathcal{L}} = \frac{\beta}{2}\sum_{n=1}^{N}(y(\vec{x}_n,\vec{w})-t_n)^2 + \frac{N}{2}\log(2\pi) - \frac{N}{2}\log\beta .$$

Define $E(\vec{w}) = \frac{1}{2}\sum_n (y(\vec{x}_n,\vec{w})-t_n)^2$,

then $\left.\frac{\partial E}{\partial \vec{w}}\right|_{\vec{w}_{ML}} = 0$ gives $\underline{\underline{\vec{w}_{ML}}}$.

ML approach ↗

[or just minimizing $E(\vec{w})$]

↑ need numerical minimizer

given $\vec{w}_{ML}$, $\left.\frac{\partial E}{\partial \beta}\right|_{\beta_{ML}} = 0$ yields

$$\beta_{ML}^{-1} = \frac{1}{N} \sum_n \left(y(\vec{x}_n, \vec{w}_{ML}) - t_n \right)^2 .$$

<u>K=2</u> <u>classification</u>:

$$C_1 : t = 1 \qquad C_2 : t = 0$$

as before, use $\sigma(\cdot)$ on the single output node, and define

$$\begin{cases} p(C_1|\vec{x}) = \underbrace{y(\vec{x}, \vec{w})}_{[0,1]} \\ p(C_2|\vec{x}) = 1 - p(C_1|\vec{x}) \end{cases}$$

Then $\mathcal{J} = \prod_n y_n^{t_n} (1 - y_n)^{1-t_n}$, where $y_n = y(\vec{x}_n, \vec{w})$;

$$E(\vec{w}) = -\log \mathcal{J} = -\sum_n \left[t_n \log y_n + (1-t_n) \log(1-y_n) \right].$$

If we have $\mathcal{N}$ separate binary classifications to perform, $\mathcal{N}$ output nodes

$$t_k = \{0, 1\}, \quad k = 1, \dots, \mathcal{N}$$

$$\mathcal{J} = \prod_{n=1}^{N} \prod_{k=1}^{\mathcal{N}} y_{nk}^{t_{nk}} (1 - y_{nk})^{1-t_{nk}},$$ where $y_{nk} = y_k(\vec{x}_n, \vec{w})$.

$$E(\vec{w}) = -\log \mathcal{J} = -\sum_n \sum_k \left[t_{nk} \log y_{nk} + (1 - t_{nk}) \log(1 - y_{nk}) \right]$$

K>2 classification:

1-of-K coding scheme: $\underset{\uparrow \text{class label}}{\overbrace{00\ldots1\ldots}^{K}}$

Then interpret $y_k(\vec{x},\vec{w}) = p(t_k=1|\vec{x})$ ⇐ K output nodes

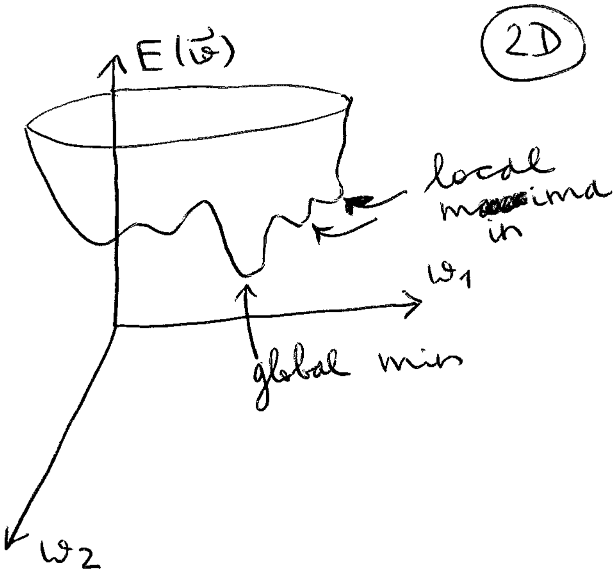
$$\mathcal{L} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}} \quad , \text{ or}$$

$$E(\vec{w}) = -\sum_{n}\sum_{k} t_{nk}\log y_{nk}$$

Output unit activation function =

= softmax : $y_k(\vec{x},\vec{w}) = \dfrac{e^{a_k(\vec{x},\vec{w})}}{\sum_{j=1}^{K} e^{a_j(\vec{x},\vec{w})}}$.

Typically, $E(\vec{w})$ is non-trivial:



Prm optimization

If $\vec{\nabla} E(\vec{w})$ is available, could make small steps in the $-\vec{\nabla} E$ direction until $\vec{\nabla} E = 0$ $\Rightarrow$ steepest descent method.

This will only find a local min in general $\Rightarrow$ could run multiple times & compare the results (i.e., get the best one).

Besides, there're symmetries in weight space (e.g. $2^M M!$ equivalent minima in a 2-layer network with M hidden units).

Local quadratic approximation:

$$E(\vec{w}) \simeq E(\hat{\vec{w}}) + (\vec{w}-\hat{\vec{w}})^T \vec{b} + \frac{1}{2} (\vec{w}-\hat{\vec{w}})^T H (\vec{w}-\hat{\vec{w}})$$

↑ Taylor expansion around $\hat{\vec{w}}$

$$\vec{b} = \vec{\nabla} E \Big|_{\hat{\vec{w}}} , \quad H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \Big|_{\hat{\vec{w}}} . \quad [H^T = H]$$

Then $\vec{\nabla} E_{(\vec{w})} \simeq \vec{b} + H(\vec{w}-\hat{\vec{w}})$

If $\hat{\vec{w}} = \vec{w}^*$ is a minimum, s.t. $\vec{b} = 0$, we get:

$$E(\vec{w}) \simeq E(\vec{w}^*) + \frac{1}{2} (\vec{w}-\vec{w}^*)^T H (\vec{w}-\vec{w}^*)$$

Consider $H\vec{u}_i = \lambda_i \vec{u}_i$ s.t. $\vec{u}_i^T \vec{u}_j = \delta_{ij}$

Now, expand $\vec{w} - \vec{w}^* = \sum_i \alpha_i \vec{u}_i$, s.t.

$$E(\vec{w}) \simeq E(\vec{w}^*) + \frac{1}{2} \sum_{kl} (w_k - w_k^*) H_{kl} (w_l - w_l^*) =$$

$$= E(\vec{w}^*) + \frac{1}{2} \sum_{k} \sum_{ij} \alpha_i \alpha_j u_{i,k} \underbrace{\sum_l H_{kl} u_{j,l}}_{\lambda_j u_{j,k}} =$$

$$= E(\vec{w}^*) + \frac{1}{2} \sum_{ij} \alpha_i \alpha_j \lambda_j \underbrace{\sum_k u_{i,k} u_{j,k}}_{\delta_{ij}} =$$

$$= E(\vec{w}^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

H is pos.-definite iff $\vec{v}^T H \vec{v} > 0$, $\forall \vec{v}$.

Using $\vec{v} = \sum_i c_i \vec{u}_i$, we obtain:

$$\vec{v}^T H \vec{v} = \sum_{ij} v_i H_{ij} v_j = \sum_{ij} \left(\sum_k c_k u_{k,i} \right) \times$$

$$\times H_{ij} \left(\sum_l c_l u_{l,j} \right) =$$

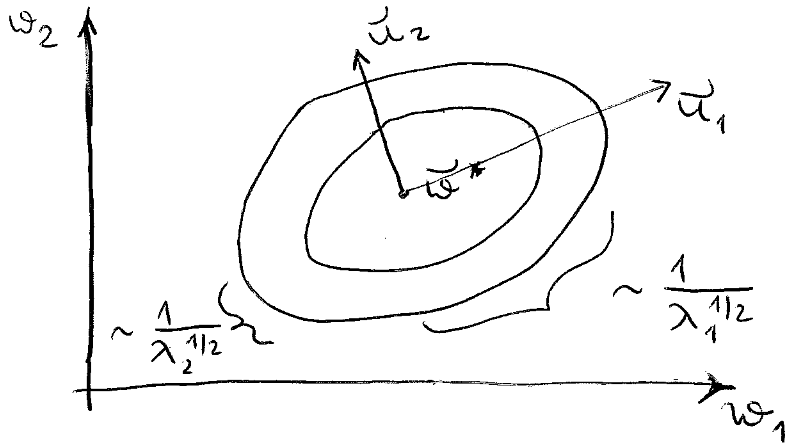
$$= \sum_{kl} c_k c_l \lambda_l \underbrace{\left(\sum_i u_{k,i} u_{l,i} \right)}_{\delta_{kl}} = \sum_l c_l^2 \lambda_l .$$

$\uparrow$

$$\sum_j H_{ij} u_{l,j} = \lambda_l u_{l,i}$$

Thus H is pos-def iff $\lambda_i > 0$, $\forall i$.
This is a requirement for $\vec{w}^*$ to be a minimum, rather than a max or a saddle point.

Contours of $E(\vec{w})$:


w_2
$\vec{u}_2$
$\vec{u}_1$
$\vec{w}^*$
$\sim \frac{1}{\lambda_1^{1/2}}$
$\sim \frac{1}{\lambda_2^{1/2}}$
w_1

Suppose $E(\vec{w}) - E(\vec{w}^*) = \Delta$, then

$$E(\vec{w}) - E(\vec{w}^*) = \frac{1}{2} \sum_i \lambda_i \alpha_i^2 = \Delta .$$

If $\vec{w} = \vec{w}^* + \underbrace{\alpha_i \vec{u}_i}_{\text{along } \vec{u}_i}$ $\Rightarrow 2\Delta = \lambda_i \alpha_i^2$, or $\alpha_i \sim \frac{1}{\lambda_i^{1/2}}$.

Steepest descent:

$$\vec{w}^{(\overset{\text{step \#}}{\tau+1})} = \vec{w}^{(\tau)} - \overset{\text{learning rate}}{\eta} \vec{\nabla} E(\vec{w}^{(\tau)})$$

Note that $E(\vec{w}) = \sum_{n=1}^{N} E_n(\vec{w})$, so that, alternatively, we can do

$$\vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} - \eta \vec{\nabla} E_n(\vec{w}^{(\tau)})$$

↑ sequential gradient descent
[cycle through datapoints]

This is a diff. algorithm b/c a local min for the whole dataset ≠ local min for each individual datapoint.