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,

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

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

($-\log \mathcal{L}$ = error f'n)

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 $\vec{w}_{ML}$.

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

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

K = 2 classification:

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

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

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

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

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

$K>2$ classification:

1-of-K coding scheme: $\overbrace{0\,0\,\ldots\,1\,\ldots}^{K}$ ($\uparrow$ class label)

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

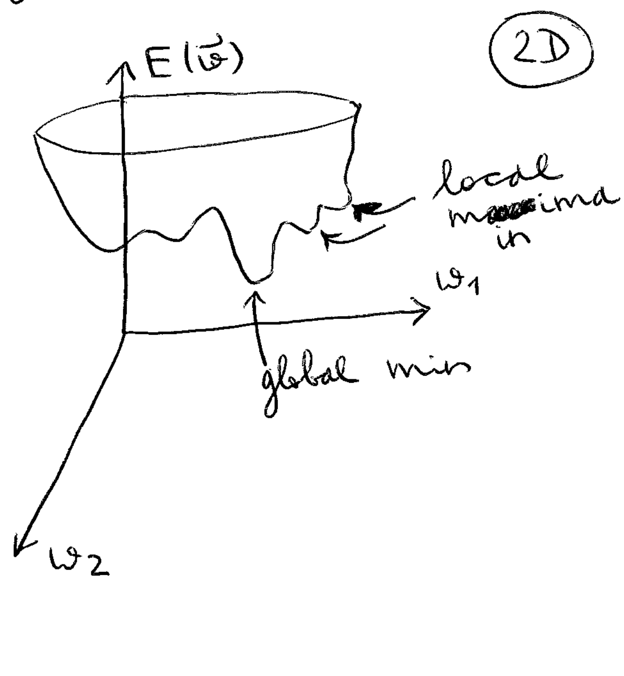
$$\mathcal{L} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}, \quad \text{or}$$

$$E(\vec{w}) = -\sum_n \sum_k t_{nk} \log y_{nk}$$

Output unit activation function = softmax:

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

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 $\hat{\vec{w}}$

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

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.

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

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

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

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

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:

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

$$\times H_{ij} \left(\sum_\ell c_\ell u_{\ell,j} \right) =$$

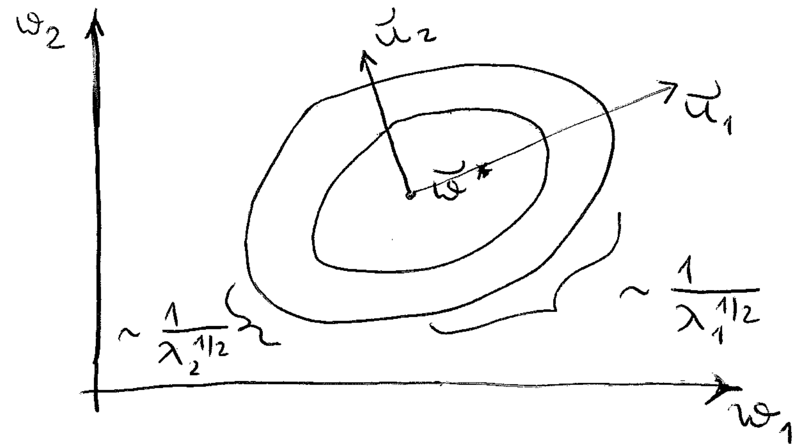
$$= \sum_{k\ell} c_k c_\ell \lambda_\ell \underbrace{\left(\sum_i u_{k,i} u_{\ell,i} \right)}_{\delta_{k\ell}} = \sum_\ell c_\ell^2 \lambda_\ell.$$

$\uparrow$ $\sum_j H_{ij} u_{\ell,j} = \lambda_\ell u_{\ell,i}$

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 $E(\vec{w})$:



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:

(step #; learning rate)

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

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.