# VII Machine learning

So far we tried to design algorithms & ansatz classes to either estimate observables or to directly represent/approx. the state. Machine learning is (in some sense) a combination of both ideas.

Consider a system which may be controlled by a set of parameters $\underline{x} \in \mathbb{R}^n$ $(n \in \mathbb{N})$. An experiment is then the measured response of the system $\underline{y} \in \mathbb{R}^m$ to some exp. procedure. Let us assume their exists a function which predicts the outcome $\underline{y}$, given $\underline{x}$ : $h(\underline{x}) = \underline{y}$. Clearly, $h$ depends on the systems properties, which specify its state.

In the following, we assume that the state can be parametrized by a set of variables $\underline{\vartheta} \in \mathbb{R}^k$ $(k \in \mathbb{N})$. Then we can pose the following question:

Can we find $h_{\underline{\vartheta}}(\underline{x})$ & a set $\underline{\vartheta}_0$ such, that $\text{dist}(h_{\underline{\vartheta}}(\underline{x}), h(\underline{x})) \stackrel{!}{=} \min$ ?

Note:

(i) The function $h_{\underline{\vartheta}}(x)$ introduces a parametrization of the functional dependency between $\underline{x}$ & $y$, i.e., a model.

(ii) The set of variables $\underline{\vartheta}_0$ then tries to fit the data $\underline{x}$ to $y$ given the model $h_{\underline{\vartheta}}(x)$ as best as possible.

# VII.1 Supervised learning

Let us consider an example: The price of a house. Let $\underline{x}$ be information about houses, e.g.:

- the squared area $\hat{=}$ $x_1$
- the nr. of bed rooms $\hat{=}$ $x_2$

A naive model for the predicted price $y = h_{\underline{\vartheta}}(\underline{x})$ would be:

$$h_{\underline{\vartheta}}(\underline{x}) = \underline{\vartheta}^t \cdot \underline{x} = \vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2$$

with $\underline{\vartheta} = (\vartheta_0 \ \vartheta_1 \ \vartheta_2)$ & $\underline{x} = (1 \ x_1 \ x_2)$.

Questions:

(i) How to optimize $\underline{\vartheta}$ to get $\mathrm{dist}(h_{\underline{\vartheta}}(x), h(\underline{x}))$ minimal?

(ii) Why should $h_{\underline{\vartheta}}(\underline{x})$ be a good model after all ?

## Regression methods (a small excerpt)

Machine learning is, to a very high degree, solving optimization problems, so we need a tool-box!

We consider a set of realizations $(\underline{x}^{(m)}, y^{(m)})$ with $m \in \{1, \dots, M\}$ & try to fit from this "training set" the optimal model parameter $\underline{\vartheta}_{\text{opt}}$.

### Least mean squares (LMS)

We consider the cost function:

$$J(\underline{\vartheta}) = \frac{1}{2} \sum_{i=1}^{n} \left( h_{\underline{\vartheta}}(\underline{x}^{(m)}) - y^{(m)} \right)^2$$

& use gradient descent to generate a seq. of model parameter $\underline{\vartheta}_k$ :

$$\underline{\vartheta}_{k+1} = \underline{\vartheta}_k - \alpha \underline{\nabla}_{\underline{\vartheta}} J \big|_{\underline{\vartheta}_k}$$

where $\alpha > 0$ is called learning rate.

The update rule can be rewritten using

$$\underline{\nabla}_{\underline{\vartheta}} J = \sum_{j=1}^{n} \underline{e}_j \frac{\partial J}{\partial \vartheta_j}$$

$$= \sum_{j=1}^{n} \underline{e}_j \frac{\partial}{\partial \vartheta_j} \frac{1}{2}\left(\underline{\vartheta}^t \cdot \underline{X} - Y\right)\left(\underline{\vartheta}^t \cdot \underline{X} - Y\right)$$

$$= \sum_{j=1}^{n} \underline{e}_j \left(\underline{\vartheta}^t \cdot \underline{X} - Y\right) X_j .$$

For the m-the realization we then choose:

$$\underline{\vartheta}_{m+1} = \underline{\vartheta}_m + \alpha \left( Y^{(m)} - \underline{\vartheta}^t \cdot \underline{X}^{(m)} \right)^t \cdot \underline{X}^{(m)}$$

This can be repeatedly applied to the M training samples $(\underline{X}^{(m)}, Y^{(m)})$ at cost $\sim \mathcal{O}(n)$ per iteration!

Remarks:

- The updates $\underline{\vartheta}_m \to \underline{\vartheta}_{m+1}$ can be defined by iterating over all $M$ samples from the training set (batch gradient descent):

$$J(\underline{\vartheta}) = \frac{1}{2}\sum_{m=1}^{M}\left(y^{(m)} - \underline{\vartheta}^t \cdot \underline{x}^{(m)}\right)^2$$

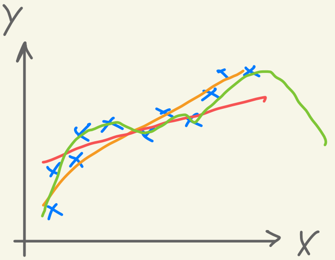
- The updates $\underline{\vartheta}_m \to \underline{\vartheta}_{m+1}$ can be defined by iterating over a randomly chosen subset $\{(\underline{x}^{(n)}, y^{(n)})\}_1^N$ of $N \leq M$ samples from the training set (stochastic gradient descent):

$$J(\underline{\vartheta}) = \frac{1}{2}\sum_{m=}^{n}\left(y^{(m)} - \underline{\vartheta}^t \cdot \underline{x}^{(n)}\right)^2$$

- $J(\underline{\vartheta})$ is quadratic in $\underline{\vartheta}$ & thus minimizing w.r.t. $\underline{\vartheta}$ always gives the global minimum! This is a consequence of the model $h_{\underline{\vartheta}}(\underline{x})$!

## Locally weighted LMS

What happens if the training set is more complicated but we still believe in our model $h_{\underline{\vartheta}}(\underline{x})$ up to small corrections?




— $\hat{=}$ best expected fit

— $\hat{=}$ underfitting

— $\hat{=}$ overfitting

We add a feature map $\varphi(\underline{x})$ such that

$$\varphi(\underline{x}) = (1 \quad x_1 \quad x_2^2)$$

& introduce local weights $w^{(m)}$

$$J(\underline{\vartheta}) = \frac{1}{2} \sum_{m=1}^{N} w^{(m)} \left( y^{(m)} - \underline{\vartheta}^t \cdot \varphi^{(m)}(\underline{x}) \right)^2$$

where the weights $w^{(m)} > 0$ can be used to fit $\underline{\vartheta}^t \cdot \varphi(\underline{x})$ only in a neighborhood of $\underline{x}$:

$$w^{(m)}(\underline{x}) = e^{-\frac{(\underline{x}^{(m)} - \underline{x})^2}{2\tau^2}}$$

## Consequences

(i) now $\underline{\vartheta}$ depends on the point we want to predict

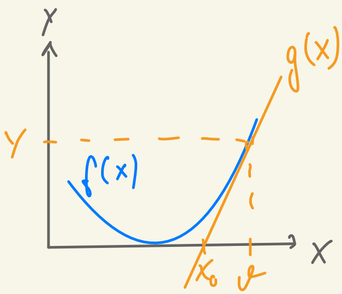
(ii) We need the training data the whole time

(iii) We automatically suppress too many features

# Newton - Raphson

We look for $\underline{\vartheta}_{opt}$ such that $\nabla_{\underline{\vartheta}} J|_{\underline{\vartheta}_{opt}} = 0$.

Let us consider a quadratic function $f: \mathbb{R} \to \mathbb{R}$ for simplicity:




$g(x) = f' \cdot x + y_0$ with $f' = \frac{\partial f}{\partial x}\Big|_{\vartheta}$

$\Rightarrow y_0 = y - f' \cdot \vartheta$

We now determine $x_0$ from $g(x_0) \equiv 0$:

$$0 = f'(x_0 - \vartheta) + y \iff x_0 = \vartheta - \frac{y}{f'} = \vartheta - \frac{f(\vartheta)}{f'(\vartheta)}$$

We thus can perform the update:

$$\vartheta_{k+1} = \vartheta_k - \frac{f(\vartheta_k)}{f'(\vartheta_k)}$$

$$= \vartheta_k - \frac{J'(\vartheta_k)}{J''(\vartheta_k)}$$

For vectors this translates to:

$$\underline{\vartheta}_{k+1} = \underline{\vartheta}_k - \left(\underline{\underline{H}}[J]\big|_{\underline{\vartheta}_k}\right)^{-1} \underline{\nabla}_{\underline{\vartheta}} J\big|_{\underline{\vartheta}_k}$$

with the Hesse matrix:

$$\underline{\underline{H}}[J] = \begin{pmatrix} \frac{\partial}{\partial\vartheta_1}\frac{\partial J}{\partial\vartheta_1} & \cdots & \frac{\partial}{\partial\vartheta_1}\frac{\partial J}{\partial\vartheta_n} \\ \vdots & & \\ \frac{\partial}{\partial\vartheta_n}\frac{\partial J}{\partial\vartheta_1} & \cdots & \frac{\partial}{\partial\vartheta_n}\frac{\partial J}{\partial\vartheta_n} \end{pmatrix}$$

Remarks:

(i) Newton-Raphson method typically converges much faster since $(\underline{\vartheta}_{k+1} - \underline{\vartheta}_k) \propto \underline{\underline{H}}^{-1}\big|_{\underline{\vartheta}_k}$, i.e., the learning rate $\alpha$ from LMS is replaced by model dependent quantity

(ii) Newton-Raphson requires evaluation of $\underline{\underline{H}}[J]$ which can be costly! Also the scaling with the number of features is $\sim \mathcal{O}(n^2)$

# Probabilistic modelling

We now turn back to the question why we expect $h_{\underline{\vartheta}}(\underline{x}) = \underline{\vartheta}^t \cdot \underline{x}$ should be a good model function.

Consider for each sample $(\underline{x}^{(m)}, y^{(m)})$ in the training set the error (residual):

$$y^{(m)} - h_{\underline{\vartheta}}(\underline{x}^{(m)}) = \varepsilon^{(m)}$$

Let us make the assumption that the errors $\varepsilon^{(m)}$ are independently & identically distributed according to a gaussian distribution:

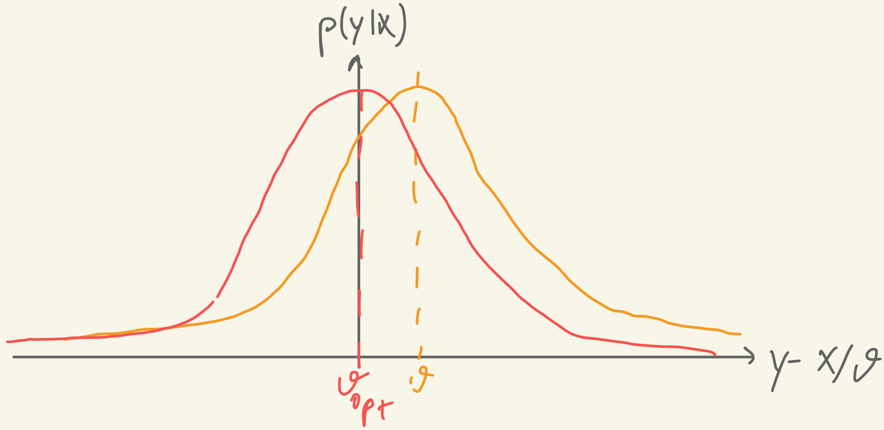
$$p(\varepsilon^{(m)}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left[\varepsilon^{(m)}\right]^2 / 2\sigma^2}$$

$$\Rightarrow \quad p(y^{(m)} | \underline{x}^{(m)}; \underline{\vartheta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(y^{(m)} - \underline{\vartheta}^t \cdot \underline{x}\right)^2}{2\sigma^2}}$$

Read $p(y^{(m)} | \underline{x}^{(m)}; \underline{\vartheta})$ as prob. to predict $y^{(m)}$ given $\underline{x}^{(m)}$ if we fix the parameters $\underline{\vartheta}$.

Then we get for the joint prob. of the errors of all training sets:

$$L(\underline{\vartheta}) = \prod_{m=1}^{M} p(y^{(m)} | \underline{x}^{(m)}; \underline{\vartheta})$$




⇒ We want to find $\underline{\vartheta}$ such that the prob to have errors $\varepsilon^{(m)}$ as small as possible, i.e., we want to maximize the Likelihood function $L(\underline{\vartheta})$.

Equivalently, we can extremize $F(L(\underline{\vartheta}))$ for any strictly monotonically increasing function $F(x)$. Let us choose $F(x) = \log(x)$:

$$\underline{\vartheta}_{opt} = \arg\max_{\underline{\vartheta}} \log(L(\underline{\vartheta}))$$

$$\Leftrightarrow \nabla_{\underline{\vartheta}} \log\left(L(\underline{\vartheta})\right) = 0$$

$$\Leftrightarrow 0 = \frac{\partial}{\partial \vartheta_j} \sum_{m=1}^{M} \log\left(p(y^{(m)} \mid \underline{x}^{(m)}; \underline{\vartheta})\right) \quad \text{for all } j \in \{1, .., n\}$$

$$= \sum_{m=1}^{M} \frac{\partial}{\partial \vartheta_j} \left( - \frac{\left(y^{(m)} - \underline{\vartheta}^t \cdot \underline{x}^{(m)}\right)^2}{2\sigma^2} - \log\left(2\pi\sigma^2\right)\right)$$

$$\Rightarrow 0 = -\frac{1}{2\sigma^2} \sum_{m=1}^{M} \nabla_{\underline{\vartheta}} \underbrace{\left(y^{(m)} - \underline{\vartheta}^t \cdot \underline{x}^{(m)}\right)^2}_{J(\underline{\vartheta})}$$

And thus we arrived just at our linear model with $h_{\underline{\vartheta}}(\underline{x}) = \underline{\vartheta}^t \cdot \underline{x}$ & cost function $J(\underline{\vartheta})$ which needs to be minimized!

Remarks:

(i) We can still choose feature vectors $\varphi(\underline{x})$ such that $\varphi(\underline{x}) = (1 \;\; x_1 \;\; x_1^2 \; \dots)$ contains higher orders in $x_j$.

(ii) This is valid only if the errors are uncorrelated! Situations with $\langle \varepsilon^{(m)} \varepsilon^{(\ell)} \rangle \neq 0$ can occur if for instance hidden features (i.e. missing parameters $x_j$) affect predictions.

## Logistic regression

So far the output (vector) was real. What happens, if we want to make a decision (classification)?

Example:

We want to learn to distinguish lemons from oranges:

lemons have elliptic cross section
lemons are mostly yellow

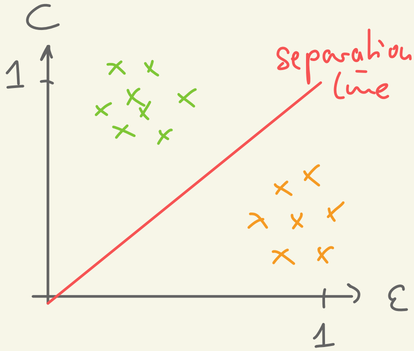
oranges have spherical crosssection
oranges are mostly orange

choose input: $x_1 \mathrel{\hat{=}}$ Ellipticity $\varepsilon$ of cross section

$x_2 \mathrel{\hat{=}}$ Color scheme value, e.g.

$c = 0 \mathrel{\hat{=}}$ red

$c = 1 \mathrel{\hat{=}}$ yellow



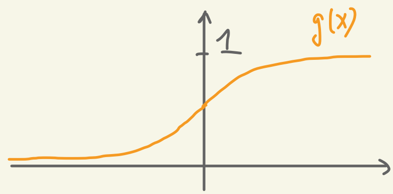

Identification of training data:

$\times \ \hat{=}$ Lemon : $y = 0$

$\times \ \hat{=}$ Orange : $y = 1$

We try to learn the separation line by approximating the binary output $y$ with the logistic (sigmoid) function:

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




We then choose our model as:

$$h_{\underline{\vartheta}}(\underline{x}) = g(\underline{\vartheta}^t \cdot \underline{x}) \equiv g(\eta)$$

$$= \frac{1}{1 + e^{-\underline{\vartheta}^t \cdot \underline{x}}}$$

We need for any optimization the derivative:

$$\frac{\partial h}{\partial \vartheta_j} = \frac{\partial g}{\partial \eta} \frac{\partial \eta}{\partial \vartheta_j} = (1 - h_{\underline{\vartheta}}(\underline{x}))\, h_{\underline{\vartheta}}(\underline{x})\, x_j$$

using:

$$\frac{\partial g}{\partial x} = \frac{\partial}{\partial x}\left(1 + e^{-x}\right)^{-1} = \frac{e^{-x}}{\left(1 + e^{-x}\right)^2}$$

$$= \underbrace{\frac{e^{-x}}{1 + e^{-x}}}_{1 - \frac{1}{1+e^{-x}}} \; \frac{1}{1 + e^{-x}} = (1 - g(x))\, g(x)$$

We now assume again that we can assign probabilities to the outcomes of our learning strategy:

$$\left.\begin{aligned} P(y=1 \mid \underline{x}; \underline{\vartheta}) &= h_{\vartheta}(\underline{x}) \\ P(y=0 \mid \underline{x}; \underline{\vartheta}) &= 1 - h_{\vartheta}(\underline{x}) \end{aligned}\right\} \; p(y \mid \underline{x}, \underline{\vartheta}) = \left[h_{\underline{\vartheta}}(\underline{x})\right]^{y} \left[1 - h_{\underline{\vartheta}}(\underline{x})\right]^{1-y}$$

Then, the likelihood is simply:

$$L(\underline{\vartheta}) = \prod_{m=1}^{M} p(y^{(m)} \mid \underline{x}^{(m)}; \underline{\vartheta})$$

Maximizing $L(\underline{\vartheta})$ is again equivalent to max. $\log L(\underline{\vartheta})$ :

$$\underline{\vartheta}_{opt} = \arg\max_{\underline{\vartheta}} \nabla_{\underline{\vartheta}} \log\left(L(\underline{\vartheta})\right)$$

$$= \arg\max_{\underline{\vartheta}} \nabla_{\underline{\vartheta}} \sum_{m=1}^{M} \left\{ y^{(m)} \log h_{\underline{\vartheta}}(\underline{x}^{(m)}) + (1-y^{(m)}) \log(1-h_{\underline{\vartheta}}(\underline{x}^{(m)})) \right\}$$

Component-wise we have

$$\frac{\partial}{\partial \vartheta_j} \left( y^{(m)} \log h_{\underline{\vartheta}}(\underline{x}^{(m)}) + (1-y^{(m)}) \log(1-h_{\underline{\vartheta}}(\underline{x}^{(m)})) \right)$$

$$= y^{(m)} \frac{(1-h_{\underline{\vartheta}}(\underline{x}^{(m)}))\, \cancel{h_{\underline{\vartheta}}(\underline{x}^{(m)})}}{\cancel{h_{\underline{\vartheta}}(\underline{x}^{(m)})}} x_j + (1-y^{(m)}) \frac{-\cancel{(1-h_{\underline{\vartheta}}(\underline{x}^{(m)}))}\, h_{\underline{\vartheta}}(\underline{x}^{(m)})}{\cancel{1-h_{\underline{\vartheta}}(\underline{x}^{(m)})}} x_j^{(m)}$$

$$= \left( y^{(m)} - h_{\underline{\vartheta}}(\underline{x}^{m}) \right) x_j^{(m)}$$

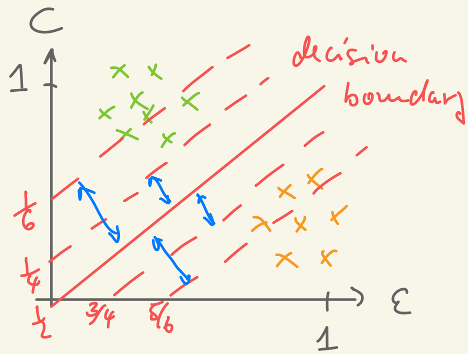
Note that this is the same result as the one we obtained from prob. modelling of linear regression! In fact, it can be shown that both regressions belong to the same class of optimization problems.

Back to lemons & oranges:

We here learn $\underline{\vartheta}^t \cdot \underline{x}$ such that:

(i) $\underline{\vartheta}^t \cdot \underline{x} > 0 \Rightarrow g(\underline{\vartheta}^t \cdot \underline{x})$ becomes $\sim \underline{1}$

(ii) $\underline{\vartheta}^t \cdot \underline{x} < 0 \Rightarrow g(\underline{\vartheta}^t \cdot \underline{x})$ become $\sim 0$

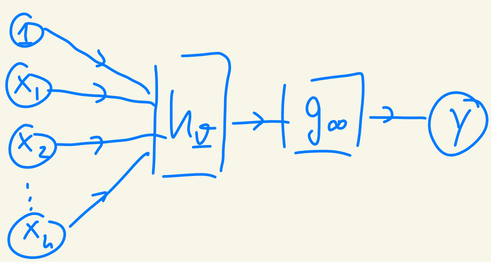



Lines parallel to separation line correspond to lines with constant $g(\underline{\vartheta}^t \cdot \underline{x})$!

More general case:

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

Large $\mu$ squeeze the step together, small $\mu$ broaden it! Extreme case: $\mu \to \infty$ yields the perceptron:




$y \in \{0, \underline{1}\}$

# Small peek into Support Vector Machines (SVM)

Let us change notation:

$$\underline{\vartheta}^t \cdot \underline{X} = \underline{\omega}^t \cdot \underline{X} + b \quad , \; \omega_j, b \in \mathbb{R}$$

where on the left side $\underline{X} = (x_1, \ldots, x_n)$.

For the lemon-orange problem we had:

(i) $\underline{\omega}^t \cdot \underline{X} + b \gg 0 \;\Rightarrow\; g(\underline{\omega} \cdot \underline{x} + b) \to 1$

(ii) $\underline{\omega}^t \cdot \underline{x} + b \ll 0 \;\Rightarrow\; g(\underline{\omega} \cdot \underline{x} + b) \to 0$

Let us now consider the perceptron case with new classifier $h(\underline{\omega}, b)$:

(i) $\underline{\omega}^t \cdot \underline{X} + b > 0 \;\Rightarrow\; h(\underline{\omega}, b) = +1$

(ii) $\underline{\omega}^t \cdot \underline{x} + b < 0 \;\Rightarrow\; h(\underline{\omega}, b) = -1$

classify according to $y \in \{-1, 1\}$

Now we define the geometric margin w.r.t. training datasets $(\underline{X}^{(m)}, y^{(m)})$:

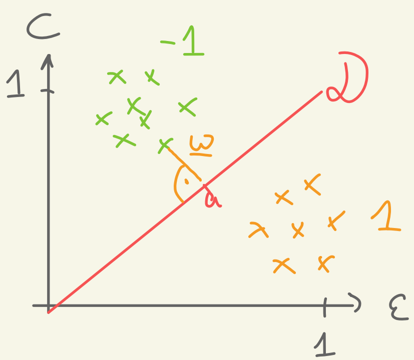
$$\gamma^{(m)} = y^{(m)} \left( \underline{\omega}^t \cdot \underline{X}^{(m)} + b \right)$$

How do we find $\underline{\omega}, b$ ?

Formulate optimization problem noting that $\gamma^{(m)} \geq 0$ always holds true with $\gamma^{(m)} = 0$ at the decision boundary $\underline{w}^t \cdot \underline{x} + b = 0$ :

Find the optimal geometric margin

$$\gamma = \min_m \gamma^{(m)}$$




Since $\gamma$ minimizes distance between points $y^{(m)}$ & the decision boundary, $\underline{w}$ must be orth. to it!

$\Rightarrow$ For each $y^{(m)}$ there is $\underline{a} \in \mathcal{D}$ on the decision boundary with:

$$\underline{a} = x^{(m)} - y^{(m)} \gamma^{(m)} \frac{\underline{w}}{\|\underline{w}\|}$$

Using that for points $\underline{a} \in \mathcal{D}$ we have

$$\underline{w}^t \cdot \underline{a} + b = 0$$

it follows:

$$\underline{w}^t \cdot \left( \underline{x}^{(m)} - y^{(m)} \gamma^{(m)} \frac{\underline{w}}{\|\underline{w}\|} \right) + b = 0$$

$$\iff \quad \underline{w}^t \cdot \underline{x}^{(m)} + b = y^{(m)} \gamma^{(m)} \frac{\underline{w}^t \cdot \underline{w}}{\|\underline{w}\|}$$

$$\underset{\underline{w}^t \cdot \underline{w} = \|\underline{w}\|^2}{\iff} \quad \gamma^{(m)} = y^{(m)} \left( \frac{\underline{w}^t}{\|\underline{w}\|} \cdot \underline{x}^{(m)} + \frac{b}{\|\underline{w}\|} \right)$$

Now from the definition of the optimal margin $\gamma$ it follow that setting $\|\underline{w}\| = 1$ we normalize the optimal parameters $\underline{w}, b$ to their values at $\gamma$: $\gamma \mapsto \frac{\gamma}{\|\underline{w}\|}$, then we optimize:

$$\underset{w, b}{\arg\max} \frac{\gamma}{\|\underline{w}\|} \quad \text{s.t.} \ \underset{m}{\forall} \ y^{(m)} \left( \underline{w}^t \cdot \underline{x}^{(m)} + b \right) \geq \gamma$$

can be made nicer by realizing that the classifier $h(\underline{w}, b)$ is invariant under rescaling $\underline{w}, b$. We can thus choose:

$$\underline{w} \mapsto \frac{\underline{w}}{\gamma}, \quad b \mapsto \frac{b}{\gamma}$$

and thus arrive at the constraint optimization

$$\min_{\underline{w}, b} \frac{1}{2} \|\underline{w}\|^2 \quad \text{s.t.} \quad \forall_m \; y^{(m)}(\underline{w}^t \cdot \underline{x}^{(m)} + b) \geq \underline{1}$$

We thus search the vectors $\underline{w}$ with minimal distance from decision boundary, i.e. the support vectors!

## $\overline{\underline{\text{VII}}}$.2 Deep learning

So far we considered single-layer models:

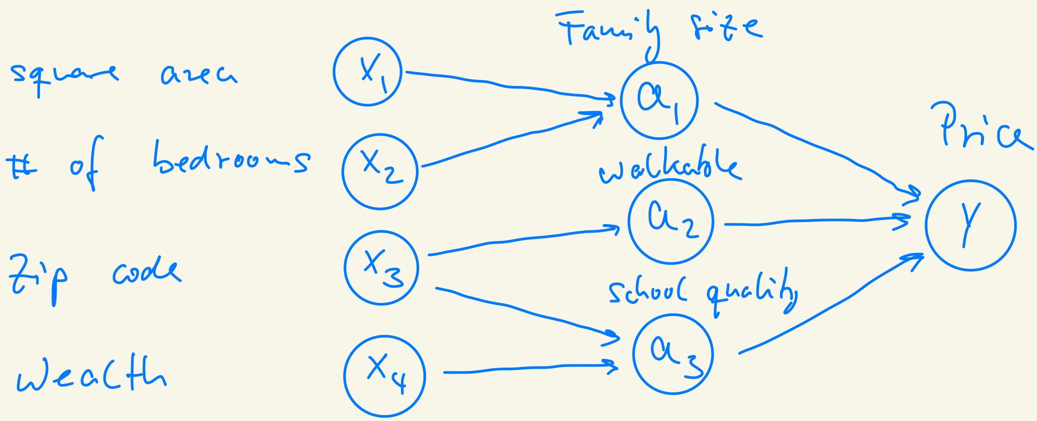
$$h_{\underline{\vartheta}}(\underline{x}) = g(\underline{\vartheta}^t \cdot \underline{x})$$

with some function (non-)linear $g: \mathbb{R} \to \mathbb{R}$.

We also saw feature parametrizations $\varphi(\underline{x})$, i.e. $\varphi(\underline{x}) = (1 \;\; x_1 \;\; x_2^2 \;\; \ldots)^t$.

Generally, we want models with features $\varphi(\underline{x})$ that are taylored to the underlying problem.

Let's go back to the housing problem & make our model more complex:




We introduced features:

$$a_1(x_1, x_2), \quad a_2(x_3), \quad a_3(x_3, x_4)$$

to express price as function of these features.

But this is an ad-hoc representation! We actually want the optimization to identify important features!

This motivates the deep neural network ansatz:

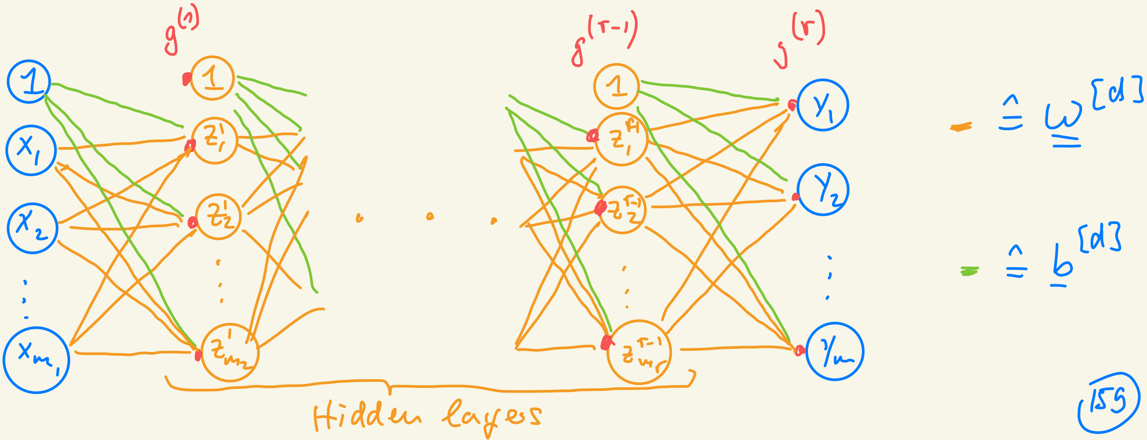
Let $r \in \mathbb{N}$ & $m_d \in \mathbb{N}$, $d \in \{1, \dots, r\}$, then we introduce weight matrices: $\underline{\underline{W}}^{[d]} \in \mathbb{R}^{m_{d+1} \times m_d}$, bias vectors $\underline{b}^{[d]} \in \mathbb{R}^{m_{d+1}}$

& activation functions $g^{(d)}: \mathbb{R}^{m_{d+1}} \to \mathbb{R}^{m_{d+1}}$, such that for input values $\underline{x} \in \mathbb{R}^{m_1}$ we parametrize output values $\underline{y} \in \mathbb{R}^{m_{r+1}}$:

$$\underline{y} = g^{(r)}\left(\underline{\underline{W}}^{[r]}\, g^{(r-1)}\left(\underline{\underline{W}}^{[r-1]}\, g^{(r-2)}(\dots)\right) + \underline{b}^{[r]}\right)$$

$$= g^{(r)}\left[\underline{\underline{W}}^{[r]}, \underline{b}^{[r]}\right] \circ g^{(r-1)}\left[\underline{\underline{W}}^{[r-1]}, \underline{b}^{[r-1]}\right] \circ \dots \circ g^{(1)}\left[\underline{\underline{W}}^{[1]}, \underline{b}^{[1]}\right](\underline{x})$$

with $g^{(d)}\left[\underline{\underline{W}}^{[d]}, \underline{b}^{[d]}\right](\underline{z}) = g^{(d)}\left(\underline{\underline{W}}^{[d]} \cdot \underline{z} + \underline{b}^{[d]}\right)$



Let us consider a quadratic cost function

$$J^{(s)}\left[\{\underline{\underline{W}}^{[d]}, \underline{b}^{[d]}\}\right](\underline{x}^{(s)}) = \frac{1}{2m}\| \underline{y}^{(s)} - \underline{y} \|^2$$

How do we minimize $J^{(s)}$ ?

## Back propagation

Consider a network with one hidden layer & $\underline{b}^{[d]} \equiv 0$ for simplicity.

We consider the last layer:

$$\underline{y} = g^{(2)}\left(\underline{\underline{W}}^{[2]} \, \underline{z}^{(1)}\right)$$

& write the cost function component wise:

$$J^{(s)} = \frac{1}{2m} \sum_{j=1}^{m} \left( y_j^{(s)} - y_j \right)^2 \equiv \frac{1}{2m} \sum_{j=1}^{m} \left[\Delta_j^{(2)}\right]^2$$

We want to minimize all $\Delta_j^{(2)} \geq 0$ with:

$$\frac{1}{2m} \Delta_j^{(2)} = \frac{1}{2m}\left( y_j^{(s)} - g^{(2)}\left( \sum_{k=1}^{m_1} w_{jk}^{[2]} z_k^{(1)} \right)\right)^2$$

We use gradient descent to update each $\underline{w}_j^{[2]}$ for fixed $j$: $\underline{w}_j^{[2]} \longleftarrow \underline{w}_j^{[2]} - \delta \underline{w}_j^{[2]}$ with

$$\delta \underline{w}_j^{[2]} = \alpha \, \underline{\nabla}_{w_j^{[2]}} \, g^{(2)} \qquad (\alpha \text{ is learning rate})$$

$$= -\frac{\alpha}{m}\left(g^{(2)}\left(\underline{w}_j^{[2]} \cdot \underline{z}^{(1)}\right) - y_j^{(s)}\right) \frac{\partial g^{(2)}}{\partial \eta} \, \underline{z}^{(1)}$$

having defined $\underline{w}_j^{[2]} = \left(w_{j1}^{[2]} \; w_{j2}^{[2]} \; \ldots \; w_{j\,m_{r+1}}^{[2]}\right)^t$.

Note that introducing:

(i) $t_j^{[2]} = y_j^{(s)}$

(ii) $o_j^{[2]} = g^{(2)}\left(\underline{w}_j^{[2]} \cdot \underline{z}^{(1)}\right)$

this can be written more conveniently:

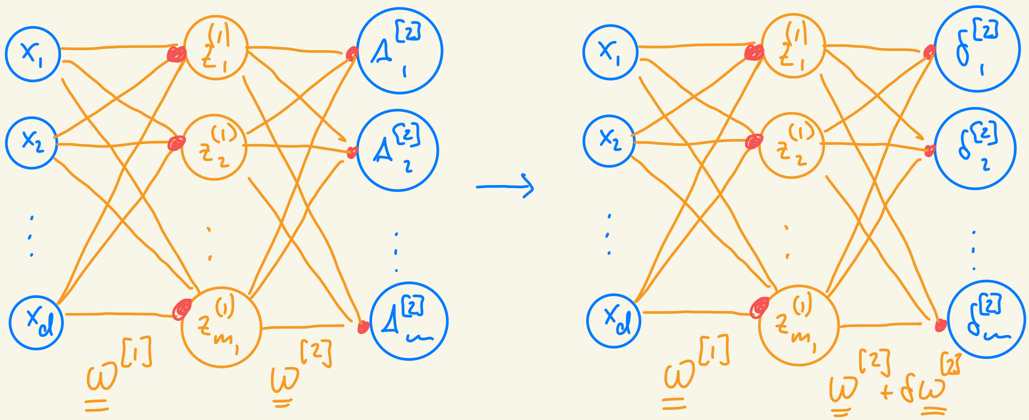
$$\delta \underline{w}_j^{[2]} = -\frac{\alpha}{m}\left(o_j^{[2]} - t_j^{[2]}\right) \frac{\partial g^{(2)}}{\partial \eta} \, \underline{z}^{(1)}$$

$$\equiv -\frac{\alpha}{m} \Delta_j^{[2]} \frac{\partial g^{(2)}}{\partial \eta} \frac{\partial \eta}{\partial \underline{w}_j^{[2]}}$$

Let us denote the updated weights by:

$$\underline{\underline{w}}^{[2]} \longleftarrow \underline{\underline{w}}^{[2]} + \delta \underline{\underline{w}}^{[2]}$$

These updated weights now generate updated errors $\Delta_j^{[1]}$




We now want to update $\underline{\underline{w}}^{(1)}$ by minimizing the errors in the central layer. What are these errors? Consider the updated cost function:

$$J^{(s)} = \frac{1}{2m} \sum_{j=1}^{m} \left( y_j^{(s)} - g^{(2)}\left( (\underline{w}_j^{[2]} + \delta \underline{w}_j^{[2]}) \cdot \underline{z}^{(1)} \right) \right)^2$$

with $z_k^{(1)} = g^{(1)}\left( \underline{w}_k^{[1]} \cdot \underline{x}^{(s)} \right)$

Minimization w.r.t. $\underline{w}_k^{[1]}$'s:

$$\nabla_{\underline{w}_k^{[1]}} J^{(s)} = \frac{1}{m} \sum_{j=1}^{m} \left( y_j^{(s)} - g^{(2)}\left( \underline{w}_j^{[2]} \cdot \underline{z}^{(1)} + \delta \underline{w}_j^{[2]} \cdot \underline{z}^{(1)} \right) \right) \frac{\partial g^{(2)}}{\partial \eta^{(2)}} \nabla_{\underline{w}_k^{[1]}} \eta^{(2)}$$

$$= \frac{1}{m}\sum_{j=1}^{m} \delta_j^{[2]} \frac{\partial g^{(2)}}{\partial y^{(2)}} \sum_{\ell=1}^{n} e_\ell \frac{\partial y^{(2)}}{\partial z_\ell^{(1)}} \nabla_{\underline{w}_k^{[1]}} z_\ell^{(1)}$$

where we just used the definition of the errors after optimizing $\underline{\underline{w}}^{[2]}$.

Using:

$$\frac{\partial y^{(2)}}{\partial \underline{z}^{(1)}} = (\underline{w}_j^{[2]} + \delta \underline{w}_j^{[2]})$$

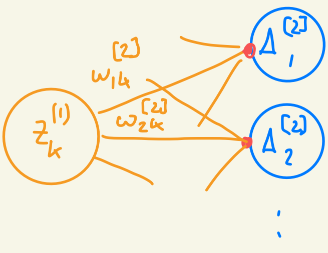
$$\nabla_{\underline{w}_k^{[1]}} z_\ell^{(1)} = \frac{\partial g^{(1)}}{\partial y_k^{(1)}} \, \underline{x} \, \delta_{k\ell}$$

We arrive at:

$$\nabla_{\underline{w}_k^{[1]}} J^{(S)} = \frac{1}{m} \sum_{j=1}^{m} \delta_j^{[2]} \frac{\partial g^{(2)}}{\partial y^{(2)}} \left(\underline{w}_j^{[2]} + \delta \underline{w}_j^{[2]}\right) \cdot \underline{x} \, \frac{\partial g^{(1)}}{\partial y^{(1)}}$$

$$= \frac{1}{m} \left[ \underbrace{\frac{\partial g^{(2)}}{\partial y^{(2)}} \underline{\delta}^{[2]} \cdot \left( \underline{\underline{w}}^{[2]} + \delta \underline{\underline{w}}^{[2]} \right)}_{\text{back-propagated error } \underline{\Delta}^{[1]}} \right] \frac{\partial g^{(1)}}{\partial y^{(1)}} \, \underline{x}$$

Note that there is a nice interpretation:




$$\Delta_1^{[2]} \approx \sum_{k=1}^{m_2} z_k^{(1)} w_{1k}^{[2]} \frac{\partial g^{(2)}}{\partial y_1}$$

$$\Delta_2^{[2]} \approx \sum_{k=1}^{m_2} z_k^{(1)} w_{2k}^{[2]} \cdot \frac{\partial g^{(2)}}{\partial y_2}$$

$\vdots$

where we expanded $g(x)$ to first order & dropped the constant $g(x)$ since it does not affect the optimization. As we look for a correction $\delta\omega_{jk}^{[2]}$, we try to distribute the errors $\delta_j^{[2]}$ over the weights:

$$\Delta_j^{[1]} = \frac{\omega_{jk}^{[2]} \frac{\partial g^{(2)}}{\partial y_j}}{\sum_{k=1}^{m_2} \omega_{jk}^{[2]} \frac{\partial g^{(2)}}{\partial y_j}} \Delta_j^{[2]}$$

$$\Rightarrow \quad \underline{\Delta}^{[1]} = \begin{pmatrix} - \left(\frac{\partial g^{(2)}}{\partial y_1} \underline{\omega}_1^{[2]}\right)^t - \\ \vdots \\ - \left(\frac{\partial g^{(2)}}{\partial y_m} \underline{\omega}_m^{[2]}\right)^t - \end{pmatrix} \underline{\Delta}^{[2]}$$

where we dropped the normalization since it is the same for each row.

Now after updating $\underline{\omega}_j^{[2]} \to \underline{\omega}_j^{[2]} + \delta\underline{\omega}_j^{[2]}$, the distributed errors become:

$$\underline{\Delta}^{[1]} = \frac{\partial g^{(2)}}{\partial y} \underline{\delta}^{[2]} \cdot \left(\underline{\underline{\omega}}^{[2]} + \delta\, \underline{\underline{\omega}}^{[2]}\right)$$

which is exactly what we derived above!

This allows to formulate the back propagation algorithm:

(1) push forward sample $\underline{x}^{(s)}$ to the last layer at cost $\sim \mathcal{O}(r \cdot m_{max}^2)$ where $m_{max} = \max_d m_d$, to obtain $\underline{y}$. Set $\underline{y}^{(s)} \equiv \underline{t}^{[r]}$, $\underline{y} \equiv \underline{o}^{[r]}$ & $\underline{\Delta}^{[r]} = \underline{t}^{[r]} - \underline{o}^{[r]}$

(2) calculate back-propagated error in d'th layer by evaluating:

$$\underline{\Delta}^{[r-1]} = \frac{\partial g^{(r)}}{\partial y^{(r)}} \underline{\delta}^{[r]} \cdot \left( \underline{\underline{w}}^{[r]} + \delta \underline{\underline{w}}^{[r]} \right)$$

where $\underline{\delta}^{[r]}$ is error after prev. optimization.

(3) calculate $\underline{\nabla}_{\underline{w}_k} J^{(s)}$ from back propagated error in d'th layer:

$$\underline{\nabla}_{\underline{w}_k} J^{(s)} = -\frac{1}{m} \underline{\Delta}^{[d-1]} \cdot \frac{\partial g^{(d-1)}}{\partial y_k^{(d-1)}} \underline{z}^{(d-1)}$$

& update: $\underline{w}_k^{[d-1]} \leftarrow \underline{w}_k^{[d-1]} - \alpha \nabla_{\underline{w}_k^{[d-1]}} J^{(s)}$

Remarks:

(i) bias nodes can be easily added & the formalism does not change:

$$\underline{w}_k^{(d)} \mapsto \begin{pmatrix} b^{[d]} & \underline{w}_k \end{pmatrix}$$

$$\underline{z}^{(d)} \mapsto \begin{pmatrix} 1 & \underline{z}^{(d)} \end{pmatrix}$$

(ii) Computational cost scale only $\mathcal{O}(r \cdot m_{max})$ where $m_{max} = \max_d m_d$ ! Can be speed up further by generalizing input/output to update using several samples (stacking)