FIRST STEPS: LINEAR CLASSIFIERS

- discussed here:
  - Perceptron
  - Adaline (adaptive linear neuron)

- good entry point to modern machine learning as it is fairly simple but generalizes to deep networks
- idea goes back to McCulloch & Pitts 1943

  "A logical calculus of the ideas immanent in nervous activity"

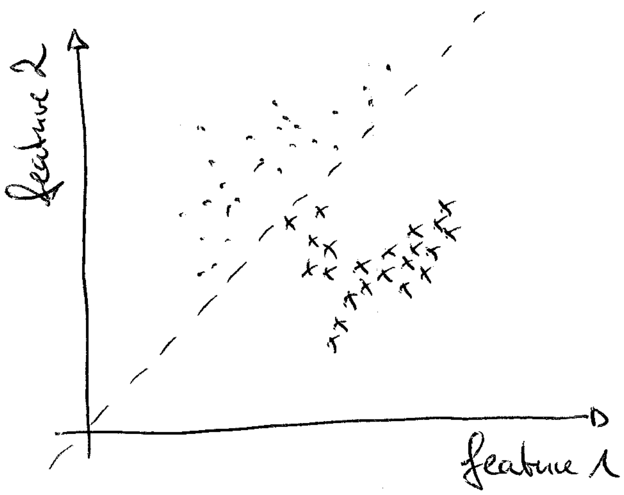
  - Rosenblatt, 1957:

    "The perceptron, a perceiving and recognizing automaton"

  - Widrow, 1960:

    "A adaptive 'Adaline' neuron using chemical memistors"

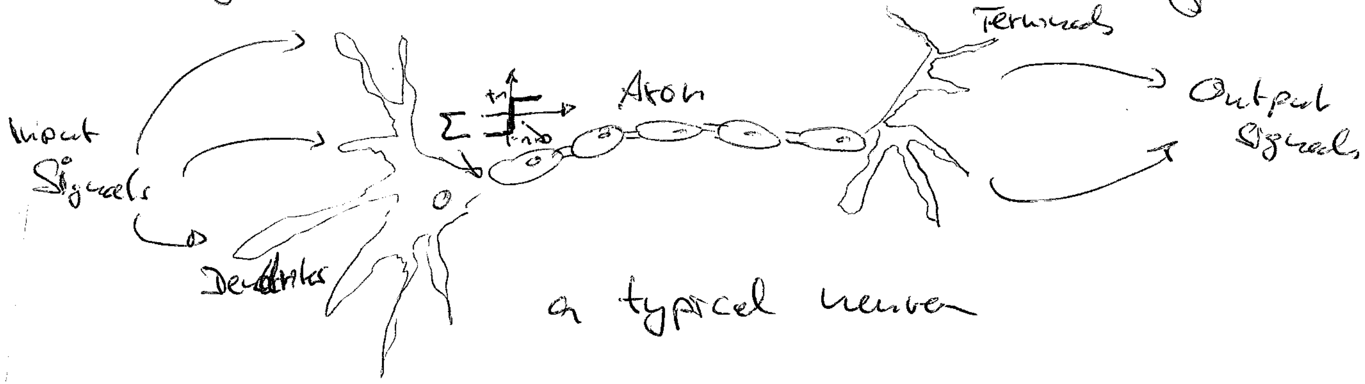
BASIC IDEA



- these data point are to be classified by assigning each data point $x^{(i)}$ a label $y^{(i)} \in \{-1,1\}$ say $Y = (y^{(i)})_{1 \leq i \leq N}$
- problem of binary classification:
  - we look for a function $f: \mathbb{R}^n \to \{-1,1\}$
  - we want the machine to find f by supervised training

- each data point consists of for example two features $X = (\underset{\mathbb{R}^2}{\underset{\wedge}{x^{(i)}}})_{1 \leq i \leq N}$

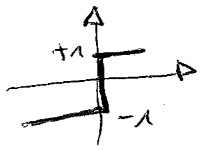
Analogy from biology:



a typical neuron

A mathematical model:

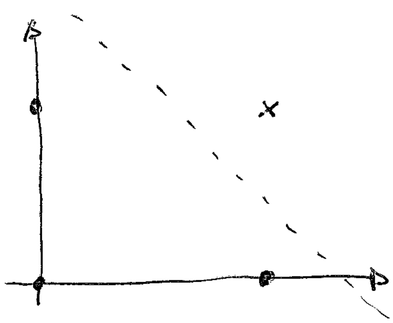
Input signal



$$\mathbb{R}^{n+1} \ni x = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} \longrightarrow \underset{W \in \mathbb{R}^{n+1}}{\overset{\Sigma}{\boxed{W \cdot x}}} \longrightarrow \boxed{\sigma(W \cdot x)} \longrightarrow y \in \{-1, 1\}$$

$$\sigma(z) := \begin{cases} +1 & \text{for } z \geq 0 \\ -1 & \text{for } z < 0 \end{cases}$$

Example:



AND-gate

IN_1	IN_2	AND
0	0	0
0	1	0
1	0	0
1	1	1

we need a linear map $z = w \cdot x$, for example

$$w = \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix}$$

$$w \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = -1.5 < 0$$

$$w \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = w \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = -0.5 < 0$$

$$w \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 0.5 > 0$$

$$\Rightarrow \quad f(x) = \sigma(w \cdot x)$$

is a representation of the AND-gate

HW: Try to implement all 16 logical gates by the perceptron. Which fail?

Next we want an algorithm that allows to find a good choice of weights:

- they should classify the training data most accurately
- however, should also generalize to new data in a good way

⚠ $x \in \mathbb{R}^{n+1}$ because $x = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$

Learning rule

- given: training data $(x^{(i)}, y^{(i)})_{1 \le i \le N}$, $x^{(i)} \in \mathbb{R}^{n+1}$, $y^{(i)} \in \{-1, 1\}$

~~ALG (Perception training)~~:

~~Start: Initialize a weight vector $w^{(0)} \in \mathbb{R}^{n+1}$ either zero or random and $k=0$~~

~~Step:~~

- Initialize weight vector $w \in \mathbb{R}^{n+1}$ to zero or at random
- for each training sample $(x^{(i)}, y^{(i)})$
  - a) compute output $y = \sigma(w x^{(i)})$
  - b) compare y and $y^{(i)}$

    if equal do nothing

    else update the weight vector w in a good way

For the perceptron the following update rule was suggested:

$$\Delta = y^{(i)} - y$$

$$\Delta w = \eta \Delta x^{(i)}$$

$\eta \in \mathbb{R}$ is called "the learning rate"

Why is this a good learning task?

- $\Delta = 0 \Rightarrow$ no error $\Rightarrow \Delta w = 0$
- $\Delta \neq 0 \Rightarrow$ error:
  a) $\Delta = 2 \Rightarrow$ wanted $y = 1$ but got $y = -1$
  $\Rightarrow \Delta w > 0$ which corrects in the right direction
  b) $\Delta = -2 \Rightarrow$ wanted $y = -1$ but got $y = 1$
  $\Rightarrow \Delta w < 0$ which again corrects in the right direction

V

Convergence in case training data is linearly separable

DEF: Two sets A, B of points in $\mathbb{R}^n$ are called: i) linearly separable iff $\exists\, w \in \mathbb{R}^n, w_0 \in \mathbb{R}$:

$$\forall a \in A: \; wa + w_0 \geq 0$$
$$\forall b \in B: \; wb + w_0 < 0$$

ii) absolutely linearly separable iff $\exists\, w \in \mathbb{R}^n, w_0 \in \mathbb{R}$:

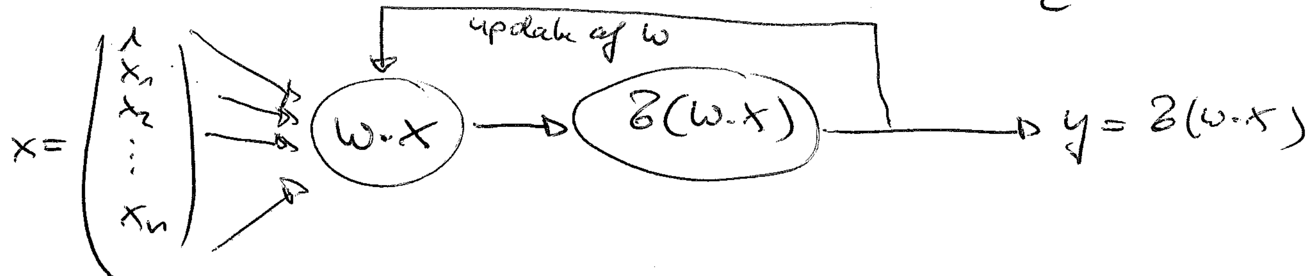
$$\forall a \in A: \; wa + w_0 > 0$$
$$\forall b \in B: \; wb + w_0 < 0$$

~~THM: If A,B a linearly separable with $w_0=0$ and finite, then the perceptron learning algorithm converges~~

~~ALG: start: $t := 0$, $w^{(0)} \in \mathbb{R}^n$ randomized~~

~~step: $x \in A, B$~~

$w \mapsto w + \eta(y^{(i)} - \sigma(w \cdot x^{(i)}))\, x^{(i)}$



III

ALG (Perceptron training)

Start: initialize weight vector $w^{(0)}$ at random, $t := 0$

Step: choose $x \in A \cup B$ at random ~~[illegible]~~)

$x \in A \wedge w^{(t)} \cdot x \geq 0$ goto Step

$x \in A \wedge w^{(t)} \cdot x \leq 0$ goto Add

$x \in B \wedge w^{(t)} \cdot x \leq 0$ goto Step

$x \in B \wedge w^{(t)} x \geq 0$ goto Sub

Add: $w^{(t+1)} := w^{(t)} + x$, $t := t+1$ goto Step

Sub: $w^{(t+1)} := w^{(t)} - x$, $t := t+1$ goto Step

Thm: If A, B are finite and linearly seperable, then the above algorithm converges in finitely many steps.

Prop: A, B finite, then A, B lin. sep $\Leftrightarrow$ A, B abs. sep.

Proof: A, B lin. sep. $\Rightarrow \exists w: \forall a \in A, \; w \cdot a \geq 0$

$\forall b \in B, \; w \cdot b < 0$

$$\varepsilon := -\max \{ wb \mid b \in B \} > 0$$

$$w' := w + \begin{pmatrix} \varepsilon/2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\Rightarrow \forall a \in A: \; w'a = \overbrace{wa}^{\geq 0} + \frac{\varepsilon}{2} > 0$$

$$\forall b \in B: \; w'b = \overbrace{wb}^{\leq -\varepsilon} + \frac{\varepsilon}{2} < 0$$

$\square$

Proof of Thm:

- WLOG: we may assume that the point vectors in A and B are normalized as $w \cdot x \gtrless 0 \iff w \cdot \frac{x}{\|x\|} \gtrless 0$
- since A, B are lin. sep., they are abs. lin. sep., and therefore there is a normalized w^* for which
  $\forall a \in A: \; w^* a > 0$
  $\forall b \in B: \; w^* b < 0$
- $P := A \cup (-1)B$

abs. lin. sep!

Say after some time $w^{(t+1)}$ is computed
$\Rightarrow$ at time t $p \in P$ was incorrectly classified and $w^{(t+1)} := w^{(t)} + p$

$$\cos\varphi = \frac{w^* \cdot w^{(t+1)}}{\|w^{(t+1)}\|}$$

but 1)
$$w^* w^{(t+1)} = w^* \cdot w^{(t)} + w^* p \geq w^* \cdot w^{(t)} + \delta$$

for $\delta := \min \{ w^* p \mid \forall p \in P \} \underline{> 0}$

by induction we get

$$w^* w^{(t+1)} \geq w^* w^0 + (t+1)\delta$$

but 2)
$$\|w^{(t+1)}\|^2 = \|w^{(t)}\|^2 + \overbrace{2 w^{(t)} \cdot p}^{\substack{\text{misclassified} \\ <0}} + \overbrace{\|p\|^2}^{\text{normed}} \leq \|w^{(t)}\|^2 + 1$$

again by induction we get

$$\|w^{(t+1)}\|^2 \leq \|w^{(0)}\|^2 + (t+1)$$

Collecting these inequalities we get

$$1 \geq \cos\varphi \geq \frac{w^{*} \cdot w^{(0)} + (t+1)\delta}{\sqrt{\|w^{(0)}\|^2 + (t+1)}} \sim \sqrt{t+1}\,\delta$$

grows like $\sqrt{t+1}\,\delta$ indefinitely

$\Rightarrow$ only finite many updates are possible

$\Rightarrow$ ALG converges in finite many steps $\square$

$\hookrightarrow$ Python code

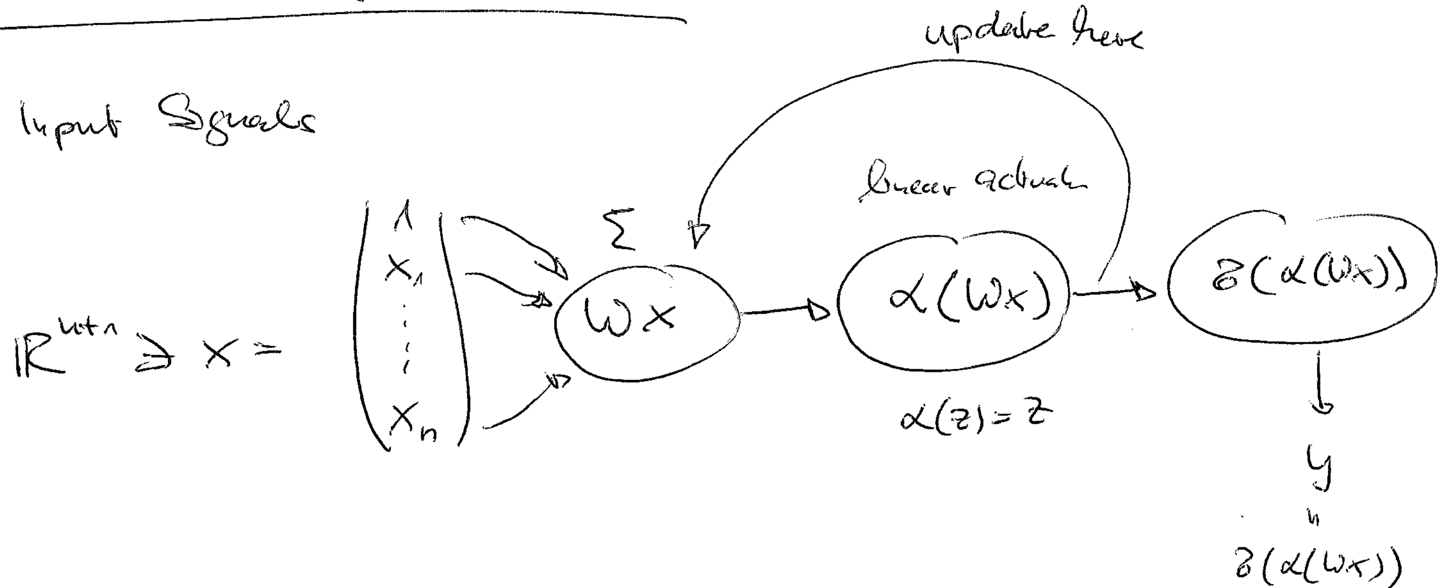
HW: 1) Implement by yourself. ~~Look for other data sets.~~
2) Visualize the weights per epoch
~~[illegible]~~ 3) learn logical gates

PROBLEMS OF THE PERCEPTRON

1) Convergence is not guaranteed in general
only when A, B are lin. sep. $\Rightarrow$ threshold for max. updates
2) If A, B are not lin. sep. at least one time every epoch w will be updated and w might start oscillate and fail to find a good decision boundary
3) If A, B are lin. sep. w stops being updated the moment everything in training data is classified correctly

HW: 1) Find such case (maybe by putting additional data points by hand)

Adaptive Linear Neuron



<u>Advantage</u>:
- The activation function α is differentiable.
- Using $\alpha(Wx)$ instead of $\sigma(\alpha(Wx))$ as indicator for an update allows to employ analytic optimization theory

<u>Learning rule</u>

def. a loss function

$$L(W) := \frac{1}{2} \sum_i \left[y^{(i)} - \alpha(W \cdot x^{(i)}) \right]^2$$

Instead of minimizing directly the number of misclassifications we may minimize $L(w)$

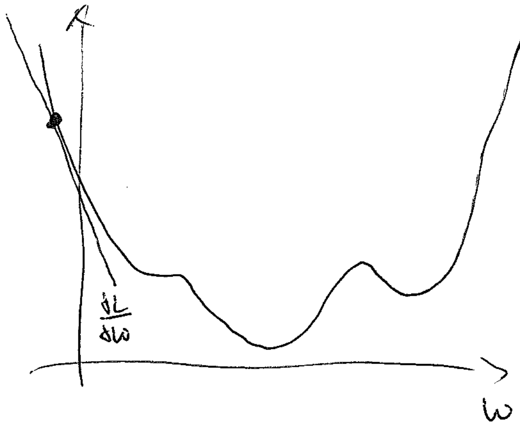
compute: $$\frac{\partial L}{\partial w} = \sum_i \left(y^{(i)} - \alpha(w \cdot x^{(i)}) \right) (-1)\, x^{(i)}$$

$$f(x^i) = \sum_i (y^i - \alpha(\overset{w \cdot x^i}{x^i}))^2$$

$$f(x) = (y-x)^2$$

$$\frac{\partial L(w)}{\partial w_j} = \frac{\partial \frac{1}{2} \sum_i (y^i - \alpha(wx^i))^2}{\partial w_j}$$

$$= \sum_i (y^i - \alpha(wx^i)) \underbrace{\left(- \frac{\partial [\alpha(wx^i)]}{\partial w} \right)}_{\alpha'(wx^i)\, x_j^i}$$



to get closer to a local minimum we may use the gradient $\frac{\partial L}{\partial w}$ to give a good direction

$$\Delta w = -\eta \frac{\partial L(w)}{\partial w}$$

for the intuition

$$L(w+\Delta w) = L(w) + \frac{\partial L(w)}{\partial w} \cdot \Delta w + \mathcal{O}(\Delta w^2)$$

choosing $\Delta w = -\eta \frac{\partial L(w)}{\partial w}$ gives

$$L(w+\Delta w) = L(w) \underbrace{- \eta \left(\frac{\partial L(w)}{\partial w}\right)^2}_{\substack{\text{linear perturbation}\\ \text{has the right direction}}} + \mathcal{O}(\Delta w^2)$$

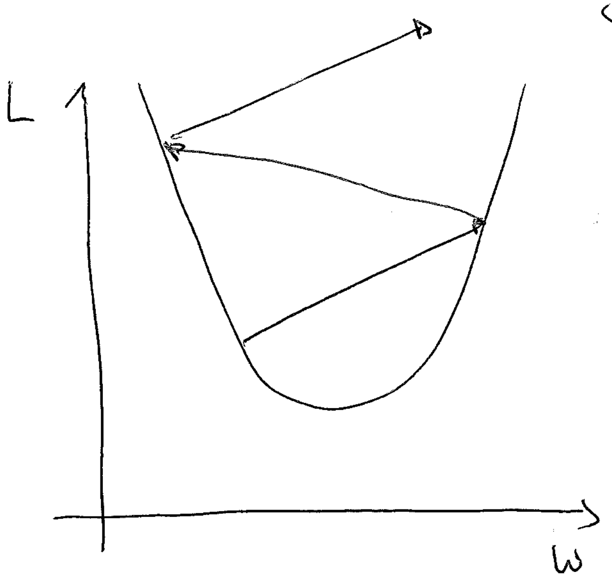
Differences to Perceptron:
- not online but batch update
- even in sep. case, no convergence guaranteed?

↳ Python code

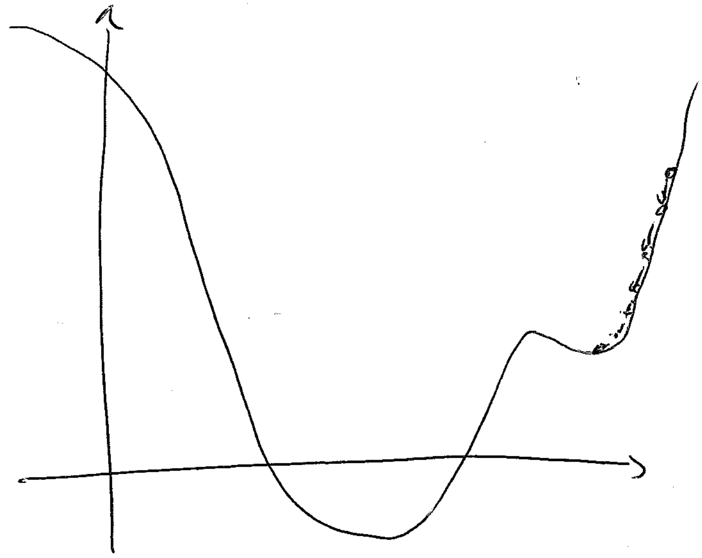
HW: ~~Try other α, and L(w)~~ Implement yourself.

~~• How could online learning be performed~~

How to choose the learning rate?



to high



too low

Constant added examples

HW: 1) Find such inappropriate learning rates. 2) Try other L and α? 3)

A word about the preparation of the data

- the choice of learning rate depends on the fluctuation of the data
  $\Rightarrow$ usually good practise to "normalize" the data

$$\tilde{x}^{(i)} = \frac{x^{(i)} - \mu}{\sigma}$$

for $\mu = \frac{1}{N} \sum_i x^{(i)}$, $\sigma = \sqrt{\frac{1}{N} \sum_i (x^{(i)} - \mu)^2}$

Compare the 9+0 data normalized or not.
use Iverne ML Library

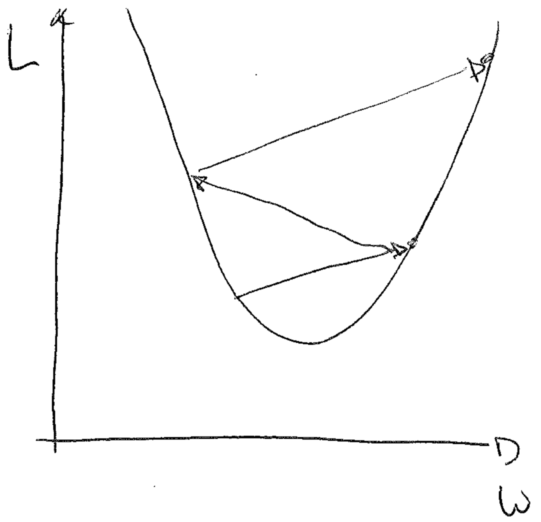
HW: ~~look for other data~~ 1) Prepare it: train, test, all, normalize and train a model. 2) choose $\eta = \frac{1}{M}$ and discuss

Online Learning & Stochastic Gradient Descent ~~[illegible]~~

- Adaline so far compute ΔW depending on all training data – can be expensive!
- Perceptron performed "online" learning after each newly encountered training data point
- ~~[illegible]~~ This could also be done for Adaline in which case it is called "stochastic gradient descent"
- again ~~[illegible]~~ to avoid to strong dependence on the sequence of learning one can do "mini-batch st. gradient descent".
- HW: 1) Make Adaline an online learner
  2) Make Perceptron a batch learner using average
  3) Make ~~both~~ mini-batch learner.

Choosing a good learning rate: gradient descent $\boxed{V}$

$$w \mapsto \tilde{w} := w - \eta \frac{\partial L(w)}{\partial w}, \quad L(w) := \sum_{i=1}^{M} \left| y^{(i)} - \alpha(w \cdot x^{(i)}) \right|^2$$



too high

too low

HW: Implement a test case for both situations in 1d, 2d and plot.

The chosen learning rate depends on the fluctuation of training data:

$\Longrightarrow$ good practice to normalize data

$$\tilde{x}^{(i)} = \frac{x^{(i)} - \mu}{\sigma}$$

$$\text{for } \mu = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}, \quad \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x^{(i)} - \mu)^2}$$

HW: • Try with Iris data set, random data, or some standard training data from UCI ML Lib.

Online vs Batch Learning

- Adaline so far computes ΔW given the batch of all training data — batch learning
- Perceptron performed an update after each misclassification — online learning
- In many situation it is less expensive and one gains convergence speed by not using the whole batch for an update

→ stochastic gradient descent "online"

$$L(w) = \sum_{i=1}^{M} L_i(w) \qquad L_i(w) = |w \cdot x^{(i)} - y^{(i)}|^2 \frac{1}{2}$$

$$w \mapsto \tilde{w} := w - \eta \frac{\partial L_i(w)}{\partial w} \quad \text{for sample } i$$

- A compromise between these two algorithms is "mini-batch" gradient descent.

$i = 1, \dots, M$ training data points divided into distinct subsets $I_1 \cdots I_m$ and m-times one performs the update
$$w \mapsto \tilde{w} := w - \eta \frac{\partial \sum_{i \in I_k} L_i(w)}{\partial w}$$

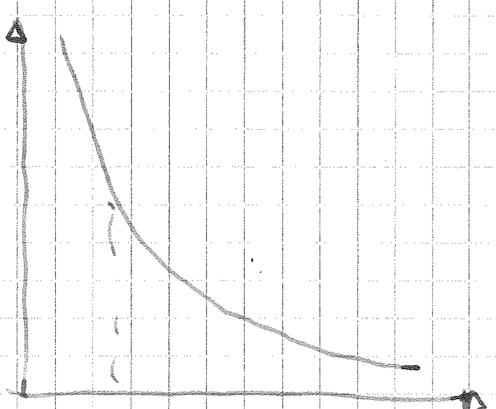
- to avoid a strong dependence on the sequence of chosen data-points one can average by factor
$$\eta = (\#\text{ sample points})^{-1}$$

HW: 1) Make Adaline an online learner
2) Make Perceptron a batch learner with averaged update
3) Make both minibatch learner.

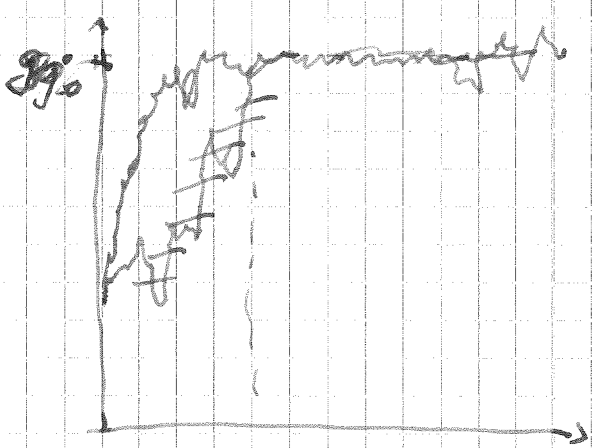
Overfitting

- as the convergence is not guaranteed in general one must find a good criterion when to stop the learning process
- for e.g. Adaline there is the danger of "overfitting":
    - presenting the same training data over and over again W gets better and better adapted to the training data
    - however, that usually means that W generalizes worse and worse to unseen data
    - overfitting training data could turn out to be a perverse way to just store the training data

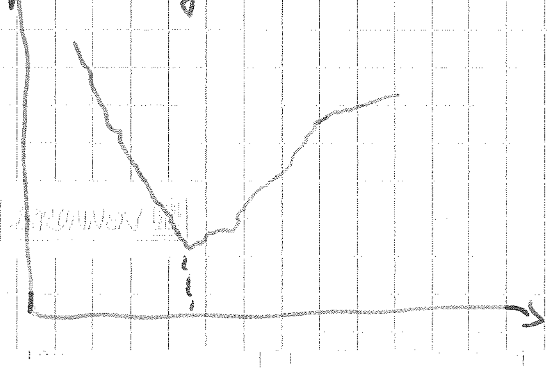
cost of training data

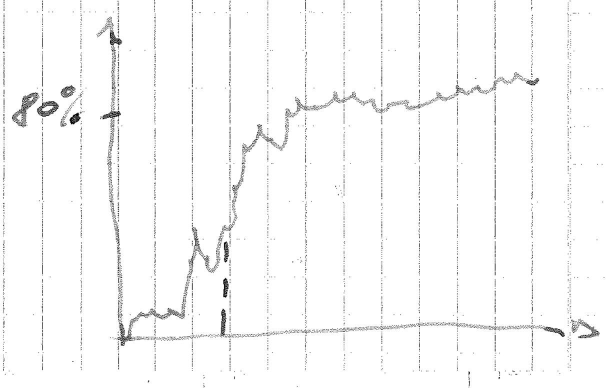


accuracy on training



cost of test data





What's next?

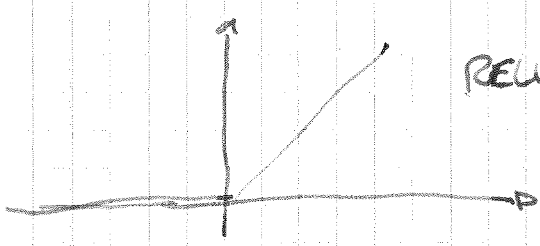
- freedom to incorporate additional terms in the loss function

  cross-entropy

  $$-\frac{1}{M}\sum_i y^{(i)}\ln f(x^{(i)}) + (1-y^{(i)})\ln(1-f(x^{(i)}))$$

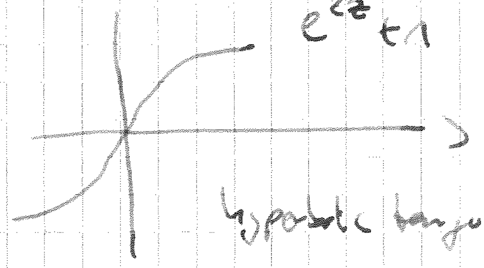
  - e.g. $\|w\|_p$ norms to control what kind of weights are preferred
  - different distance relations such as L^p norms

- different activation functions α



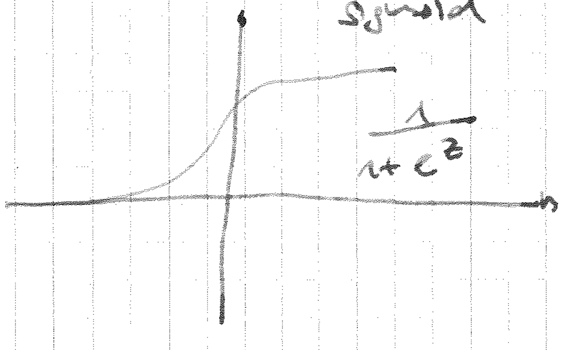
RELU

$$\frac{e^{2z}-1}{e^{2z}+1}$$



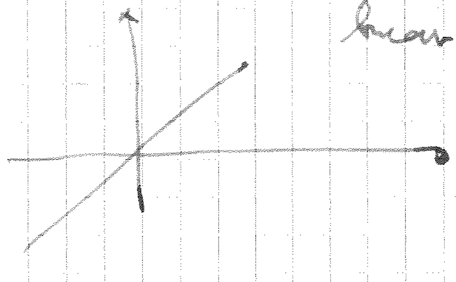
hyperbolic tangent

sigmoid



$$\frac{1}{1+e^{z}}$$

linear

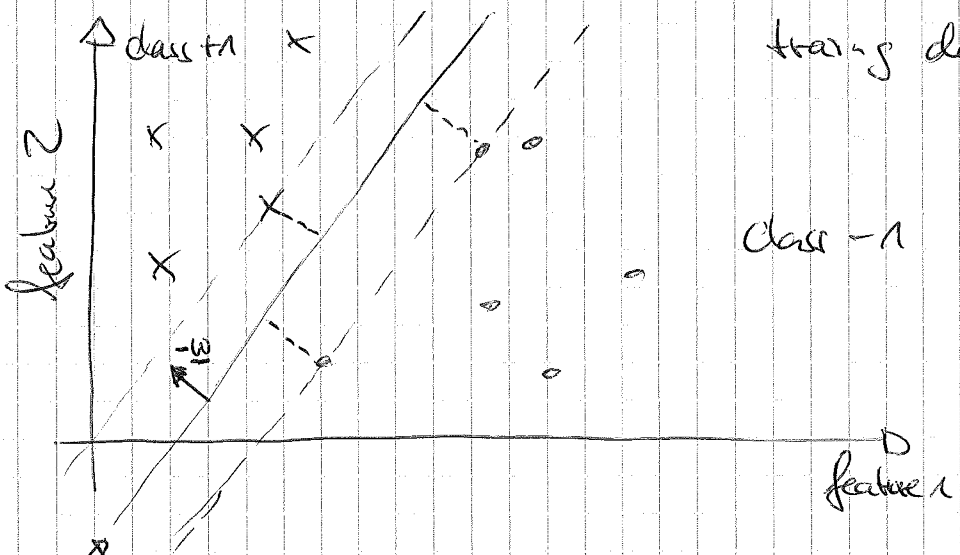


etc...

<u>HW</u>: Implement some possibilities and discuss the behavior on synth. toy training data.

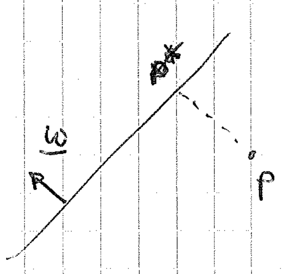
Support Vector Machines (lin. sep. case) $\boxed{\text{VII}}$

training data $(x^{(i)}, y^{(i)})_{i=1..m}$



$\mathbb{R}^{n+1}$

$w \cdot x = 0$



$$\text{dist}(p, w) = ?$$

$$= \| p - p^* \|$$

$$= \| \alpha \underline{w} \|$$

$$= \left\| \frac{w \cdot p}{\underline{w}^2} \underline{w} \right\| = \frac{|w \cdot p|}{\|\underline{w}\|}$$

$p^* = p + \alpha(\underline{w})$

for $w \cdot p^* = 0 \iff w \cdot p + \alpha \underline{w}^2 = 0$

$\iff \alpha = -\frac{w \cdot p}{\underline{w}^2}$

Want to find hyperplane w that maximizes the margin:

$$\rho_w := \min_{i=1,\dots,m} \frac{|w \cdot x^{(i)}|}{\|\underline{w}\|}$$

- for this purpose note that $w \cdot x = 0$ is scale independent $\Rightarrow$ we may scale w s.t.

$$\min_{i=1..m} |w \cdot x^{(i)}| = 1$$

- this means there could be points ~~X_+~~ and x_- the training data s.t.

$$w \cdot x_+ = +1$$
$$w \cdot x_- = -1$$

- with this we ~~defined~~

$$\gamma_w = \frac{1}{\|\underline{w}\|}$$

optimization problem

We may thus def the optimal choice w in terms of an optimization problem as follows:

I) $\underset{w \in \mathbb{R}^{n+1}}{\text{maximum}} \; \frac{1}{\|\underline{w}\|}$ subject to $\begin{cases} w \cdot x^{(i)} \geq 1 \text{ if } y^{(i)} = +1 \\ w \cdot x^{(i)} \leq -1 \text{ if } y^{(i)} = -1 \end{cases}$ for $i = 1, \ldots, M$

or equivalently

II) $\underset{w \in \mathbb{R}^{n+1}}{\text{minimum}} \; \frac{1}{2}\|\underline{w}\|^2$ subject to $y^{(i)}\, w \cdot x^{(i)} \geq 1$ for $i = 1 \ldots M$

<u>Prop.</u> If the training data is finite and lin. sep. there is a unique solution.

~~Proof: The objective function $F(w) = \frac{1}{2}\|w\|^2$ is smooth, $\partial_w F(w) = w$, Hessian $\partial_w^2 F(w) = \mathbb{1}$~~

~~$\Rightarrow$ F strictly convex~~

~~$\Leftrightarrow \forall \lambda \in (0,1), \forall w \neq \tilde{w}$:~~

~~$F(\lambda w + (1-\lambda)\tilde{w}) < \lambda F(w) + (1-\lambda) F(\tilde{w})$~~

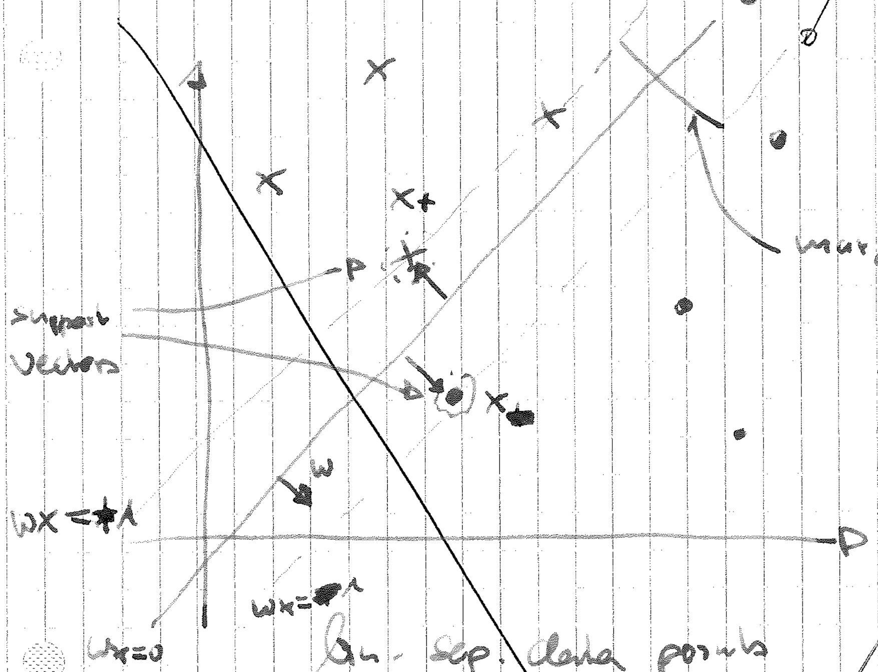
Support vector machines

VII

scale s.t. $w \cdot x_+ = +1$, $w \cdot x_- = -1$

optimal hyperplane

$0 = w \cdot x = w_0 + \underline{w} \cdot x_i$



lin. sep. data points

dist (x, w)

$x^0 = x + w\alpha$, $\quad wx = 0$

$=$

$wx^0 = 0 \Rightarrow wx + \|w\|^2\alpha = 0$

$\Rightarrow \frac{w}{\|w\|} x = -\|w\|\alpha$

$\Rightarrow \|x - x^0\| = \|\alpha w\| = \frac{|wx|}{\|w\|}$

- since $w \cdot x = 0$ also $\frac{w}{\|w\|} \cdot x = 0$
  and we may normalize w or any other scaling
- choose a scaling s.t. $w \cdot x_+ = +1$
  $w \cdot x_- = -1$

$$\frac{\underline{w}}{\|\underline{w}\|} \cdot (x_+ - x_-) = \frac{2}{\|\underline{w}\|} = \text{size of margin}$$

Hence, an optimal choice for w can be formulated by

- $\max_{w \in \mathbb{R}^{n+1}} \frac{2}{\|\underline{w}\|}$ subject to $w x^{(i)} \geq 1$ for $y^{(i)} = 1$, $w x^{(i)} \leq -1$ for $y^{(i)} = -1$, for $i = 1 \dots m$

- or equivalently

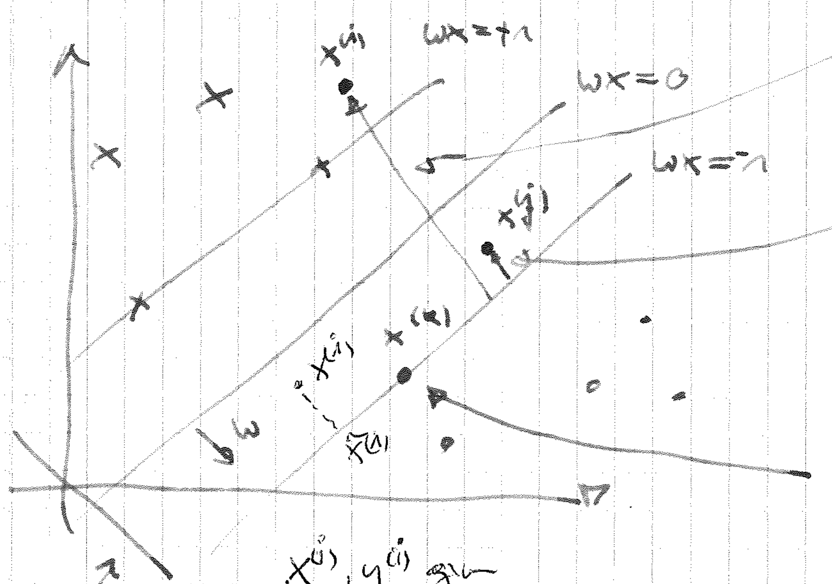
$$\min_{w \in \mathbb{R}^{n+1}} \|\underline{w}\|^2 \quad \text{subject to} \quad y^{(i)} \underline{w} x^{(i)} \geq 1 \quad \text{for } i = 1 \dots m$$

This is a quadratic optimization problem
subject to linear constraint

$\Longrightarrow$ there is a unique minimum

$F(w) = \frac{1}{2}\|w\|^2$ inf. diff. $\partial_w F = w$, $\partial_w^2 F = \mathbb{1} \Rightarrow$ strictly convex, constraint affine
$\Rightarrow$ local min = global min and since strictly convex: min is unique

The general case for only approximately lin. sep. data



$x^{(i)}, y^{(i)}$ given

$\tilde{x}^{(i)} = x^{(i)} + \alpha \begin{pmatrix} 0 \\ \underline{w} \end{pmatrix}$, $w\tilde{x}^{(i)} = y^{(i)}$

$\Rightarrow \|x^{(i)} - \tilde{x}^{(i)}\| = \frac{|1 - y^{(i)} \cdot wx^{(i)}|}{\|\underline{w}\|}$

Soft margin optimization problem

slack variable

$$\underset{\substack{w \in \mathbb{R}^{n+1} \\ \xi^{(i)} \in \mathbb{R}^+}}{\text{minimum}} \quad \|\underline{w}\|^2 + \mu \sum_{i=1}^{m} \xi^{(i)}$$

subject to $y^{(i)}(w \cdot x) \geq 1 - \xi^{(i)}$

- every constraint can be satisfied for $\xi^{(i)}$ suff. large
- $\mu \in \mathbb{R}^+$ is a regularization parameter that controls the weight of the margin
- Still a quadratic optimization problem $\Rightarrow$ unique with lin. constraints solution

OPTIMIZATION THEORY

History: 1) 1629: Fermat's theorem

$$f(x) \longrightarrow \text{extrema}$$

THM: Let $f: \mathbb{R}^n \to \mathbb{R}$ diff at $x^* \in \mathbb{R}^n$. If x^* is a local extremum, then $f'(x^*) = 0$ (stat. point).

Then discussion of boundary, non-diff., stationary points.

2) 1788: Lagrange's theorem

$f_0(x) \longrightarrow$ minimum subject to equality constraints

$$f_1(x), \dots f_m(x) = 0$$

THM: Let $f_i: \mathbb{R}^n \to \mathbb{R}$ be cont. diff in the vicinity of a local extremum, then one can find $z^* \in \mathbb{R}^{m+1}$ s.t.

$$L(x^*, z^*) = \sum_{i=0}^{m} z_i^* \cdot f_i(x^*) \quad , \quad z^* \neq 0$$

fulfills

$$\nabla_{(x,z)} L(x, z) \Big|_{(x,z) = (x^*, z^*)} = 0 \qquad (1)$$

If $\nabla f_i(x^*)$ $i = 1 \dots m$ are linear independent $z_0^* \neq 0$.

After solving (1) we find stat. point to discuss.

3) 1951: Karush-Kuhn-Tucker theorem

$f_0(x) \longrightarrow$ minimum subject to constraints

$$f_1(x), \dots, f_m(x) \leq 0$$

} minimization program

THM: Let $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 0, \dots, m$, be convex. If x^* is a minimum of above program, then $\exists\, z^* \in \mathbb{R}^{m+1}$, $z^* \neq 0$, s.t.

i) $\inf_x L(x, z^*) = L(x^*, z^*)$

ii) $z_i^* \geq 0$ for $i = 0, \dots, m$

iii) $z_i^* f_i(x^*) = 0 \quad i = 1 \dots m$

a) If $z_0^* \neq 0$, (i)–(iii) $\Rightarrow$ x^* a solution of minimization problem.

(Slater condition) b) If $\exists\, \bar{x}$ s.t. $f_i(\bar{x}) < 0 \quad i = 1 \dots m \Rightarrow z_0^* \neq 0$.

c) If the Slater cond. is satisfied ~~##~~ the Lagrange function

$$L(x, z) = f_0(x) + \sum_{i=1}^m z_i f_i(x)$$

has a saddle point at $(x^*, z^*/z_0^*)$, i.e.,

$$L\left(x, \frac{z^*}{z_0^*}\right) \geq L\left(x^*, \frac{z^*}{z_0^*}\right) \geq L\left(x^*, z/z_0\right) \quad \forall x, z, z^0 \neq 0$$

Existence of optimal solutions

Optimization program:

$$(P) \quad f_0(x) \to \text{minimum subject to} \quad \begin{array}{ll} f_i(x) \leq 0 & i = 1 \dots p \\ f_j(x) = 0 & j = p+1 \dots m \\ x \in C \subseteq \mathbb{R}^n & \end{array}$$

THM: Let C be closed, $f_i : \mathbb{R}^n \to \mathbb{R}$ cont. and f_0 coercive over $S := \{x \in C \mid f_i(x) \leq 0, f_j(x) = 0$ for $i = 1 \dots p, j = p+1 \dots m\}$

[$\Leftrightarrow$ $\forall (x_n)_{n \in \mathbb{N}}$ in S with $\|x_n\| \xrightarrow[n\to\infty]{} \infty$ also $|f(x_n)| \to \infty$,

then (P) has at least one optimal solution.

Proof: • C closed, f_i cont $\Rightarrow$ S closed

• let $(x_n)_{n \in \mathbb{N}}$ be a sequence in S s.t.

$$f_0(x_n) \xrightarrow[n\to\infty]{} \inf_{x \in S} f_0(x)$$

• since f_0 is coercive and $(f_0(x_n))_{n \in \mathbb{N}}$ obviously bounded also $(x_n)_{n \in \mathbb{N}}$ is bounded

• $\Rightarrow \exists (x_{n_k})_{k \in \mathbb{N}}$ s.t. $x_{n_k} \xrightarrow[k\to\infty]{} x^* \in S$ since S was closed

• but f_0 cont. $\Rightarrow$ $f_0(x^*) = \lim_{k\to\infty} f_0(x_{n_k}) = \inf_{x \in S} f_0(x)$

Hence, at least there is one x^* fulfilling (P). □

THM: Let f_i, $i=1 \dots m$, convex, then:

i) f_0 convex $\Rightarrow$ local min = global min

ii) f_0 strictly convex $\Rightarrow$ min. unique.

Proof: i) • let $x^* \in S$ be a local min.

$\Rightarrow \exists \varepsilon > 0 : \forall x \in B_\varepsilon(x^*) \cap S : f(x^*) \leq f(x)$

• As ~~f_i~~ f_i $i=1 \dots m$ are convex, ~~also~~ S is convex

• take $y \neq x$, $y \in S$, $\lambda \in (0,1]$ s.t.

$$x^* + \lambda (y - x^*) \in B_\varepsilon(x^*)$$

$$\Rightarrow f(x^*) \leq f(x^* + \lambda(y - x^*)) \leq (1-\lambda) f(x^*) + \lambda f(y)$$

$$\Rightarrow f(x^*) \leq f(y)$$

ii) same argument with strict inequalities. □

Proof of our Props about SVMs:

$$f_0(w) = \frac{1}{2} \|w\|^2 \Rightarrow \partial_w f_0(w) = w \quad \text{(gradient)}$$

$$\Rightarrow \partial_w^2 f_0(w) = \mathbb{1} \quad \text{(Hessian)}$$

Hence, f_0 is strictly convex, and also coercive, $\mathbb{R}^n$ closed.
Furthermore, f_i are affine $i=1 \dots m$, $\Rightarrow$ convex

$\Rightarrow$ above THMs apply and there is a unique solution.

□

Convex Optimization Problems

Proof of LEP 1-2 (I)

Optimization program:

(notation $f_0(x) = f(x)$)

$$(P) \quad \inf_{x \in \mathbb{R}^n} \quad f(x) \quad \text{subject to} \quad \begin{array}{ll} f_i(x) \le 0 & i = 1 \dots p \\ f_j(x) = 0 & j = p+1 \dots m \\ x \in C & \end{array}$$

DEF: $S := \{ x \in C \mid f_i(x) \le 0, f_j(x) = 0, i = 1 \dots p, j = p+1 \dots m \}$

ASSUMPTIONS:

(A1) $C \subseteq \mathbb{R}^n$ convex s.t. $\operatorname{supp} f_i \supseteq C$, $i = 1 \dots m$

(A2) $f_i : \mathbb{R}^n \to \overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$ convex for $i = 1 \dots p$

(A3) $f_j : \mathbb{R}^n \to \mathbb{R}$ affine for $j = p+1 \dots m$

REM: (A1)-(A3) $\Rightarrow$ S convex
and furthermore if C closed $\wedge$ $f_i|_C$, $f_j|_C$
continuous (implied by convexity) $\Rightarrow$ S closed

ASSUMPTIONS:

(A4) for convex S holds:

a) $\exists \bar{x} \in S \cap C^\circ$

b) $\forall f_i$, $i = 1 \dots p$, not affine $\exists x_i \in S$: $f_i(x_i) < 0$

This is called Slater condition.

THM: (P) fulfilling
- (A1)-(A3)
- (A4) Slater cond
- $\alpha := \inf \{ f(x) \mid x \in S \} \in \mathbb{R}$

$\Rightarrow \exists y \in \mathbb{R}^m$, $y_i \ge 0$ $i = 1 \dots p$ s.t.

$$f(x) + \sum_{i=1}^{p} y_i f_i(x) \ge \alpha \quad \text{for all } x \in C$$

REM: Only finiteness of α is assumed, not existence of $x^* \in C$ with $f(x^*) = \alpha$, however, if $x^* \subset C_i$ is the optimal value $f(x^*) = \alpha$ holds.

Proof: • Preliminary: $(A4)\ b) \Longrightarrow \exists \hat{x} \in S$ s.t. $f_i(\hat{x}) < 0$ for all non-affine functions, say, $i = 1 \dots l$

Proof: $\forall i = 1 \dots l \ \exists x_i \in S$ with $f_i(x_i) < 0$, $f_k(x_i) \leq 0$ and $f_j(x_i) = 0$ for $k \neq i, k = 1 \dots p$

$\hat{x} := \frac{1}{l}(x_1 + x_2 + \dots x_n) \in S$ due to convexity

and $f_i(\hat{x}) = f_i\left(\frac{1}{l}\sum_{n=1}^{l} x_n\right)$

$$\leq \frac{1}{l}\sum_{n=1}^{l} f_i(x_n) = \frac{1}{l}\Big(\underbrace{f_i(x_1)}_{\leq 0} + \dots + \underbrace{f_i(x_i)}_{< 0} \dots \underbrace{f_i(x_l)}_{\leq 0}\Big)$$

< 0 $\square$

• To simplify the cases, let us assume $\exists \hat{x} \in S$ with $f_i(\hat{x}) < 0$ for $i = 1 \dots p$ (also for the affine function)

(*) • By (A4) a) $\exists \bar{x} \in S \cap C^0$ so that by convexity $\bar{x}$ can be ~~selected~~ to fulfill $f_i(\bar{x}) < 0$ $i = 1 \dots p$

• WLOG $\alpha = 0$

PART 1: Without (A4) we prove that $\exists z \in \mathbb{R}^{m+1}, z \neq 0$:

$$z_i \geq 0, \ i = 1, \dots, p \ \wedge \ \sum_{i=0}^{p} z_i f(x) \geq 0 \quad \forall x \in C$$

PART 2: With (A4) we show that $z_0 > 0$

$\Longrightarrow y := (z_i / z_0)_{i = 1 \dots m}$

Proof of Part 1:

Proof of LEMMA 1-2 $\boxed{IV}$

$$A := \left\{ v \in \mathbb{R}^{m+1} \;\middle|\; \exists x \in C : \begin{array}{ll} v_0 > f_0(x) & \\ v_i \geq f_i(x) & i = 1..p \\ v_j = f_j(x) & j = p+1...m \end{array} \right\}$$

C convex $\wedge$ f_i convex $i=1..p$ $\wedge$ f_j affine $j=p+1...m$
$\Rightarrow f_j$ convex

$\Rightarrow$ • A convex
• $0 \notin A$ because of $\alpha = 0$
• $A \neq \emptyset$ as v_i can be chosen arbitrarily negative

<u>LEMMA</u>: $A \neq \emptyset$, $0 \notin A$, A convex $\subseteq \mathbb{R}^{m+1}$
$\Rightarrow \exists z \in \mathbb{R}^{m+1}, z \neq 0$ s.t. i) $z \cdot v \geq 0 \;\forall v \in A$
ii) $\exists \bar{v} \in A : z \cdot \bar{v} > 0$

• But $v \in A \Rightarrow v + \lambda w \in A \quad \forall \lambda \geq 0, w \geq 0$
and $\lim_{\lambda \to +\infty} z \cdot (v + \lambda w) \geq 0 \Rightarrow z \cdot w \geq 0 \Rightarrow z \geq 0$

• for all $x \in C$, $\varepsilon > 0$ $\exists\; v_\varepsilon := \begin{pmatrix} f_0(x) + \varepsilon \\ f_i(x) \\ f_j(x) \end{pmatrix} \in A$

$$0 \leq z \cdot v_\varepsilon = z_0 \left(f(x) + \varepsilon \right) + \sum_{i=1}^{p} z_i f_i(x)$$

$$\varepsilon \to 0 \Rightarrow \quad 0 \leq z_0 f(x) + \sum_{i=1}^{p} z_i f_i(x) \;, \; z_i \geq 0, \; z \neq 0$$

Proof of Part 2:

- Let us assume $z_0 = 0$

$$\Rightarrow \quad v = \Big(f(\hat{x})+1,\ f_{i=1 \dots p}(\hat{x}),\ \overset{0,\dots,0}{\cancel{\quad}} \Big) \in A$$

because of (A4) a) (see *)

$\Rightarrow$ $z \cdot v \geq 0$

but since $z_0 = 0$, $f_i(\hat{x}) < 0$

$\Rightarrow$ $z_0, z_1, \dots z_p = 0$

$\Rightarrow$ due to A: $\sum_{j=p+1}^{m} z_j f_j(x) \geq 0 \quad \forall x \in C$

and since $\{0\}$ and A well separated $\exists\ \tilde{x} \in C$:

$$\sum_{j=p+1}^{m} z_j f_j(\tilde{x}) > 0$$

- because $\hat{x} \in C^{\circ}$ we have $\hat{x} - \varepsilon(\tilde{x} - \hat{x}) \in C$

- since f_j affine for $j = p+1 \dots m$

$$f_j(\hat{x} - \varepsilon(\tilde{x} - \hat{x})) = \underbrace{f_j(\hat{x})}_{=0} - \varepsilon\left[f_j(\tilde{x}) - f_j(\hat{x})\right]$$

$$= -\varepsilon f_j(\tilde{x})$$

$$\Rightarrow \sum_{j=p+1}^{m} z_j f_j(\hat{x}) = -\varepsilon \sum_{j=p+1}^{m} z_j f_j(\tilde{x}) < 0 \quad ↯$$

$\Rightarrow$ $z_0 \neq 0$ but $z_0 > 0$. $\square$

Optimality cond. for constraint convex optimization

AIM: Derive cond. which allow to decide whether a point $x \in S$ is optimal or not.
→ basis for many numerical receipts

Question is "the solvability of a system of equalities and inequalities".

(P) $f(x) \longrightarrow$ minimum subject to
$f_i(x) \leq 0 \quad i = 1..p$
$f_j(x) = 0 \quad j = p+1, .. m$
$x \in C$

DEF: $S := \{x \in C \mid f_i(x) \leq 0,\ f_j(x) = 0\ \ i = 1..p,\ j = p+1..m\}$

ASSUMPTIONS:

$$(A\text{convex}) = \begin{cases} (A1) & C \subseteq \mathbb{R}^n \text{ convex},\ C \subseteq \operatorname{supp} f_i \quad i = 1..m \\ (A2) & f_i : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\} \text{ convex for } i = 1..p \\ (A3) & f_j : \mathbb{R}^n \to \mathbb{R} \text{ affine for } j = p+1..m \end{cases}$$

REM: $(A\text{convex}) \Rightarrow S$ convex and closed (by continuity)

ASSUMPTION:

$$(A\text{Slater}) = \begin{cases} (A4)\ a) & \exists\, \bar{x} \in S \cap \mathring{C} \\ \quad\ \ b) & \forall i = 1..p,\ f_i \text{ non-affine } \exists\, x_i \in S : f_i(x) < 0 \end{cases}$$

The strategy will use the hyperplane separation theorems for convex sets to arrive at the lemmas:

<u>LEM 1:</u> Let (P) fulfill (Aconvex) and let

$$\alpha := \inf_{x\in S} f_0(x) \in \mathbb{R}$$ (finiteness only not existence of an optimal $x^* \in S$)

then $\exists\, z \in \mathbb{R}^{m+1}$, $z \neq 0$, $z_i \geq 0$, $i = 1 \dots p$ s.t.

$$z_0 \left[f_0(x) - \alpha \right] + \sum_{i=1}^{m} z_i \, f_i(x) \geq 0 \quad \forall x \in C.$$

<u>LEM 2:</u> Assumptions as in LEM 1 but in addition (Aslater).
then $\exists\, y \in \mathbb{R}^m$, $y_i \geq 0$, $i = 1 \dots p$ s.t.

$$\left[f_0(x) - \alpha \right] + \sum_{i=1}^{m} y_i \, f_i(x) \geq 0 \quad \forall x \in C$$

<u>Implications:</u> Let x^* be a optimal solution of (P)
$\Rightarrow \quad \alpha = f(x^*)$

(Aconvex) $\overset{\text{LEM 1}}{\Longrightarrow}$ $\exists\, z$ ~~$[f_0(x) + f_0(x^*)]$~~ $\in \mathbb{R}^{m+1}$, $z \neq 0$, $z_i \geq 0$ for $i = 1 \dots m$ s.t.

$$z_0 \left[f_0(x) - f_0(x^*) \right] + \sum_{i=1}^{m} z_i \, f_i(x) \geq 0 \qquad (*)$$

Suppose f_i, $i = 0 \dots m$, are diff, then

$$\begin{cases} (\mathrm{I}) & \displaystyle\sum_{i=0}^{m} z_i \, \nabla f_i(x^*) = 0 \\ (\mathrm{II}) & f_i(x^*)\, z_i = 0, \quad f_i(x^*) \leq 0, \quad z_i \geq 0 \quad i = 1 \dots p \\ (\mathrm{III}) & f_j(x^*) = 0 \quad j = p+1 \dots m \end{cases}$$

IV

II

Because: (I)

- $\phi(x) = z_0 \left(f_0(x) - f_0(x^*) \right) + \sum_{i=1}^{m} z_i f_i(x)$

  is convex & diff & $\phi(x^*) \leq 0$

- but also (*) $\Rightarrow$ $\phi(x) \geq 0$

  $\rightarrow$ x^* gives rise to min. of $\phi(x)$

  $\Rightarrow$ $\nabla_x \phi(x) \big|_{x=x^*} = 0$ (Fermat's theorem)

  which implies I)

(II) (*) $\Rightarrow$ $z_i \geq 0$ $i = 0 .. p$, $f_i(\bar{x}) \leq 0$ $i = 1 .. p$

$\Rightarrow$ if $z_k f_k(x^*) \neq 0 \Rightarrow z_k f_k(x^*) < 0$

$\Rightarrow$ $\phi(x^*) < 0$ $\lightning$ $\Rightarrow$ $z_k f_k(x^*) = 0$

(III) direct from (P).

By LEMMA 2 & (Aslater), the same holds for $y = \frac{z^*}{z_0}$ because $z_0 \neq 0$:

THM: (P) fulfills (Aconvex) & (Aslater). If $x^* \in S$ is optimal and f_i are diff at x^* $i = 0, \dots, m$, then $\exists\, y \in \mathbb{R}^m$, $y_i \geq 0$:

(KKT conditions)

(I) $\sum_{i=1}^{m} y_i \nabla f_i(x^*) + \nabla f_0(x^*) = 0$

(II) $f_i(x^*)\, y_i = 0$, $f_i(x^*) \leq 0$, $y_i \geq 0$ $i = 1 .. p$

(III) $f_j(x^*) = 0$ $j = p+1 .. m$

Example 2: lin. sep. case SVM

$$f_0(w) = \frac{1}{2}\|w\|^2$$

$$f_i(w) := 1 - y^{(i)}(w \cdot x^{(i)})$$

We know there is a unique optimal solution (say $w^* \in \mathbb{R}^{n+1}$) for (P) in this case $\Rightarrow \exists\, \alpha \in \mathbb{R}^m, \alpha_i \geq 0$

(I) $\quad \partial_w f_0(w^*) + \sum_{i=1}^{m} \alpha_i \partial_w f_i(w^*) = 0$

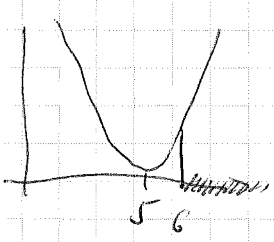
$\Rightarrow \quad w^* = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$

(II) $\quad \alpha_i f_i(w^*) = 0 \qquad i = 1 \dots m, \ \alpha_i \geq 0$

In conclusion, w^* can be given by training data by (I) but only some points actually contribute, those for which $f_i(w^*) = 0$ otherwise (II) implies $\alpha_i = 0$!

This is the reason for the name SVM.

Example 1:



A) $f_0(x) = (x-5)^2$

$f_1(x) = 6 - x$

KKT: $f_0'(x) + z_1 f_1'(x) = 0$

$\iff 2(x-5) - z_1 = 0$

$z_1 \geq 0, \ f_1(x) z_1 \geq 0, \ f_1(x) \leq 0$

$z_1 > 0 \Rightarrow f_1(x) = 0 \Rightarrow x = 6$ ✓

$z_1 = 0 \Rightarrow 2(x-5) = 0 \Rightarrow x = 5, \ f_1(x) = 1$ ↯

B) $f_0(x) = (x-5)^2$

$f_1(x) = 3 - x$

KKT: $2(x-5) - z_1 = 0$

$z_1 \geq 0, \ f_1(x) \leq 0$

$z_1 f_1(x) \leq 0$

$z_1 > 0$: $x = 3$

$\Rightarrow 2(3-5) - z_1 = 0$

$\Rightarrow z_1 = -4$ ↯

$z_1 = 0$: $x = 5$

Lagrange function

V ③

Gives a nice interpretation of the results so far

Ex: To solve $\inf_{x\in C}\{f_0(x) \mid f_1(x) \le 0\}$ (P)

we may look at a family of auxiliary problems: fix $y \ge 0$ and solve

$$\inf_{x\in C}\{f_0(x) + y f_1(x)\} \qquad (P_y)$$

Then y is the "weight" of the constraint.

Let $x^*(y)$ be optimal for (P_y) and y given, then

- $x^*(0)$ may violate $f_1(x^*(0)) \le 0$
- for y very large one may have $f_1(x^*(0)) < 0$ but $x^*(y)$ may be far from optimal
- there should be an intermediate value $\bar{y}$ s.t. $f_1(x^*(\bar{y})) = 0$
  $\rightarrow$ $x^*(\bar{y})$ is also a solution of (P).

DEF: i) Let $D = \{y \in \mathbb{R}^m \mid y_i \ge 0 \;\; i = 1..p\}$.
We call $L : C \times D \to \mathbb{R}$ Lagrange function of (P):

$$L(x,y) := f_0(x) + \sum_{i=1}^{m} y_i f_i(x)$$

ii) We call $(\bar{x},\bar{y}) \in C \times D$ saddle point of L iff

$$L(x,\bar{y}) \ge L(\bar{x},\bar{y}) \ge L(\bar{x},y) \quad \forall x \in C, y \in D$$

Theorem of KKT:

Thm: Let (P) fulfill (A_{convex}). Then the following holds:

i) $(\bar{x},\bar{y})$ saddle point of $L \Rightarrow \bar{x}$ optimal for (P) and $\bar{y}_i f_i(\bar{x}) = 0$, i.e.,

$$L(\bar{x},\bar{y}) = f_0(\bar{x})$$

ii) If $\bar{x}$ is optimal for (P) and (A_{Slater}), then $\exists\, \bar{y} \in D$ s.t. $(\bar{x},\bar{y})$ is saddle point of L.

iii) If $\alpha \in \mathbb{R}$ & (A_{Slater}) $\exists\, \bar{y} \in D$ s.t.

$$\inf\{f_0(x) \mid x \in S\} = \alpha = \inf_{x \in C} L(x,\bar{y}) = \max_{y \in D} \inf_{x \in C} L(x,y).$$

Proof: i) Let $(\bar{x},\bar{y})$ be saddle point of $L(x,y)$, then $\forall\, y \in D$: $L(\bar{x},y) \le L(\bar{x},\bar{y})$

$$\Rightarrow \forall\, y \in D \quad \sum_{i=1}^{m} y_i f_i(\bar{x}) \le \sum_{i=1}^{m} \bar{y}_i f_i(\bar{x})$$

$$y_i \to +\infty \Rightarrow f_i(\bar{x}) \le 0$$

$$y_i \to 0 \Rightarrow \bar{y}_i f_i(\bar{x}) = 0$$

and f_j affine $\Rightarrow f_j(\bar{x}) = 0 \Rightarrow \bar{x} \in S$

But $L(\bar{x},\bar{y}) \le L(x,\bar{y})$ for $x \in S$

$$f_0(x^*) \le f_0(x) + \sum_{i=1}^{p} \bar{y}_i f_i(x) + \sum_{i=p+1}^{m} \bar{y}_i f_i(x) \le f_0(x) \quad \square \text{ i)}$$

Proof: i) $(\bar{x}, \bar{y})$ saddle point of L

$\Leftrightarrow \forall x \in C, y \in D$:

$$L(x, \bar{y}) \overset{(1)}{\geq} L(\bar{x}, \bar{y}) \overset{(2)}{\geq} L(\bar{x}, y)$$

$$(2) \Leftrightarrow \sum_i \bar{y}_i f_i(\bar{x}) \geq \sum_i \bar{y}_i f_i(\bar{x})$$

$y_j \to +\infty \Rightarrow f_j(\bar{x}) \leq 0$

$y_i \to \infty \Rightarrow f_i(\bar{x}) \leq 0$, $\quad y_j \to -\infty \Rightarrow f_j(\bar{x}) \geq 0$

$y_i \to 0 \Rightarrow \sum_i \bar{y}_i \underbrace{f_i(\bar{x})}_{\leq 0} \geq 0 \Rightarrow \bar{y}_i f_i(\bar{x}) = 0$

$$(1) \Rightarrow f_0(x) + \sum_i \bar{y}_i f_i(x) \geq f_0(\bar{x}) + \sum_i \bar{y}_i f_i(\bar{x})$$

$$\Rightarrow \quad f_0(x) \geq f_0(\bar{x}).$$

Hence, $\bar{x}$ is a minimum of f_0 for all x: $f_i(x) \leq 0$
$f_j(x) = 0$.

ii) $\bar{x}$ optimal sol. of (P) $\Leftrightarrow \forall x$: $f_0(\bar{x}) \leq f_0(x)$
$f_i(\bar{x}) \leq 0 \quad i = 1 \ldots p$
$f_j(\bar{x}) = 0 \quad j = p+1 \ldots m$

$(A_{convex}) + (A_{Slater}) \overset{\text{Lem 2}}{\Longrightarrow} \exists \bar{y} \in D$ s.t. $\forall x \in C$

$$L(x, \bar{y}) = f(x) + \sum_i \bar{y}_i f_i(x) + \sum_j \bar{y}_j f_j(x) \geq f(x^*)$$

for $x = \bar{x} \Rightarrow \sum_i \bar{y}_i f_i(\bar{x}) \geq 0$ but $\bar{y}_i \geq 0$, $f_i(\bar{x}) \leq 0$
implies $\bar{y}_i f_i(\bar{x}_i) = 0$

$$\Rightarrow L(x, \bar{y}) \geq f(\bar{x}^*) = L(\bar{x}, \bar{y}) \geq f(\bar{x}) + \sum_i y_i f_i(\bar{x}) + \sum_j y_j f_j(\bar{x})$$
$$= L(\bar{x}, y)$$

iii) $\alpha := \inf_{x \in S} f(x) \in \mathbb{R}$

(A convex) + (A slater) $\overset{\text{Lem 2}}{\Longrightarrow} \exists \bar{y} \in D$

$$L(x, \bar{y}) \geq \alpha$$

$$\Rightarrow \inf_{x \in C} L(x, \bar{y}) = \inf_{x \in C} \left[f_0(x) + \sum_i \bar{y}_i f_i(x) + \sum_j \bar{y}_j f_j(x) \right]$$

$$= \inf_{x \in S} L(x, \bar{y}) = \alpha$$

$$\sup_{y \in D} L(x, y) = \begin{cases} f(x) & \text{for } x \in S \\ +\infty & \text{else} \end{cases}$$

$$\Rightarrow \inf_{x \in C} \sup_{y \in D} L(x, y) = \inf_{x \in S} f(x) = \alpha$$

$$\Rightarrow \alpha = \inf_{x \in C} \sup_{y \in D} L(x, y) \geq \sup_{y \in D} \inf_{x \in C} L(x, y)$$

$$\geq \inf_{x \in C} L(x, \bar{y}) = \alpha$$

$$\Rightarrow \alpha = \sup_{y \in D} \inf_{x \in S} L(x, y) = \max_{y \in D} \inf_{x \in S} L(x, y)$$

$\square$

Rem: note that $L(x, \bar{y}) \geq L(\bar{x}, \bar{y})$

implies because: $\nabla f_0(x) + \sum_i y_i \nabla f_i(x) + \sum_j y_j \nabla f_j(x) = 0$

$\boxed{\text{HW}}$ $(\bar{x}, \bar{y})$ saddle point of $L \Leftrightarrow$ KKT conditions hold for $(\bar{x}, \bar{y})$

PRIMAL & DUAL FORM OF SVM

Primal program

$$f_0(w,\xi) = \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{m} \xi_i \longrightarrow \min$$

subject to constraints

$$f_i(w,\xi) = 1 - y^{(i)} w\cdot x^{(i)} - \xi_{(i)} \leq 0 \qquad i = 1\dots m$$

$$g_j(w,\xi) = -\xi_{(j)} \leq 0 \qquad j = 1\dots m$$

$\Rightarrow$ Lagrange function

$$L(w,\xi,\alpha,\beta) = f_0(w,\xi) + \sum_{i=1}^{m} \alpha_i f_i(w,\xi) + \sum_{j=1}^{m} \beta_j g_j(w,\xi)$$

at optimum w,ξ,α,β we have KKT:

(I) $\begin{pmatrix}\partial_{w_0} \\ \partial_{\underline{w}} \\ \partial_{\xi_k}\end{pmatrix} \left[f_0(w,\xi) + \sum_{i=1}^{m} \alpha_i f_i(w,\xi) + \sum_{j=1}^{m} \beta_j g_j(w,\xi) \right] = 0$

(II) $\alpha_i f_i(w,\xi) = 0 \;,\quad \beta_j g_j(w,\xi) = 0 \;,\quad \alpha_i \geq 0 \;,\; \beta_j \geq 0$

for $i,j = 1\dots m$

$$\partial_{w_0} f_0(w,\xi) = 0 \;,\quad \partial_{\underline{w}} f_0(w,\xi) = \underline{w} \;,\quad \partial_{\xi_k} f_0(w,\xi) = C$$

$$\partial_{w_0} f_i(w,\xi) = -y^{(i)} \;,\quad \partial_{\underline{w}} f_i(w,\xi) = -y^{(i)}\underline{x}^{(i)} \;,\quad \partial_{\xi_k} f_i(w,\xi) = -\delta_{ik}$$

$$\partial_{w_0} g_j(w,\xi) = 0 \;,\quad \partial_{\underline{w}} g_j(w,\xi) = 0 \;,\quad \partial_{\xi_k} g_j(w,\xi) = -\delta_{jk}$$

$\boxed{11}$

$$(\mathrm{I}) \Leftrightarrow \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$\underline{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \underline{x}^{(i)}$$

$$C = \alpha_i + \beta_i \qquad i = 1 \ldots m$$

$$(\mathrm{II}) \Leftrightarrow \alpha_i \left(1 - y^{(i)} w \cdot x^{(i)} - \xi^{(i)}\right) = 0 \qquad \alpha_i \geq 0 \qquad i = 1 \ldots m$$

$$\beta_j \xi_j = 0 \qquad \beta_j \geq 0 \qquad j = 1 \ldots m$$

We note again as in the lin. sep. case that due to (I)

$$\underline{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \underline{x}^{(i)}$$

where due to (II) only vectors fulfilling $\alpha_i \neq 0$, i.e.,

$$1 - y^{(i)} w \cdot x^{(i)} - \xi^{(i)} = 0$$

contribute $\Rightarrow$ either support vectors with $\xi^{(i)} = 0$ or other vectors for $\xi^{(i)} \neq 0$. The latter however must satisfy $\beta_i \xi^{(i)} = 0 \Rightarrow \beta_i = 0 \Rightarrow \alpha_i = C$.

Hence, vectors contributing to $\underline{w}$ are either support vectors or outliers with $\alpha_i = C$.

Note however that even though w is unique as we have shown, the support vectors are not.

NOTE: This program has several advantages:

1) $\partial_\alpha^2 \tilde{f}_0(\alpha) = - y^{(i)} \underline{x}^{(i)} \cdot y^{(j)} \underline{x}^{(j)}$ which is negative semi-def.

$\Rightarrow$ $\tilde{f}_0(\alpha)$ concave

2) Constraints are all affine

$\Rightarrow$ $\exists!$ optimum and problem is always quadratic

We can furthermore express the activation output

$$h(x) = wx$$

by choosing one support vector $\bar{x}^{(*)}$, i.e., $w\bar{x}^{(*)} = \bar{y}^{(*)}$
and computing

$$w_0 = \bar{y}^{(*)} - \underline{w} \cdot \underline{\bar{x}}^{(*)}$$
$$= \bar{y}^{(*)} - \sum_{j=1}^{n} \alpha_j \, y^{(j)} \underline{x}^{(j)} \underline{\bar{x}}^{(*)}$$

$\Rightarrow$ The activation does not depend on the particular values of $x^{(i)}$ but only on inner products.

Duality

$S := \{x \in C \mid f_i(x) \le 0,\ f_j(x) = 0,\ i = 1..p,\ j = p+1..m\}$ I

minimize $f_0(x)$ $\quad x \in C$

subject to $f_i(x) \le 0$ $\quad i = 1..p$

$f_j(x) = 0$ $\quad j = p+1..m$

Lagrange function $L(x,y) = f_0(x) + \sum_{i=1}^{p} y_i f_i(x) + \sum_{j=p+1}^{m} y_j f_j(x)$

for $x \in C$, $y \in D = \{\mathbb{R}^m \mid y_i \ge 0 \ i = 1..p\}$

Dual form: $\hat{L}(y) := \inf_{x \in C} L(x,y)$

Say $\alpha := \inf_{x \in S} f_0(x)$ then

$$\hat{L}(y) \le \alpha$$

because

$$\hat{L}(y) = \inf_{x \in C} L(x,y) \le L(\overset{S}{\overset{\Psi}{x^*}}, y) \le f(x^*)$$

because the constraints are fulfilled

Hence, the optimal value of the dual problem $\delta := \sup_{y \in D} \hat{L}(y)$ fulfills

$$\delta \le \alpha$$

$\alpha - \delta$ is called the duality gap.

Example:

$$\text{min. } f_0(x) = x^2$$
$$\text{subj. } f_1(x) = ax + b \leq 0$$

$$L(x,y) = f_0(x) + y f_1(x), \; y \geq 0 \qquad \partial_x L(x,y) = 0 \iff \boxed{x = -\frac{ya}{2}}$$

$$\boxed{\tilde{L}(y) = -\frac{1}{4} y^2 a^2 + yb} \qquad y \geq 0$$

$$\tilde{L}'(y) = 0 \implies \boxed{y = \frac{2b}{a^2}}$$

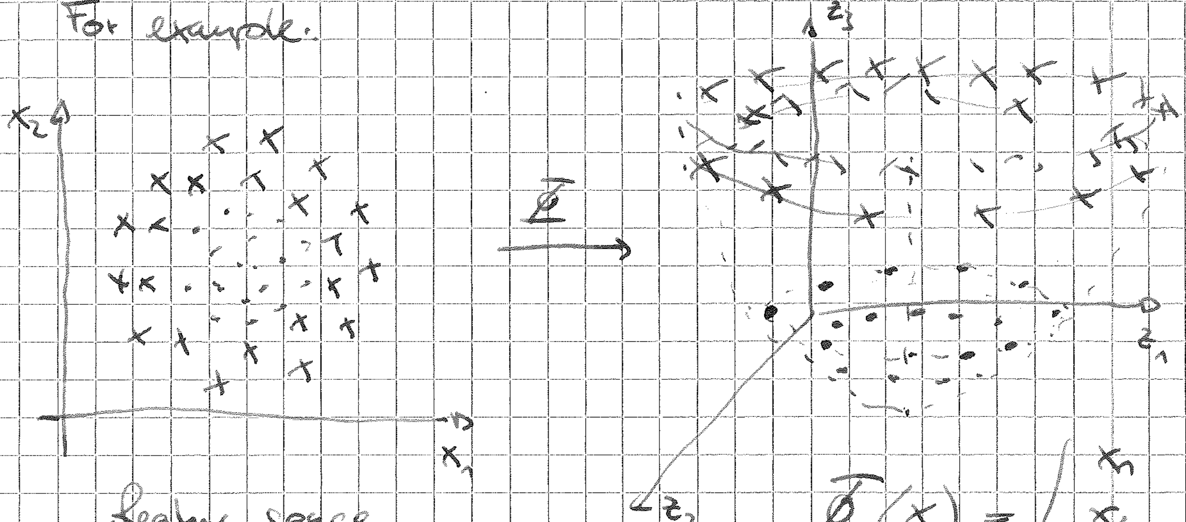
$$\implies \sup_{y \geq 0} \tilde{L}(y) = \begin{cases} \frac{2b}{a^2} & \text{for } b \leq 0 \implies x = -\frac{b}{a} \\ 0 & \text{for } b > 0 \implies x = 0 \end{cases}$$

~~$\iff$~~

NON-LINEAR CLASSIFIERS

1) General: Often it is a good strategy to map the feature space $X \in \mathbb{R}^n$ into a bigger space where the training data become lin. sep.

For example:



feature space $X \in \mathbb{R}^n$

$$\Phi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}$$

We may thus generalize Adaline as follows

$$h(x) = w \cdot \begin{pmatrix} 1 \\ \underline{\underline{\Phi}}(x) \end{pmatrix} = w \cdot \Phi(x)$$

and the loss function becomes

$$L(w) = \frac{1}{2} \sum_{i=1}^{M} \left(y^{(i)} - w \cdot \Phi(x) \right)^2$$

Learning & update rule as always

$$w \longmapsto \tilde{w} = w - \eta \frac{\partial L(w)}{\partial w}$$

$\boxed{\text{HW:}}$ Implement this with Iris data for non-lin. sep. features

Drawback: This can get very costly. Computation of $\Phi(x)$ can be $\mathcal{O}(n)$ and n is usually extremely large.

2) For SVMs we have seen that in the dual representation it only depends on inner products

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle \to \max$$

subject to $0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$

Using the map Φ we need to compute

$$\langle \Phi(x), \Phi(x') \rangle =: K(x, x')$$

K can be chosen freely as long as the existence of Φ is guaranteed and, in order to ensure concavity, K must be pos. def. symmetric:

DEF: $K: X \times X \to \mathbb{R}$ is pos. def. sym.

$\Leftrightarrow \forall\ x_1 \dots x_m \in X, \quad [K(x_i, x_j)]_{ij}$ is a pos. def. symmetric matrix in $\mathbb{R}^{m \times m}$

THM: (Mercer's cond.) $K: X \times X \to \mathbb{R}$ cont. and symmetric, $X \subset \mathbb{R}^n$ compact, then $\exists\ a_n > 0$, $\phi_n: X \to \mathbb{R}$ cont s.t.

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x')$$

iff K is the kernel of a pos. def. Hilbert-Schmidt operator, i.e.,

$$\forall f \in L^2(X): \quad \langle f, Kf \rangle \geq 0.$$

(I)

2) For SVM we have seen that in the dual representation the optimization problem depends entirely on inner products:

$$\text{maximize} \quad \tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \underline{x}^{(i)}, \underline{x}^{(j)} \rangle$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad \sum_{j=1}^{n} \alpha_j y^{(j)} = 0, \quad i = 1..n$$

Applying a map from feature space to some Hilbert space $\mathcal{H}$ with $\dim \mathcal{H} \gg n$

$$\Phi : X \longrightarrow \mathcal{H}$$

the scalar products turn into

$$\langle \Phi(\underline{x}^{(i)}), \Phi(\underline{x}^{(j)}) \rangle_{\mathcal{H}} = K(\underline{x}^{(i)}, \underline{x}^{(j)})$$

But since all depends on K and not directly on $\Phi(\cdot)$, there is no need to specify Φ, we only have to make sure K carries the properties that induces a scalar product.

~~Def the Shortcut kernel for soft-margin problem with hypothesis? [illegible]~~

<u>Advantages:</u> While $\langle \Phi(x), \Phi(y) \rangle$ usually take $\mathcal{O}(\dim \mathcal{H})$ to compute, $K(x,y)$ may take much less, e.g., $\mathcal{O}(n)$.

THM: (Mercer's cond.)

Let $X \subset \mathbb{R}^n$ be compact and $K : X \times X \to \mathbb{R}$ is a continuous and symmetric function. Then

$$T_K f := \int_X K(\cdot, z)\, dz \quad \forall f \in L^2(X)$$

is self-adjoint and there are eigenvectors/-values $\phi_1, \phi_2, \dots / \lambda_1, \lambda_2, \dots$ ordered by their geometric multiplicity.

Furthermore, (semi-positivity implies expansion and vice versa)

$$\forall f \in L^2 : \langle f, Tf \rangle_{\mathcal{X}} \geq 0 \iff K(x,y) = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(y), \quad \lambda_j \geq 0$$

where the convergence is absolute and uniform.

REM: Let $K(x,y)$ fulfill Mercer's cond. then the eigenvalue/vectors def. a map

$$\Phi(x) = \left(\sqrt{\lambda_i}\, \phi_i(x) \right)_{i \in \mathbb{N}}$$

and a Hilbert space $\mathcal{X}$ with a scalar product

$$\langle \Phi(x), \Phi(y) \rangle_{\mathcal{X}} = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(y).$$

DEF: (Pos def sym. kernels) A $K : X \times X \to \mathbb{R}$ is called pos. def. sym. if $\forall \{x_1 \dots x_m\} \subset X$ the matrix $(K(x_i, x_j))_{1 \leq i,j \leq m}$ is symmetric and pos. semidef.

(II)

Recall: A matrix M is sym. and pos. semi-def.

iff it is sym-dec and else:

i) all eigenvalues are non-neg.

ii) for any $v \in \mathbb{R}^M$: $\langle v, Mv \rangle \geq 0$

Example: (Polynomial Kernel)

$$\forall x, y \in \mathbb{R}^n: \quad K(x,y) = (x \cdot y + c)^d$$

for any $c \in \mathbb{R}$, $d \in \mathbb{N}$

For $n=2$, $d=2$ we find

$$\Phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2c}\,x_1 \\ \sqrt{2c}\,x_2 \\ c \end{pmatrix} \Rightarrow \dim(\mathcal{H}) = 6$$

$\boxed{HW}$ Show that $\dim(\mathcal{H}) = \binom{N+d}{d} = \frac{(N+d)!}{N!\,d!}$

~~addeknow~~

~~(Gaussian Kernel)~~

~~$\forall x, y \in \mathbb{R}^n: K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma}\right)$~~

~~for any $\sigma > 0$~~

Gaussian Kernel

$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma}\right) \quad \sigma > 0$$

Why pos. def. sym.

Lem Pos. def. sym. kernels are closed under sum, product, tensor product, point-wise limit and composition with power series $\sum a_n x^n$ for $a_n \geq 0$.

Proof: sum) clear

product) as $(K_{ij}) = \left(k(x_i, x_j)\right)_{1 \leq i,j \leq n}$ is pos. def. sym.

$\exists R$ s.t. $K = R^* R$

the kernel matrix associated with $k(x,y)\,k'(x,y)$ is

$(K_{ij} K'_{ij})_{1 \leq i,j \leq n}$. Hence for any $v \in \mathbb{R}^n$

$$\sum_{ij} K_{ij} K'_{ij} v_i v_j = \sum_{ij} \sum_k R_{ik} R_{jk} K'_{ij} v_i v_j$$

$$= \sum_k w_k K' w_k \geq 0 \quad \text{for } w_k = \begin{pmatrix} v_1 R_{1k} \\ \vdots \\ v_n R_{nk} \end{pmatrix}$$

and furthermore sym.

tensor product) $\tilde{k} : (x_1, x_1', x_2, x_2') \mapsto k(x_1, x_2)$

$\tilde{k}' : (x_1, x_1', x_2, x_2') \mapsto k'(x_1, x_2)$

$\Rightarrow k \otimes k'(x_1, x_1', x_2, x_2') = \tilde{k}\tilde{k}'$ pos. def. sym.

as product

point-wise limit) say $(k_n)_{n \in \mathbb{N}}$ pos. def. sym. kernels and $k_n(x,y) \to k(x,y)$ then for all $v \in \mathbb{R}^n$

$$\langle v, K_n v \rangle \geq 0 \Rightarrow \lim_{n \to \infty} \langle v, K_n v \rangle \geq 0 \text{ and sym.}$$

III

power series) Say $\sum_n a_n x^n$ converges with conv. radius ϱ. Suppose $|k(x,y)| < \varrho$ $\forall x,y \in X$, then $a_n k^n$ pos.def. sym. (by product) and $\sum_i a_i k^i$ pos.def. sym. (by limit)

~~HW: For any pos. def. sym. kernel~~

$$\cancel{k'(x,y) = \begin{cases} 0 & \text{for } k(x,x) = 0 = k(y,y) \\ \frac{k(x,y)}{\sqrt{k(x,x)\,k(y,y)}} & \text{else} \end{cases}}$$

~~is also a kernel which pos. def. sym.~~

(*)

$\boxed{\text{HW:}}$ Show that $k(x,y) = \tanh(a(x \cdot y) + b)$ is a pos. def. sym. kernel for $a \geq 0$, $b \geq 0$.

THM: (Reproducing kernel Hilbert space)

Let $k: X \times X \to \mathbb{R}$ be a pos. def. sym. kernel

Then: i) $\exists$ a Hilbert space $\mathcal{H}$ of real-valued functions and a map $\Phi: X \to \mathcal{H}$ s.t.

$$k(x,y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

ii) $\forall f \in \mathcal{H}, x \in X$:

$$f(x) = \langle f, k(x,\cdot) \rangle_{\mathcal{H}}$$

Proof: For any $x, y \in X$ def.

$$\bar{\Phi}(x)(y) = K(x,y)$$

Def. $\mathcal{H}^0 := \left\{ \sum_{i \in I} a_i \bar{\Phi}_i(x_i) \mid a_i \in \mathbb{R}, x_i \in X, |I| < \infty \right\}$

Def: $\langle \cdot, \cdot \rangle : \mathcal{H}^0 \times \mathcal{H}^0 \to \mathbb{R}$ s.t. for $f, g \in \mathcal{H}^0$,

i.e.,

$$f = \sum_{i \in I} a_i \bar{\Phi}_i(x_i)$$

$$g = \sum_{j \in J} b_j \bar{\Phi}_j(y_j),$$

$$\langle f, g \rangle := \sum_{\substack{i \in I \\ j \in J}} a_i b_j \underbrace{\bar{\Phi}_i(x_i)\bar{\Phi}_j(y_j)}_{K(x_i, y_j)} = \sum_{i \in I} a_i g(x_i)$$

$$= \sum_{j \in J} b_j f(x_j)$$

Note: i) $\langle f, g \rangle = \langle g, f \rangle$

ii) representation indep. $\Rightarrow$ well-def.

iii) $\langle f, f \rangle = \sum_{i,j \in I} a_i a_j K(x_i, x_j) \geq 0$

$\Rightarrow \langle \cdot, \cdot \rangle$ pos semi-def. bilinear form

$\Rightarrow \langle \cdot, \cdot \rangle$ pos semi-def. on $\mathcal{H}^0$ but $\forall f \in \mathcal{H}^0$

$$\langle f, \bar{\Phi}(x) \rangle^2 \leq \langle f, f \rangle \langle \bar{\Phi}(x), \bar{\Phi}(x) \rangle$$

$$= \langle f, f \rangle \underbrace{K(x,x)}_{\geq 0}$$

and $f(x) = \sum_{i \in I} a_i K(x_i, x) = \langle f, \bar{\Phi}(x) \rangle$

$$\Rightarrow |f(x)|^2 \leq \langle f, f \rangle \langle \bar{\Phi}(x), \bar{\Phi}(x) \rangle$$

IV

and therefore

$$f = 0 \iff \langle f, f \rangle = 0$$

Hence $\langle \cdot, \cdot \rangle$ is definite $\Rightarrow$ inner product on $\mathcal{H}^0$

Def $\mathcal{H} := \overline{\mathcal{H}^0}^{\langle \cdot, \cdot \rangle}$

Furthermore, $L_x : f \mapsto \langle f, \Phi(x) \rangle$ is bounded, hence, reproducing property holds on $\mathcal{H}$. $\square$

~~IV~~ V Normalization! (*)

<u>Conclusion</u>: SVM for non-lin. problem

1) $K(x,y)$ pos. def. sym.

2) maximize $\tilde{L}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} K(x^{(i)}, x^{(j)})$

subject to $0 \le \alpha_i \le C$, $\sum_i \alpha_i y^{(i)} = 0$

3) activation function (hypothesis)

$$h(x) = \sigma\left(\sum_i \alpha_i K(x_i, x) + w_0 \right)$$

$$w_0 = y^{(i)} - \sum_j \alpha_j y^{(j)} K(y^{(j)}, x^{(i)})$$

for any $x^{(i)}$ with $\alpha_i \in (0, C)$

Property 3) is quite general:

Module affect the hypothesis function is a

lin. comb. of $K(x_i, \circ)$:

<u>THM</u>: (Representer Theorem)

Let $K: X \times X \to \mathbb{R}$ pos. def. sym.,

$G: \mathbb{R} \to \mathbb{R}$ non-dec, $L: \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \Big\{ \underbrace{G(\|h\|_{\mathcal{H}}) + L(h(x_i), i = 1..n)}_{F(h)} \Big\}$$

admits sol. of the form

$$h^* = \sum_i^n a_i \, k(x_i, \circ)$$

and if G increasing only such one.

<u>Proof</u>: def. $\mathcal{H}_1 := \operatorname{span} \langle K(x_i, \circ), i = 1..n \rangle$

and split $\mathcal{H} = \mathcal{H}_1 \oplus \mathcal{H}_1^\perp$ s.t. $\forall h \in \mathcal{H}$

$\exists \, h_1 \in \mathcal{H}_1, h_1^\perp \in \mathcal{H}^\perp$ with $h = h_1 + h_1^\perp$

Clearly $G(\|h_1\|_{\mathcal{H}}) \le G(\|h\|_{\mathcal{H}})$

and by reproducing property

$$h(x_i) = \langle h, K(x_i, \circ) \rangle = h_1(x_i)$$

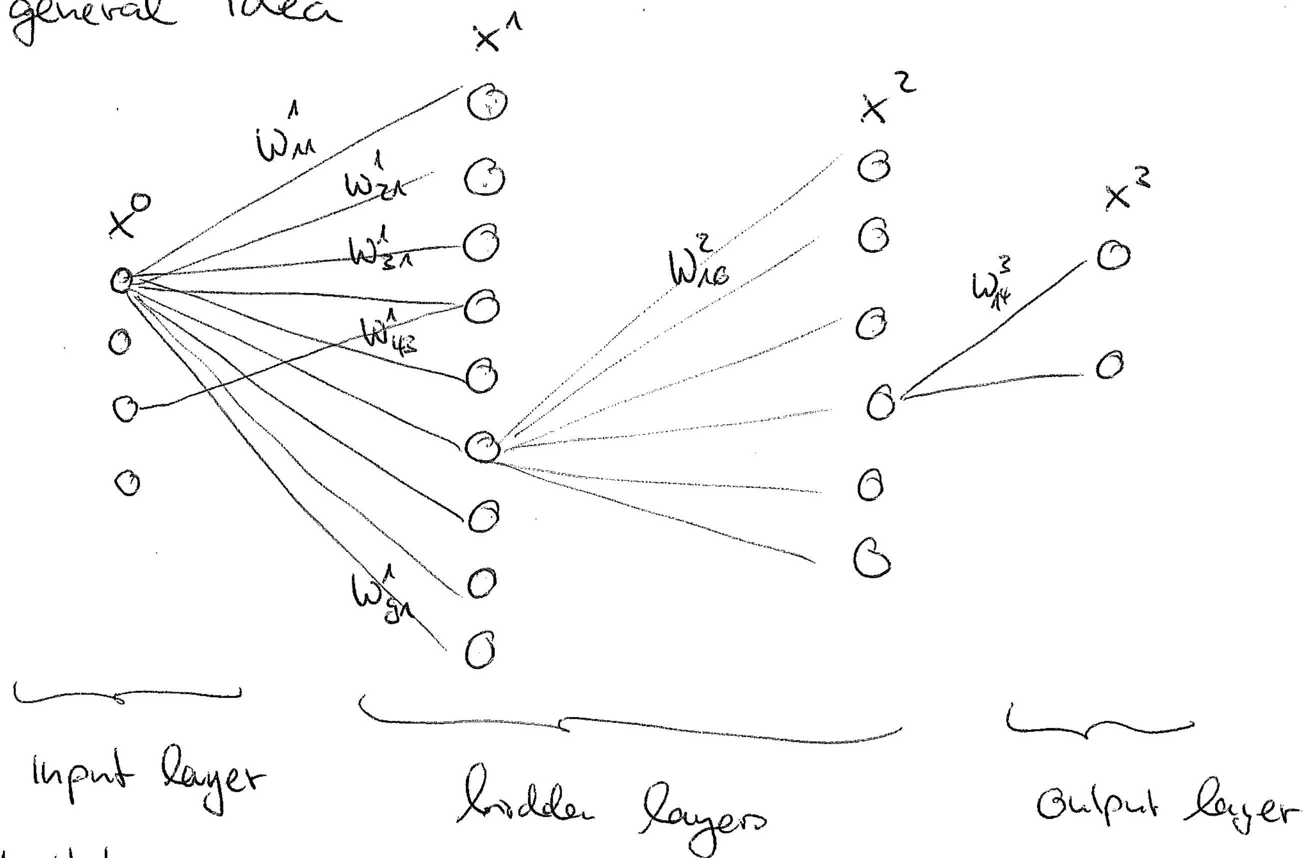
$$\Rightarrow L(h(x_i), i = 1..n) = L(h_1(x_i), i = 1..n)$$

$$\Rightarrow F(h_1) \le F(h).$$

The last assertion follows from the corresp. strict ineq. $\square$

MULTILAYER NETWORKS

general idea



network's forward pass:

$$\mathbb{R}^{n^0} \ni x^0 \longmapsto \underbrace{\overset{\mathbb{R}^{n^1\times n^0}}{\overset{\cup}{W^1}} \cdot x^1 + \overset{\mathbb{R}^{n^1}}{\overset{\cup}{b^1}}} =: z^1$$

$$\big\downarrow$$

$$x^1 := \sigma^1(z^1) \longmapsto \underbrace{W^2 x^1 + b^1} =: z^2$$

$$\big\downarrow$$

$$x^2 := \sigma^2(z^2) \longmapsto \underbrace{W^3 x^2 + b^3 =: z^3}$$

$$\big\downarrow$$

$$x^3 = \sigma^3(z^3)$$

- each layer takes as input x^{k-1} and applies the maps $x^{k-1} \mapsto z^k := W^k x^{k-1} + b^k \longrightarrow \sigma^k(z^k)$

where $W^k \in \mathbb{R}^{n^k \times n^{k-1}}$, $b^k \in \mathbb{R}^k$, $\sigma^k : \mathbb{R}^k \to \mathbb{R}^k$

$$(z_i)_{1\le i\le k} \mapsto (\bar\sigma^k(z_i))_{1\le i\le k}$$

for some $\bar\sigma^k : \mathbb{R} \to \mathbb{R}$

- the output of an N-Layer neural network is

$$x^N = \sigma(z^N) = \sigma(W^N x^{N-1} + b^N) = \dots = x^N(x^0)$$

this is what we usually called the hypothesis function $h(x^0) \equiv x^N(x^0)$

- given the training data $(x^{(i)}, y^{(i)})_{1 \le i \le M}$ with $x^{(i)} \in \mathbb{R}^{n^0}$ being the features and $y^{(i)} \in \mathbb{R}^{n^N}$ the "labels":
  - for classification we use the convention

$$\underset{\substack{\cap \\ \mathbb{N}}}{\text{class } i} \longleftrightarrow \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^{n^N}$$

i-th position

  - for regression $y^{(i)} \in \mathbb{R}^{n^N}$ are the n^N continuous values of the output that is desired

After the forward pass we want to compare the output of the network to the labels of our training data and update the weights we and biases b^l accordingly:

$$L = \frac{1}{M} \sum_{i=1}^{M} l_i \quad \text{as loss function}$$

for some function l_i depending on the i-th pair $(x^{(i)}, y^{(i)})$, for example $l_i = \frac{1}{2}\left(y^{(i)} - x^N(x^{(i)})\right)^2$

How should the weights and biases be updated? (2)

Gradient descent: let p^k be a place holder for some parameter in the k-th layer, i.e., W_{ij}^k, b_i^k for $1 \le i \le n^k$, $1 \le j \le n^{k-1}$

$$\frac{\partial L}{\partial p^k} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial l_i}{\partial p^k}$$

recall $l_i = l(y^{(i)}, x^N(x^{(i)}))$

depending on all the weights W^l and biases b^l

$$\frac{\partial l}{\partial p^k} = \sum_{i=1}^{n^N} \underbrace{\frac{\partial}{\partial x_i^N} l(y, x^N)}_{=: \Delta(x^N)} \cdot \frac{\partial x_i^N}{\partial p^k}$$

$$\frac{\partial x^N}{\partial p^k} = \frac{\partial [\sigma^N(z^N)]}{\partial p^k} = \sum_{i=1}^{n^N} \sigma^{N'}(z_i^N) \frac{\partial z_i^N}{\partial p^k}$$

$$= \sum_{i=1}^{n^N} \sigma^{N'}(z_i^N) \frac{\partial [\sum_{j=1}^{n^{N-1}} W_{ij}^N x_j^{N-1} + b_i^N]}{\partial p^k}$$

$$= \sum_{i=1}^{n^N} \sigma^{N'}(z_i^N) \left[\sum_{j=1}^{n^{N-1}} W_{ij}^N \frac{\partial x_j^{N-1}}{\partial p^k} + \sum_{j=1}^{n^{N-1}} \frac{\partial W_{ij}^N}{\partial p^k} x_j^{N-1} + \frac{\partial b_i^N}{\partial p^k} \right]$$

We note that

$$\frac{\partial x^l}{\partial p^k} = 0 \quad \text{for} \quad l \le k$$

$$\frac{\partial W^l}{\partial p^k} = 0 = \frac{\partial b^l}{\partial p^k} \quad \text{for} \quad l \ne k$$

Hence, the last expression simplifies depending on layer number k:

$$\frac{\partial x^N}{\partial p^k} = \sigma^{N'}(z^N) \odot W^N \cdot \overbrace{\frac{\partial x^{N-1}}{\partial p^k}}^{\text{if } k \leq N-1 \text{ this has to be expanded}}$$

$$+ \, \sigma^{N'}(z^N) \odot \underbrace{\left[\frac{\partial W^N}{\partial p^k} x^{N-1} + \frac{\partial b^N}{\partial p^k} \right]}_{\text{if } k \neq N \text{ this term is zero}}$$

for $k < N$:

$$\frac{\partial x^N}{\partial p^k} = \sigma^{N'}(z^N) \odot W^N \cdot \frac{\partial x^{N-1}}{\partial p^k}$$

$$= \sigma^{N'}(z^N) \odot W^N \cdot \left[\sigma^{N-1'}(z^{N-1}) \odot W^{N-1} \cdot \frac{\partial x^{N-1}}{\partial p^k} + \sigma^{N-1'}(z^{N-1}) \odot \left(\frac{\partial W^{N-1}}{\partial p^k} \cdot x^{N-2} + \frac{\partial b^{N-1}}{\partial p^k} \right) \right]$$

and, ~~hence~~, we get the following structure:

$$\frac{\partial L}{\partial p^k} = \Delta(x^N) \cdot \left[\sigma^{N'}(z^N) \odot W^N \right] \cdot \frac{\partial x^{N-1}}{\partial p^k}$$

$$= \Delta(x^N) \cdot \left[\sigma^{N'}(z^N) \odot W^N \right] \cdot \left[\sigma^{N-1'}(z^{N-1}) \odot W^{N-1} \right] \cdot [\ldots] \cdot \left[\sigma^{k+1'}(z^{k+1}) \odot W^{k+1} \right] \cdot \left[\sigma^{k'}(z^k) \odot \left(\frac{\partial W^k}{\partial p^k} \cdot x^{k-1} + \frac{\partial b^k}{\partial p^k} \right) \right]$$

An efficient update rule: Backpropagation

Algorithm: 1) Input $x^0 \leftarrow x^{(i)}$

2) Forward pass: $x^0 \mapsto z^1 \mapsto x^1 \mapsto \dots \mapsto z^N \mapsto x^N$

3) Compute $\Delta(x^N) = \ell(y^{(i)}, x^N(x^{(i)}))$

4) Compute backward pass

$$\Delta^N := \Delta(x^N)$$
$$\Delta^{k-1} := \Delta^k \cdot \left[\sigma^{k\prime}(z^k) \odot W^k \right]$$

5) Compute

$$\frac{\partial \ell_i}{\partial p^k} = \Delta^k \frac{\partial x^k}{\partial p^k}$$
$$= \Delta^k \left[\sigma^{k\prime}(z^k) \odot \left(\frac{\partial W^k}{\partial p^k} \cdot x^{k-1} + \frac{\partial b^k}{\partial p^k} \right) \right]$$

6) Compute ℓ_i for $i \in I$ (mini-batch $I \subseteq \{1 \dots n\}$)

and average

$$\frac{\partial L_I}{\partial p^k} = \frac{1}{|I|} \sum_{i \in I} \frac{\partial \ell_i}{\partial p_k}$$

7) Update weights accordingly

$$W_{ij}^k \mapsto W_{ij}^k - \eta \frac{\partial L_I}{\partial W_{ij}^k}$$

$$b_i^k \mapsto b_i^k - \eta \frac{\partial L_I}{\partial b_i^k}$$

8) repeat for all mini-batches and epochs.

$\boxed{\text{HW}}$ Derive this backpropagation algorithm from the KKT condition for the Lagrangian formulation of:

$$\text{minimize} \ \frac{1}{M}\sum_{i=1}^{M} \ell(y^{(i)}, x^N)$$

under constraints $\quad x^k = \sigma(W^k x^{k-1} + b^k)$

1) formulate Lagrangian

$$\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M} \ell(y^{(i)}, x^N) + \sum_{i=1}^{M}\sum_{k=1}^{N} \beta_i^k \cdot \left[x^k - \sigma(W^k x^{k-1} + b^k)\right]$$

2) $\dfrac{\partial \mathcal{L}}{\partial \beta_i}$ gives forward pass

3) $\dfrac{\partial \mathcal{L}}{\partial x_i}$ gives backward pass

4) $\dfrac{\partial \mathcal{L}}{\partial W^k}, \dfrac{\partial \mathcal{L}}{\partial b^k}$ give update rule

Python implementation

- MNIST data download
- Number browser
- example:
  - randomly initialized weights
  - $\eta = 3$, $n^0 = 28 \times 28 = 784$, $n^1 = 30$, $n^2 = 10$, quadratic loss

$\Rightarrow$ efficiency $\gtrsim 94\%$

BACKPROPAGATION

Network forward pass

$$x^{(i)} \xmapsto{\ id\ } x^0 \in \mathbb{R}^{n^0}$$

$$\downarrow$$

$$z^1 = W^1 x^0 + b^1 \in \mathbb{R}^{n^1}, \text{ i.e., } W^k \in \mathbb{R}^{n^k \times n^{k-1}},\ b^k \in \mathbb{R}^k$$

$$\downarrow$$

$$x^1 = \sigma^1(z^1), \text{ i.e., } \sigma^k : \mathbb{R}^k \longrightarrow \mathbb{R}^k$$

$$(z_i)_i \longmapsto \left(\tilde{\sigma}^k(z_i)\right)_i$$

where $\tilde{\sigma}^k : \mathbb{R} \to \mathbb{R}$

$$\downarrow$$

$$\vdots$$

$$x^{k-1} = \sigma^{k-1}(z^{k-1})$$

$$\downarrow$$

$$\text{k-th layer} \begin{cases} z^k = W^k z^{k-1} + b^k \\ \downarrow \\ x^k = \sigma^k(z^k) \end{cases}$$

$$\downarrow$$

$$\vdots$$

$$\downarrow$$

$$h(x^{(i)}) \xleftarrow{\ id\ } x^N = \sigma(z^N) \xleftrightarrow{\text{compare}} v(y^{(i)}), \text{ where}$$

$$v : \{0, 1, \dots, n^N\} \longrightarrow \mathbb{R}^{n^N}$$

$$i \longmapsto \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \text{- i-th}$$

Loss function

$$L = \frac{1}{M} \sum_{i=1}^{M} \cancel{\ell}\ell(v(y^{(i)}), \cancel{h}h(x^{(i)}))$$

where $\ell(v, x)$ is a "distance" or a function of it, e.g.:

$$\ell(v, x) = \frac{1}{2}(v - x)^2$$

Optimization $\quad 0 = \frac{\partial L}{\partial p^k}$ where p^k stands for $\begin{pmatrix} W^k_{ij} \\ b^k_i \end{pmatrix}$

$$\frac{\partial L}{\partial p^k} = \frac{1}{M} \sum_i \frac{\partial \ell^{(i)}}{\partial p^k}$$

$$\frac{\partial \ell}{\partial p^k} = \sum_i \ell'_i(v(y^{(i)}), h(x^{(i)})) \frac{\partial h_i}{\partial p^k} = \overbrace{\ell'(v(y^{(i)}), x^N)}^{\Delta(x^N)} \cdot \frac{\partial x^N}{\partial p^k}$$

$$\frac{\partial x^N}{\partial p^k} = \sum_i \sigma^{N'}(z^N_i) \frac{\partial z^N_i}{\partial p^k}$$

$$= \sum_i \sigma^{N'}(z^N_i) \left[\sum_j \frac{\partial W^N_{ij}}{\partial p^k} x^{N-1}_j + \sum_j W^N_{ij} \frac{\partial x^{N-1}}{\partial p^k} + \frac{\partial b^{N-1}}{\partial p^k} \right.$$

$$= \sigma^{N'}(z^N) \odot W^N \cdot \frac{\partial x^{N-1}}{\partial p^k}$$

$$+ \sigma^{N'}(z^N) \odot \cdot \left[\frac{\partial W^N}{\partial p^k} \cdot x^{N-1} + \frac{\partial b^{N-1}}{\partial p^k} \right]$$

$$\frac{\partial x^\ell}{\partial p^k} = 0 \quad \text{for} \quad \ell \leq k$$

$$\frac{\partial W^\ell}{\partial p^k} = 0 \quad \text{for} \quad \ell \neq k$$

$\frac{\partial b^\ell}{\partial p^k}$ too

Hence, we get the following structure by induction

$$\frac{\partial \ell}{\partial p^k} = \Delta(x^N) \cdot \left[\sigma^{N\prime}(z^N) \odot W^N \right] \cdot \frac{\partial x^{N-1}}{\partial p^k}$$

$$= \Delta(x^N) \cdot \left[\sigma^{N\prime}(z^N) \odot W^N \right]$$

$$\cdot \left[\sigma^{N-1\prime}(z^{N-1}) \odot W^{N-1} \right]$$

$$\cdot \;\; \cdots$$

$$\cdot \left[\sigma^{k+1\prime}(z^{k+1}) \odot W^{k+1} \right]$$

$$\cdot \left[\sigma^{k\prime}(z^k) \odot \left(\frac{dW^k}{dp^k} x^{k-1} + \frac{db^k}{dp^k} \right) \right]$$

Backpropagation algorithm

(1) Compute forward pass

$$x^{(i)} = x^0 \mapsto z^1 = W^1 x^0 + b^1 \mapsto x^1 = \sigma^1(z^1)$$
$$\mapsto \cdots \mapsto x^{k-1} \mapsto z^k = W^k x^{k-1} + b^k$$
$$\mapsto x^k = \sigma^k(z^k) \mapsto \cdots \mapsto x^N = \sigma(z^N)$$

(2) Compute $\Delta(x^{(i)}) = \ell'(v(y^{(i)}), h(x^{(i)}))$

(3) Compute backward pass

$$\Delta^N := \Delta(x^{(i)})$$

$$\Delta^{k-1} := \Delta^k \cdot \left[\sigma^{k\prime}(z^k) \odot W^k \right]$$

and with that

$$\frac{dl}{dp^k} = \Delta^{k} \cdot \frac{\partial x^k}{\partial p^k}$$

$$= \Delta^k \cdot \left[\sigma^{k\prime}(z^k) \odot \left(\frac{\partial W^k}{\partial p^k} \cdot x^{k-1} + \frac{\partial b^k}{\partial p^k} \right) \right]$$

LEARNING BEHAVIOR

① Consider one neuron with quadratic loss

$$\mathbb{R} \ni x^0 \circ \longrightarrow \bigcirc \longrightarrow \circ \; x^1 = \sigma(z^1) = \sigma(W^1 x^0 + b^1) \in \mathbb{R}$$

dropping the indices

$$h(x) = \sigma(Wx+b) \quad \text{for} \quad W, b \in \mathbb{R}$$

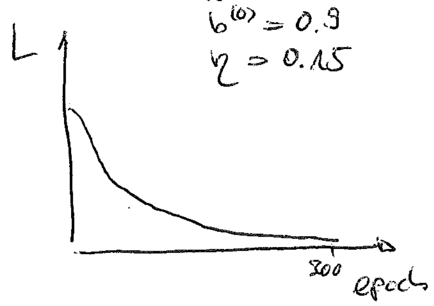
Cost function for one training data point $(x^{(1)}, y^{(1)}) = (1, 0)$

$$L(W,b) = \frac{1}{2}(0 - h(1))^2 = \frac{1}{2} h(1)^2$$

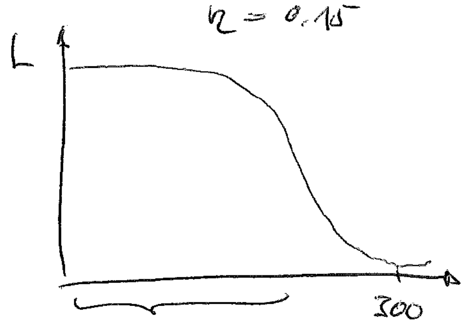
$$\frac{\partial L}{\partial W} = h(1) \frac{\partial}{\partial W} \sigma(W\cancel{x}+b) = h(1)\, \sigma'(W+b) \cdot 1$$

$$\frac{\partial L}{\partial W} = h(1)\, \sigma'(W+b) \cdot 1$$

$W^{(0)} = 0.6$
$b^{(0)} = 0.9$
$\eta = 0.15$



$W^{(0)} = 2.0$
$b^{(0)} = 2.0$
$\eta = 0.15$



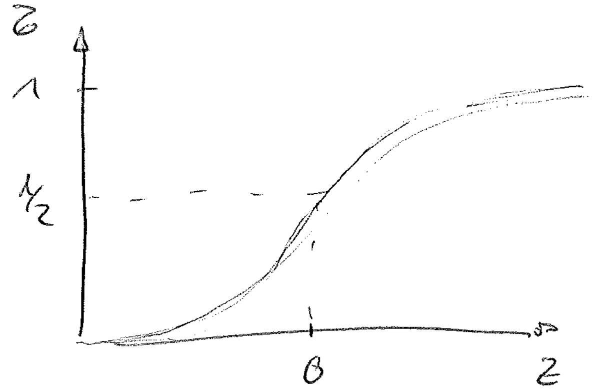
even though we are predicting "very" wrongly we only take little notice and only adapt the learning in the last 100 epochs !

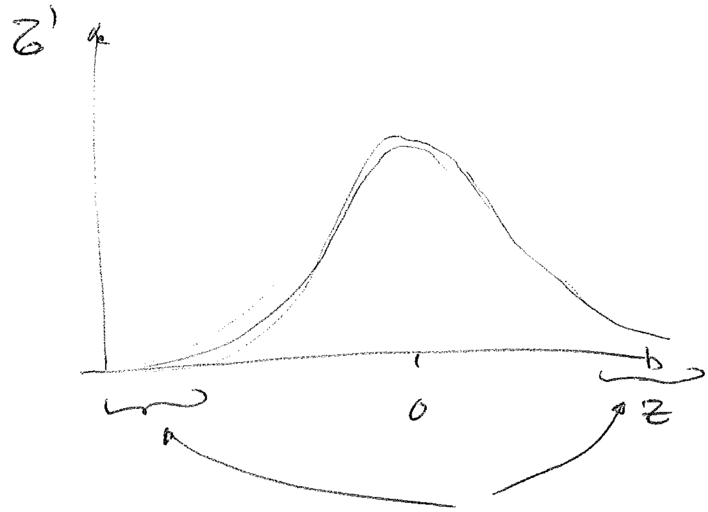
updates per epoch

$$W \mapsto W - \eta \frac{\partial L}{\partial W}$$
$$b \mapsto b - \eta \frac{\partial L}{\partial W}$$

→ Show plots for weights, loss, output

Recall the sigmoid





$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\Rightarrow \quad \sigma'(z) = \sigma(z)(1-\sigma(z))$$

region where the learning is really slow because $\sigma'(z)$ is very small

<u>Conclusion</u>: In order to avoid this slow learning curves we have to change either 1) $\sigma(z)$ or 2) L

$\sigma(z)$ is however desireable
So let's try to find a better L.

ENTROPIE

IV

$$H(P) = -\sum_{\omega \in \Omega} p(\omega) \underbrace{\log_2 p(\omega)}$$

je überraschender
das Ereignis umso
weniger Information
→ ihrer Anzahl unabhängiger Bits

Bsp.: • Wurf einer Münze $P_{fair}(Kopf) = \frac{1}{2} = P_{fair}(Zahl)$

$$H(P_{fair}) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right)$$

$$= 1$$

• Wurf eines Würfels $P_{fair}(Augen = a) = \frac{1}{6} \quad \forall a \in \{1..6\}$

$$H(P_{fair}) = -\log\frac{1}{6} = 2{,}58\ldots$$

Bem: 8-seitige Würfel entsprechen $H(P_{fair}) = 3$

So ist $H(P)$ ein Maß für die durchschnittliche Überraschung, oder die Anzahl der Bits, die im Durchschnitt benötigt werden.

Bsp.: • Unfairer Würfel

ω	1	2	3	4	5	6
P	1/4	1/16	1/16	1/16	1/16	1/2

$H(P) = 2$ Die Augenzahl 6 kommt im Durchschnitt natürlich viel häufiger vor, deshalb brauchen wir im ∅ weniger Bits.

Entropie ist also ein Maß für die Gleichmäßigkeit einer Wahrscheinlichkeitsverteilung P.

DEF: Für zwei W-Räume (Ω, P) und (Ω, Q) im diskreten Fall def. wir die Kreuzentropie

$$H[P,Q] = -\sum_{\omega \in \Omega} P(\omega) \log_2 Q(\omega)$$

Interpretation: Q ist eine Schätzung von P. Der Informationsgehalt ~~von~~ $\omega \in \Omega$ ist nach unserer Schätzung $-\log_2 Q(\omega)$ dieser tritt aber $P(\omega)$ verteilt auf, also ist $H[P,Q]$ der erwartete Informationsgehalt nach Q.

THM: $H[P,Q] \geq H(P,P) = H(P)$

$\cancel{V}$ $\ell(x) = -y \ln x - (1-y) \ln(1-x)$
$\ell'(x) = -\frac{y}{x} + \frac{1-y}{1-x} = 0 \Rightarrow x = y$
$\ell''(x) = \frac{y}{x^2} + \frac{1-y}{(1-x)^2} > 0$

Interpretation von Neuron

Output $h(x) = \sigma(\omega x + b) \in [0,1]$

bei input $x^{(i)}$ wird output $y^{(i)} \in \{0,1\}$ erwartet, also ist $P_i(x^{(i)}) = y^{(i)}$, $P_i(\text{not } x^{(i)}) = 1-y^{(i)}$
$Q_i(x^{(i)}) = h(x^{(i)})$ und $Q_i(\text{not } x^{(i)}) = 1-h(x^{(i)})$

damit

$$H[P,Q] = -\frac{1}{M} \sum_{i=1}^{M} y^{(i)} \log_2 h(x^{(i)}) + (1-y^{(i)}) \log_2 (1-h(x^{(i)}))$$

② One neuron with cross-entropy

$\boxed{IV}$

$$L(w,b) = -y \ln h(x) - (1-y)\ln(1-h(x)), \quad z = \sigma(wx+b)$$

(for training data $x=1$, $y=0$)

$$\frac{\partial L}{\partial w} = -\left[\frac{y}{h(x)} - \frac{1-y}{1-h(x)}\right]\frac{\partial h(x)}{\partial w} = -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\sigma'(z)\cdot x$$

$$\frac{\partial L}{\partial b} = \text{—''—}$$

but $\sigma'(z) = \sigma(z)(1-\sigma(z))$

$$\Rightarrow \quad \frac{\partial L}{\partial w} = -\left[y(1-\sigma(z)) - (1-y)\sigma(z)\right]\cdot x$$

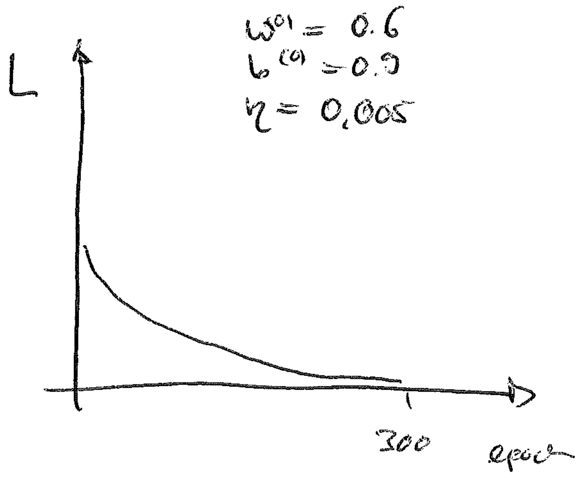
$$= \left[\sigma(z) - y\right]x$$

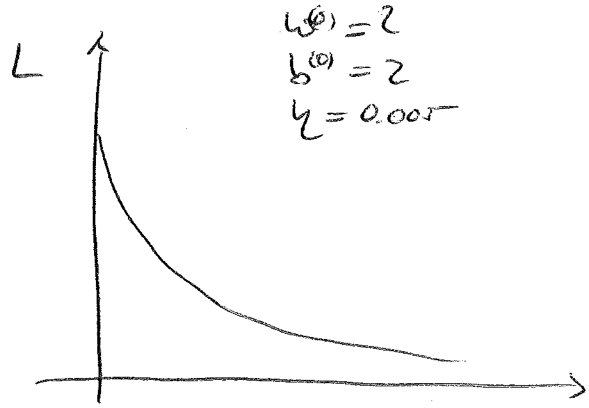
and hence for $x=1$, $y=0$

$$\frac{\partial L}{\partial w} = \sigma(z)$$

$$\frac{\partial L}{\partial b} = \sigma(z)$$

Hence, the neuron learns the weights proportional to the error made.





For the general case of $x^{(i)} \in \mathbb{R}^n$, $y^{(i)} \in \mathbb{R}^c$
we may define

$$L(W^{(1)} \dots W^{(N)}, b^{(1)} \dots b^{N})$$
$$= -\frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{c} \left(y_j^{(i)} \ln h_j(x^{(i)}) + (1-y_j^{(i)}) \ln h_j(x^{(i)}) \right)$$

Rem: • Cross-entropy nearly always a better choice for sigmoid neurons

• however for $\sigma(z) = z$, i.e., in the linear neuron case the quadratic loss performs equally well

$$L(w,b) = \frac{1}{2}(wx+b-y)^2$$
$$\frac{\partial L}{\partial w} = (wx+b-y)x \overset{x=1,\,y=0}{=} w+b$$
$$\frac{\partial L}{\partial b} = (wx+b-y) = w+b$$

General case for cross entropy is:

$$\frac{\partial L}{\partial p^k} = \underbrace{\Delta(x^N)}\cdot[\sigma^{N'}(z^N) \odot W^N] \cdot \dots \cdot [\sigma^{k'}(z^k) \odot (\frac{\partial W^k}{\partial p_k} \cdot x^k + \frac{\partial b^k}{\partial p_k})]$$

$$\ell_i'(x) = \cancel{\ }- \left[\frac{y_i}{\sigma(z_i^N)} + \frac{1-y_i}{1-\sigma(z_i^N)}\right] \overbrace{\sigma'(z_i^N)}^{= \sigma(z_i^N)(1-\sigma(z_i^N))}$$
$$= (\sigma(z_i^N) - y_i) = \cancel{\ } (x^N - y_i)$$

Back to MNIST

30 hidden neurons, 10 sized mini-batches,

$\eta = 0.5$, epochs 30

gives results close to quadratic loss $\sim 95.42\%$

100 hidden neurons and everything else the same $\sim 96.59\%$

REGULARIZATION

$$L' = L + \underbrace{\frac{\lambda}{N} \sum_{\ell} \frac{1}{2} \| W^{\ell} \|_2^2}_{R(W^1 \dots W^N, \overbrace{b^1 \dots b^N}^{\text{for all generality}})}$$

$$\frac{\partial R}{\partial W^k} = \frac{\lambda}{N} W^k$$

new update rule:

$$W^k \mapsto W^k - \eta \frac{\partial L'}{\partial W^k} = \left(1 - \eta \frac{\lambda}{N}\right) W^k - \eta \frac{\partial L}{\partial W^k}$$

Example: $\lambda = 5$, same parameters as for 100 hidden neurons and cross-entropy

for $\eta = 0.1$ we break the 98% barrier

Other methods of regularization:

1) L_1 regularization $\quad R = \frac{\lambda}{N} \sum_{l=1}^{N} \| W^l \|_1$

$$w \mapsto w - \eta \frac{\lambda}{N} \operatorname{sgn}(w) - \eta \frac{\partial L}{\partial w}$$

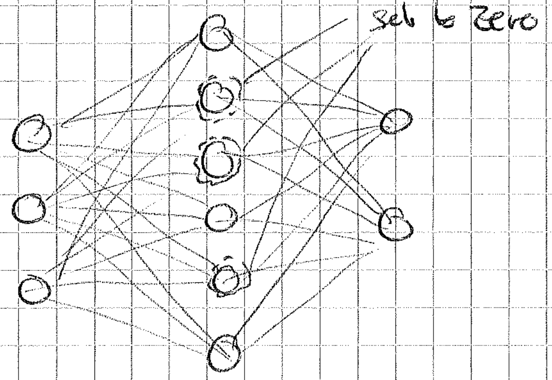
except for $w=0$

which shrinks the weights by a constant towards zero;

opposed to L2 which is proportional to w itself.

$\Rightarrow$ L_1 tends to concentrate on high number of high-importance connections

2) Dropout (2012)



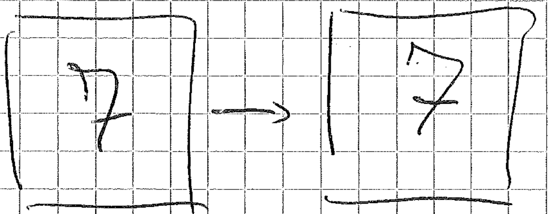
1) choose half of the hidden neurons and switch them off; leave weights
2) do this process again for other randomly chosen neurons deleted
3) average the weights of both networks

Idea: • superposition of networks, averaging vote
• minimizes co-adaptation

VII

(2012) Dropout + L_2 regularization gave 98.7%

3) artificially expanding the training set



- rotate a bit
- translate ..
- skew .. 98.9%
- elastic distortions 99.3%

Other improvements

Weight initialization

Ex: Say we have 1000 neurons in a layer

Say for 500 i's, $W_{ij} = 0$

$$z_i = \underbrace{\sum_j W_{ij} x_j + b_i}$$

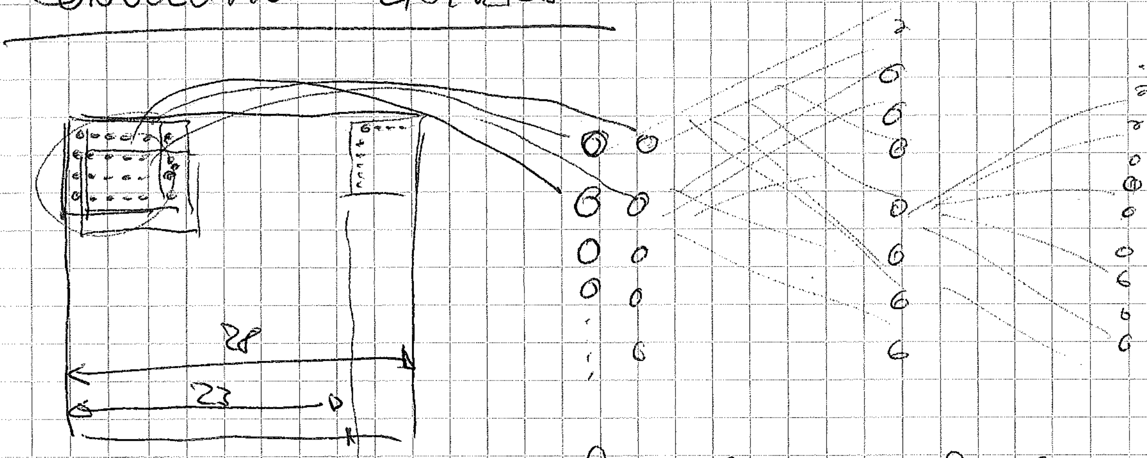
$\sim$ random variable gaussian with mean zero and $\sigma \sim \sqrt{501} \approx 22.4$

$\Rightarrow$ sigmoids will be saturated

Solution: Initialize weights by

$$\mathbb{R}^{n^{k-1}, n^k} \ni W_{ij}^k \sim \mathcal{N}\left(0, \frac{1}{\sqrt{n^k}}\right)$$

Convolution Layers



hidden layer dense layer

input layer

X^0 X^1

convolution layer

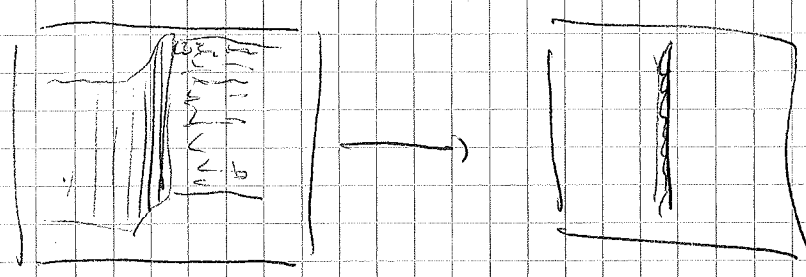
$$\left(x^0_{ij}\right)_{ij} \longmapsto \left(z^1_{ij}\right)_{ij} \quad \text{for} \quad z^1_{ij} = \sum_{k=0}^{4} \sum_{l=0}^{4} w_{kl}\, x^0_{i+k,j+l} + b$$

$$\downarrow$$

$$\left(\sigma\left(z^1_{ij}\right)\right)_{ij}$$

What does a convolution layer do?

$$w = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$ Laplace filter



TUNING

1 LEARNING BEHAVIOR

Let us consider one neuron with quadratic loss:

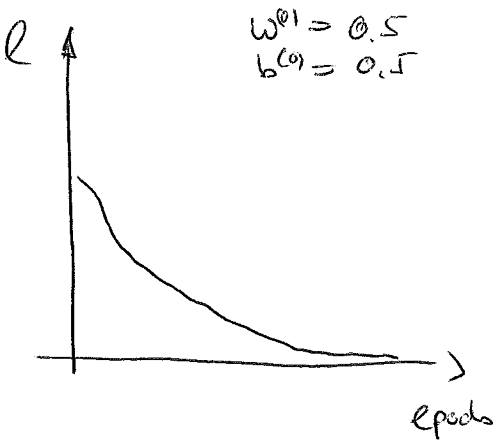
$$\mathbb{R} \ni x^0 \mapsto z^1 := \underset{\mathbb{R}}{\underset{\cap}{w}} x^0 + \underset{\mathbb{R}}{\underset{\cap}{b}} \mapsto x^1 = \sigma(z^1) \in \mathbb{R}$$

loss $\quad \ell(y,x) = \frac{1}{2}\left(y - x^1(x)\right)^2$

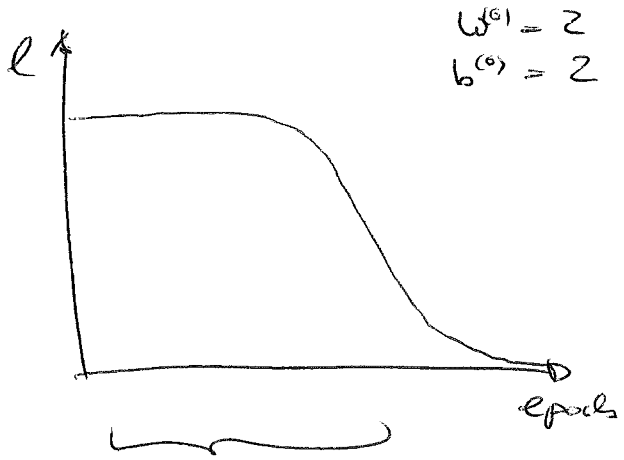
$$\Rightarrow \quad \frac{\partial \ell}{\partial w} = [y - x^1(x)](-1)\,\sigma'(z^1)\,x^0$$

$$\frac{\partial \ell}{\partial b} = [y - x^1(x)](-1)\,\sigma'(z^1)$$

let's use that to learn $x^0 = 1 \mapsto x^1 = 0$



Same η



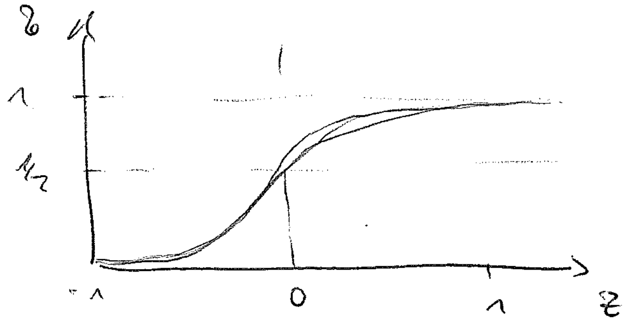
steady learning

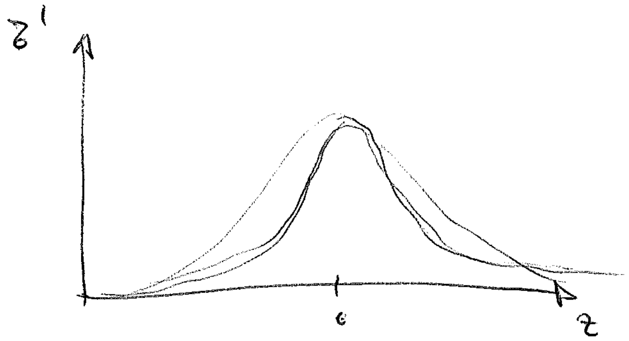
long phase where not much is happening

using $\quad w \mapsto w - \eta \frac{\partial \ell}{\partial w}$

$\quad b \mapsto b - \eta \frac{\partial \ell}{\partial w}$

Recall the sigmoid function $\quad \sigma(z) = \frac{1}{1+e^{-z}}, \quad \sigma'(z) = \sigma(z)(1-\sigma(z))$





Conclusion: slow learning due to saturation of the neuron.

Hence, either we change σ or l!

σ is, however, desirable, therefore, let's change l.

New loss function chosen that σ' is compensated:

$$l(y, x^1) = -y \ln x^1 - (1-y) \ln (1-x^1)$$

is called the cross entropy, for which we find

$$\frac{\partial l}{\partial w} = -\left[\frac{y}{x^1} - \frac{1-y}{1-x^1}\right] \frac{\partial x^1}{\partial w}$$

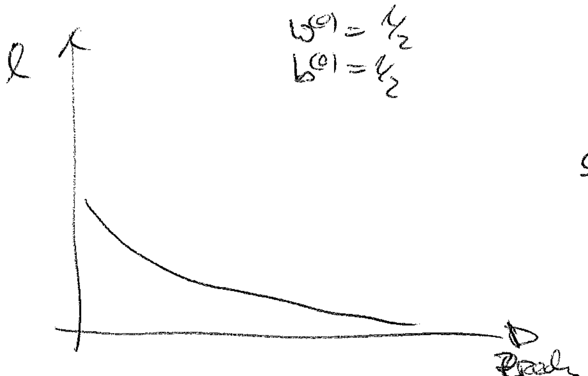
$$= -\left[\frac{y}{\sigma(z^1)} - \frac{1-y}{1-\sigma(z^1)}\right] \sigma'(z^1)\, x^0$$

$$\frac{\partial l}{\partial b} = -\left[\ \text{——} \ \text{"} \ \text{——}\ \right] \sigma'(z^1)$$

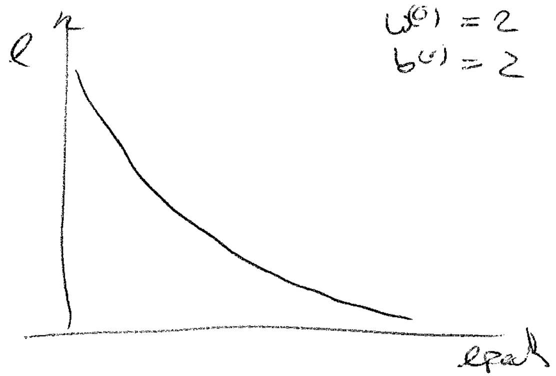
but since $\sigma'(z) = \sigma(z)(1-\sigma(z))$ we find

$$\frac{\partial l}{\partial w} = -\left[y - x^1\right] x^0$$

$$\frac{\partial l}{\partial b} = -\left[y - x^1\right]$$



same η



Where does the entropy come from?

Let $(P, \Omega, \mathcal{P}(\Omega))$ be a probability space for a discrete Ω.

DEF: (Entropy) $H(P) = - \sum_{\omega \in \Omega} p(\omega) \underbrace{\log_2 p(\omega)}_{\text{\# of independent bits needed to store the results}}$

On average with respect to measure P

Example:
- Tossing of a coin $P(\text{head}) = \frac{1}{2} = P(\text{tails})$

$$H(P) = -\left[\frac{1}{2}\log_2 \frac{1}{2} + \frac{1}{2}\log_2 \frac{1}{2}\right] = 1$$

- Casting a dice $P(\text{shows } a) = \frac{1}{6} \quad \forall a = 1 \dots 6$

$$H(P) = -\log_2 \frac{1}{6} \approx 2.58$$

- Unfair dice

ω	1	2	3	4	5	6
P_{unfair}	1/4	1/16	1/16	1/16	1/16	1/2

$$H(P_{\text{unfair}}) = 2$$

The number 6 shows up more often, hence on average we need less bits to store the outcome

$\Rightarrow$ Entropy is a measure of the uniformity of the probabilities

<u>DEF</u>: Let $(\Omega, P, \mathcal{P}(\Omega))$ and $(\Omega, Q, \mathcal{P}(\Omega))$ be two probability spaces for a discrete Ω we define the cross-entropy

$$H[P, Q] = -\sum_{\omega \in \Omega} P(\omega) \log_2 Q(\omega)$$

<u>Interpretation</u>: If Q is an estimate of the measure P then $-\log_2 Q(\omega)$ is the # of bits needed to store the information (according to the estimate). $H[P, Q]$ is the average w.r.t. P of this estimates.

<u>THM</u>: $H[P, Q] \geq H[P, P] = H[P, \cancel{P}]$

Back to our σ-neurons:

We may interpret the input $x^{(i)}$ as probability event and the output $y^{(i)}$ as probability. The prediction of our neuron $X^1(x^{(i)})$ is the estimate to the real probability. The resulting cross-entropy is

$$\ell(y^{(i)}, X^1(x^{(i)})) = -y^{(i)} \log_2 X^1(x^{(i)}) - (1 - y^{(i)}) \log_2 (1 - X^1(x^{(i)}))$$

$\boxed{HW}$ 1) Does it play a role whether we take $\log_2$ or $\ln$?
2) Prove $\ell(y, x)$ has min at x and is convex.

⑥

Hence, for the general case we have

$$L = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{n^N} (-1) \left\{ y_j^{(i)} \ln \left(\cancel{\max} X_j^N(x^{(i)}) \right) + (1-y_j^{(i)}) \ln \left(1 - X_j^N(x^{(i)}) \right) \right\}$$

HW: Is the cross entropy always better concerning the slow learning? Consider for example the last layer to be linear-neurons.

② Initializing the weights and biases

- It makes sense to initialize the weights at random
- consider the output $z_i^k = \sum_j (W_{ij}^k x_j^{k-1}) + b_i^k$, with $\underbrace{\sum_j}_{n^{k-1} \text{ summands}}$

 If these where i.i.d. with Variance $= 1$ we would get Variance $= n_{k-1}$ $\Longrightarrow$ Z-neuron saturates easily

- hence it makes sense to normalize the initial random weights

$$W_{ij}^k \sim \mathcal{N}(0, 1/\sqrt{n^{k-1}})$$

HW: Compare the cases:

- all weights and biases $\sim \mathcal{N}(0,1)$
- ——— " ——— $= 0$

③ Regularization

- In order to prevent that some weight become very dominant while others are seldomly used we may change our loss function

$$L' := L + \frac{\lambda}{N}\sum_{\ell=1}^{N}\frac{1}{2}\|W^\ell\|_2^2$$

which changes the update rule according to

$$W^k \mapsto W^k - \eta\frac{\partial L'}{\partial W} = \left(1-\eta\frac{\lambda}{N}\right)W^k - \eta\frac{\partial L}{\partial W^k}$$

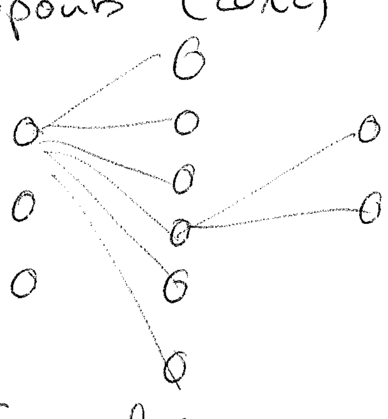
<u>Python implementation</u>: $\lambda = 5$, $\eta = 0.1$, cross-entropy, L_2 regularization breaks 98%

- $\boxed{\text{HW}}$ Consider $\|W^\ell\|_1$ instead of L_2 norm

$$W^k \mapsto W^k\left(1-\eta\frac{\lambda}{N}\,\mathrm{sign}(W^k)\right) - \eta\frac{\partial L}{\partial W^k}$$

What is the main difference?

- Dropout (2012)



hidden layer

- artificially expanding the training data

1) choose half neurons of hidden layer and delete them
2) learn weights
3) choose other half, delete them learn weights
4) average over the two networks

→ superposition, minimization of co-adaption

APPROXIMATION BY NEURAL NETWORKS $\boxed{I}$

Representation of boolean functions

$\overset{\vee}{=}$ DEF: neural net, vocabulary, feed forward

The goal is to take an arbitrary boolean function
$$f : \{0,1\}^d \longrightarrow \{0,1\}$$

and find a neural network, i.e., activation function σ^i, weights W^i and biases b^i for $i = 1..N$ layers s.t.

$$\begin{aligned} f(x) = x^N(x) &= \sigma^N(W^N x^{N-1}(x) + b^N) \\ &= \sigma^N(W^N \sigma^{N-1}(W^{N-2} x^{N-3}(x) + b^{N-2}) + \dots) \\ &\dots \\ &= \sigma^N(\dots\dots\dots (\sigma(W^1 x + b^1)))) + \dots) \end{aligned}$$

THM: Every boolean function f can be exactly represented by a neural network with $N \geq 2$ and at most 2^d neurons in the hidden layer and $\sigma(z) = \begin{cases} 1 & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases}$.

Proof: Let us denote $f^{-1}(\{1\}) = \{a^1, a^2, \dots, a^m\}$

Then we find a representation
$$f(x) = \sigma\left(-1 + \sum_{i=1}^{m} \delta_{x,a^i}\right).$$

Next, we find a representation for $\delta_{x,a} = \begin{cases} 1 & \text{for } x = a \\ 0 & \text{for } x \neq a \end{cases}$ for $x, a \in \{0,1\}^d$.

For this we consider for $d=1$

a	b	$2ab-a-b$
0	0	0
0	1	-1
1	0	-1
1	1	0

Hence, $\delta_{x,a} = \sigma(2xa - x - a)$ and for $d \geq 1$ we get

$$\delta_{x,a} = \sigma\Big(\sum_{j=1}^{d} 2x_j a_j - x_j - a_j\Big)$$

which allows to recast f into

$$f(x) = \sigma\Big(-1 + \sum_{i=1}^{m} \sigma\Big(\sum_{j=1}^{d} 2x_j a_j^i - x_j - a_j^i\Big)\Big)$$

Here we can read off the weights and biases

- $y_i^1 = W^1 x + b^1 \overset{!}{=} \sum_{j=1}^{d} 2x_j a_j^i - x_j - a_j^i$

$$\Rightarrow \quad W_{ij}^1 = 2a_j^i - 1 \qquad 1 \leq i \leq m,\ 1 \leq j \leq d$$

$$b_i^1 = -\sum_{j=1}^{d} a_j^i \qquad 1 \leq i \leq m$$

$$x_i^1 = \sigma(y_i^1)$$

- $y^2 = W^2 x^1 + b^2 \overset{!}{=} \sum_{i=1}^{m} x_i^1 - 1$

$$\Rightarrow \quad W_i^2 = 1$$

$$b^2 = -1$$

and finally

$$x^2 = \delta(y^2)$$

Hence, we need weights and biases

$$W^1 \in \mathbb{R}^{m \times d}, \quad b^1 \in \mathbb{R}^m$$

$$W^2 \in \mathbb{R}^m, \quad b^2 \in \mathbb{R}, \quad \text{and} \quad N = 2$$

where d is the number of input neurons
m is the number of hidden neurons

and for m we get the estimate

$$0 \leq m \leq 2^d$$

for the worst case where the function f fulfills $f(x) = 1 \ \forall x$. $\square$

$\boxed{\text{HW:}}$ Implement XOR gate by hand.

REM:
- This construction implies an exponential increase in hidden layer neurons.
- Any network with $N > 2$ is of course also able to represent the binary function, too.
- Bound can be sharpened to

$$0 \leq m \leq \min\left(|f^{-1}(\{0\})|, |f^{-1}(\{1\})|\right)$$

Representation of binary classification in $\mathbb{R}^d$

The goal is to take an arbitrary ~~test~~-data set ~~s~~: $(x^{(i)}, y^{(i)})_{1\le i\le M}$ and find a function $f: \mathbb{R}^d \to \{-1, +1\}$ s.t.

$$f(x^{(i)}) = y^{(i)} \quad \forall\ 1 \le i \le M \qquad (*)$$

that can be represented by a neural network

<u>THM</u>: Given test data $(x^{(i)}, y^{(i)})_{1\le i\le M}$ there is an $f: \mathbb{R}^d \to \{-1, 1\}$ that fulfills $(*)$ and can be represented by a neural network with $N=2$ and not more than $2M$ hidden neurons and $\sigma(z) = \operatorname{sign}(z)$.

<u>Proof</u>: Def. $\{a_1, \dots, a_m\} = \{x^{(i)} \mid y^{(i)} = 1,\ 1 \le i \le M\}$

Since this set is finite, for each $1 \le i \le m$ we find $W_i, b_i \in \mathbb{R}^d$ s.t.

$$h_i : z \mapsto W_i z + b_i \quad \text{with} \quad \operatorname{range} h_i \cap \{a_1 \dots a_m\} = a_i$$

furthermore, also due to finiteness, there is an $\varepsilon_i > 0$ (e.g. half of min distance to nearest a_j, $j \ne i$) s.t.

$$I_i : z \mapsto \sigma(\varepsilon_i + (W_i z + b_i)) + \sigma(\varepsilon_i - (W_i z + b_i))$$

fulfills $\quad I_i(a_i) = 2$

$\quad I_i(a_j) = 0 \quad$ for $j \neq i$

With this we construct

$$f(z) = \sigma\left(-1 + \sum_{i=1}^{m} I_i(z)\right)$$

which by construction fulfills

$$f(a_i) = 1 \quad , \quad f(z) = 0 \quad \forall \; z \neq a_i, \; 1 \leq i \leq n$$

$$\Rightarrow \quad f(x^{(i)}) = y^{(i)}$$

From f we read off the appropriate weights and biases of the corresponding neural network with $N = 2$

$$\mathbb{R}^d \ni x^0 = z$$

$$\downarrow$$

$$\mathbb{R}^m \ni y^1 = \left(y^0_i\right)_{1 \leq i \leq 2m} \quad \text{with}$$

for $1 \leq l \leq m$

$$y_{2l-1} = \varepsilon_l + (W_l x^0 + b_l)$$

$$y_{2l} = \varepsilon_l - (W_l x^0 + b_l)$$

$$= W^1 x^0 + b^1 \quad \text{for} \quad W^1_{2l-1,j} = W_{lj}, \quad b_{2l-1} = b_l + \varepsilon_l$$

$$-W^1_{2l,j} \qquad b_{2l} = -b_l + \varepsilon_l$$

$$\downarrow$$

$$x^1 = \sigma(y^1)$$

$$\downarrow$$

$$y^2 = W^2 x^1 + b^2 \quad \text{for} \quad W^2_i = 1, \; b^2 = -1$$

$$\downarrow$$

$$x^2 = \sigma(y^2)$$

Clearly the number of hidden neurons is bounded by

$$0 \leq 2m \leq 2M.$$

$\square$

REM: • The bound on the number of hidden neurons can easily be improved if test data is distributed in general position, e.g.,

Approximation of real-valued functions

Goal is to show that a feedforward neural network can approximate any function $f \in \mathcal{C}^0([a,b])$.

The set of hypothesis of a $N=2$ feed forward neural network with m hidden neurons, $\sigma^1(z) = \begin{cases} 1 & \text{for } z \to +\infty \\ 0 & \text{for } z \to -\infty \end{cases}$ and bounded and $\sigma^2(z) = z$ is given by

$$\mathcal{H}_m^\sigma = \Big\{ g: \mathbb{R} \to \mathbb{R} \;\Big|\; g(x) = \sum_{i=1}^{m} W_i^2 \, \sigma(W_i^1 x + b_i^1)$$

$$\text{for } W^1 \in \mathbb{R}^{m \times 1},\, W^2 \in \mathbb{R}^{1 \times m} \text{ and } b^1 \in \mathbb{R}^m \Big\}$$

THM: Let $\sigma: \mathbb{R} \to \mathbb{R}$ bounded s.t. $\lim_{z \to \infty} \sigma(z) = 1$ and $\lim_{z \to -\infty} \sigma(z) = 0$. Then $\exists\, C(\sigma) < \infty$ s.t. $\forall m \in \mathbb{N}$: $\inf_{g_m \in \mathcal{H}_m^\sigma} \| f - g_m \|_\infty \leq C(\sigma) \sup_{|x-y| < \frac{1}{m}} |f(x) - f(y)|$.

REM:
- $C(\sigma) = \|\sigma\|_\infty$
- Existence of limits and $\lim_{z \to +\infty} \sigma(z) \neq \lim_{z \to -\infty} \sigma(z)$ suffices
- f Lipschitz $\Rightarrow \sup_{|x-y| \leq \frac{1}{m}} |f(x) - f(y)| \leq \frac{L}{m}$

IV

Proof: It suffices to construct a tester

$$g_m \in \mathcal{H}_m$$

to estimate the infimum

$$\inf_{g \in \mathcal{H}_m} \|f - g\|_\infty \leq \|f - g_m\|_\infty$$

WLOG $f \in \mathcal{C}^0([0,1])$, i.e., $a = 0$, $b = 1$

def. $x_i = \frac{i}{m}$ $\quad i = 0, \ldots, m+1$

and $f_m(x) = f(x_i)$ for $x \in [x_i, x_{i+1})$

which can be written as

$$f_m(x) = f(x_0) + \sum_{i=1}^{m} [f(x_i) - f(x_{i-1})] \mathbb{1}_{i \leq mx}$$

which we need to estimate with an element $g_m \in \mathcal{H}_m$, which for $\alpha > 0$ we take

$$g_m(x) = f(x_0)\, \sigma(\alpha) + \sum_{i=1}^{m} [f(x_i) - f(x_{i-1})]\, \sigma(\alpha(mx - i))$$

$\cap$

$\mathcal{H}_m$

(!!!)

Rem: • σ may not be chosen polynomial simply because $(\mathcal{H}^{\sigma}_{m})_m$ would still be only polynomials of the order and therefore cannot be dense in $\mathcal{C}^0([a,b])$
• also increasing the number of hidden layers does not help
• on the contrary $(\mathcal{H}^{\sigma}_{m})_m$ is dense in $\mathcal{C}^0([a,b])$ iff σ is not a polynomial — we however proved this for a certain subset of non-polynomial σ only.

THM: For σ as before, $\bigcup_m \mathcal{H}^{\sigma}_m$ is dense in $\mathcal{C}^0(\mathbb{R}^n, \mathbb{R}^c)$

Proof: WLOG we may assume $c=1$ because if we can approximate each $f_i : \mathbb{R}^n \to \mathbb{R}$ of $f = (f_i)_{i=1..c}$ to accuracy $\|f_i - \hat{f}_i\|_\infty \leq \frac{\varepsilon}{c}$ with m neurons, we can approximate f with $\|f - \hat{f}\|_\infty < \varepsilon$ with $m \cdot c$ neurons.

In order to use our results for $\mathcal{C}^0([a,b], \mathbb{R})$ we use that

$$\mathcal{D} := \operatorname{span}\{ e^{w \cdot x} \mid w \in \mathbb{R}^d \}$$

is dense in $\mathcal{C}^0([a,b]^n, \mathbb{R})$ according to Stone-Weierstrass, since

i) $\mathcal{D}$ forms an algebra under multiplication and linear combination (Vector space closed under multiplication)

ii) $1 \in \mathcal{D}$ (non-zero constant function) (Non-zero constant function)

iii) $\forall x, y \in [a,b]^n \; \exists \, w = x - y$ s.t. $e^{wx} \neq e^{wy}$ (separation of points)

$$\Rightarrow \forall \varepsilon > 0 \ \exists \ m \in \mathbb{N}, \ v_1 \ldots v_m \in \mathbb{R}^n, \ c_1 \ldots c_m \in \mathbb{R}$$

s.t.

$$\sup_{x \in [a,b]^n} \left| f(x) - \sum_{i=1}^m c_i e^{v_i \cdot x} \right| < \frac{\varepsilon}{2}$$

Now we only need to approximate the exponentials, which can be done using our last theorem: $\exists \ \tilde{a} < \tilde{b}$ s.t.

$$[\tilde{a}, \tilde{b}] \supseteq \bigcup_{i=1}^m \{ v_i \cdot x \mid x \in [a,b]^n \}$$

Hence, for all $\varepsilon > 0 \ \exists \ f_\ell(y) = \sum_{j=1}^{\ell} w_j^2 \, \sigma(w_j^1 y + b_j^1)$ with

$$\underbrace{\| e^y - f_\ell \|_{\tilde{\infty}} < \frac{\varepsilon}{2m \max_{i=1..m} |c_i|}}_{\in \mathcal{T}_m^2}$$

$$\Rightarrow \sup_{x \in [a,b]^n} \left| f(x) - \sum_{i=1}^m c_i \underbrace{\sum_{j=1}^{\ell} w_j^2 \, \sigma(w_j^1 v_i x + b_j^1)}_{= f_\ell} \right|$$

$$\leq \sup_{x \in [a,b]^n} \underbrace{\left| f(x) - \sum_{i=1}^m c_i e^{v_i \cdot x} \right|}_{< \frac{\varepsilon}{2}}$$

$$+ \sup_{x \in [a,b]^n} \underbrace{\left| \sum_{i=1}^m c_i \left(e^{v_i \cdot x} - f_\ell(v_i \cdot x) \right) \right|}$$

$$\leq m \max_{i=1..m} |c_i| \underbrace{\sup_{y \in [\tilde{a}, \tilde{b}]} | e^y - f_\ell(y) |}_{< \frac{\varepsilon}{2m \max_{i=1..m} |c_i|}}$$

$$< \varepsilon \qquad \square$$

REM:
- each time step looks like a usual feed forward network with two layers
- however, information about the last step may "flow through time" by dependence on h_{t-1}
- memory certainly depends on size of hidden layer

Learning and update rule

We define a loss function for each sequence $(x_1, \bar{y}_1), \dots, (x_N, \bar{y}_N)$

$$L = \sum_{t=1}^{N} \ell(y_t(x_t), \bar{y}_t) \quad , \quad \ell(y, \bar{y}) = \frac{1}{2}(y - \bar{y})^2$$

and perform a mini-batch gradient descent update over all sequences of length N:

The free parameters are W, b, V:

$$W \in \mathbb{R}^{h \times d}, \quad b \in \mathbb{R}^{h}$$
$$V \in \mathbb{R}^{c \times h}$$

$$\begin{pmatrix} W \\ b \\ V \end{pmatrix} \longmapsto \begin{pmatrix} W \\ b \\ V \end{pmatrix} - \eta \begin{pmatrix} \frac{\partial L}{\partial W} \\ \frac{\partial L}{\partial b} \\ \frac{\partial L}{\partial V} \end{pmatrix}$$

for learning rate η.

Algorithm: For length $\ell = 1 \dots N$:

Mini-batch grad. desc. for all seq. of length ℓ.

Backpropagation through time

In order to compute the update efficiently we regard an unravelling of the recurrent network

$$z_t := W_t x_t + U_t h_{t-1} + b_t$$
$$h_t := \sigma(z_t)$$
$$y_t := V h_t$$

This is the same network except for the fact that the weights are not shared. Hence, the loss function is a function of

$$\tilde{L} = \sum_{t=1}^{N} \tilde{\ell}(y_t(W_1 \dots W_t, U_1 \dots U_t, b_1 \dots b_t, V, x_1 \dots x_t), \bar{y}_t)$$

However, the loss function of the recurrent network is given by

$$L = \tilde{L} \Big|_{W_t = W,\, U_t = U,\, b_t = b \text{ for } t = 1 \dots N}$$

This implies that

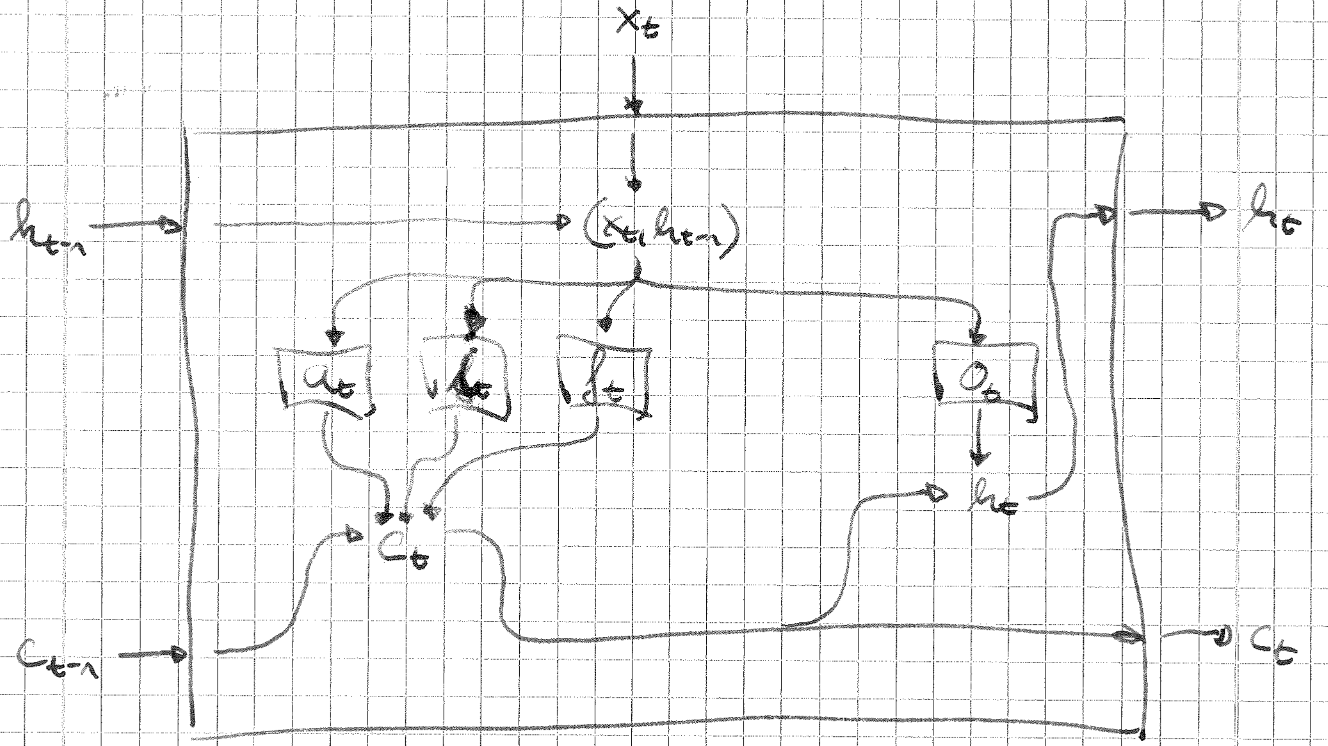
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{N} \frac{\partial \tilde{\ell}(y_t, \bar{y}_t)\big|_{W_i = W \; i = 1 \dots t}}{\partial W}$$
$$= \sum_{t=1}^{N} \sum_{k=1}^{t} \frac{\partial \tilde{\ell}(y_t, \bar{y}_t)}{\partial W_k} \Bigg|_{W_i = W \; i = 1 \dots t}$$

and we can use ordinary backpropagation for the unravelled network summing the gradients.

REM: Since the unravelled network become as deep as the length of the sequence, problems in the training become significant.

$\frac{\partial \tilde{L}}{\partial W_h}$ expands to a long product of matrices:
- region of slow learning are extremely slow
- region of fast learning explode.

LONG SHORT TERM MEMORY NETWORKS



add $a_t = \tanh(W_a x_t + U_a h_{t-1})$

input $i_t = \sigma(W_i x_t + U_i h_{t-1})$

forget $f_t = \sigma(W_f x_t + U_f h_{t-1})$

output $O_t = \sigma(W_o x_t + U_o h_{t-1})$

Control how much of the input to write to C_t, how much to forget.

Controls how much to put out from C_t

memory operation $C_t = i_t \odot a_t + f_t \odot C_{t-1}$

$h_t = O_t \odot \tanh(C_t)$