ETH zürich

Department of Mechanical and Process Engineering

Version: 30.07.2020

About this Summary:

This summary should cover most topics from the lecture and the exercises.
However, in the last days before the exam, there may still be some minor changes; so be sure to check back for new versions regularly on:
n.ethz.ch/~kiten/MAD

If you have any comments, corrections, or improvements, please contact me at:
KITEN@ETHZ.CH

Thank You!

D MAVT

---

0. Changelog and Versions

July 13, 2020

Initial Version

July 17, 2020

- Corrected **typos** in section *3.2 Cubic Splines* → Tri-Diagonal Matrix Algorithm:
  - Changed c_1 to c_2 in second row of TDMA
  - Corrected typo in b_i: Changed $b_i = \frac{\Delta_{i-1} - \Delta_i}{3}$ to $b_i = \frac{\Delta_{i-1} + \Delta_i}{3}$

  (Thanks to Lucas for spotting the errors!)
- **Changed** section *4.9 Gauss Quadrature* and **added** method of undetermined coefficients

July 20, 2020

- Corrected **error** in section *6.6 Gradient Descent* → Learning parameter η:
  - (a) and (b) were wrongly switched in previous version

  (Thanks to Irma for spotting the error!)
- **Added** definition of $f'(x)$ to section *3.2 Cubic Splines* because the $\ominus$-sign might be a pitfall at the exam!
- **Minor edits:**
  - Section *4.9 Gauss quadrature*: added integral bounds a and b to 2-point Gauss rule
  - *4.12 Quadrature in multiple dimensions*: added clause "order **of accuracy** s" (as opposed to order of error)

July 22, 2020

- Corrected **error** in section *2.3 Secant Method*:
  - Changed descent term to $\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$.
    Previously was $\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$!!!

  (Big thank you to Eric for this!)
- **Minor edits:**
  - Fixed numerous typos
  - Now enforces single-spacing across all pages (last page was not necessarily single-spaced before)

July 26, 2020

- Corrected **error** in section *4.8 Adaptive Quadrature*:
  - Changed recursion term from
    INTEGRAL(a, m) + INTEGRAL(m, b)
    to ADAPINT(a, m) + ADAPINT(m, b)
  - else return now outputs more accurate $I_i(h/2)$

  (Thanks to Irma and Manuel for pointing this out!)

July 28, 2020

- **Added** more information in section *4.11 N-point Gauss rule*:
  - Quadrature points x_k as roots of Legendre polynomial
  - Basic examples for Legendre polynomials and Bonnet's recursion formula

July 29, 2020

- Minor Edit:
  - Fixed typo in section *4.11 N-point Gauss rule*: Lagrange-polynomial $P_1 = x$ previously missed =-sign

  Thanks to Simon for this tip!

July 30, 2020

Added two things since I noticed that many people were confused by it:
- Section *4.5 Newton-Cotes*:
  Added formula for integration over entire domain (I from a to b) in addition to I_i from x_i to x_{i+1}
- Section *4.11.1 N-point Gauss rule recipe*:
  Added a comprehensive recipe for Gauss integration.
  For space reasons, it is on another page than the rest of N-point Gauss rule (section *4.11.2*).

This will probably be the last version before the exam. Good luck!

MAD Cheat Sheet

Klemens Iten - kiten@ethz.ch
Pascal Auf der Maur - pascalau@ethz.ch

Version: July 30, 2020

1. Function Fitting

1.0 Cost Functions

L_2-Norm (average):
$$\|e\|_2 = \sqrt{\sum_{i=1}^{N} e_i^2} = \sqrt{\sum_{i=1}^{N} (y_i - f(x_i))^2}$$

L_1-Norm (median):
$$\|e\|_1 = \sum_{i=1}^{N} |e_i| = \sum_{i=1}^{N} |y_i - f(x_i)|$$

L_∞-Norm (maximum): and In general: p-Norm
$$\|e\|_\infty = \max |e_i| = \max |y_i - f(x_i)|, \quad \|e\|_p = \left(\Sigma_i^N e_i^p\right)^{\frac{1}{p}}$$

1.0 Data and Function Fitting

Data Points: $\{(x_i, y_i)\}_{i=1}^N$

Fitted function: $f(x; \mathbf{w}) = \sum_{k=1}^{M} w_k \varphi_k(x)$
where $\varphi_k(x)$ are linear independent basis functions.
The functions can be non linear (ex: $e^{\beta x}$) but the unknown weights w must enter linearly. Usually $M < N$.

1.1 Linear Least Squares

Cost Function (L_2-Norm):
$$E(\mathbf{w}) = \sum_{i=1}^{N} e_i^2(\mathbf{w}) = \sum_{i=1}^{N} (y_i - f(x_i; \mathbf{w}))^2$$

Goal: Find weights $\mathbf{w}^\star = \arg\min_\mathbf{w} E(\mathbf{w})$
In Matrix Formulation and with case distinction:
$$A \cdot \mathbf{w} = \mathbf{y} \qquad A \in \mathbb{R}^{N \times M}, \mathbf{w} \in \mathbb{R}^M \text{ and } \mathbf{y} \in \mathbb{R}^N$$

1. $M = N$: unique solution, $\mathbf{w} = A^{-1}\mathbf{y}$
2. $M > N$: underdetermined, infinitely many solutions
3. $M < N$: overdetermined, $\partial E/\partial \mathbf{w} = 0$ for $\mathbf{w}^\star$:
$$\mathbf{w}^\star = \arg\min_\mathbf{w} E(\mathbf{w}) = (A^T A)^{-1} A^T \mathbf{y}$$

In Matrix Formulation:
$$\begin{aligned} E &= e^T \cdot e = (y - Aw)^T (y - Aw) \\ &= y^T y - 2w^T A^T y + w^T A^T A w \\ \partial E/\partial w &= 0 \Rightarrow A^\top A \mathbf{w}^\star = A^\top \mathbf{y}, \text{ normal eq.} \end{aligned}$$

Inverse Matrix Formulas:
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

$$(\mathbb{R}^{3\times3})^{-1} = \frac{1}{\det A} \cdot \begin{pmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ dh - eg & bg - ah & ae - bd \end{pmatrix}$$

Formulas for linear case

$$\boxed{x_1 + x_2 t_i \approx b_i}$$

$$x_1 = \frac{\sum t_i^2 \sum b_i - \sum t_i \sum t_i b_i}{N \sum t_i^2 - (\sum t_i)^2} \qquad x_2 = \frac{N \sum t_i b_i - \sum t_i \sum b_i}{N \sum t_i^2 - (\sum t_i)^2}$$

$$b = X_1 + X_2(t - \bar{t}) \qquad X_1 = \frac{\sum b_i}{N} \qquad X_2 = \frac{\sum(t_i - \bar{t})b_i}{\sum(t_i - \bar{t})^2}$$

Geometrical Interpretation

From normal equations (vector $\mathbf{e}^\star$ is orthogonal to A):
$$A^\top (\mathbf{y} - A\mathbf{w}^\star) = A^T \mathbf{e}^\star = 0$$

$$P = A(A^T A)^{-1} A^T$$
$$M = I - A(A^\top A)^{-1} A^\top$$
$$\mathbf{y} = (P + M)\mathbf{y} = A\mathbf{w}^\star + \mathbf{e}^\star$$

Properties of the projection matrices:
1. symmetric: $P^T = P, M^T = M$
2. idempotent: $P^2 = P, M^2 = M$
3. $P + M = I$
$$\mathbf{y} = (P + M)\mathbf{y} = A\mathbf{w}^\star + \mathbf{e}^\star$$

1.2 Numerical Solutions

With computed weights $\tilde{\mathbf{w}}$ and error $\delta\mathbf{w} = \mathbf{w} - \tilde{\mathbf{w}}$, error in output is $\delta\mathbf{y} = \mathbf{y} - \tilde{\mathbf{y}} = \mathbf{y} - A\tilde{\mathbf{w}} = A\mathbf{w} - A\tilde{\mathbf{w}}$.
$$\frac{\|\delta\mathbf{w}\|}{\|\mathbf{w}\|} \leqslant \|A^{-1}\| \|A\| \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} = \kappa(A) \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|}$$

The condition number
$$\boxed{\kappa(A) = \|A\|\|A^{-1}\| = \frac{\sigma_{max}(A)}{\sigma_{min}(A)} \qquad 1 \leqslant \kappa(A) < \infty}$$

- Tells us how stable a fit is: the smaller κ, the better
- Used norm: L_2-norm
- For our approach: $\kappa(A^T A) = \kappa(A)^2$.
  By solving the normal equation, the condition number gets squared due to A being twice in the inverse.
- A more stable solution would be to apply a QR or SVD decomposition where the inverse first gets normalized.

QR-Decomposition
$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 & 0 \end{bmatrix}^T$$
$$\boxed{\mathbf{w}^\star = R_1^{-1} Q_1^T \mathbf{y}} \qquad \kappa(A) = \kappa(R_1)$$

Singular Value Decomposition
$$A = \begin{bmatrix} U_r & U_n \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_r^\top \\ V_n^\top \end{bmatrix} = U\Sigma V^\top$$

Moore-Penrose Pseudo-Inverse of A:
$$A^+ = V \cdot \Sigma^+ \cdot U^T$$

Pseudo-Inverse of Σ:
$$\begin{aligned} \Sigma^+ &= (\text{diag}(\sigma_1, \sigma_2 \dots, 0)^+ \\ &= \text{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \dots, 0) \end{aligned}$$

$$\boxed{\mathbf{w}^\star = V\Sigma^+ U^T \mathbf{y}} \qquad \kappa(A) = \kappa(\Sigma V^\top)$$

2. Nonlinear Systems

2.0 Preliminaries

Comparison between methods

	Bisection	Newton's Mthd.	Secant Mthd.
rate	1	2	1.618/1.839
cost	cheap	expensive	middle
robust	yes	no	no

Root Finding Problem
Any equation $g(x) = h(x)$ can be rewritten as
$$g(x) - h(x) = f(x) = 0.$$
The solution to this is the root (or zero) $x^\star$, i.e. $f(x^\star) = 0$

Bolzano's Theorem
For a function $f(x) \in [a, b]$, if $f(a) \cdot f(b) < 0$,
i.e. $\text{sgn}(f(a)) \neq \text{sgn}(f(b))$, then according to the intermediate value theorem
$$\exists\, x^\star \in (a, b) \text{ with } f(x^\star) = 0$$

Condition Number
$$\kappa = \frac{|\delta y|}{|\delta x|} = \frac{|f(x + \delta x) - f(x)|}{|\delta x|} \approx |f'(x)|$$

With $f(x^\star) = 0 \Leftrightarrow x^\star = f^{-1}(0)$:
$$\kappa = \left|\frac{\partial}{\partial y} f^{-1}(y)\right|\Big|_{y=0} = \frac{1}{|f'(f^{-1}(0))|} = \frac{1}{|f'(x^\star)|}$$

Order of Convergence
With $e_k = x_k - x^\star$, if $x_k \overset{k\to\infty}{=} x^\star$, there exists:
$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^r} = C \quad \text{with} \begin{cases} r, & \text{rate of convergence} \\ C, & \text{order of convergence} \end{cases}$$

- $r = 1$: if $0 < C < 1$ linear convergence
  $C = 0$ superlinear, $C = 1$ sublinear.
- $r = 2$: quadratic convergence.
- $r \approx \log\left|\frac{e_{k+2}}{e_{k+1}}\right| / \log\left|\frac{e_{k+1}}{e_k}\right|$

2.1 Bisection Method

1. Start with two function eval's $\text{sgn}(f(a)) \neq \text{sgn}(f(b))$
2. Halve the interval length, until a solution has been isolated within a prescribed accuracy

- $C = 1$, $r = 1/2$
- Certain to converge (but slow), uses only signs of f
- f doesn't need to be differentiable, just continuous
- Initial interval $[a, b]$ may be hard to find

2.2 Newton's Method

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

- Follows from taylor expansion around root $x^\star$, ignoring hot
- Exact for linear functions only
- Quadratic convergence, but not guaranteed
- Sensitive to initial conditions
- Can get stuck in local minimum
- Requires function eval and derivative for each iteration

2.3 Secant Method

approximate $f'(x_k) \approx (f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$
$$\boxed{x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}}$$

- No calc. of derivative, only one f evaluation at each step
- Convergence rate is not quadratic
- Requires two initial approximations x_0, x_1

2.4 Set of Equations

A general system with N equations and $x_{1\dots M}$ unknowns:
$$F(\mathbf{x}^\star) = \begin{bmatrix} f_1(\mathbf{x}^\star) & \cdots & f_N(\mathbf{x}^\star) \end{bmatrix}^T = \vec{0}$$
Solution approximated by taylor series:
$$f_i(\mathbf{x} + \mathbf{y}) = f_i(\mathbf{x}) + \sum_{j=1}^{M} \frac{\partial f_i(\mathbf{x})}{\partial x_j} y_j + O(\|\mathbf{y}\|^2)$$
$$\boxed{F(\mathbf{x} + \mathbf{y}) = F(\mathbf{x}) + J(\mathbf{x})\mathbf{y} + O(\|\mathbf{y}\|^2)} \quad \kappa = \|J^{-1}(\mathbf{x}^\star)\|$$

Jacobian Matrix
$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(\mathbf{x})}{\partial x_1} & \frac{\partial f_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_N(\mathbf{x})}{\partial x_M} \end{bmatrix}$$

$N = M$: **Newton-Raphson Method**
$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - J^{-1}(\mathbf{x}_k) F(\mathbf{x}_k)}$$

Solve $J(\mathbf{x}_k)\mathbf{z} = -F(\mathbf{x}_k)$ for $\mathbf{z} \quad \to \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}$

$N \neq M$: **pseudo-Newton Method**
$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - J^+(\mathbf{x}_k) F(\mathbf{x}_k)}$$

$$J_{M>N}^+ = (J^T J)^{-1} J^T \qquad J_{M<N}^+ = J^T (JJ^T)^{-1}$$

2.5 Non-Linear Optimization

$$\mathbf{x}^\star = \arg\min_\mathbf{x} E(\mathbf{x})$$

with unknowns $\mathbf{x} = (x_1, \dots, x_M)^T$ and $E: \mathbb{R}^M \to \mathbb{R}$.
Verify if $\mathbf{x}^\star$ is local minimum with **Hessian Matrix**:
$$\boxed{\nabla^2 E(\mathbf{x}^\star) > 0} \quad \text{with } F(\mathbf{x}) = \nabla E(\mathbf{x}) = 0$$

$$\nabla^2 E(\mathbf{x}^\star) = H(\mathbf{x}^\star) = \begin{bmatrix} \frac{\partial^2 E(\mathbf{x}^\star)}{\partial x_1^2} & \cdots & \frac{\partial^2 E(\mathbf{x}^\star)}{\partial x_1 \partial x_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\mathbf{x}^\star)}{\partial x_M \partial x_1} & \cdots & \frac{\partial^2 E(\mathbf{x}^\star)}{\partial x_M^2} \end{bmatrix}$$

Newton's Method
Search root $F(\mathbf{x}) = \nabla E(\mathbf{x}) = 0$, with $J(\mathbf{x}) = \nabla^2 E(\mathbf{x})$
$$\boxed{\begin{aligned} \text{Solve for } \mathbf{z}: \ & \nabla^2 E(\mathbf{x}_k)\mathbf{z} = -\nabla E(\mathbf{x}_k) \\ \Rightarrow\ & \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z} \end{aligned}}$$

3. Interpolation and Extrapolation

3.1 Lagrange Interpolation

Key Idea

Fit N polynomials of degree $N-1$ s.t. $l_i(x_j) = \delta_{ij}$:

$$l_k(x) = \prod_{i=1, i\neq k}^{N} \frac{x-x_i}{x_k-x_i}$$

$$= \frac{(x-x_1)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N)}{(x_k-x_1)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_N)}$$

$$\boxed{f(x) = \sum_{k=1}^{N} y_k \cdot l_k(x)}$$

Properties
- Can cause huge oscillations in the edge regions (Runge's phenomenon)
- Not accurate especially for local trends
- Small fluctuations in data result in re-fitting the whole model over the whole domain
- Sensitive to noise

Approximation Error
$$|y(x) - f(x)| = \left| \frac{y^{(n)}(\xi)}{n!} \prod_{k=1}^{n} (x - x_k) \right|$$

3.2 Cubic Splines

Key Idea and Derivation

Piecewise local polynomials with different parameters for each interval, split data into smaller independent intervals.

Construct $f(x) \in [x_1 < x < x_n]$, $f(x_i) = y_i$ from $f_i(x) \in [x_i, x_{i+1}]$ with N unknowns $f_i'' = f''(x_i)$:

$$f''(x) = f_i'' \frac{(x_{i+1}-x)}{\Delta_i} + f_{i+1}'' \frac{(x-x_i)}{\Delta_i}$$

$$f'(x) = -f_i'' \frac{(x_{i+1}-x)^2}{2\Delta_i} + f_{i+1}'' \frac{(x-x_i)^2}{2\Delta_i} + C_i$$

$$f(x) = f_i'' \frac{(x_{i+1}-x)^3}{6\Delta_i} + f_{i+1}'' \frac{(x-x_i)^3}{6\Delta_i} + C_i(x-x_i) + D_i$$

From boundary conditions:

$$C_i = \frac{y_{i+1}-y_i}{\Delta_i} - (f_{i+1}'' - f_i'')\frac{\Delta_i}{6}$$

$$D_i = y_i - f_i'' \frac{\Delta_i^2}{6}$$

$$\boxed{\begin{aligned} \frac{\Delta_{i-1}}{6}f_{i-1}'' + \left(\frac{\Delta_{i-1}+\Delta_i}{3}\right)f_i'' + \frac{\Delta_i}{6}f_{i+1}'' \\ = \frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i-y_{i-1}}{\Delta_{i-1}} \end{aligned}}$$

Initial Conditions
- Natural spline: $f_1'' \equiv f_N'' = 0$
- Parabolic runout: Set $f_1'' = f_2''$ and $f_N'' = f_{N-1}''$
- Clamping: Set $f'(x_1) = f'(x_N) = 0$

Tri-Diagonal Matrix Algorithm (TDMA)

$N-2$ equations and 2 initial conditions in $N \times N$-Matrix

$$\begin{bmatrix} b_1 & c_1 & & & & 0 \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & c_{N-1} \\ 0 & & & & a_N & b_N \end{bmatrix} \begin{bmatrix} f_1'' \\ f_2'' \\ f_3'' \\ \vdots \\ \vdots \\ f_N'' \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_N \end{bmatrix}$$

$$a_i = \frac{\Delta_{i-1}}{6}, \quad b_i = \frac{\Delta_{i-1}+\Delta_i}{3}$$

$$c_i = \frac{\Delta_i}{6}, \quad d_i = \frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i-y_{i-1}}{\Delta_{i-1}}$$

---

4. Numerical Integration

4.0 Key Idea

Split a definite integral which may be analytical or not into N consecutive intervals

$$I = \int_a^b f(x)\mathrm{d}x = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x$$

and approximate $f(x)$ by $p(x)$:

$$I \approx \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} p_i(x)\mathrm{d}x$$

4.1 Rectangle rule

$$p_i(x) = f(x_i) \quad \rightarrow \quad I_{R_i} = f(x_i)\Delta_i$$

$$\boxed{I \approx \Delta_x \sum_{i=0}^{N-1} f(x_i)} \quad \text{with error in second order}$$

4.2 Midpoint rule

$$p_i(x) = f\left(\frac{x_i+x_{i+1}}{2}\right) \quad \rightarrow \quad I_{M_i} = f(x_{i+1/2})\Delta_i$$

$$\boxed{I \approx \Delta_x \sum_{i=0}^{N-1} f\left(\frac{x_i+x_{i+1}}{2}\right)}$$

Error in third order: $E_{M_i} = \frac{1}{24}f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5)$

4.3 Trapezoidal rule

$$p_i(x) = \frac{(f(x_{i+1})-f(x_i))\cdot x}{\Delta_i} + f(x_i)$$

$$I_{T_i} = \frac{f(x_i)+f(x_{i+1})}{2}\Delta_i$$

$$= I_{M_i} + \frac{1}{8}f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5)$$

$$\boxed{I \approx \frac{\Delta_x}{2}\left(f(x_0) + 2\cdot\left(\sum_{i=1}^{N-1} f(x_i)\right) + f(x_N)\right)}$$

Error in third order: $E_{T_i} = -\frac{1}{12}f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5)$

4.4 Simpson's Rule

Approximate $f(x)$ with a parabola $p_i = f(x^3)$:

$$I_{S_i} = \frac{f(x_i)+4f((x_i+x_{i+1})/2)+f(x_{i+1})}{6}\Delta_i$$

$$= \frac{2}{3}I_{M_i} + \frac{1}{3}I_{T_i}$$

$$\boxed{\begin{aligned} I \approx \frac{\Delta_x}{3}\Big(f(x_0) + 4\sum_{i=1,\text{ odd}}^{N-1} f(x_i) + \dots \\ + 2\sum_{i=2,\text{ even}}^{N-2} f(x_i) + f(x_N)\Big) \end{aligned}}$$

Error in fifth order: $E_{S_i} = O(\Delta_i^5) + \cdots$

Attention:
Error is reduced by one order if it is evaluated over a domain.

4.5 Newton-Cotes

$$l_k^M(x) = \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_M)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_M)}$$

$$I_i \approx \int_{x_i}^{x_{i+1}} p_i(x)\mathrm{d}x = \sum_{k=0}^{M} f(x_k)\int_{x_i}^{x_{i+1}} l_k^M(x)\mathrm{d}x$$

$$\boxed{I_i \approx \Delta_i \sum_{k=0}^{M} C_k^M f(x_k) \text{ with } C_k^M = \frac{1}{\Delta_i}\int_{x_i}^{x_{i+1}} l_k^M(x)\mathrm{d}x}$$

$$\boxed{I \approx (b-a)\sum_{k=0}^{M} C_k^M f(x_k) \text{ with } C_k^M = \frac{1}{b-a}\int_a^b l_k^M(x)\mathrm{d}x}$$

C_k^M : averaged Lagrange basis over the interval

Properties of C_k^M: $\sum C_k^M = 1$ and $C_k^M = C_{M-k}^M$

4.6 Richardson Extrapolation

The absolute value G is approximated with $G \approx G(h)$, which is dependant on discretisation h.

Taylor Expansion of Quantity for $G(h) \xrightarrow{h\to 0} G$

$$G(h) = G(0) + c_1 h + c_2 h^2 + \cdots$$

$$G(h/2) = G + \frac{1}{2}c_1 h + \frac{1}{4}c_2 h^2 + \cdots$$

Richardson's Combination of $G(h)$ and $G(h/2)$

$$G_1(h) = 2G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \cdots$$

$$G_2(h) = \frac{1}{3}(4G_1(h/2) - G_1(h)) = G + O(h^3)$$

$$\boxed{\begin{aligned} G_n(h) = \frac{1}{2^n-1}(2^n G_{n-1}(h/2) - G_{n-1}(h)) \\ = G + O(h^{n+1}) \end{aligned}}$$

Error estimation

$$\epsilon(h/2) \approx G(h/2) - G(h)$$
$$\text{Relative Tolerance: } \epsilon(h/2) < 3\cdot \text{tol}\cdot\frac{h}{h_0}$$
$$\text{Error order: } E_{n-1}(h) > E_{n-1}(h/2) > E_n(h)$$

4.7 Romberg integration

Trapezoidal rule with spacing $h = \frac{b-a}{n}$.

Starting with one interval, the number of intervals is doubled with every iteration until the desired accuracy is achieved.

$$I_0^n = \frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f(a + jh)\right]$$

$$I_0^n = I - c_1 h^2 - c_2 h^4 - c_3 h^6$$

$$I_0^{2n} = I - c_1\frac{h^2}{4} - c_2\frac{h^4}{16} - c_3\frac{h^6}{64}\cdots$$

$$I_1^n = \frac{4I_0^{2n} - I_0^n}{3} = I + \frac{1}{4}c_2 h^4 + \frac{5}{16}c_3 h^6\cdots$$

Richardson applied to Trapezoidal and Simpson Rule:

$$I_k^n = \frac{4^k I_{k-1}^{2n} - I_{k-1}^n}{4^k - 1} \qquad I_k^n = \frac{4^{k+1} I_{k-1}^{2n} - I_{k-1}^n}{4^{k+1} - 1}$$

4.8 Adaptive Quadrature

- Optimize quadrature by sampling the funtion non-uniformly.
- Evaluate the integral with more precision at points with sudden changes.
- Use Rhomberg integration and error estimation to evaluate locally.

Pseudocode: Adaptive Integration

```
function ADAPINT(a, b)
    I_i(h) = INTEGRAL(a, b)
    m = (a + b)/2
    I_i(h/2) = INTEGRAL(a, m) + INTEGRAL(m, b)
    ε = I_i(h) − I_i(h/2)
    if ε > desired :
        return ADAPINT(a, m) + ADAPINT(m, b)
    else return I_i(h/2)
```

4.9 Gauss quadrature

Key idea

$$I = \int_a^b f(x)\,\mathrm{d}x \approx \sum_i \left(c_i \cdot f(x_i)\right)$$

- We choose c_i and x_i to minimize the error.
- Inspect coefficients to find i unknowns $c_1\dots c_i$, $x_1\dots x_i$.

Undetermined Coefficients

This method is exact for integrals of a straight line and recovers the trapezoidal rule.

$$\int_a^b f(x)\mathrm{d}x = \int_a^b (a_0 + a_1 x)\,\mathrm{d}x \approx c_1 f(a) + c_2 f(b)$$

We get: $c_1 = c_2 = \frac{b-a}{2}$

2-point Gauss rule

For two points, we require the equation to exactly integrate a cubic polynomial.

$$\int_a^b a_0 + a_1 x + a_2 x^2 + a_3 x^3 dx \approx c_1 f(x_1) + c_2 f(x_2)$$

We get: $c_{1,2} = \frac{b-a}{2}$, $x_{1,2} = \left(\frac{b-a}{2}\right)\left(\mp\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}$

4.10 Hermite Interpolation

Interpolate through N points $\{x_i, y_i\}_{i=1}^N$ by fitting a polynomial of degree $2N-1$ s.t. $f(x_i) = y_i$ and $f'(x_i) = y_i'$

$$\boxed{f(x) = \sum_{k=1}^{n} U_k(x)y_k + \sum_{k=1}^{n} V_k(x)y_k'} \quad \begin{aligned} U_k(x_j) &= \delta_{jk} \\ U_k'(x_j) &= 0 \\ V_k(x_j) &= 0 \\ V_k'(x_j) &= \delta_{jk} \end{aligned}$$

$$U_k(x) = [1 - 2L_k'(x_k)(x - x_k)]L_k^2(x)$$

$$V_k(x) = (x - x_k)L_k^2(x)$$

4.11.1 N-point Gauss rule - Recipe

Goal: Evaluate $I = \int_a^b f(x)\mathrm{d}x$ using N points

1. Change the boundary of the integral using change of variables: $z = \frac{2x}{b-a} + 1 - \frac{2b}{b-a}$ for

$$I = \int_{-1}^1 \frac{b-a}{2}f\left(\frac{b-a}{2}(z - 1) + b\right)\mathrm{d}z$$

2. Read out integration points z_i and weights w_i from tables

3. Evaluate $I \approx \frac{b-a}{2}\sum_{i=1}^{N} w_i f\left(\frac{b-a}{2}(z_i - 1) + b\right)$

4.11.2 N-point Gauss rule

Key Idea

Transform integration interval $I \in (a, b)$ to $I' \in (-1, 1)$:

$$x = \frac{2\xi - (a+b)}{b-a} \quad \text{with } \zeta \in (a, b),\ x \in (-1, 1)$$

Approximate $\int_{-1}^{1} f(x)\mathrm{d}x$ with Hermite polynomials.
Accurate for polynomials of order $2N-1$.

Derivation

$$\int_{-1}^{1} f(x)\mathrm{d}x = \sum_{k=1}^{n} y_k \int_{-1}^{1} U_k(x)\mathrm{d}x + \sum_{k=1}^{n} y_k' \int_{-1}^{1} V_k(x)\mathrm{d}x$$

$$\int_{-1}^{1} f(x)\mathrm{d}x = \sum_{k=1}^{n} u_k f(x_k) + \sum_{k=1}^{n} v_k f'(x_k) = \sum_{i=1}^{n} w_i f(x_i)$$

$$u_k = \int_{-1}^{1} U_k(x)\mathrm{d}x, \quad v_k = \int_{-1}^{1} V_k(x)\mathrm{d}x \overset{!}{=} 0$$

Use Legendre polynomials to ensure
$$v_k = C_k \int_{-1}^{1} F(x) L_k(x)\mathrm{d}x = 0 \text{ to receive}$$

$$\int_{-1}^{1} f(x)dx = \sum_{k=1}^{n} u_k f(x_k) \quad u_k = \frac{2}{(1 - x_k^2)(P_n'(x_k))^2}$$

with x_k, the roots of the n-th Legendre polynomial.

Legendre Polynomials

$$P_n(x) = \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \frac{(2n-2k)!}{(n-k)!(n-2k)!k!2^n} x^{n-2k}$$

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{1}{2}(3x^2 - 1)$$

$$n \cdot P_n(x) = (2n-1)x P_{n-1}(x) - (n-1)P_{n-2}(x)$$

Error with n abscissas

$$\varepsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\xi)$$

4.12 Quadrature in multiple dimensions

Goal:

Integrate a function in D dimensions:
$$I = \int_{a_1}^{b_1} \cdots \int_{a_D}^{b_D} f(x_1, \ldots, x_D)\, \mathrm{d}x_1 \cdots \mathrm{d}x_D$$

Using Quadrature in every dimension with N gridpoints:

$$\int_{a_d}^{b_d} f(x_d)\,\mathrm{d}x_d \approx \sum_{i_d=1}^{N} w_{i_d} f(x_{i_d})$$

gives:
$$I \approx \sum_{i_1=1}^{N} \cdots \sum_{i_N=1}^{N} w_{i_1} \ldots w_{i_N}\, f(x_{i_1}, \ldots, x_{i_D})$$

Curse of dimensionality:

Quadrature in D dimensions requires $M = N^D$ function evaluations

Additionally, order of accuracy depends on dimension D, one-dimensional order of acc. s and grid spacing $h = \frac{b-a}{N}$:
$$I - I_Q = \mathcal{O}(h^s) = \mathcal{O}(N^{-s}) = \mathcal{O}\left(M^{-s/D}\right)$$

4.13 Monte Carlo Integration

Key Idea

Multiply D-dimensional integral and multiply with domain $|\Omega|$ and uniform distribution $p_{\mathcal{U}}(x) = \frac{1}{|\Omega|}$:

$$I = \int_{\Omega} f(x)\mathrm{d}x = |\Omega| \int_{\Omega} f(x)p_{\mathcal{U}}(x)\mathrm{d}x = |\Omega| \cdot \underset{x \sim p_{\mathcal{U}}}{\mathbb{E}[f(X)]}$$

With independent and identically distributed samples
$$\{f(x^{(1)}), \ldots, f(x^{(M)})\}$$
with mean $\mathbb{E}[f(\mathbf{x})] = \langle f \rangle_M$ and $\mathrm{Var}[f(\mathbf{x})] = \sigma^2 < \infty$, the *Central Limit Theorem* follows:

$$\sqrt{M}\left(\frac{1}{M}\sum_{i=1}^{M} f\left(x_1^{(i)}, \ldots, x_D^{(i)}\right) - I\right) \to \mathcal{N}\left(0, \sigma^2\right)$$

Estimate of the Value of the Integral

With expectation value of the integrand f
$$\langle f \rangle = \frac{1}{|\Omega|} \int_{\Omega} f(\vec{x})\mathrm{d}\vec{x}$$

over the domain $\Omega = \int_{\Omega} d\vec{x}$:
$$I = |\Omega| \langle f \rangle$$

Use samples from uniform distribution from $i = 1 \ldots M$
$$\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_D^{(i)}\right) \sim U\left([a_1, b_1] \times \cdots \times [a_D, b_D]\right)$$

and approximate $\langle f \rangle \approx \langle f \rangle_M = \frac{1}{M}\sum_{i=1}^{M} f(\vec{x}_i)$
Then the integral value estimate is

$$I \approx I_M = |\Omega| \cdot \frac{1}{M} \sum_{i=1}^{M} f\left(x_1^{(i)}, \ldots, x_D^{(i)}\right)$$

Error Estimate

$$\epsilon = \sqrt{\mathrm{Var}\left[\langle f \rangle_M - \langle f \rangle\right]} = \sqrt{\mathrm{Var}\left[\langle f \rangle_M\right]}$$

$$= \sqrt{\frac{\mathrm{var}[f]}{M}} \propto \mathcal{O}\left(M^{-1/2}\right)$$

Note: Error is independent of dimension D!

Summary

1. Sample points $\mathbf{x}_i$ from a uniform distribution and evaluate the integrand f to get random variables $f(\mathbf{x}_i)$
2. Store the number of samples, the sum of values, and the sum of squares: $\quad M, \sum_{i=1}^{M} f(\mathbf{x}_i), \sum_{i=1}^{M} f(\mathbf{x}_i)^2$
3. Compute the mean as the estimate of the expectation (normalized integral):
$$\frac{I}{|\Omega|} = \langle f \rangle \approx \langle f \rangle_M = \frac{1}{M}\sum_{i=1}^{M} f(\mathbf{x}_i)$$
4. Estimate the variance using the unbiased sample variance:
$$\mathrm{Var}[f] \approx \frac{M}{M-1}\left(\frac{1}{M}\sum_{i=1}^{M} f(\mathbf{x}_i)^2 - \langle f \rangle_M^2\right)$$
$$= \frac{M}{M-1}\left(\left\langle f^2 \right\rangle_M - \langle f \rangle_M^2\right)$$
5. Estimate the error:
$$\varepsilon_M = \sqrt{\frac{\mathrm{Var}[f]}{M}} \approx \sqrt{\frac{1}{M-1}\left(\langle f^2 \rangle_M - \langle f \rangle_M^2\right)}$$

4.14 Inverse Transform Sampling

Key Idea

A random variable X with PDF $p_X(x)$ and CDF $F_X(x)$ can be generated by samples $u^{(i)}|_{i=1\ldots N}$ from a uniformly distributed value $U \sim \mathcal{U}([0,1])$ with a transformation $x = g(u)$.

Transformation

$$F_X(x) = u \quad \Leftrightarrow \quad x = F_X^{-1}(u)$$

$$x^{(i)} = F_X^{-1}\left(u^{(i)}\right) \quad F(x) = \int_0^x p(x)\mathrm{d}x$$

4.15 Accept-Reject Sampling

General Method

Goal: Get samples from distribution $p(x)$. We have a distribution $h(x)$ which we know how to sample from. $h(x)$ bounds $p(x)$ such that $p(x) < \lambda \cdot h(x)$

1. draw a sample x from $h(x)$
2. draw a uniform random number u from [0,1]
3. accept the sample if $u < \frac{p(x)}{\lambda \cdot h(x)}$, otherwise forget x
4. repeat until enough samples are obtained

Rejection sampling for Integration

To integrate a function in D dimensions, sample
$$\mathbf{x}^{(i)} = \left(x_1^{(i)}, \ldots, x_D^{(i)}\right) \sim U(\Omega) \text{ from } i = 1 \ldots M$$

Evaluate function $f\left(\mathbf{x}^{(i)}\right)$ for $i = 1 \ldots M$

Accept sample if $f\left(\mathbf{x}^{(i)}\right) \leqslant y^{(i)}$, *reject* otherwise:

$$I \approx \frac{\#\text{ accepted samples}}{M} \cdot |\Omega|$$

5. Probability Theory

5.1 Basics

A probability density $P : \mathcal{F} \to [0, 1]$ assigns probabilities to events with $P(\Omega) = 1$.

- State/sample space $\mathcal{F}$ (range of values of RV)
- Discrete case: $\sum_i P(x_i) = 1$
- Continuous case: $\int_{\Omega} dP = \int_{\Omega} p(x)\mathrm{d}x = 1$ with $p(x)$, the probability density function
- $P(a \leqslant X \leqslant b) = \int_a^b p(x)\,\mathrm{d}x$
- Cumulative Distribution Function:
  $F_X(x) = P(X \leqslant a) = \int_{-\infty}^{a} p(x)\,\mathrm{d}x$

5.2 Set Theory

Consider two subsets A and B of a whole sample set $\mathcal{F}$.

- Complacement of a set: $A^c = \mathcal{F} \backslash A$
- Union of sets: $A \cup B$
- Intersection of sets: $A \cap B$
- Disjoint sets: $A \cap B = \varnothing$
- Special case: IF $A^c = B$ AND $A \cup B = \mathcal{F} : A \cap B = \varnothing$

5.3 Statistical Terms

Expectation value :
- Discrete case (mean): $\mu = \sum_x x \cdot P(x) \quad \boxed{\mu = \mathbb{E}[X]}$
- Continuous case: $\mathbb{E}[h(X)] = \int_a^b h(x)p(x)dx$
- Special case: $h(x) = X \quad \Rightarrow \quad \mathbb{E}(X) = \mu$

Variance $(h(X) = (X - \mu)^2)$:

$$\sigma^2 = \mathrm{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2$$

5.4 Empirical Distributions

Definition: Consider a set of N independent, identically distributed data points of an unknown distribution function.

Probability Density Function (PDF):
$$\hat{p}_n(x) = \frac{\#\text{elements in the sample with value} = x}{N \text{ (size of sample set)}} = \frac{\sum_{i=1}^{N} \mathbb{1}|_{X_i = x}}{N}$$

Cumulative Distribution Function (CDF):
$$\hat{F}_n(x) = \frac{\#\text{elements in the sample with value} \leqslant x}{N \text{ (size of sample set)}} = \frac{\sum_{i=1}^{N} \mathbb{1}|_{X_i \leqslant x}}{N}$$

Unbiased Estimator:

An unbiased estimator of a statistical parameter means that the expected value equals the true value of the parameter, e.g. $\mathbb{E}[\hat{\mu}] = \mu$, $\quad \mathbb{E}\left[\hat{\sigma}^2\right] = \sigma^2$

$$\hat{\mu} = \frac{1}{N}\sum_i x_i \qquad \hat{\sigma}^2 = \frac{1}{N-1}\sum_i (x_i - \hat{\mu})^2$$

5.5 Common Distributions

Uniform Distribution
$$p_{\mathcal{U}}(x) = \frac{1}{b-a}$$

Binomial Distribution
$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Normal Distribution
$$p_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Exponential Distrib.
$$p(x) = \lambda e^{-\lambda x}$$

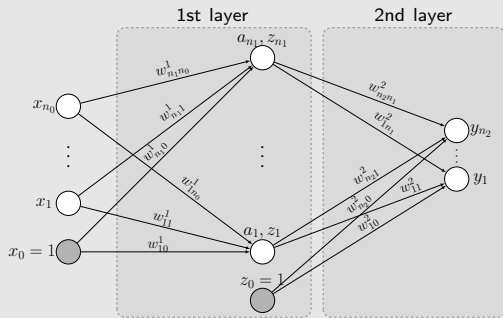# 6. Neural Networks

6.1 General Structure

$$\mathbf{y} = F(\mathbf{x}, \mathbf{w}), \quad \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$$
$$\text{output} = F(\text{input}, \text{weight})$$

Different Types of Neural Networks
- Fully Connected Neural Networks
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

6.2 2-Layer Networks

For each layer:
1. Input x_i is weighted by w_i
2. Summed
3. Activation function φ is applied

Map Input to First Layer:
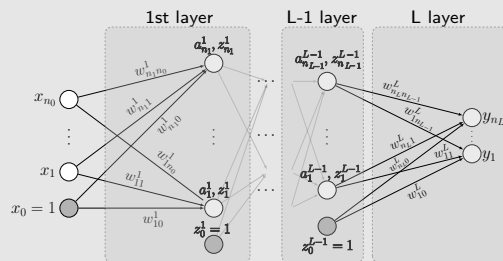$$a_j^1 = \sum_{i=0}^{n_0} w_{ji}^1 x_i \quad \text{and} \quad z_j^1 = \varphi_1\left(a_j^1\right)$$

Map First to Second Layer / Output
$$a_j^2 = \sum_{i=0}^{n_1} w_{ji}^2 z_i^1 \quad \text{and} \quad y_j = z_j^2 = \varphi_2\left(a_j^2\right)$$

Compact Notation
$$\boldsymbol{y}(\boldsymbol{x}; \boldsymbol{w}) = \varphi_2\left(W^2 \varphi_1\left(W^1 \boldsymbol{x}\right)\right)$$

6.3 L-Layer Networks

Compact Notation for $\boldsymbol{y}(\boldsymbol{x}; \boldsymbol{w}) =$
$$\varphi_L\left(W^L \varphi_{L-1}\left(W^{L-1} \varphi_{L-2}\left(\cdots W^2 \varphi_1\left(W^1 \boldsymbol{x}\right)\right)\right)\right)$$

6.4 Activation Functions

Heaviside	Ramp (ReLU)	Hyperbolic Tangent	Logistic
$\varphi(x) = \begin{cases} 0, & x<0 \\ 1, & x \geq 0 \end{cases}$	$\varphi(x) = \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases}$	$\varphi(x) = \tanh(x)$	$\varphi(x) = \frac{1}{1+e^{-x}}$
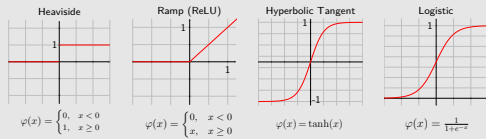
6.5 Training

Goal:
Update the weights w so that the output y_n given an input x_n matches a target $\hat{y}_n$.

Steps:
1. Build a model $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ with the initial weights $\mathbf{w} = \left\{W^1, W^2, \ldots, W^L\right\}$
2. Perform the forward pass, i.e. produce the output $\mathbf{y}_n$ for all $\mathbf{x}_n$ in the dataset
3. Compute the loss with respect to the target:
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(\hat{\mathbf{y}}_n - \mathbf{y}(\mathbf{x}_n, \mathbf{w})\right)^2 = \sum_{n=1}^{N} E_n$$
4. Perform the backward pass, i.e. update weights (see 6.6)
5. Repeat until you reach a minimum: $\mathbf{w}^\star = \arg\min E(\mathbf{w})$

6.6 Gradient Descent (GD) and Variations

Key Idea:
Use derivatives (gradient) of the cost function E with respect to the weights $\mathbf{w}$ to update the parameters:
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} E\left(\boldsymbol{w}^{(k)}\right)$$
with *iteration index* k and *learning parameter* η

Stochastic Gradient Descent (SGD):
Alternative to GD with derivative of local error E_n related to the pair $\{x_n, \hat{y}_n\}$:
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} E_n\left(\boldsymbol{w}^{(k)}\right)$$
with sequential or random choice of E_n.

Batch Stochastic Gradient Descent (batchSGD):
Method between GD and SGD with gradient on subset $\mathcal{I}$, with $\mathcal{I} \subset \{1, 2, \ldots, N\}$, chosen randomly.
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} \sum_{n \in \mathcal{I}} E_n\left(\boldsymbol{w}^{(k)}\right)$$

Learning Parameter η
- Crucial hyper-parameter in deep learning
- Not a priori clear how to be chosen
- (a) η too high/fast: oscillates between suboptimal values
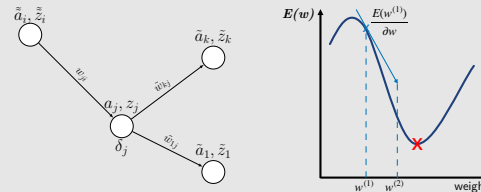- (b) η too low/slow: takes too many iterations to reach $\mathbf{w}^\star$
- (c) desired value of η
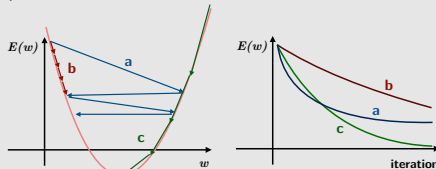
6.7 Backpropagation

Update weights using Gradient Descent:
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} E\left(\boldsymbol{w}^{(k)}\right)$$
and rewrite gradient in terms of $a_j = \sum_k w_{jk} \tilde{z}_k$ (chain rule):
$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \tilde{z}_i = \delta_j \tilde{z}_i$$

Derivative of the Error δ_j
$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial a_j} = \sum_k \tilde{\delta}_k \frac{\partial \tilde{a}_k}{\partial a_j}$$

with $\tilde{a}_k = \sum_j \tilde{w}_{kj} z_j = \sum_j \tilde{w}_{kj} \varphi(a_j)$:
$$\frac{\partial \tilde{a}_k}{\partial a_j} = \varphi'(a_j) \tilde{w}_{kj} \Rightarrow \delta_j = \varphi'(a_j) \sum_k \tilde{w}_{kj} \tilde{\delta}_k$$

Last Layer of Neural Network, i.e. $a_j = y_j$
$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \frac{\partial}{\partial y_j} \frac{1}{2} \|\boldsymbol{y}(\boldsymbol{x}_n; \boldsymbol{w}) - \hat{\boldsymbol{y}}_n\|^2$$
$$= \frac{\partial}{\partial y_j} \frac{1}{2} \sum_k \left(y_k(\boldsymbol{x}_n; \boldsymbol{w}) - \hat{y}_{ni}\right)^2 = y_j(\boldsymbol{x}_n; \boldsymbol{w}) - \hat{y}_{nj}$$

Key Idea of Backpropagation
- At last layer, gradients $\frac{\partial E_n}{\partial w_{ji}}$ and $\frac{\partial E_n}{\partial a_j}$ don't depend on Neural Network
- Calculate δ_j at last layer first, then back-propagate to acquire the δ_j's at every previous layer

6.8 Overfitting

Bias-Variance-Tradeoff
- Overfitted: Model fits behaviour of noise and does not generalize efficiently (model estimation errors)
- Underfitted: Too few parameters, model ignores meaningful data (model mismatch errors)

Key Idea:
Introduce subset of data (~10%) as a test set and run SGD on both training and test data. Plot error for both subsets over iterations:
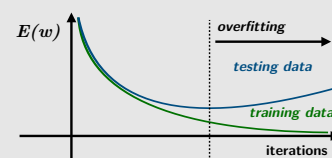
7. Dimensionality Reduction

7.1 Principal Component Analysis (PCA)

Goal
- Decrease dimension of the data while either explaining most of the variance or minimizing the reconstruction loss.
- Changes the coordinate system of the data while aligning the axes to the directions with the most variance
- Data must have zero mean, i.e. $\tilde{x}_n = x_n - \bar{x}$
  $\Rightarrow$ centered data matrix $X \in \mathbb{R}^{N \times D}$

Maximum Variance Formulation
Find direction $\mathbf{v}_1^\star = \operatorname{argmax}_{\gtrsim 1} \mathbf{v}_1^T C \mathbf{v}_1$ s.t. variance is max.:
$$\sigma_1^2 = \frac{1}{N-1} \sum_{n=1}^{N} \left(\mathbf{x}_n^T \mathbf{v}_1\right)^2 = \mathbf{v}_1^T C \mathbf{v}_1$$

with Covariance matrix $C = \frac{1}{N-1} X^T X \in \mathbb{R}^{D \times D}$:
$$C \mathbf{v}_1^\star = \lambda_1^\star \mathbf{v}_1^\star \quad, \quad \mathbf{v}_1^T \mathbf{v}_1 = 1$$

The Principal Comp'ts are the eigenvectors of the eigenvalues of C sorted from the biggest to the smallest:
$$C = V \Lambda V^{-1}, \quad \Lambda = \operatorname{diag}\left(\{\lambda_i\}\right), \quad V^{-1} = V^T \in \mathbb{R}^{D \times D}$$

Compression of data by using only first $r < D$ p.c.:
$$Y_r = V_r^T X$$

Reconstruct Data
$$\tilde{X} = V_r Y_r \in \mathbb{R}^{D \times N} \quad \text{with } \%_{\text{retained variance}} = \frac{\sum_{i=1}^{t} \lambda_i}{\sum_{i=1}^{D} \lambda_i}$$

Kernel PCA
Nonlinear cluster of data made linearly separable by transforming the data by using some kernel functions ϕ:
$$C \text{ changes to } C = \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T$$

7.2 Auto-Associative Neural Network

Key Idea
- Use Neural Network to learn dimension reduction
- Map input $\mathbf{x}_n \in \mathbb{R}^D$ onto an output $\tilde{\mathbf{x}}^n \in \mathbb{R}^D$ through an intermediate layer $\mathbf{y}_n \in \mathbb{R}^r$ using a matrix $W \in \mathbb{R}^{r \times D}$
$$\mathbf{y}_n = W \mathbf{x}_n, \quad \tilde{\mathbf{x}}_n = W^T \mathbf{y}_n$$

Error Function
Find optimal weights by minimizing the error, i.e. the difference between input and output:
$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - \underbrace{W^T W \mathbf{x}_n}_{\mathbf{y}_n}\|_2^2$$

Nonlinear Problems
Capture non-linear problems by adding non-linear activation function φ and more intermittent layers:
$$\mathbf{y}_n = \varphi_L\left(W_L \varphi_{L-1}\left(\cdots W_2 \varphi_1\left(W_1 \mathbf{x}_n\right)\right)\right)$$
$$\tilde{\mathbf{x}}_n = W_1^\top \varphi_2\left(\cdots W_{L-1}^\top \varphi_L\left(W_L^\top \mathbf{y}_n\right)\right)$$

Note: Deeper networks increase expressiveness but are easier to overfit and memorize the training dataset.