Nevrale nettverk og Dyplæring

ELMED219: Momentliste D01–D13

ELMED219

Vår 2026

Oversikt

1. Grunnleggende nevrale nettverk
2. Trening av nevrale nettverk
3. Konvolusjonelle nevrale nettverk (CNN)
4. Regularisering og avanserte teknikker

D01: Sammenligne biologiske og kunstige nevroner

Biologisk nevron:

- **Dendritter:** Mottar signaler
- **Cellekropp:** Prosesserer
- **Akson:** Sender videre
- **Synapse:** Kobling til neste nevron
- Kompleks elektrokjemisk aktivitet

Kunstig nevron (perseptron):

- **Inputs** $x_1, x_2, \ldots$: Mottar data
- **Vekter** $w_1, w_2, \ldots$: Synapsestyrke
- **Sum:** $z = \sum w_i x_i + b$
- **Aktivering:** $a = f(z)$ (f = aktiveringsfunksjon)
- Matematisk forenkling

Nøkkellikhet

Begge summerer inngående signaler og "fyrer" (aktiverer) hvis total stimulering overstiger en terskel.

Viktig forskjell

Kunstige nevroner er **kraftig forenklede modeller** – de fanger ikke hjernens fulle kompleksitet.

D02: Beskrive oppbygningen av et multilags perseptron (MLP)

MLP-arkitektur:

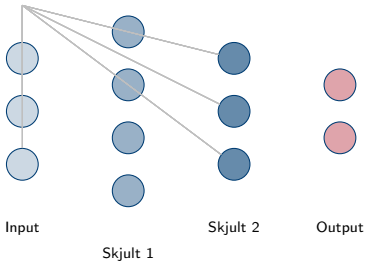
- **Inputlag:** Mottar features (ikke prosessering)
- **Skjulte lag:** 1 eller flere lag med nevroner
- **Outputlag:** Produserer prediksjon

Fully connected (dense):

- Hvert nevron i ett lag er koblet til **alle** nevroner i neste lag
- Mange parametre (vekter og bias)

"Dyp" læring:

- ≥ 2 skjulte lag = dyp modell
- Flere lag $\rightarrow$ mer abstrakte representasjoner



D03: Forklare hva en aktiveringsfunksjon er (ReLU, sigmoid)

Hvorfor aktiveringsfunksjoner? Uten dem: Hele nettverket = lineær transformasjon. Introduserer **ikke-linearitet**.

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

- Output: $(0, 1)$
- Problem: *vanishing gradients*
- Brukes i output for binær klassifisering

ReLU: $\text{ReLU}(z) = \max(0, z)$

- Output: $[0, \infty)$
- Moderne standard for skjulte lag
- Rask, unngår vanishing gradients

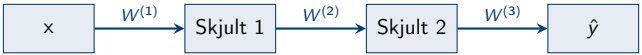
Softmax (multi-klasse output)

$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ – gir sannsynlighetsfordeling over klasser (summerer til 1)

D04: Forstå konseptet forward propagation

Forward propagation = fremoverberegning

1. **Input:** Data x (features) legges inn i inputlaget
2. **For hvert lag:**
   - Beregn vektet sum: $z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}$
   - Appliser aktivering: $a^{(l)} = f(z^{(l)})$
3. **Output:** Prediksjon $\hat{y}$ fra siste lag



Nøkkelpunkt

Forward propagation beregner prediksjon – ingen læring skjer her. Læring skjer ved backpropagation og oppdatering av vekter.

D05: Forklare backpropagation på et konseptuelt nivå

Backpropagation = bakoverberegning av feil

Hovedidé:

1. **Beregn feil:** Sammenlign prediksjon $\hat{y}$ med fasit y
2. **Propager bakover:** Hvor mye bidro hver vekt til feilen?
3. **Kjerneregelen:** Deriver feil m.h.p. hver vekt i nettverket
4. **Oppdater vekter:** Juster for å redusere feil

Intuisjon

Tenk deg at du justerer en kompleks maskin: Backpropagation forteller deg hvilke skruer (vekter) du skal skru på, og i hvilken retning, for å forbedre resultatet.

Matematisk (forenklet)

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w}$

(Kjerneregelen for derivasjon propagerer gradienter bakover gjennom nettverket)

D06: Forstå gradient descent og læringsrate

Gradient descent:

- Optimiseringsalgoritme for å **minimere loss-funksjonen**
- Følger den **negative gradienten** (bratteste nedoverbakke)

Oppdateringsregel:

$$w_{\text{ny}} = w_{\text{gammel}} - \eta \cdot \frac{\partial L}{\partial w}$$

Læringsrate η:

- Bestemmer **steglengde**
- For stor: Hopper over minimum
- For liten: Treg konvergens
- Typisk: 0.001–0.01

Varianter:

- **Batch GD:** Alle data per oppdatering
- **Stochastic GD:** Ett eksempel om gangen
- **Mini-batch GD:** Små grupper (32–128)
- **Adam:** Adaptiv læringsrate (populær!)

Epoch

En **epoch** = én gjennomgang av hele treningsdatasettet

D07: Kjenne til loss functions (cross-entropy, MSE)

Loss function (tapsfunksjon):

- Måler **hvor feil** modellens prediksjoner er
- Målet: Minimere loss under trening

MSE (Mean Squared Error):

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Brukes for **regresjon**
- Straffes hardt for store feil
- Kontinuerlig output

Cross-Entropy:

$$L = -\sum_{i} y_i \log(\hat{y}_i)$$

- Brukes for **klassifisering**
- Binary: $-[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
- Måler avvik mellom sannsynlighetsfordelinger

Velge riktig loss

Regresjon: MSE/MAE **Binær:** Binary CE **Multi-klasse:** Categorical CE

D08: Beskrive et konvolusjonelt nevralt nettverk (CNN)

CNN = spesialisert for bildedata

- Utnytter **romlig struktur** i bilder
- Langt færre parametre enn fullt koblet nettverk
- Translasjonsinvariant – gjenkjenner mønstre uansett posisjon

Typisk CNN-arkitektur:

Hierarkisk læring:

- Tidlige lag: Enkle features (kanter, teksturer)
- Senere lag: Komplekse features (former, objektdeler)
- Siste lag: Høynivå konsepter (objekter, kategorier)

D09: Forklare hva et konvolusjonsfilter gjør

Konvolusjonsfilter (kernel):

- Liten matrise (f.eks. 3×3) som **glir over bildet**
- Beregner punktprodukt mellom filter og bildepatch
- Produserer et **feature map**

Eksempel: Kantdeteksjon

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- Detekterer horisontale kanter
- Ulike filtre $\rightarrow$ ulike features
- CNN **lærer** optimale filtre!

Nøkkelbegreper:

- **Stride:** Hvor langt filteret flyttes
- **Padding:** Legge til piksler i kanten
- **Kanal:** RGB = 3 kanaler
- **Feature map:** Output fra konvolusjon

Vektdeling

Samme filter brukes over hele bildet $\rightarrow$ dramatisk færre parametre enn fullt koblet nettverk.

D10: Beskrive pooling-lag og deres funksjon

Pooling = nedskalering av feature maps

Vanligste type: Max pooling

- Velger **maksverdi** i hvert område (f.eks. 2×2)
- Reduserer romlig størrelse med faktor 2

Fordeler med pooling:

- **Reduserer beregning** – færre parametre
- **Translasjonsinvarians** – litt skift påvirker ikke output
- **Unngår overtilpasning** (overfitting)

Eksempel (2×2 max pool):

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \rightarrow 4$$

$$\begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix} \rightarrow 5$$

Andre pooling-typer

- **Average:** Gjennomsnitt
- **Global avg:** Feature map $\rightarrow$ én verdi

D11: Kjenne til batch normalization og dropout

Batch Normalization:

- Normaliserer aktiveringer i hvert mini-batch
- $\hat{x} = \frac{x-\mu}{\sigma}$, deretter skaler/skift
- Fordeler:
  - Raskere trening
  - Stabiliserer læring
  - Tillater høyere læringsrate
- Plasseres typisk etter konvolusjon, før aktivering

Dropout:

- "Slår av" tilfeldige nevroner under trening
- Typisk dropout rate: 0.2–0.5
- Fordeler:
  - Reduserer **overtilpasning**
  - Tvinger nettverk til å være robust
  - Fungerer som ensemble
- Brukes kun under trening!

Regulariseringsteknikker

Både batch normalization og dropout hjelper med å unngå overtilpasning og forbedre generaliseringsevne.

D12: Forstå konseptet transfer learning

Transfer learning:

- Gjenbruk en modell trent på ett problem til et nytt problem
- Spesielt nyttig når du har **lite data**

Typisk fremgangsmåte:

1. Ta en pretrent modell (f.eks. trent på ImageNet – millioner av bilder)
2. Frys tidlige lag (generelle features)
3. Erstatt/tren siste lag for din spesifikke oppgave
4. (Valgfritt) Fintun hele modellen med lav læringsrate

Hvorfor fungerer det?

Tidlige lag lærer **generelle features** (kanter, teksturer) som er nyttige for mange oppgaver. Kun de siste lagene er oppgavespesifikke.

Eksempel i medisin

Tren på millioner av naturlige bilder → Fintun på tusenvis av røntgenbilder → Bedre resultat enn å trene fra scratch!

D13: Kjenne til avanserte arkitekturer (ResNet, ViT)

ResNet (Residual Networks):

- Introduserte **skip connections**
- Lar gradienter flyte direkte
- Muliggjør svært dype nettverk (50–152+ lag)
- $y = F(x) + x$ (residual block)

Viktig bidrag:

- Løste problemet med degradering i dype nettverk
- Standard for medisinsk bildeanalyse

ViT (Vision Transformer):

- Anvender transformer-arkitektur på bilder
- Deler bilde i patches ("tokens")
- Bruker self-attention
- Skalerer godt med data og compute

Moderne trend:

- Overgår CNN på store datasett
- Brukes i state-of-the-art modeller

Andre viktige arkitekturer

U-Net: Segmentering (medisinsk klassiker) | **EfficientNet**: Balansert skalering | **DenseNet**: Tette koblinger

Oppsummering: Dyplæring

Grunnleggende:

- D01–D03: Nevroner, MLP, aktiveringsfunksjoner
- D04–D07: Forward/backprop, gradient descent, loss functions

CNN:

- D08–D10: Arkitektur, konvolusjonsfiltre, pooling
- Hierarkisk læring av bildefeatures

Avansert:

- D11: Batch normalization, dropout (regularisering)
- D12: Transfer learning (gjenbruk av pretrent kunnskap)
- D13: ResNet (dype nettverk), ViT (transformers for bilder)

Lab 2 – Praktisk erfaring

Bygg, tren og evaluer CNN med PyTorch. Bruk Grad-CAM for å forstå modellbeslutninger.