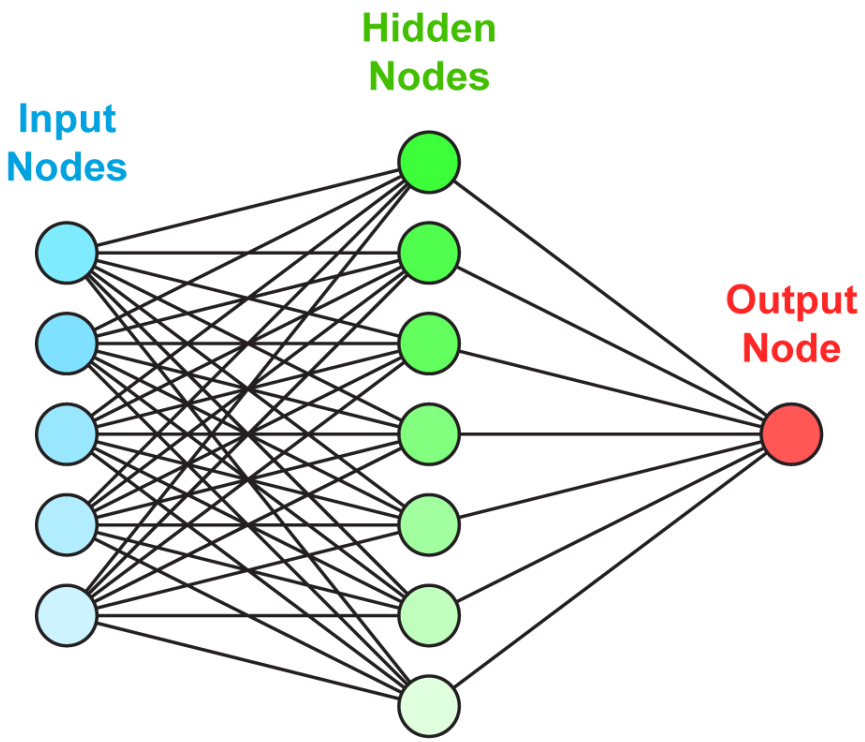
Oppgaver til workshop med Westerdals

Kursinnhold

- Praktisk introduksjon til maskinlæring i Java
  - Laste inn og forhåndsprosessere et dataset
  - Trene opp et neural nettverk
  - Validere resultat
- Gi forståelse for hvordan maskinlæring kan brukes i prosjekter
- Gi kjennskap til noen utfordringer/fallgruver som finnes



Koden

Git

Git-repo: https://github.com/netcompanyno/fagkveld-ml.git

```
git clone https://github.com/netcompanyno/fagkveld-ml

cd fagkveld-ml

mvn clean install

cd src/main/java/com/netcompany/machinelearning/data/
```

Pakk ut datasettet:

Mac:

```
gunzip -c mnist_png.tar.gz | tar xopf -
```

MinGW (git bash)/MSYS eller Cygwin:

```
tar -xzvf mnist_png.tar.gz
```

Datasettet

Hele datasettet (MNIST) består av 70,000 bilder av håndskrevne tall, delt inn i trenings- og testdata. De er 28x28 piksler store og lagret som .png. Bildene blir lest inn som en tre-dimensjonal array:

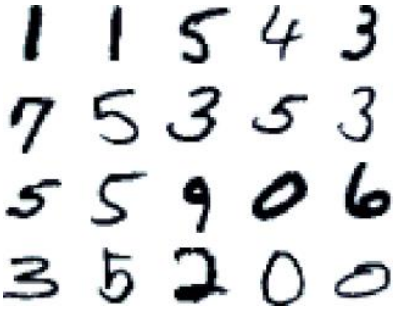
```
data = array[antall bilder][høyde][bredde]
```

Eksempel: data[3][0][0] vil være pikselen øverst til venstre i det 4. bildet.

MINIST er et kjent datasett for benchmarking innen maskinlæring (bildegjenkjenning) og brukes ofte for å måle ytelsen til forskjellige algortimer (i alle fall tidligere – idag er det rett og slett litt for enkelt). De beste har en suksessrate på over 99,7 %. Mer info finnes her https://en.wikipedia.org/wiki/MNIST_database

Data:

- Hvert bilde er på 28x28 piksler
- Treningsdata: 60,000 bilder.
- Testdata: 10,000 bilder.
- Ti klasser (0 - 9)



Oppgavene

Det er fem oppgaver i nedenfor. Er du usikker eller trenger hint, kan man sjekke ut neste oppgave som vil inneholde et løsningsforslag til forrige oppgave.

```
git checkout Oppgave1
git checkout Oppgave2
git checkout Oppgave3
git checkout Oppgave4
git checkout Oppgave5
```

De fire første oppgavene utføres i `Oppgaver` klassen under `nevraltNettverk` pakken.

Øverst i `main`-metoden i `Oppgaver` finner du et `lesInnHeleDatasettet`-flagg som du kan brukes ved testing/debugging, slik at du slipper å vente på at alle 70'000 bildene lastes inn hver gang.

Innlest data finner du i dataLaster objektet.

Oppgave 1: Preprosessering

```
git checkout Oppgave1
```

1. Før vi kan fôre bilder inn i et nevralt nettverk, må vi ha dataene på et format nevrale nettverk forstår. Vi begynner derfor med å flate ut bildene:
    - Nå representeres hvert bilde i en to-dimensjonal representasjon. Konverter representasjonen av hvert bilde til èn dimensjon.

2. Deretter vil vi at opptreningen av nettet vårt skal konvergere raskest mulig. For å få til dette, lønner det seg å normalisere dataene (gjøre så alle features (her: piksler) får verdier fra samme "range")

    1.
    - Konverter pikselene i bildene fra nåværende verdier (gråskala [0, 255]) til flyttall i range [-1, 1].
        - Ekstra: Hvis du vil være ekstra pro -> gjør så dataene får standardavvik = 1 og gjennomsnitt 0. (Kanskje lurt å spare dette som en ekstraoppgave til slutt, siden det ikke er absolutt nødvendig...)

Oppgave 2: Bygg nettverket

Vi har basert koden på et maskinlæringsrammeverk kalt Deeplearning4j. For at dette ikke skal bli et kurs i dl4j, har vi abstrahert det bort og eksponerer kun konsepter du nesten alltid vil komme borti i maskinlæring. Om du er nysgjerrig på hvordan ting gjøres i dl4j, så er det fritt frem for å grave i koden! For å skille mellom hva som er "vår" funksjonalitet og hva som er dl4j sin, har vi skrevet vårt "rammeverk" på norsk.

1. I koden finner du en `NevraltNettverkBygger` som du kan bruke til å lagvis bygge et nevralt nettverk, fra toppen (input) og ned (output).

    - Bestem hvor mange lag nettverket ditt skal ha og hvor mange noder hvert enkelt lag skal inneholde (husk at hvert lag må ha samme antall verdier inn som laget over hadde verdier ut)
    - Sett deretter antall epoker (hvor mange ganger skal dataene itereres over ved trening) og batch-størrelse (hvor mange datainstanser (her: bilder) skal leses mellom hver oppdatering av vektene i nettet). Du finner litt mer info om disse i Javadoc'en.

Oppgave 3: Tren og test

1. Du har nå data på riktig format og et nevralt nettverk i stand til å spise dem, så da er det egentlig bare å lære det opp! I `Oppgaver` finner du et `lesInnHeleDatasettet`-flagg som du kan brukes ved testing/debugging, slik at du slipper å vente på at alle 70'000 bildene lastes inn hver gang.

    Nå som du har et ferdig trent nettverk, er det på tide å se hvordan det presterer på testdataene.
2. `NevraltNettverk` har metoden `evaluer` for dette. Bruk denne til å evaluere modellen på testdataene. Input skal være på samme format som ved trening.

Oppgave 4: Lek deg med parametre

Lek deg og se betydning av forskjellige parametre og arkitekturer for nettet (arkitektur = antall lag og noder). Se f.eks. på hvordan kjøretid og ytelse påvirkes.

Og for all del: **La det gå konkurranse i å få beste resultat!** Innen maskinlæring elsker vi nemlig konkurranser.

Merk at dette datasettet er veldig "enkelt". Med andre ord er det vanskelig å få direkte dårlige resultater.

Lei av å se grafen som dukker opp hver gang du kjører? Kommenter ut linje 94 i NevraltNettverk klassen.

Oppgave 5: Implementer AI i app

I `SifferKlassifikatorApp` tilbyr vi et rimelig frekt GUI for å vise og klassifisere bilder av håndskrevne siffer. Dessverre var denne appen laget før maskinlæring ble populært, og derfor utfører den klassifisering som i gamle dager – kun ved å gjette et tall mellom 0 og 9 😟

Kan DU hjelpe oss å implementere AI i denne, slik at vi kan holde tritt med resten av industrien? Dette bør la seg gjøre ved å endre på `SifferKlassifikator`-klassen. Start appen ved å kjøre `SifferKlassifikatorApp.start()`.

Lag dine egne håndskrevne tall med en bildeeditor (online paint versjon her, Chrome anbefales) og mat appen med tallene dine. Må være 28x28 pixler, fargelegg bakgrunnen svart og skriv inn tallet med hvit farge. Eller bruk et tilfeldig bilde fra trening eller test-settet.

Ekstraoppgaver:

- **DL4J Tutorial:** Prøv forskjellige eksempler på DL4J eller fortsett og lek med MNIST med flere parametre her.
- Om du har en Nvidia-GPU som støtter CUDA, kan du sjekke om du får dl4j til å kjøre på GPU-en din (nå gjøres kalkulasjoner av CPU). Treningen av nettet ditt bør gå utrolig mye raskere om du får det til.
    - https://deeplearning4j.org/gpu

Hvor gode er dine resultater?

Her finner du en oversikt over datasetet som er brukt sammen med de beste resultatene som er oppnådd med en rekke forskjellige implementasjoner.