NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Eksamensoppgave i TDT4102 – Prosedyre- og objektorientert programmering

Faglig kontakt under eksamen: Lasse Natvig

Tlf.: 906 44 580

Eksamensdato: 15 August 2018

Eksamenstid (fra-til): kl. 0900 - 1300

Hjelpemiddelkode/Tillatte hjelpemidler: C: Spesifiserte trykte og håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt. Walter Savitch, Absolute C++

Målform/språk: Bokmål

Antall sider: 10 inkludert vedlegg

Vedlegg 1: Ark om C++11 (1 side)

Kontrollert av:

Informasjon om trykking av eksamensoppgave
Originalen er:
1-sidig ☐ **2-sidig** ☐
sort/hvit ☐ **farger X**
skal ha flervalgskjema ☐

Dato Sign

Merk! Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Generell introduksjon

Les gjennom oppgavetekstene nøye. Noen av oppgavene har lengre tekst, men dette er for å gi kontekst, introduksjon og eksempler til oppgavene.

Når det står "*implementer*" eller "lag" skal du skrive en fungerende implementasjon: hvis det handler om en funksjon skal du skrive deklarasjonen med returtype og parametertype(r) og hele funksjons-kroppen.

Når det står "*deklarer*" er vi kun interessert i funksjons- eller klassedeklarasjonen. Typisk vil dette være deklarasjoner du vanligvis finner i header-filer.

Hvis det står "*forklar*" står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte tekst-forklaringer og vær kort og presis.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger du finner nødvendig.

Hver enkelt oppgave er ikke ment å være mer omfattende enn det som er beskrevet. Noen oppgaver fokuserer bare på enkeltfunksjoner og da er det utelukkende denne funksjonen som er tema. Andre oppgaver er "oppskriftsbasert" og vi spør etter funksjoner som utgjør deler i et program, eller forskjellige deler av en eller flere klasser. Du kan velge selv om du vil løse dette trinnvis, eller om du vil lage en samlet implementasjon, men sørg for at det går tydelig frem hvilke spørsmål du har svart på hvor i koden din. Husk at funksjonene du lager i en deloppgave ofte er ment å skulle brukes i andre deloppgaver.

All kode skal være i C++. Det er ikke viktig å huske helt korrekt syntaks for bibliotekfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt godt kjent med i øvingsopplegget, eller som står beskrevet i læreboka.

Det er ikke nødvendig å ha med include-statement eller vise hvordan koden skal lagres i filer.

Hele oppgavesettet er arbeidskrevende og det er ikke forventet at alle skal klare alt. Tenk strategisk i forhold til ditt nivå og dine ambisjoner!

Deloppgavene i de "tematiske" oppgavene er organisert i en logisk rekkefølge, men det betyr ikke at det er direkte sammenheng mellom vanskelighetsgrad og nummereringen av deloppgavene.

Hoveddelene av eksamensoppgaven teller i utgangspunktet med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan likevel bli justert ved sensur basert på hvordan oppgavene har fungert. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

Oppgave 1: Kodeforståelse (18 %)

Svar på denne måten på et vanlig svarark:
1a) … ditt svar her
1b) … ditt svar her … osv.

1a) Hva skrives ut ?

```
int a = 10;
int b = 20;
int c = ++b;
int d = b--;
cout << c << " " << d << " ";
cout << 'D' - 'A' << " " << 20 % 3;
```

1b) Funksjonen funFunc er definert i del (i), hva skrives ut av koden i del (ii)?

(i)

```
int funFunc(int &a, int b, int* c) {
      a = 3;
      b = -200;
      *c = 1;
      return (a + b + *c);
}
```

(ii)

```
int i = 0;
int *k = new int(20);
cout << funFunc(i, i, k)
     << " ";
cout << i << " " << *k;
```

1c) Hva skrives ut ?

```
string s1 = "abcd";
string s2("123");
string s3, s4;
s3 = s1 + s2;
s4 = s1.substr(2, 2);
cout << s3 << " " << s3.length() << " ";
cout << s3[2] << " " << s4;
```

1d) Funksjonen charSum er definert i del (i), hva skrives ut av koden i del (ii)?

(i)

```
int charSum(char* cString) {
    int sum = 0;
    if (*cString != '\0') {
       int diff = 0;
       diff = (*cString - 'A');
       cString++;
       sum = diff + charSum(cString);
    }
    return(sum);
}
```

(ii)

```
char letters[5] = "ABCD";
cout << charSum(letters);
```

1e) Funksjonen divide er definert i del (i), hva skrives ut av koden i del (ii)?

(i)

```
int divide(int numerator,
           int denominator) {
   if (denominator == 0) {
      throw  logic_error(
      "Denominator is 0");
   }
   return numerator /
          denominator;
}
```

(ii)

```
try {
   cout << divide(10, 2) << " ";
   cout << divide(10, 0) << " ";
   cout << divide(10, 1) << " ";
}
   catch (logic_error &e) {
      cout << "Exception: "
           << e.what() << endl;
}
```

1f) Hva skrives ut ?

```
vector<double> vec = { 11.1, 22.2, 33.3 };
vector<double>::iterator it = vec.begin();
while (it != vec.end()) {
        cout << *it << " ";
        it++;
}
```

1g) Funksjonen test er definert i del (i), hva skrives ut av koden i del (ii)?

(i)

```
template<class T>
void test(vector<T> vec) {
   set<T> nums;
   for (auto e : vec) {
      if (nums.find(e) == nums.end()) {
         nums.insert(e);
      }
      else {
         cout << "found " << e;
      }
   }
}
```

(ii)

```
vector<int> intVec =
   {11, 12, 15, 14, 10};
vector<double> dVec =
   {1.1,1.2,1.5,1.5,3.0};
vector<char> charVec =
   { 'a', 'b', 'c' };

test(intVec);
test(dVec);
test(charVec);
```

1h) Hva skrives ut ?

```
queue<int> q;
q.push(3);
q.push(2);
q.push(1);
cout << q.front() << " " << q.size() << " ";
q.pop();
q.push(4);
cout << q.back();
```

Oppgave 2: Observasjoner (12 %)

2a) Det er deklarert en datatype `Obs` som vist til venstre i Figur 1. Skriv et hovedprogram som forsøker å åpne en tekstfil med navnet `input.txt`. Dersom åpning av filen feiler skal programmet skrive ut en feilmelding og så terminere. Dersom det lykkes kan du anta at filen inneholder en eller flere observasjoner, der hver observasjon er gitt som to positive heltall som gir plassering av observasjonen som ett x-koordinat og ett y-koordinat. Etter siste observasjon er det en linje med tallet -1. Programmet ditt skal lese alle observasjonene i filen og lagre hver i en instans av datatypen `Obs`. Samlingen av alle innleste observasjoner skal lagres i en `vector<Obs>`. Et eksempel på filen som skal leses er til høyre i Figur 1.

```
struct Obs {
       int x;
       int y;
};
```

```
20 30
22 33
44 55
22 33
22 34
23 33
20 30
-1
```

Figur 1.

2b) Definer `operator<` for `Obs` slik at en observasjon A er mindre enn observasjon B dersom A sin x-koordinat er mindre enn B sin x-koordinat. Dersom A og B har like x-koordinater skal det sammenliknes på y-koordinatet.

2c) Skriv en funksjon `void report(vector<Obs> vec, int threshold)` som tar inn som første argument en vector av observasjoner. Funksjonen skal skrive ut x- og y-koordinater for de steder (x,y-par) som har *flere* enn `threshold` forekomster i filen. (Hint: Det er mange måter å løse dette på. Du kan anta at oppgave 2b) er riktig løst slik at du kan bruke `map` dersom du ønsker det). Hvis input-fila er som i Figur 1 og threshold har verdien 1 så skal linjene `"20 30 2"` og `"22 33 2"` skrives ut.

Oppgave 3: Bildebehandling (40%)

I denne oppgaven skal du skrive kode for digital bildebehandling. Et digitalt bilde består av en mengde piksler (bildeelementer) i et rutenett. Hver piksel har en farge basert på RGB, dvs. en blanding av de tre primærfargene **R**ød, **G**rønn og **Blå**. Hver av de tre fargekomponentene er representert som et heltall mellom 0 og 255. For eksempel er RGB(127, 0, 127) en blanding mellom rødt og blått, dvs. fiolett. Fargen RGB(0, 0, 0) er svart mens RGB(255, 255, 255) er hvitt.

3a) Skriv en funksjon `float clip(float n, float lower, float upper)` som "klipper" verdien n til området `lower` - `upper`. Dvs. hvis n er mindre eller lik lower, så skal funksjonen returnere lower. Tilsvarende for upper: om n er større eller lik upper, skal funksjonen returnere upper. Hvis n ligger mellom lower og upper, så skal funksjonen returnere n uendret.

3b) Deklarer en `struct Color` med tre medlemsvariabler r, g, b for de tre farge-komponentene rød, grønn og blå. Hvilken datatype er hensiktsmessig å bruke for r, g, b? Forklar kort.

Deklarer og implementer to konstruktører for Color:
`Color()` skal sette fargen til svart.
`Color(int red, int green, int blue)` setter fargeverdiene som gitt i parameterlista.

Sørg for at alle verdiene er gyldige ved å klippe dem til området 0-255.

3c) Vi deklarerer klassen `Image` som representerer et bilde, og koden er vist nedenfor. Vi bruker en endimensjonal tabell (`data`) til å lagre fargeverdier til alle pikslene. Vi må altså regne om fra en todimensjonal koordinat (x, y) i bildet til en endimensjonal indeks i tabellen `data`. (Se også oppgave 3g).

Implementer konstruktøren `Image::Image(int width, int height)` som initialiserer alle medlemsvariable og allokerer en ny tabell med plass til width x height antall piksler.

```
class Image {
private:
	Color * data; // Array of pixels
	unsigned int width; // Width of image
	unsigned int height; // Height of image

public:
	Image(int width, int height);
	Image(const Image &other);
	~Image();
	Image& operator=(Image rhs);
	Color getPixel(int x, int y);
	void setPixel(int x, int y, Color c);
	Image grayscale();
	Image threshold(unsigned int t);
	Image operator+(Image other);
	Color applyKernel(int x, int y, Kernel k);
	Image convolve(Kernel k);
};
```

3d) Implementer kopikonstruktøren.

3e) Implementer destruktøren.

3f) Hvorfor må vi implementere tilordningsoperatoren? *Forklar* kort.

3g) En piksels plassering angis av koordinatet (x, y) hvor (0,0) er øverst i venstre hjørne mens (width-1, height-1) er nederst i høyre hjørne. Tabellen `data` inneholder fargeverdiene til alle pikslene i bildet hvor radene ligger etter hverandre (først kommer alle pikslene i rad 0, deretter pikslene i rad 1 osv). Implementer funksjonen Image::getPixel() som returnerer fargeverdien til pikselen (x, y). Implementer også funksjonen Image::setPixel() som setter fargeverdien for pikselen (x, y).

3h) Vi skal nå skrive kode for enkel bildebehandling. Når farge-komponentene R,G,B alle er like, blir fargen en gråtone. For eksempel er RGB(0, 0, 0) svart, RGB(127, 127, 127) grått og RGB(255, 255, 255) hvitt. En enkel måte å konvertere et fargebilde til svart-hvitt er ved å endre alle pikslene til gjennomsnittet av fargekomponentene R,G,B. Implementer funksjonen `Image::grayscale()` som returnerer en svart-hvitt kopi av bildet.

3i) En vanlig bildeoperasjon er «thresholding» som sammenlikner fargeverdiene til alle pikslene opp mot en terskelverdi (Eng. threshold). For hver fargekomponent, hvis verdien er større eller lik terskelverdien, blir den nye fargekomponenten 255, og 0 ellers. Implementer funksjonen `Image::threshold(unsigned int t)` som returnerer en kopi av bildet der pikslene har blitt modifisert som beskrevet over, hvor `t` angir terskelverdien.

3j) Vi kan kombinere to bilder til et nytt bilde ved å legge sammen pikselverdiene til begge bildene og dele på 2. Med andre ord de to pikselene i samme posisjon i bilde A og bilde B kombineres til en piksel-verdi i samme posisjon i resultatbildet C . Overlagre `operator+` for Image som returnerer et nytt bilde som er denne kombinasjonen av de to bildene. Hvis bildene ikke er like store skal det kastes et unntak.

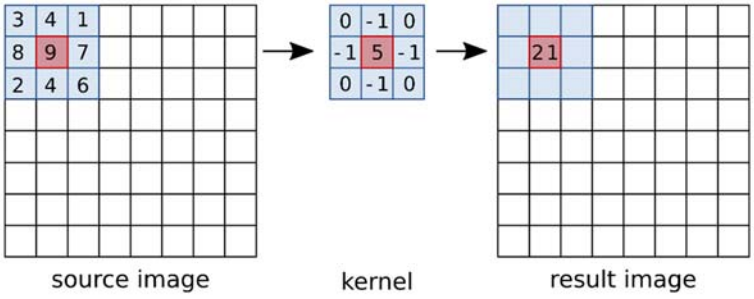
3k) Så langt har vi jobbet med bildeoperasjoner der den nye verdien til hver piksel bestemmes basert på kun den enkelte piksels verdi. En kraftigere bildeoperasjon er *konvolusjon* (Eng. convolution) der også nabopikslene er med på å bestemme den nye fargeverdien. Med konvolusjon kan man f.eks. gjøre bildet skarpere, uskarpt, finne kanter, osv.

Et eksempel på konvolusjon er et såkalt "sobelfilter" som fremhever kanter i bildet. Hvis vi bruker et slikt filter på bildet i Figur 2 så blir resultatet som vist i Figur 3.

Figur 2: Originalbilde (Foto: Lasse Natvig)

Figur 3: Resultat etter konvolusjon med et "sobelfilter"

I konvolusjon blir den nye pikselverdien beregnet som *en vektet sum* av pikselen selv og nabopikslene. Dette gjøres for hver av de tre fargekomponentene. Vektene bestemmes av en "kernel" som vist i Figur 4. Den midterste verdien i kernelen er vekten til den gjeldende pikselen, mens verdiene rundt er vekten til nabopikslene. Vi skal i denne oppgaven fokuserer på 3x3 kerneler.



Figur 4 Konvolusjon av en piksel med en 3x3 kernel.

For eksempelet i figuren blir den nye verdien til en av de tre fargekomponentene til pikselen (merket med rødt):

$$0 \cdot 3 - 1 \cdot 4 + 0 \cdot 1 - 1 \cdot 8 + 5 \cdot 9 - 1 \cdot 7 + 0 \cdot 2 - 1 \cdot 4 + 0 \cdot 6 = 21$$

Denne prosedyren repeteres *individuelt for hver fargekomponent* for alle pikslene i bildet. Vi deklarerer en datatype Kernel som er en 3x3 tabell:

```
typedef float Kernel[3][3];
```

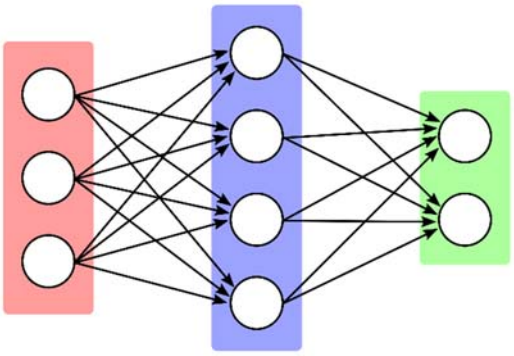
Merk at vi bruker flyttall fordi det ofte er nyttig med flyttallsvekter.

Implementer funksjonen `Color Image::applyKernel(int x, int y, Kernel k)` som regner ut og returnerer den nye fargeverdien for én piksel (x, y) gitt en kernel k. Du kan anta at (x,y) ikke er en av pikslene på kanten, dvs 0 < x < width-1 og 0 < y < height-1.

3l) Implementer funksjonen `Image::convolve(Kernel k)` som går gjennom alle pikslene i bildet og anvender kernelen k. Funksjonen skal returnere et nytt bilde med resultatet av operasjonen. Merk at for pikslene i kanten av bildet så vil noen av naboene ligge utenfor bildet. Det er flere måter å håndtere dette på, men i denne oppgaven skal du la pikslene i kanten forbli uendret.

Oppgave 4: Nevralt nettverk (30%)

Nevrale nett er allsidige datamodeller inspirert av hjernen vår. De kan for eksempel brukes til å kjenne igjen håndskrevne bokstaver eller til å styre roboter. I denne oppgaven skal du implementere en klasse for en vanlig type nevralt nett kalt feedforward nevralt nett (FFNN). Du trenger ikke ha noe kunnskap om temaet for å forstå oppgaven da vi forenkler mye og forklarer hva programkoden skal gjøre.



Figur 5. Nevralt nett med 3 lag. Første laget har 3 noder, neste lag har 4 noder og siste lag har 2 noder.

Figur 5 viser et eksempel på et FFNN. Et FFNN består av et sett med noder (sirkler i figuren) som er organisert i et eller flere lag (firkanter i figuren). Nodene i et lag er koblet sammen med nodene i det neste med såkalte synapser (pilene i figuren). Hver synapse har en vekt som betegner styrken til koblingen. Input flyter fra nodene i første lag (her rød firkant med tre noder) til nodene i neste lag (blått), helt til det når det siste laget (grønt) som output.

Et FFNN består altså av ett eller flere lag, hvor hvert lag har N antall noder (antallet noder er ofte ulikt fra lag til lag).

Outputen a_j til en node beregnes ut fra nodene i det forrige laget a_i (som da blir input) som følger:

$$a_j = f\left(\sum_{i=1}^{M} w_{ij}\, a_i\right)$$

Her er a_j outputen til noden i det gjeldende laget, a_i er outputen til node nummer i fra det forrige laget, og w_{ij} er vekten til synapsen mellom dem. Funksjonen $f(\,)$ kalles *aktiveringsfunksjonen*. Σ betyr summen over alle synapser som går fra node a_i til node a_j. (Det vil for eksempel være en sum med fire ledd for hver av de to nodene i det grønne laget).

I denne oppgaven er lagene fullt sammenkoblede, dvs. at alle noder i et lag er koblet sammen med alle noder i neste lag. Et lag med N noder og M input-synapser har dermed MxN vekter.

4a) Deklarer en klasse `Layer` som representerer et lag i det nevrale nettet.

Klassen skal ha medlemsvariabler for antall noder `N` (int), antall input-synapser `M` (int), en vektor `a` for outputen til nodene (vektor av double) og en vektor `w` for vektene (vektor av double).

Deklarer konstruktøren. Deklarer og implementer inline get-funksjoner for `N` og `M`: getSize() og getInputSize().

NB: vi kommer til å legge til flere funksjoner i de neste deloppgavene, men disse trenger du ikke ta med i deklarasjonen her.

4b) Implementer konstruktøren `Layer::Layer(int N, int M)`. Her er N antall noder i laget og M er antall synapser inn til hver node. Konstruktøren skal opprette vektoren a for outputen til de N nodene og vektoren w for de assosierte NxM vektene. Verdiene i a skal initialiseres til 0 mens verdiene i w skal initialiseres til 1.

4c) Vektene lagres i en 1-dimensjonal vektor der de første M vektene hører til node 0, de neste M vektene hører til node 1 osv. Implementer medlemsfunksjonene setWeight(j, i, weight) og getWeight(j, i) som henholdsvis setter og returnerer vekt nummer i inn til node nummer j.

4d) I første omgang skal vi bruke den enkle aktiveringsfunksjonen f(x) = x. Senere ønsker vi å kunne endre denne i en subklasse av Layer. Vi lager funksjonen Layer::fn(x) for aktiveringsfunksjonen:

```
virtual double Layer::fn(double x) { return x; }
```

Hvorfor er det viktig at denne funksjonen er deklarert som virtual? Forklar kort.

4e) Vi skal nå implementere `Layer::forward(const vector<double> &input)` som brukes til å oppdatere nodene i laget. Funksjonen tar en vektor med input-verdier og oppdaterer a[j] som beskrevet i innledningen av oppgaven. Sjekk at størrelsen på input-vektoren er M. Om den ikke er det skal det kastes et unntak.

4f) Implementer en overlagret versjon av Layer::forward() som tar som argument en peker til ett Layer-objekt som skal brukes som input.

4g) Deklarer klassen `SigmoidLayer` som arver fra `Layer`. Klassen skal være identisk med Layer bortsett fra at aktiveringsfunksjonen f(x) skal defineres slik: f(x) = 1 / (1 + exp(-x)) (Dette kalles en sigmoid-funksjon). Implementer konstruktøren og aktiveringsfunksjonen inline.

4h) Vi vil nå lage en klasse `NeuralNet` som inneholder et eller flere lag:

```
class NeuralNet {
protected:
    vector<Layer *> layers;
public:
    NeuralNet() {}
    void addLayer(Layer *layer);
    Layer *forward(const vector<double> &input);
};
```

Implementer medlemsfunksjonen addLayer() som legger til et lag i enden av nettverket (bakerst, på output-siden). Om det allerede finnes lag i nettverket, skal funksjonen sjekke at størrelsen på det siste laget stemmer med input-størrelsen på det nye laget. Bruk assert.

4i) Implementer funksjonen forward() som oppdaterer hele nettverket. Dvs. først gjør forward på første lag med input, deretter neste lag osv. Funksjonen skal returnere en peker til det siste laget. Du kan anta at nettverket har minst ett lag.

…---ooo0ooo---…